BERICHTE

AUS DEM DEPARTMENT FÜR INFORMATIK
der Fakultät II - Informatik, Wirtschafts- und Rechtswissenschaften

Herausgeber: Die Professorinnen und Professoren
des Departments für Informatik

Structural Stationarity in the π-Calculus

Roland Meyer

Dissertation

Nummer 02-09 – Februar 2009

ISSN 1867-9218

Gutachter: Prof. Dr. E.-R. Olderog
Prof. Dr. E. Best
Prof. Dr. D. Sangiorgi (Uni Bologna)

Datum der Einreichung: 29.11.2008
Datum der Verteidigung: 20.02.2009

Author´s address:
Roland Meyer
Fakultät II, Department für Informatik
Abteilung „Entwicklung korrekter Systeme“
26111 Oldenburg
Germany

E-mail: Roland.Meyer@liafa.jussieu.fr

CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Structural Stationarity in the π-Calculus

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

vorgelegt von

Dipl.-Inform. Roland Meyer

Gutachter:

Prof. Dr. Ernst-Rüdiger Olderog
Prof. Dr. Eike Best
Prof. Dr. Davide Sangiorgi

Tag der Disputation: 20. Februar 2009

Abstract

Dynamically reconfigurable systems (DRS) permeat our daily lifes, and their importance in critical application areas where correct functionality is essential increases. Home banking systems where a secure channel has to be established between client and server prior to money transfers, may serve as an example. Already the classical concurrent systems are difficult to design correctly. DRS add to concurrency the problem of evolving connections between system components. To ensure desired system behaviour, program verification techniques are an established means. In particular computer-aided verification has received much attention. This thesis presents finite representations for DRS modelled in the π-Calculus, thus bridging the gap to existing automated verification techniques.

DRS are infinite-state systems with an unbounded number of components and connections. Despite this unboundedness, a large class of DRS exhibits only finitely many patterns of connections at runtime. These systems are called *structurally stationary*. We propose a semantical translation of structurally stationary systems into finite place/transition Petri nets, which highlights the connection patterns. With this *structural semantics*, structurally stationary systems inherit all verification techniques and tools for Petri nets. To demonstrate that our translation-based approach to verification is feasible in practice, we establish correctness properties of different kinds for two *industrial case studies*.

To judge the expressiveness of structurally stationary systems, we present two *complete characterisations*. The first proves structural stationarity for well-known DRS classes from the literature and for *finite handler* systems, which we design to model the client-server architectures in our case studies. The second characterisation shows that structural stationarity is equivalent to boundedness in the novel functions *depth* and *breadth*. The breadth of a DRS corresponds to the connection degree of the components, while the depth measures their interdependence. Searching for finite representations of these larger classes, we find that systems of bounded depth have *well-structured transition systems*, where properties can be decided on a finite prefix of the computation tree. For systems of bounded breadth, we show *Turing completeness*.

Inspired by the decidability result, we aim at recovering a translation into finite place/transition Petri nets for systems of bounded depth. The approach

is to combine the newly developed structural semantics with classical concurrency semantics. Although the resulting *mixed semantics* generalises the previous translations, it does not cover all processes of bounded depth. By proving *undecidability of reachability*, we show that a Petri net translation for the full class does not exist. The undecidability result relies on a class of systems just beyond the capabilities of the mixed semantics. In this sense, we find the *borderline* between DRS and finite place/transition Petri nets.

Zusammenfassung

Dynamisch rekonfigurierbare Systeme (DRS) sind allgegenwärtig und werden zusehends selbst in kritischen Anwendungsgebieten eingesetzt, wo eine korrekte Funktionsweise unerlässlich ist. Als Beispiel sind Home-Banking-Systeme zu nennen, bei denen vor einer jeden Überweisung zunächst eine sichere Verbindung zwischen Client und Server zu erstellen ist. Schon das Design nebenläufiger Systeme ist als sehr schwierig bekannt, bei DRS kommen noch die Probleme der sich ändernden Verbindungsstrukturen hinzu. Um dennoch das gewünschte Systemverhalten sicherzustellen, haben sich Programmverifikationstechniken etabliert. Insbesondere der computergestützten Verifikation wurde viel Aufmerksamkeit geschenkt. In dieser Arbeit werden endliche Darstellungen für DRS vorgestellt, welche die bestehende Lücke zwischen DRS und existierenden automatischen Verifikationsmethoden schließen. Als Modellierungssprache für DRS werden dabei die Prozesse des π-Kalküls genutzt.

DRS sind zustandsunendliche Systeme, bei denen weder die Anzahl der Komponenten noch der Verbindungen beschränkt ist. Trotz dieser Unbeschränktheit zeigen viele DRS während der Laufzeit nur endlich viele Verbindungsmuster. Diese DRS heißen *strukturell stationär*. In der Arbeit wird eine semantische Übersetzung von strukturell stationären Systemen in endliche Stellen-Transitions-Petri-Netze vorgestellt, die die Verbindungsmuster betont. Mit dieser *strukturellen Semantik* erben strukturell stationäre Systeme alle Verifikationstechniken und Tools für Petri-Netze. Um zu belegen, dass der vorgeschlagene übersetzungsbasierte Ansatz zur Verifikation tatsächlich für Systeme der Praxis durchführbar ist, werden verschiedene Korrektheitseigenschaften für zwei *industrielle Fallstudien* nachgewiesen.

Um die Ausdrucksmächtigkeit strukturell stationärer Systeme zu beurteilen, werden zwei *vollständige Charakterisierungen* der Eigenschaft vorgestellt. Die erste beweist strukturelle Stationarität für wohl-bekannte Klassen von DRS aus der Literatur sowie für *Finite-Handler-Systeme*, die vorgestellt werden, um die Client-Server-Architekturen der Fallstudien zu modellieren. Die zweite Charakterisierung zeigt, dass die Eigenschaft der strukturellen Stationarität äquivalent zur Beschränktheit des Systems in den neuen Funktionen *Tiefe* und *Breite* ist. Die Breite eines DRS entspricht dem Verbindungsgrad der Komponenten des System,

während die Tiefe deren wechselseitige Abhängigkeit misst. Bei der Suche nach endlichen Darstellungen dieser größeren Systemklassen zeigt sich, dass Systeme beschränkter Tiefe *wohl-strukturierte Transitionssysteme* haben, die es erlauben, Systemeigenschaften auf endlichen Anfangsstücken der Berechnungsbäume zu entscheiden. Für Systeme beschränkter Breite wird *Berechnungsvollständigkeit* nachgewiesen.

Motiviert durch das Entscheidbarkeitsresultat, wird versucht, eine endliche Petri-Netz Darstellung für die Systeme beschränkter Tiefe zu finden. Der Ansatz ist, die neu entwickelte strukturelle Semantik mit klassischen Nebenläufigkeitssemantiken zu verknüpfen. Obwohl die resultierende *gemischte Semantik* die vorherigen Übersetzungen verallgemeinert, kann sie doch nicht alle Systeme beschränkter Tiefe übersetzen. Durch einen Beweis der *Unentscheidbarkeit der Erreichbarkeit* für Systeme beschränkter Tiefe wird gezeigt, dass eine Übersetzung der gesamten Klasse nicht existiert. Das Unentscheidbarkeitsresultat bedient sich einer Klasse von Systemen, die gerade außerhalb der Fähigkeiten der gemischten Semantik liegen. In diesem Sinne zeigt die vorliegende Arbeit die *Grenzlinie* zwischen DRS und endlichen Stellen-Transitions-Petri-Netzen auf.

Acknowledgements

I am deeply indebted to Ernst-Rüdiger Olderog. As the professor of my very first lectures he introduced me to the areas of program verification and concurrency theory, as the supervisor of my thesis he freely shared his precious insights about research with me. These discussions shaped the way I work and I am grateful for everything I learned from him. Thank you for the beautiful years in your group, the trips, the lunches and dinners, and all the fun we had!

My second supervisor Eike Best is the source of my fascination for Petri nets. Thank you for guiding my interest to the topic! Eike and Maciej Koutny arranged a visit for me to Newcastle from November to December 2008 and I am indebted to both for the wonderful weeks. In Newcastle I met Victor Khomenko, to whom I am grateful for all I learned in our joint work, for the beautiful dinners, and for hosting my stay in March 2008. On a summer school 2006 I met Davide Sangiorgi. His interest in an early version of my Petri net translation may have saved me from discarding the idea. He later agreed to to review this thesis and visited us in Oldenburg for the colloquium. I thank you for your efforts! With Roberto Gorrieri I had numerous email discussions about expressiveness aspects of process algebras, for which I am grateful.

Within the *Correct System Design Group*, my colleagues created a wonderful working atmosphere. From Andreas Schäfer I learned a good deal about life and work and I am happy we became good friends. Tim Strazny agreed to write his Master's thesis under my supervision and spent an incredible amount of work on implementing the results presented here. As colleagues we shared whatever lay on our hearts and I would not like to miss a single minute of discussion or laughter. Thank you! To Sven Linker I am grateful for taking over my group of mentees. I am indebted to Johannes Faber, Jochen Hoenicke, and Andrey Rybalchenko for the (real-)time we shared in the *AVACS* R1 research project. The *AVACS* S2 project on reconfigurable systems admitted me as a spy and I benefitted from discussions with Tobe Toben, Bernd Westphal, and Jörg Kreiker.

My thanks also go to the remaining members of the theory division. To Michael Möller for teaching me about graphics, to Hans Fleischhack and Elke Wilkeit for remarks on results in this thesis, to Harro Wimmel for convincing me to take Mathematics as subsidiary subject, to Sibylle Fröschle for arranging my visit

to Edinburgh in December 2007, to André Platzer, Ingo Brückner, Jan-David Quesel, Mani Swaminathan, and Margarete Muhle for lunch discussions, and to Andrea Göken for help in administrative issues. I especially thank Annegret Habel for introducing me to the area of reconfigurable systems.

As a PhD student in the Graduate school *TrustSoft*, I was surrounded by friends who shared the ups and downs of being a PhD student. For pleasant trips to Dagstuhl as well as nice parties I thank André van Hoorn, Astrid Rakow, Christian Storm, Heiko Koziolek, Henrik Lipskoch, Jens Happe, Karl-Heinz Pennemann, Kinga Kiss-Jakob, Malte Diehl, and Timo Warns. I thank our secretary Ira Wempe for her friendship, travelling organisation, and moral support.

During my studies and throughout my PhD, I was supported by my friends and family at home in Ostfriesland. I am indebted to Thorsten Müller, who kept in touch no matter how many meetings I missed. I thank my mother Lydia Meyer for insisting on and providing me with a good education. Thanks for everything you took upon you! My foremost thanks go to my girlfriend Katrin Lambertus for her constant love and support. You took all the worries from me and I would not have made it through this thesis without you! I love you and feel the greatest respect for you!

Contents

List of Figures

List of Tables

Part I

Structural Stationarity

1 Introduction

Finally the checker has to verify that the process comes to an end.

A. M. Turing 1949

The sentence is quoted from Turing's paper *Checking a Large Routine* [MJ84]. The purpose of early computers was the calculation of mathematical functions. Turing observed a large gap between the mathematical formulation and the actual computation of a function with the commands available in a computer. To ensure that a program computed the function it was designed for, Turing suggested to give a mathematical proof. He advocated the following approach to program development.

A programmer should annotate the code by assertions on the values of variables in the different program locations. A mathematician, called the *checker*, then proves the assertions and shows that they entail correctness of the program's return value. According to Turing, correctness of a program should follow *easily* from the assertions. As an example, he considers a program for computing the factorial function and provides the corresponding table of assertions. They are given in a detail, which makes the correctness proof a plain and simple calculation. It may have come to Turing's mind to automate the procedure.

Turing's article turned out prophetic. 60 years later, *program verification* is an established branch of computer science. A prominent research topic in this area is *computer-aided* verification, searching for algorithms that help a developer proving correctness of a program. In 2007, the ACM Turing award was given to the founders of a fully-automated verification technique, the roots of which lie in the late 70s and early 80s where operating systems required new approaches to verification.

In his seminal work, Turing pays particular attention to proving termination of his factorial program. With the development of new operating systems, the style

computers were used changed. Instead of one, several programs were executed concurrently—either on one processor by interleaving their threads of execution or on separate processors. Moreover, the new operating systems were not meant to terminate but to *process smoothly a continuous flow of user programs* [Dij68]. As a result, new problems—unique to *concurrent systems*—joined the classical ones of correct computation and termination of the separate programs. It had to be guaranteed that every program is eventually assigned the processor, and that different programs synchronise their access to peripheral devices.

In [Pnu77], Pnueli unified the verification of both sequential and terminating as well as concurrent and non-terminating programs, a contribution for which he was awarded the ACM Turing award in 1996. Pnueli's approach understands computing systems in terms of their execution sequences. He then suggests to specify correctness properties in the language of *temporal logic*, which explicitly talks about temporal dependences between events in executions. Pnueli shows that temporal logic captures in a natural way termination in sequential and non-starvation or mutual exclusion in concurrent programs. Shortly afterwards, in 1981 Clarke and Emerson [CE81] and independently in 1982 Queille and Sifakis [QS82] proposed the first algorithms that fully-automatically prove satisfaction of a temporal logic property by a transition system. After 35 years, these still called *model checkers* provided a substitute for Turing's mathematician.

In 2008, it is common practice to realise client-server architectures, even for critical applications like banking systems, over web interfaces. These new *dynamically reconfigurable systems (DRS)* are still concurrent in the sense that they consist of several interacting programs. Different from the classical notion, the number of interacting programs together with their connections—the *configuration*—is not static but evolves over time.[1] Like concurrent systems enjoy problems not present in sequential programs, DRS introduce their own set of obstacles. Most notably, correctness crucially depends on the connection topology, not only on the interaction of programs. For example, a secure channel has to be established between client and server before money can be transferred. Supported by this observation, we claim that DRS form the class of systems, program verification has to face today. This thesis establishes a basis for their computer-aided verification.

1.1 Contribution

Concurrent programs consist of finitely many components, each of which having a finite state space. This finiteness assumption no longer holds for DRS, e.g.

[1] We argue that *statically reconfigurable* systems determine their configuration at startup, e.g. operating systems configure according to the available peripheral devices.

the number of clients trying to access a server is not bounded. In fact, DRS are typically infinite-state systems.

Any automatic verification algorithm requires a *finite representation* of the infinite state space of a DRS. This thesis provides such finite representations, inspired by the following elementary observation in client-server systems. Although the number of connections between clients and server threads is not bounded, there are (essentially) *finitely many patterns* of connections. DRS which satisfy this constraint are called *structurally stationary*. Figure 1.1 gives an example of a structurally stationary DRS. We stress that this decomposition of states into connection patterns focuses on the distinctive feature in DRS: the evolving connection structure. The interaction between system components, which was the main concern in classical concurrent systems, has taken a back seat.

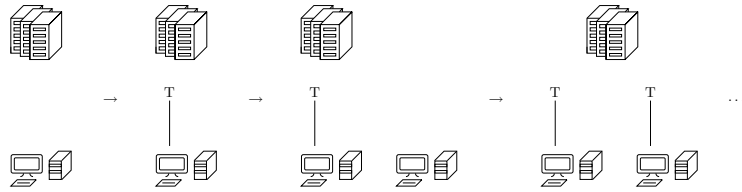


Figure 1.1:
A structurally stationary client-server system. A client (shown at the bottom of the figure) contacts a server (shown at the top), which in response spawns a thread (indicated by T) that handles the requests of the client. There are three patterns of connections in the system: the client alone, the server alone, and the connection between client and thread.

Exploiting the decomposition, we observe that structurally stationary DRS are adequately reflected by *finite place/transition Petri nets*.[2] The translation is rather intuitive. For every possible connection pattern reachable in the DRS, a place is created. For every occurrence of the structure in a state, a token is added to the place. Transitions imitate the interactions between the groups of programs, which lead to state changes. We call the resulting Petri net the *structural semantics* of the DRS, to highlight the difference to the ordinary transition system semantics. For example, the structural semantics of the client-server system above is depicted in Figure 1.2.

Our main results states that the structural semantics does not lose information. The transition systems of DRS and Petri net are isomorphic, and the states of the DRS can be obtained from the states of the Petri net. Hence, to establish

[2] A reader not familiar with Petri nets will find an introduction in Section 2.2. *Finiteness* of a Petri net means finiteness of the sets of places and transitions, the state-space may be *infinite*.

correctness of a DRS, we can compute its Petri net representation and verify the latter with existing techniques. This approach has a number of advantages. It does not depend on a concrete verification algorithm, but—with the structural semantics—structurally stationary systems inherit all techniques and tools for place/transition Petri nets. In particular, the positive results on verification of infinite state Petri nets become applicable for structurally stationary system. Finally, we demonstrate that correctness of connections, a crucial property in DRS, can be established efficiently on our Petri net representation.

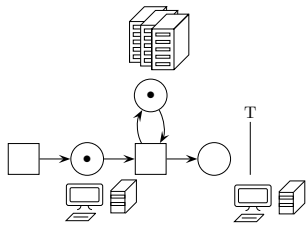


Figure 1.2:
Structural semantics of the structurally stationary client-server system in Figure 1.1. Note that the local connection between client and thread yields a place.

To justify the claim that our translation-based approach in fact permits efficient automatic verification of structurally stationary systems, we conduct a number of experiments. Most notably, we establish correctness of two realistic case studies.

Since not every DRS is structurally stationary, we strive for an intuitive explanation of the property. The main result is a complete characterisation, which shows that structural stationarity is equivalent to boundedness in two dimensions. Phrased differently, we prove that there are precisely two counterexamples to structural stationarity. The first class of systems that fail build lists as illustrated in Figure 1.3 (top). We say that these systems are not bounded in *depth*. In systems of unbounded *breadth*, single programs are connected with an unbounded number of components, Figure 1.3 (bottom).

Since important classes of systems, e.g. concurrent Java programs with broadcast mechanisms, are bounded in depth but not bounded in breadth, we investigate decidability in systems where only one of the dimensions, *depth* or *breadth*, is bounded. The outcome is that list structures (of unbounded depth but bounded breadth) are Turing complete. For DRS of *bounded depth but unbounded breadth*, we obtain a positive result: their transition systems are *well-structured*. Although not finitely factorisable, the states can be equipped with an ordering

relation, which allows us to compute a finite prefix of the infinite state space and draw conclusions about all computations.

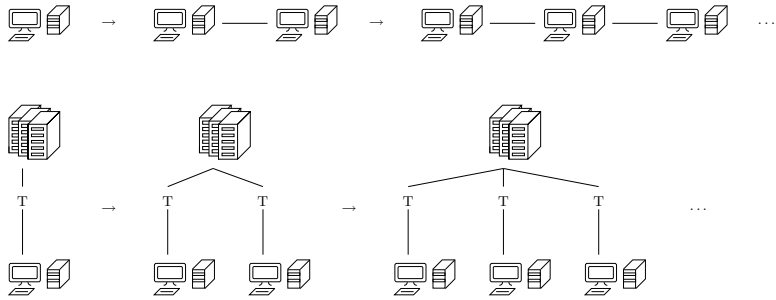


Figure 1.3:
The system above is not bounded in depth but bounded in breadth, the system below is bounded in depth but not bounded in breadth. Both fail to be structurally stationary. For the system below, we recover a mixed Petri net translation.

To maximise the benefit of Petri net verification techniques, we try to recover a translation into Petri nets for systems of bounded depth. For concurrent systems, Petri net semantics exist that highlight the interactions of programs. We define a corresponding translation for DRS and observe that it finitely represents a class of systems, which is incomparable with structurally stationary ones. We then show that both, structural and concurrency semantics, can be combined to a mixed translation. To give an example, the mixed translation detects the invariant that threads and server are always connected in Figure 1.3 (bottom). The connection is dropped and the system is translated into a Petri net similar to that in Figure 1.2. The main result is that this mixed view yields the precise borderline between DRS and Petri nets. Beyond this class, the transition systems of DRS can no longer be represented by place/transition Petri nets. Note that the border divides the systems of bounded depth.

To conclude the sketch of our contribution, we remark that before this thesis no classification of DRS existed. As soon as they became infinite state, they were considered Turing complete and approximate verification techniques were applied. This thesis contains the first presentation of decidable infinite-state DRS classes: structurally stationary, bounded in depth, bounded in breadth, and mixed bounded. We believe such a classification is an indispensable tool to judge hardness of the verification problem one is faced with, and to develop computer-aided verification techniques for the classes beyond the scope of Petri nets.

The introduction is kept general to emphasise that the theory of structural stationarity is independent of the modelling language for DRS. We develop it for the π-Calculus, but it should be extendable to graph grammars and object-oriented programs as well.

1.2 Structure of the Thesis

The contributions in this thesis are obtained according to a list of criteria that representations of DRS transition systems have to satisfy in order to be useful for verification purposes. Every chapter is dedicated to a different aspect.

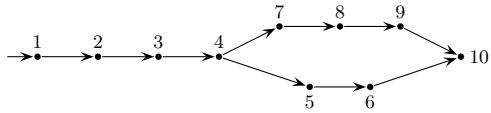


Figure 1.4: Organisation of the thesis.

After having recalled the basics on π-Calculus and Petri nets in Chapter 2, we turn to the aspect of *retrievability*. In Chapter 3, we define the structural semantics and show that a process and its Petri net representation have isomorphic transition systems. Moreover, the reachable process terms can be retrieved from the markings of the Petri net. This shows that the structural semantics contains all information necessary to verify properties of the process.

To apply automatic verification techniques, the Petri net representation has to be *finite*. We show that precisely the structurally stationary processes are finitely represented under the structural semantics in Chapter 4. In order to analyse a wide range of systems with help of our Petri net translation, we study the *expressiveness* of the class of structurally stationary processes. The main finding is a complete characterisation of structural stationarity, which shows that important classes of processes known from the literature satisfy this constraint.

In Chapter 5, we develop a concrete verification approach to demonstrate that properties of processes can be inferred efficiently using the structural semantics. In Chapter 6, we verify two larger case studies, which also supports our claim for *analysability* of the Petri net representation.

For the user of our translation, an *intuitive* understanding of whether it is able to cope with the system class of interest is indispensable. In Chapter 7, we prove that structural stationarity can be decomposed into boundedness in depth and boundedness in breadth. For both process classes, we establish intuitive graph-theoretic characterisations.

For systems of bounded depth and systems of bounded breadth, we study finite representations in Chapter 8. It turns out that systems of bounded depth have well-structured transition systems, and thus properties like termination and infinity of states can be *decided* on a finite prefix of the state space. Systems of bounded breadth are shown to be Turing complete.

Inspired by the decidability results for systems of bounded depth, we aim at maximising our translation into Petri nets. A Petri net semantics is *maximal* if no immediate extension exists that translates a larger class of processes into finite nets. In Chapter 9, we show how to combine the structural and the concurrency view to DRS to a translation which is maximal in a strong sense. The class of processes it finitely represents is complete with respect to finite place/transition Petri nets. Chapter 10 concludes the thesis.

The dependency graph of the chapters is shown in Figure 1.4.

1.3 Related Approaches

While we discuss approaches related to our work at the end of every chapter, this section gives a broad overview of related work on automatic verification techniques for DRS. We stress that all of the discussed approaches are semi-decision procedures based on abstractions. Our technique yields precise finite representations of infinite-state systems that allow for decidability results.

Graph Grammars Graph grammars model states of DRS as graphs. Transitions are defined by rewriting rules that identify a subgraph and replace it by a different one. In [Bau06], the reachable graphs of a graph grammar are abstracted to finitely many instances by identifying neighbouring vertices of the same type. Rensink suggests a combination of type graphs with logical formulae to abstract graphs [Ren04]. In [KK06], an abstraction refinement technique for graph grammars based on Petri nets is presented. The reachable graphs are abstracted to a shape graph by merging vertices. This shape graph is accompanied by a Petri net that has the edges of the shape graph as places so that tokens in the Petri net count the occurrences of edges in concrete graphs. If the abstraction yields a counterexample that cannot be concretised, the abstraction is refined by computing new shape graphs that merge less vertices. An approach orthogonal to verification is pursued in [EEHP06]. From a given safety property, application conditions for rewriting rules are computed so that the modified graph grammar is guaranteed to satisfy the requirement.

Graph-labelled Transition Systems In [Wes08], infinite-state graph-labelled transition systems are abstracted to finite instances by a so-called spotlight abstraction. It chooses a set of vertices to reflect precisely (those in the spot-

light) and abstracts the remaining ones to a single entity [WW07]. In [BTW07], the spotlight abstraction is refined by invariants generated from the analysis in [Bau06]. Toben extends the spotlight abstraction method by a refinement cycle [Tob08]. If the abstraction is to coarse to establish a temporal logic property, two refinement techniques are applied. The spotlight is enlarged to keep track of more entities and the abstract part is refined by taking into account the counter-example.

Petri Nets In [DFS98], extended Petri net models are proposed where the cardinality of an arc depends on the marking of the place. Dufourd et. al. show that important problems like coverability remain decidable in these extensions of place/transition Petri nets. The relationship of extended Petri nets with constructs in multithreaded JAVA programs is established by Delzanno and Raskin in [DRB02]. They show that the broadcast mechanism `notify all` as well as the non-blocking `notify` can be modelled adequately by Petri nets with transfer and propose symbolic verification techniques. They continue with an investigation of decidability of linear-time logics for Petri nets with transfer, which they settle negatively in [RB04]. Further decidability and expressiveness results as well as algorithmic improvements are given in the thesis of Geeraerts [Gee07].

2 Preliminaries

Contents

At the heart of the theory of structural stationarity is a mapping of π-Calculus processes into place/transition Petri nets. Therefore, we recall both models in this section. As we shall make extensive use of the structural congruence relation for the π-Calculus we give it a clean definition and carefully investigate its properties. In particular, we define a function to compute Milner's standard form, which allows us to characterise structural congruence by an equivalence relation.

For place/transition Petri nets, we recall the theory of S-invariants, unfoldings, and the coverability tree construction. While we need all of them in Part II on verification, the coverability tree also serves us as an analogy for the finite reachability tree introduced in Chapter 8.

2.1 π-Calculus

The π-Calculus is a *process algebra* for modelling DRS. The origins of process algebras date back to the 1970s with Hoare's algebra of *Communicating Sequential Processes (CSP)* [Hoa85] and Milner's *Calculus of Communicating Systems (CCS)* [Mil89]. Both lines of research were devoted to the study of the semantics of concurrent systems—with the following observation. *Communication*, i.e., sending and simultaneous receiving of messages, is the fundamental computation mechanism in concurrent systems. More complex mechanisms, e.g. semaphores, can be derived from elementary communications.

In CSP and CCS, a communication corresponds to a synchronisation on a channel. The π-Calculus was proposed by Milner, Parrow, and Walker [MPW92] as extension of CCS, where communications exchange messages over channels. For example, to transmit its IP address to a server located at some URL, a client uses the *output action* $\overline{url}\langle ip\rangle$. Here, *url* is the channel on which the message *ip* is sent. The *input action* $url(x)$ of the server listens on channel *url* and replaces variable x by the incoming message.

Transmission of data values was known to be encodable in CSP and CCS. The idea in the π-Calculus is to let message and channel have the same type: they are just *names*. Consequently, a message that is received in one communication may serve as channel in the following. We extend the model of the server to $S = url(x).\overline{x}\langle ses\rangle$. The server receives a channel x on *url* from the client. As a reply to this message it sends a session *ses* on the received channel, i.e., to the client. We also extend the client to receive the session: $C = \overline{url}\langle ip\rangle.ip(y)$.

The concurrent execution of client and server is reflected by the *parallel composition* operator $\_ \mid \_$. In the scenario above, the parallel composition is $C \mid S = \overline{url}\langle ip\rangle.ip(y) \mid url(x).\overline{x}\langle ses\rangle$. Since a communication of C and S forms a computation step, it is denoted by the transition arrow:

$$\overline{url}\langle ip\rangle.ip(y) \mid url(x).\overline{x}\langle ses\rangle \rightarrow ip(y) \mid \overline{ip}\langle ses\rangle.$$

Milner suggests to understand processes as *flow graphs* [MM79, Mil79]. The sequential processes yield nodes in the graph. The names in a process are mapped to labelled points on the border of the node. Points with the same name are connected by an edge. Intuitively, the names define the interface of a process, over which it communicates with other processes. The flow graphs of the client/server example are depicted in Figure 2.1. Note that the communication changes the link structure. While in $C \mid S$ client and server share channel *url*, they are connected by *ip* in the next step.

In the example, the communication does not change the number of entities in the system. To model *object creation*, the parallel composition operator can be nested under prefixes. Combined with the observation that new connections

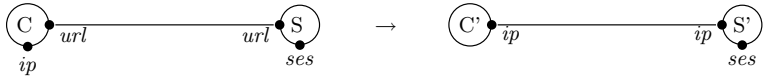


Figure 2.1:
Interpretation of the client/server system as flow graphs. The graph of $C \mid S$ with $C = \overline{url}\langle ip\rangle.ip(y)$ and $S = url(x).\overline{x}\langle ses\rangle$ is depicted to the left. A communication yields the process $C' \mid S'$ with $C' = ip(y)$ and $S' = \overline{ip}\langle ses\rangle$. The corresponding flow graph is depicted to the right.

can be established, we conclude that the evolution of connection structures over time—the characteristic feature of DRS—is well-reflected in the π-Calculus.

Much research on process algebras—CSP, CCS, or the π-Calculus—has been devoted to semantic models, their equivalences, and ordering relations between them. We do not pursue this line of research, but take the operational semantics of processes defined in terms of transition systems for granted. The contribution of the present thesis are (1) *representations* of the transition systems and (2) *analysis methods* for the transition systems that are based on the representation.

Before we turn to the definition, we remark that we work with a π-Calculus variant with parameterised recursion that is proposed by Sangiorgi and Walker in [SW01]. Our presentation and the notation follow the conventions therein.

2.1.1 Syntax

The basic elements of processes are *names* a, b, x, y in the infinite *set of names* $\mathcal{N}$. They are used as channels and messages in communications. Technically, we define *prefixes* π of the form

$$\pi ::= \overline{x}\langle y\rangle \;\mid\; x(y) \;\mid\; \tau.$$

The *output action prefix* $\overline{x}\langle y\rangle$ sends name y along channel x, the *input action prefix* $x(y)$ receives a name via x that replaces y, and the *silent prefix* τ performs an internal action.

Since we exchange single names in communications, the calculus is called *monadic*. In a polyadic π-Calculus, lists of names can be sent in one communication. The adaptation of the theory in this thesis to the polyadic π-Calculus is straightforward. Details can be found in the Master's thesis of Tim Strazny [Str07], who implemented the translation of π-Calculus into Petri nets for the polyadic version in the tool Petruchio [SM08]. We prefer the basic model for its simpler syntax. Since there is an encoding of polyadic into bisimilar monadic processes [Mil99, SW01], monadic and polyadic π-Calculus are in fact equally expressive.

A *finite sequence of names* $a_1, \ldots, a_n$ is abbreviated by $\tilde{a}$ and treated as a set where required, i.e., $\tilde{a} = \bigcup_{i=1}^{n} \{a_i\}$. To define parameterised recursion, we use *process identifiers* K, L in the *set of process identifiers* $\mathcal{ID}$. A process identifier represents a process P via a recursive definition $K(\tilde{x}) := P$, where the elements in $\tilde{x}$ are pairwise distinct. The term $K\lfloor\tilde{a}\rfloor$ is a *call* to the process identifier, which results in the process P with the names $\tilde{x}$ replaced by $\tilde{a}$. The remaining operators are standard; we recall their meaning.

The symbol $\mathbf{0}$ represents the *stop process* without any behaviour. A *prefixed process* $\pi.P$ offers the prefix π for communication and behaves like P when the prefix is consumed. The *choice* between the prefixed processes in M and N is represented by $M + N$. If a prefix $\pi.P$ is chosen in a composition $\pi.P + M$, the alternatives in M are forgotten. In a *parallel composition* $P \mid Q$, the processes P and Q communicate via pairs of send and receive prefixes. The *restriction* operator $\nu a.P$ converts the name a in P into a private name. It is different from all names in other processes.

Definition 2.1.1 (Syntax of the π-Calculus)
π-Calculus processes are typically denoted by P or Q. Their syntax is defined inductively in two steps:

$$\begin{aligned} M &::= \mathbf{0} \mid \pi.P \mid M_1 + M_2 \\ P &::= M \mid K\lfloor\tilde{a}\rfloor \mid P_1 \mid P_2 \mid \nu a.P. \end{aligned}$$

Every process relies on finitely many process identifiers K, each defined by an equation $K(\tilde{x}) := Q$. The *set of all π-Calculus processes* is $\mathcal{P}$. ♦

Convention 2.1.2 (Abbreviations and Precedences)
We use the following syntactic abbreviations and operator precedences.

1. A choice composition $M = \mathbf{0} + \ldots + \mathbf{0}$ is called *empty* and is denoted by $M^{=\mathbf{0}}$. A *non-empty* choice composition contains a prefixed process $\pi.P$. To indicate a choice composition is non-empty, we denote it by $M^{\neq\mathbf{0}}$.
2. We omit any pending $\mathbf{0}$, which means we write π instead of $\pi.\mathbf{0}$.
3. A prefix $\overline{a}.P$ denotes $\overline{a}\langle a\rangle.P$. The counterpart is $a.Q$, which stands for $a(x).Q$ where x is unimportant for process Q. With Definition 2.1.9 this means x is not in the free names of Q.
4. The natural numbers $\mathbb{N}$ contain 0, i.e, $\mathbb{N} := \{0, 1, 2, \ldots\}$.
5. With $k, m, n \in \mathbb{N}$ so that $n \geq m$ we define parallel compositions of multiple terms:

$$\Pi^k P := \underbrace{P \mid \ldots \mid P}_{k \text{ times}}$$

$$\begin{aligned} \Pi_{i=m}^{n} P_i &:= P_m \mid \ldots \mid P_n \\ \Pi_{i \in I} P_i &:= P_{i_0} \mid \ldots \mid P_{i_m}, \end{aligned}$$

where the index set $I = \{i_0, \ldots, i_m\} \subseteq \mathbb{N}$ is finite with $i_0 < \ldots < i_m$. If $k = 0$, we let $\Pi^0 P := \mathbf{0}$ and similarly $\Pi_{i \in \emptyset} P_i := \mathbf{0}$.

6. A sequence of restrictions $\nu a_1 \ldots \nu a_n.P$ is abbreviated by $\nu \tilde{a}.P$, where $\tilde{a} := a_1, \ldots, a_n$.

To avoid brackets, we define that (1) prefix π binds stronger than choice composition $+$ and (2) choice composition as well as restriction νa bind stronger than parallel composition $\mid$. ♦

Our definition of choice composition uses only prefixed processes $\pi.P$ as alternatives. In the literature, this well-accepted restriction is called *guarded choice*. It ensures that before process P_i can be executed in $\pi_1.P_1 + \ldots + \pi_n.P_n$ prefix π_i has to be consumed. The definition excludes choices of the form $(P_1 \mid P_2) + P_3$ or $(\nu a.P) + Q$. For the former process, it is known to be hard to define suitable Petri net semantics [Old91] and it is considered of minor practical importance [SW01]. The latter process causes problems in the definition of normal forms. A more elaborate structural congruence would allow for using the general syntax, but also complicate the theoretical development. Moreover, the decision for guarded choice does not delimit the computational expressiveness of the calculus.

Restricting the use of the remaining operators—restriction, recursion, and parallel composition—yields three syntactic subclasses of π-Calculus. *Restriction-free* processes are built without using the restriction operator. Amadio and Meyssonnier proved them to be computationally equivalent to Petri nets in [AM02]. We recall their construction in Section 4.5 when we investigate the size of the Petri nets resulting from our translation of structurally stationary processes.

Definition 2.1.3 (Restriction-Free Process)
A process $P \in \mathcal{P}$ is *restriction-free*, if it is built from the syntax in Definition 2.1.1 without using $\nu a.P$. ♦

Recursion-free processes, also known as *finite terms*, do not use any recursion. Much research has been devoted to finding axiomatisations and proof systems for behavioural relations (e.g. bisimilarity) on these terms. A presentation of the main results can be found in [SW01]. In [DKK06a], a translation of recursion-free processes into high-level Petri nets is given. We discuss the work of Koutny et. al. in Section 3.6.

Definition 2.1.4 (Recursion-Free Process)
A process $P \in \mathcal{P}$ is *recursion-free*, it is built from the syntax in Definition 2.1.1 except calls to process identifiers $K\lfloor\tilde{a}\rfloor$. ♦

Mads Dam gave a model checking algorithm and a sound and complete proof system for verifying so-called *finite control processes* against modal μ-calculus formulas in [Dam96]. Finite control processes restrict the use of parallel compositions.

Definition 2.1.5 (Finite Control Process)
A *finite control process* has the form $\nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ where the P_i do not use the parallel composition operator, i.e., they are built from the syntax in Definition 2.1.1 without $P_1 \mid P_2$. In particular, parallel compositions are forbidden within recursive definitions, i.e., in Q where $K(\tilde{x}) := Q$. ♦

To study the size of the Petri net translation, we define the *size of a process*. We sum up the lengths of the terms in all defining equations and the length of the main process. Note that prefixes π yield length two for channel and message.

Definition 2.1.6 ($\| - \| : \mathcal{P} \to \mathbb{N}$)
Consider process $P \in \mathcal{P}$ which uses the defining equations $K_i(\tilde{x}_i) := Q_i$ with $1 \leq i \leq n$. The *size* of P is $\|P\| := len(P) + \Sigma_{i=1}^{n} len(K_i(\tilde{x}_i) := Q_i)$, where function $len : \mathcal{P} \to \mathbb{N}$ is defined by

$$\begin{aligned}
len(\mathbf{0}) &:= 1 & len(\pi.P) &:= 2 + len(P)\\
len(M + N) &:= len(M) + 1 + len(N) & len(K\lfloor\tilde{a}\rfloor) &:= 1 + |\tilde{a}|\\
len(P \mid Q) &:= len(P) + 1 + len(Q) & len(\nu a.P) &:= 1 + len(P)\\
len(K(\tilde{x}) := P) &:= 1 + |\tilde{x}| + len(P).
\end{aligned}$$

Here, $|\tilde{x}|$ is the length of the list, for example n for $\tilde{x} = x_1, \ldots, x_n$. ♦

2.1.2 Names and Substitutions

We mentioned that a name a, which occurs in the scope of a restriction νa, is different from all other names in the process under consideration. To ensure this disjointness, we define ν to *bind* the name a. We then allow for renaming bound names by α-conversion. Similarly, in a prefixed process $a(y).P$ the receive action $a(y)$ *binds* the name y in P. Intuitively, y is a variable which has not yet received a concrete value and should be assumed different from all other names in the process.

Definition 2.1.7 ($bn : \mathcal{P} \to \mathbb{P}(\mathcal{N})$)
The function $bn : P \to \mathbb{P}(\mathcal{N})$ computes the set of *bound names* in a process as defined in Table 2.1. ♦

$$
\begin{aligned}
bn(\mathbf{0}) &:= \emptyset & bn(\tau.P) &:= bn(P)\\
bn(\overline{a}\langle b\rangle.P) &:= bn(P) & bn(a(y).P) &:= \{y\} \cup bn(P)\\
bn(M+N) &:= bn(M) \cup bn(N) & bn(K\lfloor\tilde{a}\rfloor) &:= \emptyset\\
bn(P \mid Q) &:= bn(P) \cup bn(Q) & bn(\nu a.P) &:= \{a\} \cup bn(P).
\end{aligned}
$$

$$
\begin{aligned}
arn(M) &:= \emptyset & arn(K\lfloor\tilde{a}\rfloor) &:= \emptyset\\
arn(P \mid Q) &:= arn(P) \cup arn(Q) & arn(\nu a.P) &:= \{a\} \cup arn(P).
\end{aligned}
$$

$$
\begin{aligned}
fn(\mathbf{0}) &:= \emptyset & fn(\tau.P) &:= fn(P)\\
fn(\overline{a}\langle b\rangle.P) &:= \{a,b\} \cup fn(P) & fn(a(y).P) &:= \{a\} \cup (fn(P) \setminus \{y\})\\
fn(M+N) &:= fn(M) \cup fn(N) & fn(K\lfloor\tilde{a}\rfloor) &:= \tilde{a}\\
fn(P \mid Q) &:= fn(P) \cup fn(Q) & fn(\nu a.P) &:= fn(P) \setminus \{a\}.
\end{aligned}
$$

Table 2.1: Definition of bn, arn, and fn.

Of particular interest in the theory of structurally stationary processes are those restricted names that are not covered by a prefix. We call them *active restricted names* or just *active restrictions*. For example, in the process

$$\nu a.(\overline{a}\langle b\rangle.\nu c.\overline{a}\langle c\rangle \mid a(x) \mid K\lfloor b\rfloor)$$

the restriction νa is active while νc is not as it is covered by the prefix $\overline{a}\langle b\rangle$.

Definition 2.1.8 ($arn : \mathcal{P} \to \mathbb{P}(\mathcal{N})$)
The set of *active restricted names* in a process is computed by the function arn in Table 2.1. ♦

The active restricted names are a subset of the bound names of a process, i.e., $arn(P) \subseteq bn(P)$ holds. Active restrictions connect the processes that use the name. In the example above, νa connects $\overline{a}\langle b\rangle.\nu c.\overline{a}\langle c\rangle$ and $a(x)$, but not $K\lfloor b\rfloor$. In Section 3.2, we formalise the idea of connecting processes by active restrictions. To make the notion of *using* a name precise, we define *free* names. A name that

is not bound by an input action or a restriction is *free* in a process. The function fn collects all free names; it is the counterpart to bn defined above.

Definition 2.1.9 ($fn : \mathcal{P} \to \mathbb{P}(\mathcal{N})$)
The function $fn : \mathcal{P} \to \mathbb{P}(\mathcal{N})$ yields the set of *free names* in a process as defined in Table 2.1. We say that process P *uses the name* a, if $a \in fn(P)$ holds. ♦

In the literature, processes without free names are known as *closed processes*. In Section 3.3, we show that our structural semantics translates them into a subclass of Petri nets.

Definition 2.1.10 (Closed Process)
A process $P \in \mathcal{P}$ is *closed* if $fn(P) = \emptyset$. ♦

Since we will permit α-conversion of bound names, we assume without loss of generality (1) that all bound names are different and (2) that bound names and free names do not interfere. So, we forbid the following two processes

$$\nu a.\overline{a}\langle b\rangle \mid b(a) \qquad\qquad b(a) \mid a(x).$$

In the first, the name a is bound twice; in the second process, the name a occurs bound in $b(a)$ and free in $a(x)$. With α-conversion we can rewrite the first process to $\nu a.\overline{a}\langle b\rangle \mid b(c)$ and the second to $b(c) \mid a(x)$, both of which respect the conventions.

Convention 2.1.11 (Disjointness of Names)
Consider process $P \in \mathcal{P}$ with the defining equations $K_i(\tilde{x}_i) := P_i$ for $1 \leq i \leq n$. We formalise three requirements.

(1) The free names in a defining process are included in the parameter list, $fn(P_i) \subseteq \tilde{x}_i$ for all $1 \leq i \leq n$.

(2) Bound and free names are always disjoint, i.e., $fn(Q) \cap bn(R) = \emptyset$ for all $Q, R \in \{P, P_1, \ldots, P_n\}$.

(3) The main process as well as all defining equations use disjoint sets of bound names: $bn(Q) \cap bn(R) = \emptyset$ for all $Q, R \in \{P, P_1, \ldots, P_n\}$ with $Q \neq R$

Unless otherwise stated, a name is bound at most once in a process. This means for $\nu x.P$ and $a(x).P$ we have $x \notin bn(P)$. For $P \mid Q$ we get $bn(P) \cap bn(Q) = \emptyset$. Combined with Requirement (3) this ensures that a name is bound at most once in a process and in all the defining equations. ♦

Technically, α-conversion of a bound name a to c means changing the process $\nu a.P$ to $\nu c.P'$, where every free occurrence of a in P is replaced by c in P'. As

an example, $\nu a.a(x)$ can be α-converted to $\nu c.c(x)$. To rename free names in a process, we use *substitutions*.

Definition 2.1.12 ($\sigma : \mathcal{N} \to \mathcal{N}$)
A *substitution* σ is a mapping from names to names, $\sigma : \mathcal{N} \to \mathcal{N}$. Let $x\sigma$ denote the image of x under σ. If we give domain and codomain, $\sigma : A \to B$ with $A, B \subseteq \mathcal{N}$, we demand $x\sigma \in B$ if $x \in A$ and $x\sigma = x$ otherwise. An explicitly defined substitution $\sigma = \{a_1, \ldots, a_n / x_1, \ldots, x_n\}$ maps x_i to a_i, i.e., $\sigma : \{x_1, \ldots, x_n\} \to \{a_1, \ldots, a_n\}$ with $x_i\sigma = a_i$. ♦

An application of a substitution σ to a process P results in a new process $P\sigma$, where all free names in P are changed according to σ. For example, applying $\sigma = \{a, b/x, y\}$ to $\nu c.\overline{x}\langle y\rangle$ yields $(\nu c.\overline{x}\langle y\rangle)\sigma = \nu c.\overline{a}\langle b\rangle$. To ensure that substitutions do not introduce new bindings, we assume that the names in the substitution σ do not interfere with the bound names in the process σ is applied to.

Convention 2.1.13 (Substitution)
If we apply a substitution $\sigma : A \to B$ to a process P, we demand the names in σ to be disjoint from the bound names in P, i.e., $(A \cup B) \cap bn(P) = \emptyset$. ♦

Definition 2.1.14 (Application of Substitutions)
Consider a substitution $\sigma : A \to B$ and a process $P \in \mathcal{P}$ with $(A \cup B) \cap bn(P) = \emptyset$. The *application of σ to P* results in a new process $P\sigma$ defined by

$$\begin{aligned}
\mathbf{0}\sigma &:= \mathbf{0} & (\tau.P)\sigma &:= \tau.(P\sigma) \\
(x(y).P)\sigma &:= x\sigma(y).(P\sigma) & (\overline{x}\langle y\rangle.P)\sigma &:= \overline{x\sigma}\langle y\sigma\rangle.(P\sigma) \\
(M + N)\sigma &:= M\sigma + N\sigma & K\lfloor\tilde{a}\rfloor\sigma &:= K\lfloor\tilde{a}\sigma\rfloor \\
(P \mid Q)\sigma &:= P\sigma \mid Q\sigma & (\nu a.P)\sigma &:= \nu a.(P\sigma).
\end{aligned}$$

♦

In Section 4.2, we construct substitutions σ and have to show that their codomain is correct, i.e., we prove $\sigma : fn(P) \to A$ for some set A. The following lemma provides a proof technique for this problem. To show σ maps $fn(P)$ to A it is sufficient to apply σ to P and then check whether the free names in $P\sigma$ are in A.

Lemma 2.1.15 (Proof Technique for the Codomain of Substitutions)
For every process $P \in \mathcal{P}$ and every substitution σ with domain $fn(P)$ we have: $\sigma : fn(P) \to A$ if and only if $fn(P\sigma) \subseteq A$.

Lemma 2.1.15 follows from the compatibility of applications of substitutions

with the computation of the free names in a process. Lemma 2.1.16 can be shown by an induction on the structure of processes.

Lemma 2.1.16
For every process $P \in \mathcal{P}$ and substitution σ the equality $fn(P\sigma) = fn(P)\sigma$ holds.

Proof (of Lemma 2.1.15)
Consider $P \in \mathcal{P}$ and substitution σ with domain $fn(P)$. The following equivalences hold:

$$\begin{aligned} & & \sigma : fn(P) \to A \\ (\text{ Def. (co)domain }) & \Leftrightarrow & fn(P)\sigma \subseteq A \\ (\text{ Lemma 2.1.16 }) & \Leftrightarrow & fn(P\sigma) \subseteq A. \end{aligned}$$

This proves the claim. ■

2.1.3 Structural Congruence

To give an operational semantics to a process algebra, the behaviour of every process has to be defined. To keep the definition of the transition relation simple, Berry and Boudol suggested to define only the transitions of representative terms and use a second relation to relate processes with representatives [BB90]. It is then demanded that a process behaves like its representative. Intuitively, the definition of the operational semantics is factorised into the definition of a transition and a structural relation.

Berry and Boudol called the approach *chemical abstract machine* with the following idea. Processes are chemical molecules that change their structure. Changing the structure heats molecules up or cools them down. Only heated molecules react with one another, which changes their state.

The π-Calculus semantics that exploits the chemical abstract machine idea was introduced by Milner in [Mil92]. He called the relation to identify processes with representatives *structural congruence* and the name is still in use. Many results in this thesis exploit the invariance of the transition relation under structural rewriting.[1] Without the idea of Berry and Boudol and Milner's adaptation to the π-Calculus, the results in this thesis would not have been possible.

Before we turn to the definition of structural congruence $\equiv \subseteq \mathcal{P} \times \mathcal{P}$, we recall that a *congruence relation* is an equivalence which is compatible with the

[1] For example the definition of the restricted form and the structural semantics in Chapter 3, or the theory of depth and breadth and anchored fragments in Chapter 7.

operators of the algebra under study. That $\equiv$ is an *equivalence* means we have

$$\forall P \in \mathcal{P} : P \equiv P \qquad \text{(Reflexivity)}$$
$$\forall P, Q \in \mathcal{P} : P \equiv Q \text{ implies } Q \equiv P \qquad \text{(Symmetry)}$$
$$\forall P, Q, R \in \mathcal{P} : P \equiv Q \text{ and } Q \equiv R \text{ implies } P \equiv R. \qquad \text{(Transitivity)}$$

That structural congruence is a congruence means it is preserved under composition, using any of the operators:

$$\forall P, Q, M \in \mathcal{P} : \forall \pi : P \equiv Q \text{ implies } \pi.P + M \equiv \pi.Q + M$$
$$\forall P, Q, R \in \mathcal{P} : P \equiv Q \text{ implies } P \mid R \equiv Q \mid R$$
$$\forall P, Q \in \mathcal{P} : \forall a \in \mathcal{N} : P \equiv Q \text{ implies } \nu a.P \equiv \nu a.Q.$$

Note that the operands of choice compositions are guarded in our setting, hence $P + M$ is no valid term unless $P = \pi.P'$. For the latter case, congruence is demanded by the first implication. Note also that this implication in particular requires $\pi.P \equiv \pi.Q$ when $M = \mathbf{0}$.

Definition 2.1.17 (Structural Congruence)
Structural congruence $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ is the least congruence relation on processes, which allows for α-converting bound names, i.e.,

$$\nu x.P \equiv \nu y.(P\{y/x\}) \qquad a(x).P \equiv a(y).(P\{y/x\}),$$

where in both cases $\{y\} \cap (fn(P) \cup bn(P)) = \emptyset$, where $+$ and $\mid$ are commutative and associative with $\mathbf{0}$ as neutral element, i.e.,

$$M + \mathbf{0} \equiv M \qquad M_1 + M_2 \equiv M_2 + M_1$$
$$M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3$$
$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1$$
$$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3,$$

and restriction is a commutative quantifier that is absorbed by $\mathbf{0}$ and whose scope can be shrunk and extruded over processes not using the quantified name:

$$\nu x.\nu y.P \equiv \nu y.\nu x.P \qquad \nu x.\mathbf{0} \equiv \mathbf{0}$$
$$\nu x.(P \mid Q) \equiv P \mid (\nu x.Q), \text{ if } x \notin fn(P).$$

The latter law is called *scope extrusion.* ♦

For the implementation of our Petri net translation it is important to note that structural congruence is decidable in our setting. In particular, we encode calls to process identifiers $K\lfloor a \rfloor$ into the reaction relation (Definition 2.1.34) to obtain this decidability. For a structural congruence that is extended by $K\lfloor \tilde{a} \rfloor \equiv P\{\tilde{a}/\tilde{x}\}$ where $K(\tilde{x}) := P$ decidability is far from trivial.

Theorem 2.1.18 ([KM09])
For all $P, Q \in \mathcal{P}$ it is decidable whether $P \equiv Q$ holds and the problem is graph isomorphism complete. If P and Q are (1) restriction-free or (2) contain neither parallel nor choice composition, then the problem $P \equiv Q$ can be decided in time polynomial in the size of P and Q.

To reduce $P \equiv Q$ to graph isomorphism, Khomenko and the author give a reduction to a term equality problem that is known to be reducible to labelled digraph isomorphism [Bas94]. In practical tools for π-Calculus verification like MWB [VM94], HAL [FGMP03], the Spatial Logic Model Checker [Cai04], or Petruchio [SM08], structural congruence is a basic task to be solved when computing the state space of a process, or the Petri net representation in case of Petruchio. So the efficiency of a tool crucially depends on the efficiency of the structural congruence checker. Therefore, in [KM09] reduction techniques for the graphs resulting from Basin's construction are presented, which exploit specific features of π-Calculus terms. Example graphs were reduced from 60/63 to 26/38 vertices and edges. Off-the-shelf graph isomorphism checkers could then be used in black-box fashion to decide graph isomorphism of the reduced graphs.

We shall need some properties of structural congruence. The relation allows for removing unused restricted names, i.e., $\nu a.P \equiv P$ if $a \notin fn(P)$ is a derived rule. The fact is well-known; we quote the proof from [SW01]:

$$\nu a.P \equiv \nu a.(P \mid \mathbf{0}) \equiv P \mid \nu a.\mathbf{0} \equiv P \mid \mathbf{0} \equiv P.$$

We also need that structural congruence preserves the free names in a process. We quote this result from Milner's book and remark that he uses a structural congruence that expands the definition of process identifiers. Hence, our congruence is included in his version and the result still holds in our setting.

Lemma 2.1.19 (Invariance of fn under $\equiv$, [Mil99])
For all $P, Q \in \mathcal{P}$ we have: $P \equiv Q$ implies $fn(P) = fn(Q)$.

2.1.4 Sequential Processes

Of particular interest in the theory of structural stationarity are *sequential processes*, i.e., non-empty sums $M^{\neq \mathbf{0}}$ and calls to process identifiers $K\lfloor \tilde{a} \rfloor$. Intuitively, the behaviour of sequential processes determines the behaviour of composed processes. It should be noted that a process $K\lfloor \tilde{a} \rfloor$ always yields a reaction in the reaction relation defined below. Therefore it is justified to call it sequential regardless of its definition, which may be $K(\tilde{x}) := P \mid Q$. We use the function $\mathcal{S} : \mathcal{P} \to \mathbb{P}(\mathcal{P})$ to refer to the sequential processes inside a given process.

Definition 2.1.20 ($\mathcal{S} : \mathcal{P} \to \mathbb{P}(\mathcal{P})$)
The set of *sequential processes in a process* $P \in \mathcal{P}$ is $\mathcal{S}(P)$ defined by

$$\begin{aligned} \mathcal{S}(M^{=\mathbf{0}}) &:= \emptyset & \mathcal{S}(M^{\neq\mathbf{0}}) &:= \{M^{\neq\mathbf{0}}\} \\ \mathcal{S}(K\lfloor\tilde{a}\rfloor) &:= \{K\lfloor\tilde{a}\rfloor\} & \mathcal{S}(P \mid Q) &:= \mathcal{S}(P) \cup \mathcal{S}(Q) \\ \mathcal{S}(\nu a.P) &:= \mathcal{S}(P). \end{aligned}$$

♦

The application of substitutions is compatible with the computation of the sequential processes.

Lemma 2.1.21
Given $P \in \mathcal{P}$ and $\sigma : fn(P) \to \mathcal{N}$, then $\mathcal{S}(P\sigma) = \mathcal{S}(P)\sigma$ holds.

We also need that the free names of a sequential process in P are included in the active restrictions and free names of P.

Lemma 2.1.22
For every process $P \in \mathcal{P}$ and every $Q \in \mathcal{S}(P)$ we have $fn(Q) \subseteq fn(P) \cup arn(P)$.

Formally, the number of sequential processes is defined by counting the non-empty sums and process identifiers, e.g. $\|\nu b.(b(x) \mid K\lfloor a, b\rfloor)\|_{\mathcal{S}} = 2$.

Definition 2.1.23 ($\| - \|_{\mathcal{S}} : \mathcal{P} \to \mathbb{N}$)
The *number of sequential processes* inside $P \in \mathcal{P}$ is $\|P\|_{\mathcal{S}}$, defined inductively:

$$\begin{aligned} \|M^{=\mathbf{0}}\|_{\mathcal{S}} &:= 0 & \|M^{\neq\mathbf{0}}\|_{\mathcal{S}} &:= 1 \\ \|K\lfloor\tilde{a}\rfloor\|_{\mathcal{S}} &:= 1 & \|P \mid Q\|_{\mathcal{S}} &:= \|P\|_{\mathcal{S}} + \|Q\|_{\mathcal{S}} \\ \|\nu a.P\|_{\mathcal{S}} &:= \|P\|_{\mathcal{S}}. \end{aligned}$$

♦

The function is invariant under structural congruence. This follows from an induction on the derivations of structural congruence.

Lemma 2.1.24 (Invariance of $\| - \|_{\mathcal{S}}$ under $\equiv$)
Consider $P, Q \in \mathcal{P}$. If $P \equiv Q$ then $\|P\|_{\mathcal{S}} = \|Q\|_{\mathcal{S}}$.

2.1.5 Standard Form

The chemical abstract machine approach only defines transitions of representative processes. In the π-Calculus, processes in Milner's *standard form* are well-known representatives [Mil99, SW01]. Lemma 2.1.43 shows that the reaction relation is essentially defined for processes in standard form. The behaviour of the remaining processes can be derived from theirs using structural congruence.

The idea of the standard form is to maximise the scopes of active restricted names. This yields processes of the form $\nu\tilde{a}.P$, where P is a parallel composition of choices and calls to process identifiers. Empty choices are then removed from P. Similarly, names $a \in \tilde{a}$ that are not used in P are erased. For example, $\nu c.(a(x) \mid \mathbf{0}) \mid \nu b.\overline{a}\langle b\rangle$ is not in standard form but $\nu b.(a(x) \mid \overline{a}\langle b\rangle)$ is.

Definition 2.1.25 (Process in Standard Form)
A *process in standard form* is typically denoted by P^{sf} or Q^{sf} and built from the following syntax:

$$\begin{array}{rcl} P^{\neq\nu} & ::= & M^{\neq\mathbf{0}} \;\mid\; K\lfloor\tilde{a}\rfloor \;\mid\; P_1^{\neq\nu} \mid P_2^{\neq\nu} \\ P^{sf} & ::= & \mathbf{0} \;\mid\; P^{\neq\nu} \;\mid\; \nu a.P^{sf} \end{array}$$

with $a \in fn(P^{sf})$. The *set of all processes in standard form* is $\mathcal{P}_{sf}$. ♦

The name suggests that the syntax in Definition 2.1.25 is a normal form for processes. This means any process is related to a process in standard form by structural congruence. In the proof, we use the function sf.

Definition 2.1.26 ($sf : \mathcal{P} \to \mathcal{P}_{sf}$)
The function $sf : \mathcal{P} \to \mathcal{P}_{sf}$ computes for every $P \in \mathcal{P}$ a process $sf(P) \in \mathcal{P}_{sf}$ as defined in Table 2.2. We call $sf(P)$ the *standard form of* P. ♦

Example 2.1.27 ($sf : \mathcal{P} \to \mathcal{P}_{sf}$)
Consider the process $\nu c.(a(x) \mid \mathbf{0}) \mid \nu b.\overline{a}\langle b\rangle$. The definition of $sf(P \mid Q)$ recursively computes

$$\begin{array}{rcl} sf(\nu c.(a(x) \mid \mathbf{0})) & = & sf(a(x) \mid \mathbf{0}) = a(x) \\ sf(\nu b.\overline{a}\langle b\rangle) & = & \nu b.sf(\overline{a}\langle b\rangle) = \nu b.\overline{a}\langle b\rangle. \end{array}$$

The function then maximises the scope of νb:

$$sf(\nu c.(a(x) \mid \mathbf{0}) \mid \nu b.\overline{a}\langle b\rangle) \quad = \quad \nu b.(a(x) \mid \overline{a}\langle b\rangle).$$

♦

$$sf(M^{=\mathbf{0}}) := \mathbf{0} \qquad sf(M^{\neq\mathbf{0}}) := M^{\neq\mathbf{0}} \qquad sf(K\lfloor\tilde{a}\rfloor) := K\lfloor\tilde{a}\rfloor$$

$$sf(\nu a.P) := \begin{cases} \nu a.sf(P), & \text{if } a \in fn(P) \\ sf(P), & \text{if } a \notin fn(P) \end{cases}$$

$$sf(P \mid Q) := \begin{cases} \mathbf{0}, & \text{if } sf(P) = \mathbf{0} = sf(Q) \\ sf(P), & \text{if } sf(P) \neq \mathbf{0} = sf(Q) \\ sf(Q), & \text{if } sf(P) = \mathbf{0} \neq sf(Q) \\ \nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}), & \text{if } sf(P) = \nu\tilde{a}_P.P^{\neq\nu} \\ & \text{and } sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}. \end{cases}$$

Table 2.2: Definition of function sf.

Lemma 2.1.28 shows that the codomain of sf is correct, i.e., $sf(P)$ is in fact in $\mathcal{P}_{sf}$. It furthermore proves that a process and its standard form are related by structural congruence as required. We shall also need that sf does not change the sequential processes as well as the active restrictions. To see this, recall that we assume a name to be bound at most once and the bound names to be disjoint with the free names. Hence, α-conversion is not required in the computation of $sf(P)$ and the statement holds. Finally, sf is the identity on processes in standard form.

Lemma 2.1.28 (Properties of sf)
For every $P \in \mathcal{P}$ we have the following properties: $sf(P) \in \mathcal{P}_{sf}$, $P \equiv sf(P)$, $\mathcal{S}(P) = \mathcal{S}(sf(P))$, $arn(sf(P)) \subseteq arn(P)$, and $sf(P\sigma) = sf(P)\sigma$. For processes $P^{sf} \in \mathcal{P}_{sf}$ the equality $sf(P^{sf}) = P^{sf}$ holds.

Proof
We proceed by induction on the structure of processes. The base cases of empty sums $M^{=\mathbf{0}}$, non-empty sums $M^{\neq\mathbf{0}}$, and calls to process identifiers $K\lfloor\tilde{a}\rfloor$ are trivial. We turn to the induction step and assume the desired properties hold for P and Q.

Case $P \mid Q$ Since $sf(P)$ and $sf(Q)$ are included in $\mathcal{P}_{sf}$, we have $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ or $sf(P) = \mathbf{0}$ and similar for $sf(Q)$. Let both standard forms be different from $\mathbf{0}$, the remaining cases are trivial. We compute

$$sf(P \mid Q) = \nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}) \in \mathcal{P}_{sf}.$$

To see that the inclusion holds, we observe that $\tilde{a}_P \subseteq fn(P^{\neq\nu})$ by the hypothesis

and $fn(P^{\neq\nu}) \subseteq fn(P^{\neq\nu} \mid Q^{\neq\nu})$ by definition of fn. For $\tilde{a}_Q$ we argue similarly. We now prove structural congruence:

$$\begin{array}{rcl} & & P \mid Q \\ (\ P \equiv sf(P) \text{ by hypothesis, } sf(P) = \nu\tilde{a}_P.P^{\neq\nu}\ ) & \equiv & \nu\tilde{a}_P.P^{\neq\nu} \mid Q. \end{array}$$

We assume $bn(P \mid Q) \cap fn(P \mid Q) = \emptyset$, so in particular $arn(P) \cap fn(Q) = \emptyset$. By the hypothesis, we have $\tilde{a}_P = arn(sf(P)) \subseteq arn(P)$. Thus, we conclude $\tilde{a}_P \cap fn(Q) = \emptyset$ and we can extrude the scope of $\tilde{a}_P$:

$$\begin{array}{rcl} (\text{ Scope extrusion }) & \equiv & \nu\tilde{a}_P.(P^{\neq\nu} \mid Q) \\ (\ Q \equiv sf(Q) \text{ by hypothesis, } sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}\ ) & \equiv & \nu\tilde{a}_P.(P^{\neq\nu} \mid \nu\tilde{a}_Q.Q^{\neq\nu}). \end{array}$$

To extrude the scope of $\tilde{a}_Q$, we need to ensure that $\tilde{a}_Q \cap fn(P^{\neq\nu}) = \emptyset$. Consider a name $a \in fn(P^{\neq\nu})$ with $a \in \tilde{a}_P$. We assume that a name is bound at most once in $P \mid Q$, so $bn(P) \cap bn(Q) = \emptyset$. Since $\tilde{a}_P = arn(sf(P)) \subseteq arn(P) \subseteq bn(P)$ and similar for $\tilde{a}_Q$, we conclude that $\tilde{a}_P \cap \tilde{a}_Q = \emptyset$. Hence, $a \notin \tilde{a}_Q$. Assume that $a \notin \tilde{a}_P$, then $a \in fn(\nu\tilde{a}_P.P^{\neq\nu}) = fn(sf(P))$. With the congruence $sf(P) \equiv P$ and the invariance of free names under structural congruence we conclude $a \in fn(P)$. Since $\tilde{a}_Q \subseteq bn(Q)$ and $bn(Q) \cap fn(P) = \emptyset$, we conclude $a \notin \tilde{a}_Q$. Summing up, we have $\tilde{a}_Q \cap fn(P^{\neq\nu}) = \emptyset$, which allows us to extrude the scope of $\tilde{a}_Q$:

$$\begin{array}{rcl} (\text{ Scope extrusion }) & \equiv & \nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}) \\ (\text{ Def. } sf\text{, form of } sf(P) \text{ and } sf(Q)\ ) & = & sf(P \mid Q). \end{array}$$

For the sequential processes we compute

$$\begin{array}{rcl} & & \mathcal{S}(P \mid Q) \\ (\text{ Def. } \mathcal{S}\ ) & = & \mathcal{S}(P) \cup \mathcal{S}(Q) \\ (\text{ Hypothesis }) & = & \mathcal{S}(sf(P)) \cup \mathcal{S}(sf(Q)) \\ (\ sf(P) = \nu\tilde{a}_P.P^{\neq\nu} \text{ and } sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}\ ) & = & \mathcal{S}(\nu\tilde{a}_P.P^{\neq\nu}) \cup \mathcal{S}(\nu\tilde{a}_Q.Q^{\neq\nu}) \\ (\text{ Def. } \mathcal{S}\ ) & = & \mathcal{S}(P^{\neq\nu}) \cup \mathcal{S}(Q^{\neq\nu}) \\ (\text{ Def. } \mathcal{S}\ ) & = & \mathcal{S}(\nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu})) \\ (\text{ Def. } sf\text{, form of } sf(P) \text{ and } sf(Q)\ ) & = & \mathcal{S}(sf(P \mid Q)). \end{array}$$

To check the active restrictions, we start with the standard form of $P \mid Q$ and show that the active restrictions are included in $arn(P \mid Q)$:

$$\begin{array}{rcl} & & arn(sf(P \mid Q)) \\ (\text{ Def. } sf\text{, form of } sf(P) \text{ and } sf(Q)\ ) & = & arn(\nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu})) \\ (\text{ Def. } arn\text{, } P^{\neq\nu}\text{, and } Q^{\neq\nu}\ ) & = & \tilde{a}_P \cup \tilde{a}_Q \\ (\text{ Def. } arn\text{, form of } sf(P) \text{ and } sf(Q)\ ) & = & arn(sf(P)) \cup arn(sf(Q)) \end{array}$$

$$\begin{aligned} (\text{ Hypothesis }) \quad &\subseteq \quad arn(P) \cup arn(Q) \\ (\text{ Def. } arn \text{ }) \quad &= \quad arn(P \mid Q). \end{aligned}$$

We need to take care of substitutions. With the hypothesis, we have

$$sf(P\sigma) = sf(P)\sigma = (\nu\tilde{a}_P.P^{\neq\nu})\sigma = \nu\tilde{a}_P.(P^{\neq\nu}\sigma),$$

where the second equation holds with the assumption on the form of $sf(P)$ and the third uses the definition of substitution application. We compute

$$\begin{aligned} & & sf(P \mid Q)\sigma \\ (\text{ Def. } sf\text{, form of } sf(P) \text{ and } sf(Q) \text{ }) \quad &= \quad (\nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}))\sigma \\ (\text{ Applic. } \sigma \text{ }) \quad &= \quad \nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu}\sigma \mid Q^{\neq\nu}\sigma) \\ (\text{ Observation above, def. } sf \text{ }) \quad &= \quad sf(P\sigma \mid Q\sigma) \\ (\text{ Applic. } \sigma \text{ }) \quad &= \quad sf((P \mid Q)\sigma). \end{aligned}$$

Case $\nu a.P$ We distinguish between $a \notin fn(P)$ and $a \in fn(P)$ and begin with the latter. By definition of sf and the assumption that $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ we derive

$$sf(\nu a.P) = \nu a.sf(P) \in \mathcal{P}_{sf}.$$

To justify the inclusion $\nu a.sf(P) \in \mathcal{P}_{sf}$, we argue that $a \in fn(sf(P))$ with the invariance of fn under structural congruence and the hypothesis $P \equiv sf(P)$. Structural congruence $\nu a.P \equiv sf(\nu a.P)$ is immediate with the hypothesis $P \equiv sf(P)$ and the fact that $\equiv$ is a congruence:

$$\nu a.P \equiv \nu a.sf(P) = sf(\nu a.P).$$

The following equations show that the sequential processes coincide:

$$\mathcal{S}(\nu a.P) = \mathcal{S}(P) = \mathcal{S}(sf(P)) = \mathcal{S}(\nu a.sf(P)) = \mathcal{S}(sf(\nu a.P)).$$

They hold with the definition of $\mathcal{S}$, the hypothesis, again the definition of $\mathcal{S}$, and the definition of sf.

For the active restrictions, we first apply the definition of sf and then the definition of arn. This yields the first of the following equations. We continue with an application of the hypothesis. The last equation again holds by definition of arn:

$$arn(sf(\nu a.P)) = \{a\} \cup arn(sf(P)) \subseteq \{a\} \cup arn(P) = arn(\nu a.P).$$

To show that $sf((\nu a.P)\sigma) = sf(\nu a.P)\sigma$, we apply the substitution and then use the definition of sf, which gives the first two equations. We continue with an

application of the hypothesis to justify the third equation. The fourth holds with the definition of substitution application and the definition of sf:

$$sf((\nu a.P)\sigma) = sf(\nu a.(P\sigma)) = \nu a.sf(P\sigma) = \nu a.(sf(P)\sigma) = sf(\nu a.P)\sigma.$$

In case $a \notin fn(P)$ we have $sf(\nu a.P) = sf(P)$. So in this case the subset relation but not equality holds between $arn(sf(\nu a.P))$ and $arn(\nu a.P)$. The remaining properties are established similar to the preceding case.

Identity on $\mathcal{P}_{sf}$ We use induction on the structure of processes in standard form. Recall that $\mathbf{0}$ is an empty sum $M^{=\mathbf{0}}$, so the claim holds with $sf(M^{=\mathbf{0}}) = \mathbf{0}$. For $M^{\neq\mathbf{0}}$ and $K\lfloor\tilde{a}\rfloor$ equality holds by definition of sf.

In the induction step, we assume that $sf(P^{\neq\nu}) = P^{\neq\nu}$ and similar for $Q^{\neq\nu}$. For the parallel composition, we get $sf(P^{\neq\nu} \mid Q^{\neq\nu}) = P^{\neq\nu} \mid Q^{\neq\nu}$ since both sequences of names, $\tilde{a}_P$ and $\tilde{a}_Q$, are empty.

In case of restriction $\nu a.P^{sf}$ with $a \in fn(P^{sf})$, we have

$$sf(\nu a.P^{sf}) = \nu a.sf(P^{sf}) = \nu a.P^{sf}.$$

The first equation holds by definition of sf, the second relies on the hypothesis. This concludes the proof. ■

Structurally congruent processes do not have the same standard form, i.e., $P \equiv Q$ does not imply $sf(P) = sf(Q)$.

Example 2.1.29 (sf is not invariant under $\equiv$)
Consider the following structurally congruent processes $P \equiv Q_1 \equiv Q_2 \equiv Q_3$:

$$\begin{aligned} P &= a(x) \mid \nu b.\overline{a}\langle b\rangle & sf(P) &= \nu b.(a(x) \mid \overline{a}\langle b\rangle) \\ Q_1 &= \nu b.\overline{a}\langle b\rangle \mid a(x) & sf(Q_1) &= \nu b.(\overline{a}\langle b\rangle \mid a(x)) \\ Q_2 &= a(y) \mid \nu b.\overline{a}\langle b\rangle & sf(Q_2) &= \nu b.(a(y) \mid \overline{a}\langle b\rangle) \\ Q_3 &= a(x) \mid \nu c.\overline{a}\langle c\rangle & sf(Q_3) &= \nu c.(a(x) \mid \overline{a}\langle c\rangle). \end{aligned}$$

The standard form of P is syntactically different from the standard form of any Q_i, i.e., $sf(P) \neq sf(Q_i)$ for $i = 1, 2, 3$. ♦

Example 2.1.29 suggests that the standard forms of structurally congruent processes P and Q differ in three respects. If $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ and $sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}$, then the sequential processes in $P^{\neq\nu}$ may be rearranged in $Q^{\neq\nu}$. The second example shows that sequential processes in $P^{\neq\nu}$ may be replaced by structurally congruent ones in $Q^{\neq\nu}$. Finally, if $P \equiv Q$ is derived with α-conversion then there is a substitution σ renaming $\tilde{a}_Q$ to $\tilde{a}_P$. The relation that permits these transformations on processes in standard form is called *standard equivalence*.

Definition 2.1.30 (Standard Equivalence)
Standard equivalence $\equiv_{sf} \subseteq \mathcal{P}_{sf} \times \mathcal{P}_{sf}$ is the smallest equivalence on processes in standard form where parallel composition is commutative and associative,

$$\nu\tilde{a}.(P_1^{\neq\nu} \mid P_2^{\neq\nu}) \equiv_{sf} \nu\tilde{a}.(P_2^{\neq\nu} \mid P_1^{\neq\nu})$$
$$\nu\tilde{a}.(P_1^{\neq\nu} \mid (P_2^{\neq\nu} \mid P_3^{\neq\nu})) \equiv_{sf} \nu\tilde{a}.((P_1^{\neq\nu} \mid P_2^{\neq\nu}) \mid P_3^{\neq\nu}),$$

where restriction is commutative and restricted names may be alpha-converted,

$$\nu x.\nu y.P^{sf} \equiv_{sf} \nu y.\nu x.P^{sf} \qquad \nu x.P^{sf} \equiv_{sf} \nu y.(P^{sf}\{y/x\})$$

with $\{y\} \cap (fn(P^{sf}) \cup bn(P^{sf})) = \emptyset$, and where non-empty choices may be replaced by structurally congruent ones

$$\nu\tilde{a}.(M^{\neq\mathbf{0}} \mid P^{\neq\nu}) \equiv_{sf} \nu\tilde{a}.(N^{\neq\mathbf{0}} \mid P^{\neq\nu})$$

with $M^{\neq\mathbf{0}} \equiv N^{\neq\mathbf{0}}$ and $P^{\neq\nu}$ optional. ♦

It is immediate to check $sf(P) \equiv_{sf} sf(Q_i)$ for $i = 1, 2, 3$ in Example 2.1.29. Proposition 2.1.31 shows that the standard forms of structurally congruent processes are always related by standard equivalence, $P \equiv Q$ implies $sf(P) \equiv_{sf} sf(Q)$. Function sf is invariant under structural congruence up to standard equivalence. While the proof of this implication is a cumbersome induction on the derivations of structural congruence, the reverse direction, $sf(P) \equiv_{sf} sf(Q)$ implies $P \equiv Q$, holds by definition of $\equiv_{sf}$ and Lemma 2.1.28. Combined, the implications show that *standard equivalence* of $sf(P)$ and $sf(Q)$ *characterises structural congruence* of P and Q.

Proposition 2.1.31 (Characterisation of $\equiv$ with $\equiv_{sf}$)
For all $P, Q \in \mathcal{P}$ we have $P \equiv Q$ if and only if $sf(P) \equiv_{sf} sf(Q)$.

With Proposition 2.1.31 and Lemma 2.1.28, structural congruence and standard equivalence coincide on processes in standard form.

Corollary 2.1.32 ($\equiv$ and $\equiv_{sf}$ coincide on $\mathcal{P}_{sf}$)
For $P^{sf}, Q^{sf} \in \mathcal{P}_{sf}$ we have $P^{sf} \equiv Q^{sf}$ if and only if $P^{sf} \equiv_{sf} Q^{sf}$.

Proof
With Proposition 2.1.31, we have $P^{sf} \equiv Q^{sf}$ if and only if $sf(P^{sf}) \equiv_{sf} sf(Q^{sf})$. According to Lemma 2.1.28, sf is the identity on $\mathcal{P}_{sf}$, i.e., $sf(P^{sf}) = P^{sf}$. Hence, $sf(P^{sf}) \equiv_{sf} sf(Q^{sf})$ if and only if $P^{sf} \equiv_{sf} Q^{sf}$. ■

Standard equivalence enjoys the property that the active restrictions in $\nu\tilde{a}_P.P^{\neq\nu}$ and $\nu\tilde{a}_Q.Q^{\neq\nu}$ can be assumed to be identical. The proof of Lemma 2.1.33 is by

induction on the derivations of standard equivalence and requires the observation that standard equivalence is preserved under the application of substitutions.

Lemma 2.1.33
Consider $P^{sf}, Q^{sf} \in \mathcal{P}_{sf}$ with $P^{sf} = \nu\tilde{a}_P.P^{\neq\nu} \equiv_{sf} \nu\tilde{a}_Q.Q^{\neq\nu} = Q^{sf}$. Then there is a bijective substitution $\sigma : \tilde{a}_Q \to \tilde{a}_P$ so that $Q^{\neq\nu}\sigma \equiv_{sf} P^{\neq\nu}$ holds.

Proof (of Proposition 2.1.31)
$\Rightarrow$ We proceed by induction on the derivations of structural congruence.

Base Cases We consider the axioms of structural congruence.

Case α-conversion Consider $\nu x.P \equiv \nu y.P\{y/x\}$ with $\{y\} \cap (fn(P) \cup bn(P)) = \emptyset$. The case $x \notin fn(P)$ is trivial. If $x \in fn(P)$ we get $y \in fn(P\{y/x\})$ and thus

$$\begin{array}{rcl} & & sf(\nu x.P) \\ (\text{ Def. } sf \text{ }) & = & \nu x.sf(P) \\ (\text{ } \alpha\text{-conversion in } \equiv_{sf} \text{ }) & \equiv_{sf} & \nu y.(sf(P)\{y/x\}) \\ (\text{ Lemma 2.1.28: } sf(P)\{y/x\} = sf(P\{y/x\}) \text{ }) & = & \nu y.sf(P\{y/x\}) \\ (\text{ Def. } sf \text{ }) & = & sf(\nu y.(P\{y/x\})). \end{array}$$

α-conversion of input prefixes yields structurally congruent choices that are non-empty. Thus $\equiv_{sf}$ follows with $M^{\neq\mathbf{0}} \equiv_{sf} N^{\neq\mathbf{0}}$ where $M^{\neq\mathbf{0}} \equiv N^{\neq\mathbf{0}}$.

Case $+$ The rule $M^{\neq\mathbf{0}} \equiv_{sf} N^{\neq\mathbf{0}}$ if $M^{\neq\mathbf{0}} \equiv N^{\neq\mathbf{0}}$ also yields standard equivalence for rewriting non-empty choices using associativity or commutativity of $+$ or $\mathbf{0}$ as neutral element. For empty choices equality holds by definition of sf.

Case $\mid$ The proof for $\mathbf{0}$ as neutral element is straightforward. We consider commutativity of parallel composition, $P \mid Q \equiv Q \mid P$, the proof for associativity is similar. Let $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ and $sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}$, the remaining cases when at least one of the standard forms is $\mathbf{0}$ are trivial:

$$\begin{array}{rcl} & & sf(P \mid Q) \\ (\text{ Def. } sf\text{, form of } sf(P) \text{ and } sf(Q) \text{ }) & = & \nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}) \\ (\text{ Commut. } \mid \text{ and } \nu \text{ in } \equiv_{sf} \text{ }) & \equiv_{sf} & \nu\tilde{a}_Q.\nu\tilde{a}_P.(Q^{\neq\nu} \mid P^{\neq\nu}) \\ (\text{ Def. } sf\text{, form of } sf(Q) \text{ and } sf(P) \text{ }) & = & sf(Q \mid P). \end{array}$$

Case ν The proofs for commutativity and $\mathbf{0}$ as absorbing element are immediate with the definition of sf. The last axiom is scope extrusion. We assume $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ and $sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}$ with $a \in fn(Q)$ and $a \notin fn(P)$. The

remaining cases are simpler:

$$\begin{aligned}
& & & sf(\nu a.(P \mid Q)) \\
& (\text{ Def. } sf, \text{ form of } sf(P) \text{ and } sf(Q) \text{ }) & = & \nu a.\nu\tilde{a}_P.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}) \\
& (\text{ Commut. } \nu \text{ in } \equiv_{sf} \text{ }) & \equiv_{sf} & \nu\tilde{a}_P.\nu a.\nu\tilde{a}_Q.(P^{\neq\nu} \mid Q^{\neq\nu}) \\
& (\text{ Def. } sf, \text{ form of } sf(P) \text{ and } sf(Q) \text{ }) & = & sf(P \mid \nu a.Q).
\end{aligned}$$

Induction Step We assume that $P \equiv Q$ implies $sf(P) \equiv_{sf} sf(Q)$ and similar for $Q \equiv R$. Since $\equiv_{sf}$ is an equivalence, we immediately have that for symmetry $sf(Q) \equiv_{sf} sf(P)$ and for transitivity $sf(P) \equiv_{sf} sf(R)$ holds. The congruence rule $\pi.P + M \equiv \pi.Q + M$ is trivial.

Case $\mid$ For the parallel composition $P \mid R \equiv Q \mid R$, let $sf(R) = \nu\tilde{a}_R.R^{\neq\nu}$ and $sf(P) = \nu\tilde{a}_P.P^{\neq\nu} \equiv_{sf} \nu\tilde{a}_Q.Q^{\neq\nu} = sf(Q)$:

$$\begin{aligned}
& & & sf(Q \mid R) \\
& (\text{ Def. } sf, \text{ form of } sf(Q) \text{ and } sf(R) \text{ }) & = & \nu\tilde{a}_Q.\nu\tilde{a}_R.(Q^{\neq\nu} \mid R^{\neq\nu}) \\
& (\text{ } \alpha\text{-convert } \tilde{a}_Q \text{ to } \tilde{a}_P \text{ with } \sigma, \text{ Lemma 2.1.33 }) & \equiv_{sf} & \nu\tilde{a}_P.(\nu\tilde{a}_R.(Q^{\neq\nu} \mid R^{\neq\nu}))\sigma \\
& (\text{ Applic. } \sigma \text{ }) & = & \nu\tilde{a}_P.\nu\tilde{a}_R.(Q^{\neq\nu}\sigma \mid R^{\neq\nu}\sigma) \\
& & = & \nu\tilde{a}_P.\nu\tilde{a}_R.(Q^{\neq\nu}\sigma \mid R^{\neq\nu}).
\end{aligned}$$

The last equation requires some explanation. Since we consider $Q \mid R$, we have $bn(Q) \cap bn(R) = \emptyset$ and $bn(Q) \cap fn(R) = \emptyset$ by Convention 2.1.11. The domain of σ is $\tilde{a}_Q = arn(sf(Q)) \subseteq arn(Q) \subseteq bn(Q)$. The free names in $R^{\neq\nu}$ are either in $\tilde{a}_R \subseteq bn(R)$ or in $fn(R)$. In both cases, the domain of σ is disjoint with the free names of $R^{\neq\nu}$ and $R^{\neq\nu}\sigma = R^{\neq\nu}$ holds.

To continue our equivalence we assume $Q^{\neq\nu} = \Pi_{i=1}^{n} Q_i$, where the Q_i are choices or calls to process identifiers. With the definition of substitution application we derive $Q^{\neq\nu}\sigma = \Pi_{i=1}^{n} Q_i\sigma$. Since $Q^{\neq\nu}\sigma \equiv_{sf} P^{\neq\nu}$ by Lemma 2.1.33, the definition of standard equivalence yields $P^{\neq\nu} = \Pi_{i=1}^{n} P_i$ so that there is a bijection ϕ between the $Q_i\sigma$ and the P_i with $Q_i\sigma \equiv_{sf} P_{\phi(i)}$. Without loss of generality, we assume the processes P_j ordered so that $Q_i\sigma \equiv_{sf} P_i$. We continue the equivalence by replacing all $Q_i\sigma$ by processes P_i using the rule $\nu\tilde{a}.(M^{\neq\mathbf{0}} \mid P^{\neq\nu}) \equiv \nu\tilde{a}.(N^{\neq\mathbf{0}} \mid P^{\neq\nu})$ with $M^{\neq\mathbf{0}} \equiv N^{\neq\mathbf{0}}$. Note that a similar rule holds for calls to process identifiers as $K\lfloor\tilde{a}\rfloor$ is only structurally congruent with itself[2] and $\nu\tilde{a}.(K\lfloor\tilde{a}\rfloor \mid P^{\neq\nu}) \equiv_{sf} \nu\tilde{a}.(K\lfloor\tilde{a}\rfloor \mid P^{\neq\nu})$ by reflexivity of standard equivalence:

$$\begin{aligned}
& (\text{ Form of } Q^{\neq\nu} \text{ }) & = & \nu\tilde{a}_P.\nu\tilde{a}_R.(Q_1\sigma \mid \Pi_{i=2}^{n} Q_i\sigma \mid R^{\neq\nu}) \\
& (\text{ Discussion above: } Q_1\sigma \equiv P_1 \text{ }) & \equiv_{sf} & \nu\tilde{a}_P.\nu\tilde{a}_R.(P_1 \mid \Pi_{i=2}^{n} Q_i\sigma \mid R^{\neq\nu})
\end{aligned}$$

[2]This holds because we removed all restrictions, in general $K\lfloor\tilde{a}\rfloor \equiv \nu b.K\lfloor\tilde{a}\rfloor$ with $b \notin fn(K\lfloor\tilde{a}\rfloor)$.

$$
\begin{array}{rcl}
(\text{ Assoc. and commut. } \mid \ ) & \equiv_{sf} & \nu\tilde{a}_P.\nu\tilde{a}_R.(Q_2\sigma \mid P_1 \mid \Pi_{i=3}^{n} Q_i\sigma \mid R^{\neq\nu}) \\
(\text{ Replace the remaining } Q_i\sigma \ ) & \equiv_{sf} & \nu\tilde{a}_P.\nu\tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu}) \\
(\text{ Def. } sf\text{, form } sf(P) \text{ and } sf(R) \ ) & = & sf(P \mid R).
\end{array}
$$

Case ν We consider the restrictions $\nu a.P \equiv \nu a.Q$ where $a \in fn(P) = fn(Q)$. If a is not in the free names, standard equivalence follows from the definition of sf and the hypothesis. Let $sf(P) = \nu\tilde{a}_P.P^{\neq\nu}$ and $sf(Q) = \nu\tilde{a}_Q.Q^{\neq\nu}$. Since by the hypothesis $\nu\tilde{a}_P.P^{\neq\nu} \equiv_{sf} \nu\tilde{a}_Q.Q^{\neq\nu}$, Lemma 2.1.33 gives a substitution $\sigma : \tilde{a}_Q \to \tilde{a}_P$ so that $Q^{\neq\nu}\sigma \equiv_{sf} P^{\neq\nu}$. We then replace $Q^{\neq\nu}\sigma$ by $P^{\neq\nu}$ like we did in the case of parallel composition. More precisely, the following equations prove $sf(\nu a.Q) \equiv_{sf} sf(\nu a.P)$:

$$
\begin{array}{rcl}
 & & sf(\nu a.Q) \\
(\text{ Def. } sf\text{, form of } sf(Q) \ ) & = & \nu a.\nu\tilde{a}_Q.Q^{\neq\nu} \\
(\text{ Commut. } \nu \text{ in } \equiv_{sf} \ ) & \equiv_{sf} & \nu\tilde{a}_Q.\nu a.Q^{\neq\nu} \\
(\ \alpha\text{-convert } \tilde{a}_Q \text{ to } \tilde{a}_P \text{ with } \sigma \text{ in Lemma 2.1.33 }) & \equiv_{sf} & \nu\tilde{a}_P.(\nu a.Q^{\neq\nu})\sigma \\
(\text{ Applic. } \sigma \ ) & \equiv_{sf} & \nu\tilde{a}_P.\nu a.(Q^{\neq\nu}\sigma) \\
(\text{ Replace } Q^{\neq\nu}\sigma \text{ by } P^{\neq\nu} \ ) & \equiv_{sf} & \nu\tilde{a}_P.\nu a.P^{\neq\nu} \\
(\text{ Commut. } \nu \ ) & \equiv_{sf} & \nu a.\nu\tilde{a}_P.P^{\neq\nu} \\
(\text{ Def. } sf\text{, form of } sf(P) \ ) & = & sf(\nu a.P).
\end{array}
$$

$\Leftarrow$ All rules making up standard equivalence hold for structural congruence. Hence, $sf(P) \equiv_{sf} sf(Q)$ implies $sf(P) \equiv sf(Q)$. Lemma 2.1.28 yields $P \equiv sf(P)$ and similar for Q. Transitivity of structural congruence gives $P \equiv Q$. ■

2.1.6 Reaction Relation

The behaviour of π-Calculus processes is defined by the so-called *reaction relation* $\to \subseteq \mathcal{P} \times \mathcal{P}$. The inclusion $(P, Q) \in \to$, typically denoted by $P \to Q$, represents a communication of sequential processes in P, which results in process Q. Hence, the reaction relation models *internal* system behaviour. The systems we consider are also called *closed* as they do not interact with an external environment.

Besides the reaction relation, the *labelled transition relation* is commonly used as operational semantics for the π-Calculus (see, e.g. [Mil99, SW01]). Here, processes receive messages from an unknown environment and send messages outside the system. So, the labelled transition relation models *open* systems that communicate with any environment they are plugged into.

We prefer the reaction semantics for two reasons. First, the definition of the labelled transition relation is more involved, which distracts the reader from the

basic ideas underlying the theory of structural stationarity—which is very much concerned with the structure of processes. Second, open systems can be imitated in the reaction semantics by explicitly composing an environment process in parallel with the system under study.

The definition of the reaction relation uses the *structural approach to operational semantics* [Plo81]. Plotkin argues that the states of a transition system, like that of a program or that of a π-Calculus process, have a syntactic structure. They are compositions of basic elements using a set of operators. He then proposes to define transitions between these structured states by a proof system: a transition exists iff it is provable in the proof system. In order to define the behaviour of every state, the proof system uses induction on their structure. It comprises (1) axioms that define the transitions of basic elements and (2) proof rules that define the transitions of composed states from the transitions of the operands. The benefit of structural operational semantics is their simplicity and elegance combined with the possibility to establish properties of transitions by induction on the derivations.

Definition 2.1.34 (Reaction Relation)
The *reaction relation* $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ is defined by the rules in Table 2.3. Let the reflexive and transitive closure be $\rightarrow^* \subseteq \mathcal{P} \times \mathcal{P}$. For a process $P \in \mathcal{P}$, we define the *set of reachable processes* to be $Reach(P) := \{Q \in \mathcal{P} \mid P \rightarrow^* Q\}$. ♦

$$\text{(Tau)} \quad \tau.P + M \rightarrow P$$

$$\text{(React)} \quad x(y).P + M \mid \overline{x}\langle z\rangle.Q + N \rightarrow P\{z/y\} \mid Q$$

$$\text{(Const)} \quad K\lfloor\tilde{a}\rfloor \rightarrow P\{\tilde{a}/\tilde{x}\}, \text{ if } K(\tilde{x}) := P$$

$$\text{(Par)} \quad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \text{(Res)} \quad \frac{P \rightarrow P'}{\nu a.P \rightarrow \nu a.P'}$$

$$\text{(Struct)} \quad \frac{P \rightarrow P'}{Q \rightarrow Q'}, \text{ if } P \equiv Q \text{ and } P' \equiv Q'.$$

Table 2.3: Rules defining the reaction relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$.

Different from Plotkin's classical approach where the proof system only relies on the transition relation, Definition 2.1.34 makes use of the chemical abstract machine idea (cf. Section 2.1.3). All rules except (Struct) define the transitions of representative processes. Rule (Struct) then postulates that a process can do all transitions of the representative it is related to by structural congruence.

Example 2.1.35 (Reaction Relation)
Consider $b(y).y(z) \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle$. Scope extrusion and neutrality of $\mathbf{0}$ for $+$ yield

$$b(y).y(z) \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle \equiv \nu h.(b(y).y(z) + \mathbf{0} \mid \overline{b}\langle h\rangle.\overline{h}\langle b\rangle + \mathbf{0}).$$

We apply Axiom (React) followed by Rule (Res):

$$\frac{b(y).y(z) + \mathbf{0} \mid \overline{b}\langle h\rangle.\overline{h}\langle b\rangle + \mathbf{0} \to h(z) \mid \overline{h}\langle b\rangle}{\nu h.(b(y).y(z) + \mathbf{0} \mid \overline{b}\langle h\rangle.\overline{h}\langle b\rangle + \mathbf{0}) \to \nu h.(h(z) \mid \overline{h}\langle b\rangle).}$$

Using structural congruence, we may insert arbitrarily many $\mathbf{0}$:

$$\nu h.(h(z) \mid \overline{h}\langle b\rangle) \equiv \nu h.(h(z) \mid \overline{h}\langle b\rangle) \mid \mathbf{0} \equiv \nu h.(h(z) \mid \overline{h}\langle b\rangle) \mid \mathbf{0} \mid \mathbf{0}.$$

The reaction and the two congruences allow us to apply Rule (Struct):

$$b(y).y(z) \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle \to \nu h.(h(z) \mid \overline{h}\langle b\rangle) \mid \mathbf{0} \mid \mathbf{0}.$$

♦

Before we continue with deeper investigations of the reaction relation, we define terminating processes and state the fact that reactions do not generate free names.

Definition 2.1.36 (Terminating Process)
A process $P \in \mathcal{P}$ *terminates* if every reaction sequence $P \to P' \to P'' \to \ldots$ is finite. ♦

Lemma 2.1.37
For all $P, Q \in \mathcal{P}$ with $P \to Q$ the inclusion $fn(Q) \subseteq fn(P)$ holds.

In Example 2.1.35, we were lucky to rewrite the process $b(y).y(z) \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle$ in a way that revealed a reaction. To establish properties about all reactions of a process requires a notion of completeness, a certainty that we considered every behaviour the process can exhibit. Proposition 2.1.38 completely characterises all possible reactions of a process in standard form. We explain the intuition.

In a process $P^{sf} = \nu\tilde{a}.P^{\neq\nu}$ in standard form, the scopes of the active restrictions $\nu\tilde{a}$ are maximal. Hence, they do not prevent communications. Processes that are covered by prefixes do not have a reaction. So, we expect P^{sf} to perform a reaction $P^{sf} \to Q$ if and only if one of the following holds.

(1) The process contains a choice composition with a τ prefix that is consumed by the reaction, i.e., we have

$$P^{sf} \quad = \quad \nu\tilde{a}.(P_1^{\neq\nu} \mid M + \tau.P + N \mid P_2^{\neq\nu})$$

$$Q \quad \equiv \quad \nu\tilde{a}.(P_1^{\neq\nu} \mid P \mid P_2^{\neq\nu})$$

for some (possibly empty) set of names $\tilde{a}$ and some processes $P_1^{\neq\nu}, P_2^{\neq\nu}, M, N$ that are optional, i.e., that can be missing.

(2) The second possibility for a reaction is that a process identifier K with $K(\tilde{x}) := P$ calls its defining equation. In this case, we have

$$P^{sf} = \nu\tilde{a}.(P_1^{\neq\nu} \mid K\lfloor\tilde{a}\rfloor \mid P_2^{\neq\nu})$$
$$Q \equiv \nu\tilde{a}.(P_1^{\neq\nu} \mid P\{\tilde{a}/\tilde{x}\} \mid P_2^{\neq\nu}),$$

again for some set of names $\tilde{a}$ and some processes $P_1^{\neq\nu}, P_2^{\neq\nu}$ that may be missing.

(3) The last possibility is that a send prefix $\overline{x}\langle z\rangle.P_1$ sends the name z to a prefixed process $x(y).P_2$. In this case, P^{sf} and Q are of the following form:

$$\begin{aligned} P^{sf} &= \nu\tilde{a}.(P_1^{\neq\nu} \mid M_1 + \overline{x}\langle z\rangle.P_1 + N_1 \mid P_2^{\neq\nu} \mid M_2 + x(y).P_2 + N_2 \mid P_3^{\neq\nu}) \\ Q &\equiv \nu\tilde{a}.(P_1^{\neq\nu} \mid P_1 \mid P_2^{\neq\nu} \mid P_2\{z/y\} \mid P_3^{\neq\nu}), \end{aligned}$$

for some set $\tilde{a}$, some names x, y, z, and some processes $P_1^{\neq\nu}, P_2^{\neq\nu}, P_3^{\neq\nu}$, M_1, M_2, N_1, N_2 that are all optional. It is not necessary that the sending process $M_1 + \overline{x}\langle z\rangle.P_1 + N_1$ precedes the receiving one in the parallel composition. We may as well have

$$\begin{aligned} P^{sf} &= \nu\tilde{a}.(P_1^{\neq\nu} \mid M_2 + x(y).P_2 + N_2 \mid P_2^{\neq\nu} \mid M_1 + \overline{x}\langle z\rangle.P_1 + N_1 \mid P_3^{\neq\nu}) \\ Q &\equiv \nu\tilde{a}.(P_1^{\neq\nu} \mid P_2\{z/y\} \mid P_2^{\neq\nu} \mid P_1 \mid P_3^{\neq\nu}). \end{aligned}$$

but without loss of generality we only consider the first form.

Proposition 2.1.38
For $P^{sf} \in \mathcal{P}_{sf}$ and $Q \in \mathcal{P}$ we have: $P^{sf} \to Q$ if and only if (1), (2), or (3) holds.

We defer the proof of Proposition 2.1.38 until the separate Section 2.1.7. Example 2.1.35 illustrates that the use of structural congruence within the reaction relation and the ability to introduce processes $\mathbf{0}$ under structural congruence yields infinitely many Q with $P \to Q$. The reaction relation is not *image-finite*. But the processes Q in the example are all structurally congruent. In fact, an application of Proposition 2.1.38 shows that for a process P^{sf} in standard form the reaction relation is *image-finite up to structural congruence*. This means P^{sf} can only react to finitely many different processes $Q_1, \ldots, Q_n$, where different means they are not structurally congruent.[3] Since an arbitrary process $P \in \mathcal{P}$

[3] Proposition 2.1.38 reveals the finitely many processes Q_i that have to be chosen. They are given by the right hand sides of the structural congruences in (1), (2), and (3).

reacts to process Q if and only if its standard form $sf(P) \in \mathcal{P}_{sf}$ does the reaction, we conclude that the reaction relation is in general image-finite up to structural congruence.

Lemma 2.1.39
The reaction relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ is image-finite up to structural congruence, i.e., for every process $P \in \mathcal{P}$ the following holds:

$$\exists\{Q_1, \ldots, Q_n\} \subseteq \mathcal{P} : \forall Q \in \mathcal{P} : P \rightarrow Q \text{ implies } \exists i : Q \equiv Q_i.$$

Lemma 2.1.39 is important in the theory of structurally stationary processes as it guarantees finiteness of the Petri net representation. It also indicates that the transition system created by the reaction relation should be factorised along structural congruence to give semantics to a process. Without the factorisation, the transition system is infinitely branching but as the branches are structurally congruent they show the same behaviour by Rule (Struct).

Technically, a *transition system* is a triple $(S, \rightsquigarrow, s_0)$ with *states* S, *transition relation* $\rightsquigarrow \subseteq S \times S$, and *initial state* $s_0 \in S$. We draw transition systems as graphs where the states are vertices, transitions are directed edges, and the initial state has an incoming edge. An example is given in Figure 3.6.

Definition 2.1.40 (Transition System of a Process)
The *transition system of a process* P is $\mathcal{T}(P) := (Reach(P)/_{\equiv}, \rightarrow_{\mathcal{T}}, [P])$ where $\rightarrow_{\mathcal{T}} \subseteq Reach(P)/_{\equiv} \times Reach(P)/_{\equiv}$ with $[Q] \rightarrow_{\mathcal{T}} [Q']$ iff $Q \rightarrow Q'$. ♦

2.1.7 Proof of Proposition 2.1.38

Due to the use of structural congruence within the reaction relation, the proof of Proposition 2.1.38 is intricate. In a monadic π-Calculus with replication, we could apply the *Harmony Lemma* of Sangiorgi and Walker [SW01]. It states that the reactions of the π-Calculus correspond to the τ-labelled transitions modulo structural congruence. We could then conduct an induction on the derivations of transitions to establish the form of processes resulting from τ-transitions.

For the theory of structurally stationary processes, it is beneficial to employ a variant of the π-Calculus with recursion. Thus, the Harmony Lemma cannot be applied directly. We could reprove it in our setting and then proceed as sketched above. The drawback is that this requires the introduction of the labelled transition relation. Furthermore, the proof of the Harmony Lemma is long and non-trivial.

Therefore, we give a different proof of Proposition 2.1.38. In Lemma 2.1.43, we show that any reaction of a process corresponds to a reaction of a process

in standard form, which can be derived without (Struct). More precisely, the process in standard form uses a so-called *standard form reaction relation* $\rightarrow_{sf} \subseteq \mathcal{P}_{sf} \times \mathcal{P}$. It is inspired by Sangiorgi's and Walker's idea of *normalised derivations*, which follow a particular pattern. Rule (Par) is always applied before (Res), and only in the end Rule (Struct) is used.

Definition 2.1.41 ($\rightarrow_{sf} \subseteq \mathcal{P}_{sf} \times \mathcal{P}$)
The *standard form reaction relation* $\rightarrow_{sf} \subseteq \mathcal{P}_{sf} \times \mathcal{P}$ consists of the Axioms (Tau_{sf}), (React_{sf}), and (Const_{sf}), which are identical with (Tau), (React), and (Const) but for $\rightarrow_{sf}$, and the Rules (Par_{sf}) and (Res_{sf}) in Table 2.4. ♦

(Par_{sf})	$\dfrac{P^{\neq\nu} \rightarrow_{sf} P'}{P^{\neq\nu} \mid Q^{\neq\nu} \rightarrow_{sf} P' \mid Q^{\neq\nu}}$	(Res_{sf})	$\dfrac{P^{sf} \rightarrow_{sf} P'}{\nu a.P^{sf} \rightarrow_{sf} \nu a.P'}, a \in fn(P^{sf})$

Table 2.4:
Rules defining the standard form reaction relation $\rightarrow_{sf} \subseteq \mathcal{P}_{sf} \times \mathcal{P}$.

The relation is actually well-defined, i.e., the left hand side of $P \rightarrow_{sf} Q$ is always in standard form. Since $\rightarrow_{sf}$ does not rely on structural congruence, we can establish the precise form of processes Q resulting from standard form reactions by induction.

Lemma 2.1.42
For all $P^{sf} \in \mathcal{P}_{sf}$ and $Q \in \mathcal{P}$ we have $P^{sf} \rightarrow_{sf} Q$ if and only if

(a) $P^{sf} = \nu\tilde{a}.(K\lfloor\tilde{a}\rfloor \mid P^{\neq\nu})$ and $Q = \nu\tilde{a}.(P\{\tilde{a}/\tilde{x}\} \mid P^{\neq\nu})$ with $K(\tilde{x}) := P$,

(b) $P^{sf} = \nu\tilde{a}.(\tau.P + M \mid P^{\neq\nu})$ and $Q = \nu\tilde{a}.(P \mid P^{\neq\nu})$, or

(c) $P^{sf} = \nu\tilde{a}.(x(y).P_1 + M_1 \mid \overline{x}\langle z\rangle.P_2 + M_2 \mid P^{\neq\nu})$ and
$Q = \nu\tilde{a}.(P_1\{z/y\} \mid P_2 \mid P^{\neq\nu})$

for some process $P^{\neq\nu}$ and some set of names $\tilde{a}$ that are both optional.

We now state the mentioned correspondence of the reaction and the standard form reaction relation.

Lemma 2.1.43 (Characterisation of $\rightarrow$ by $\equiv\rightarrow_{sf}\equiv$)
For all $P, Q \in \mathcal{P}$ the following equivalence holds: $P \rightarrow Q$ if and only if $P^{sf} \rightarrow_{sf} Q'$ for some $P^{sf} \in \mathcal{P}_{sf}$ and $Q' \in \mathcal{P}$ with $P \equiv P^{sf}$ and $Q \equiv Q'$.

Proof
$\Leftarrow$ Since $P^{sf} \rightarrow_{sf} Q'$ implies $P^{sf} \rightarrow Q'$, we get $P \rightarrow Q$ with Rule (Struct).

$\Rightarrow$ We use induction on the derivations of $\rightarrow$. The base cases are given by the Axioms (Tau), (React), and (Const), each of which is countered by the corresponding axiom indexed by sf.

Induction Step Assume we have for $P \rightarrow Q$ a derivation $P^{sf} \rightarrow_{sf} Q'$ with $P \equiv P^{sf}$ and $Q \equiv Q'$.

Case $P \mid R \rightarrow Q \mid R$ We compute the standard form $R \equiv sf(R) = \nu\tilde{a}_R.R^{\neq\nu}$. Let P^{sf} be the process $\nu\tilde{a}_P.P^{\neq\nu}$, where we can assume $\tilde{a}_P \cap (fn(R) \cup bn(R)) = \emptyset$ with Lemma 2.1.42. The lemma also reveals that $\nu\tilde{a}_P.P^{\neq\nu} \rightarrow_{sf} Q'$ implies $Q' = \nu\tilde{a}_P.Q''$ with $P^{\neq\nu} \rightarrow_{sf} Q''$. An application of Rule (Par_{sf}) followed by several applications of (Res_{sf}) that we contract to one gives:

$$\frac{\dfrac{P^{\neq\nu} \rightarrow_{sf} Q''}{P^{\neq\nu} \mid R^{\neq\nu} \rightarrow_{sf} Q'' \mid R^{\neq\nu}}}{\nu\tilde{a}_P.\nu\tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu}) \rightarrow_{sf} \nu\tilde{a}_P.\nu\tilde{a}_R.(Q'' \mid R^{\neq\nu}).}$$

It remains to show structural congruence of $P \mid R$ and $\nu\tilde{a}_P.\nu\tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu})$. By the hypothesis $P \equiv P^{sf} = \nu\tilde{a}_P.P^{\neq\nu}$. We showed above that $R \equiv \nu\tilde{a}_R.R^{\neq\nu}$ holds. Scope extrusion now yields the congruence (cf. the proof of Lemma 2.1.28 to see why the disjointness assumption validates the scope extrusion). Similarly, we get $Q \mid R \equiv \nu\tilde{a}_P.\nu\tilde{a}_R.(Q'' \mid R^{\neq\nu})$.

Case $\nu a.P \rightarrow \nu a.Q$ If $a \notin fn(P)$ then $a \notin fn(Q)$ by Lemma 2.1.37. We thus have $\nu a.P \equiv P \equiv P^{sf}$ and $\nu a.Q \equiv Q \equiv Q'$ with $P^{sf} \rightarrow_{sf} Q'$. If $a \in fn(P)$, we have $a \in fn(P^{sf})$ by the invariance of free names under structural congruence. Hence, $\nu a.P^{sf}$ is in standard form. Furthermore, $\nu a.P^{sf} \rightarrow_{sf} \nu a.Q'$ with (Res_{sf}). That $\nu a.P \equiv \nu a.P^{sf}$ and $\nu a.Q' \equiv \nu a.Q$ by the hypothesis concludes the case.

For a derivation with Rule (Struct) the claim holds trivially by the hypothesis and transitivity of structural congruence. ■

With the help of Lemma 2.1.43 and Lemma 2.1.42, we establish Proposition 2.1.38.

Proof (of Proposition 2.1.38)
The implication from right to left is immediate. We prove the reverse direction. With Lemma 2.1.43, $P^{sf} \rightarrow Q$ implies there is a process $R^{sf} \rightarrow_{sf} Q'$ with $P^{sf} \equiv R^{sf}$ and $Q \equiv Q'$. Lemma 2.1.42 gives precise information about the form of R^{sf} and Q'. We only consider Case (c) the remaining cases are simpler:

$$R^{sf} = \nu\tilde{a}_R.(x(y).R_1 + M_1 \mid \overline{x}\langle z\rangle.R_2 + M_2 \mid R^{\neq\nu})$$
$$Q' = \nu\tilde{a}_R.(R_1\{z/y\} \mid R_2 \mid R^{\neq\nu}).$$

By Corollary 2.1.32, structural congruence $P^{sf} \equiv R^{sf}$ implies standard equivalence $P^{sf} \equiv_{sf} R^{sf}$. Let P^{sf} be the process $\nu\tilde{a}_P.P^{\neq\nu}$. Lemma 2.1.33 guarantees the existence of a substitution $\sigma : \tilde{a}_R \to \tilde{a}_P$ so that

$$\begin{array}{rcl} & & (x(y).R_1 + M_1 \mid \overline{x}\langle z\rangle.R_2 + M_2 \mid R^{\neq\nu})\sigma \\ (\text{ Applic. } \sigma) & = & x\sigma(y).R_1\sigma + M_1\sigma \mid \overline{x\sigma}\langle z\sigma\rangle.R_2\sigma + M_2\sigma \mid R^{\neq\nu}\sigma \\ (\text{ Choice of } \sigma) & \equiv_{sf} & P^{\neq\nu}. \end{array}$$

By definition of standard equivalence, there is a bijection between the sequential processes in $x\sigma(y).R_1\sigma + M_1\sigma \mid \overline{x\sigma}\langle z\sigma\rangle.R_2\sigma + M_2\sigma \mid R^{\neq\nu}\sigma$ and the sequential processes in $P^{\neq\nu}$. This means we have

$$P^{\neq\nu} = P_1^{\neq\nu} \mid N_1 \mid P_2^{\neq\nu} \mid N_2 \mid P_3^{\neq\nu},$$

where $P_1^{\neq\nu} \mid P_2^{\neq\nu} \mid P_3^{\neq\nu}$ is structurally congruent with $R^{\neq\nu}\sigma$. Since $R^{\neq\nu}\sigma$ need not be split into three parts, each of the $P_i^{\neq\nu}$ may be missing. Moreover, N_1 is structurally congruent with $x\sigma(y).R_1\sigma + M_1\sigma$ and N_2 is structurally congruent with $\overline{x\sigma}\langle z\sigma\rangle.R_2\sigma + M_2\sigma$ or vice versa. By definition of structural congruence, $N_1 \equiv x\sigma(y).R_1\sigma + M_1\sigma$ implies N_1 is a choice composition

$$N_1 = N_1^1 + x\sigma(y').R_1' + N_1^2,$$

where N_1^1 and N_1^2 may be missing and $R_1' \equiv R_1\sigma\{y'/y\}$ holds. We argue similarly for N_2 and get

$$N_2 = N_2^1 + \overline{x\sigma}\langle z\sigma\rangle.R_2' + N_2^2,$$

again with optional choices N_2^1 and N_2^2 and $R_2' \equiv R_2\sigma$. This shows that P^{sf} is of the desired form:

$$P^{sf} = \nu\tilde{a}_P.(P_1^{\neq\nu} \mid N_1^1 + x\sigma(y').R_1' + N_1^2 \mid P_2^{\neq\nu} \mid N_2^1 + \overline{x\sigma}\langle z\sigma\rangle.R_2' + N_2^2 \mid P_3^{\neq\nu}).$$

We now consider Q and show that it is structurally congruent with the process

$$\nu\tilde{a}_P.(P_1^{\neq\nu} \mid R_1'\{z\sigma/y'\} \mid P_2^{\neq\nu} \mid R_2' \mid P_3^{\neq\nu}).$$

The idea is to apply the substitution $\sigma : \tilde{a}_R \to \tilde{a}_P$ to process Q' from Lemma 2.1.43 and show that Q' is structurally congruent with a process of the desired form. Then, by transitivity of structural congruence, the statement holds for Q:

$$\begin{array}{rcl} & & Q' \\ (\text{ Form of } Q') & = & \nu\tilde{a}_R.(R_1\{z/y\} \mid R_2 \mid R^{\neq\nu}) \\ (\ \alpha\text{-conversion }) & \equiv & \nu\tilde{a}_P.((R_1\{z/y\} \mid R_2 \mid R^{\neq\nu})\sigma) \\ (\text{ Applic. } \sigma) & = & \nu\tilde{a}_P.(R_1\{z/y\}\sigma \mid R_2\sigma \mid R^{\neq\nu}\sigma) \\ (\text{ Explained below }) & = & \nu\tilde{a}_P.(R_1\sigma\{z\sigma/y\} \mid R_2\sigma \mid R^{\neq\nu}\sigma) \end{array}$$

$$
\begin{aligned}
(\text{ Composition of } \sigma\ ) &= \nu\tilde{a}_P.(R_1\sigma\{y'/y\}\{z\sigma/y'\} \mid R_2\sigma \mid R^{\neq\nu}\sigma)\\
(\ R_1\sigma\{y'/y\} \equiv R_1'\ ) &\equiv \nu\tilde{a}_P.(R_1'\{z\sigma/y'\} \mid R_2\sigma \mid R^{\neq\nu}\sigma)\\
(\ R_2\sigma \equiv R_2'\ ) &\equiv \nu\tilde{a}_P.(R_1'\{z\sigma/y'\} \mid R_2' \mid R^{\neq\nu}\sigma)\\
(\ R^{\neq\nu}\sigma \equiv P_1^{\neq\nu} \mid P_2^{\neq\nu} \mid P_3^{\neq\nu}\ ) &\equiv \nu\tilde{a}_P.(R_1'\{z\sigma/y'\} \mid R_2' \mid P_1^{\neq\nu} \mid P_2^{\neq\nu} \mid P_3^{\neq\nu})\\
(\text{ Ass. and commut. } \mid\ ) &\equiv \nu\tilde{a}_P.(P_1^{\neq\nu} \mid R_1'\{z\sigma/y'\} \mid P_2^{\neq\nu} \mid R_2' \mid P_3^{\neq\nu}).
\end{aligned}
$$

Since y is bound in $x(y).R_1 + M_1 \mid \overline{x}\langle z\rangle.R_2 + M_2 \mid R^{\neq\nu}$, we have $y\sigma = y$ and $a\sigma \neq y$ for $a \neq y$ by Convention 2.1.13. This justifies the equation above and concludes the proof. ■

2.2 Place/Transition Petri Nets

In the early 1960s, Carl Adam Petri searched for a theoretical model of information flow in the upcoming concurrent systems. In his PhD thesis [Pet62], he proposed an automata-theoretic formalism that became the most prominent model in concurrency theory. The idea of *Petri nets* is to generalise finite automata by *distributed states* and *explicit synchronisation* of transitions.

To explain *distributed states*, consider the place/transition Petri net in Figure 2.2. It loosely reflects the client/server system in Section 2.1.[4] The left part of the net models the states of the client, the right part the behaviour of the server. More precisely, Petri nets use *places*, depicted by circles, that correspond to states in a finite automaton. A dot on a place, called a *token*, means the system part is in the marked state. The *state* of the full system is given by a function that counts the tokens in all places. *Transitions* are boxes with incoming and outgoing arcs. When a transition is executed, also called *fired*, a token is removed for each incoming arc. Similarly, for each outgoing arc a token is produced. In the example, transition t_0 changes the client's state from s_0 to s_2, transition t_1 changes the state of the server from s_1 to s_3. When the system starts in state (a), executing the two transitions in any order yields state (b).

Transition t_2 models the registration of the client at the server. Since the transition removes a token from the client's state s_2 and from the server's state s_3, it *explicitly synchronises* both systems. The result is state (c) with a token on s_4. Intuitively, the place represents a computation unit consisting of client and server. Transition t_3 lets the unit break apart, which reproduces the initial state (a).

The example shows that the execution of transitions (like t_2 or t_3) changes the amount of tokens in a Petri net. If a Petri net admits execution sequences where the number of tokens grows arbitrarily, then it reaches infinitely many states.

[4] The precise relationship between π-Calculus and Petri nets will the topic of Chapter 3 and Chapter 9. The client/server system is investigated in Chapter 5.

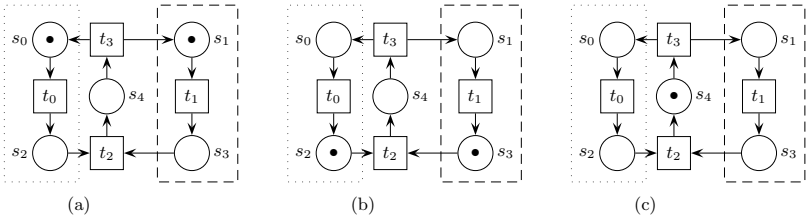


Figure 2.2:
Three states of a place/transition Petri net modelling a client/server system. The client's state space is surrounded by dotted frames, the server states by dashed frames. Transitions between the states are explained in the text.

Although infinite-state, interesting verification problems remain decidable for Petri nets. The most famous result is decidability of the reachability problem, which was settled independently by Mayr 1981 and Kosaraju 1982 and had been open for twenty years [May84, Kos82].

In this thesis, we investigate translations of π-Calculus models for DRS into place/transition Petri nets. We develop a theory that allows us to draw conclusions about a process from the verification of its Petri net representation. The study of Petri nets themselves is not subject to this thesis. They serve us as background theory to which we reduce problems in DRS. However, the *processes of bounded depth* we investigate in Chapter 7 and 8 illustrate how the theory of coverability trees is lifted from Petri nets to the more general well-structured transition systems of Finkel and Abdulla [Fin90, FS01, AČJT00].

We use standard notions of place/transition Petri nets with weights on arcs that follow the presentation in [Rei85]. Variants of the basic model, in particular high-level Petri nets with data values as tokens, are not considered in this thesis. Therefore, whenever we refer to Petri nets, we mean *place/transition* Petri nets. The presentation of unfoldings is taken from the book of Esparza and Heljanko [EH08]. Instead of defining the more common coverability graphs, we use *coverability trees* along the lines of [PW03].

2.2.1 Syntax and Semantics

As the states of Petri nets are functions, we formally define the notations we use.

Definition 2.2.1 (Functions)
The *set of all functions* $\phi : A \to B$ from *domain* A into *codomain* B is B^A. The *image* of A under ϕ is $\phi(A) := \{b \in B \mid b = \phi(a) \text{ for some } a \in A\}$. We

only consider *total functions*. Two functions $\phi, \psi \in B^A$ are *equal*, $\phi = \psi$, if $\phi(a) = \psi(a)$ for all $a \in A$. Similarly, function $\phi \in \mathbb{N}^A$ is *smaller than* $\psi \in \mathbb{N}^A$, $\phi \leq \psi$, if it is componentwise smaller, i.e., $\phi(a) \leq \psi(a)$ for all $a \in A$. We write $\phi < \psi$ to indicate that $\phi \leq \psi$ but $\phi \neq \psi$. The *sum* of ϕ and ψ in $\mathbb{N}^A$ is the function $\phi + \psi \in \mathbb{N}^A$ with $(\phi + \psi)(a) := \phi(a) + \psi(a)$. Finally, the *support* of $\phi \in \mathbb{N}^A$ is the set of elements that are mapped to a value greater zero, i.e., $supp(\phi) = \{a \in A \mid \phi(a) > 0\}$. ♦

Definition 2.2.2 (Place/Transition Petri Net)
An *unmarked place/transition Petri net* is a triple (S, T, W), where

S is a (potentially infinite) set of *places*,

T is a (potentially infinite) set of *transitions* disjoint from S,

$W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a total *weight function* giving the number of arcs from $s \in S$ to $t \in T$ and vice versa.

The states of Petri nets are functions also called *markings* $M, N \in \mathbb{N}^S$. We refer to an (unmarked) place/transition Petri net with initial marking M_0 as *marked* place/transition Petri net or Petri net $\mathcal{N} = (S, T, W, M_0)$. We denote the *set of all Petri nets* by $\mathcal{PN}$. We call a Petri net *finite* if S and T are finite. ♦

Convention 2.2.3
In a Petri net $\mathcal{N} = (S, T, W, M_0)$ the places always carry indices in the natural numbers, i.e., there is an index set $I \subseteq \mathbb{N}$ with $S = \{s_i \mid i \in I\}$. ♦

As explained in the introduction, we draw places by circles, transitions by boxes, and tokens by dots in the circles. For arcs, we obey the following convention. If $W(s,t) = 0$, no arc from s to t is inserted. We draw unlabelled arcs as long as $W(s,t) = 1$, and label arcs by k for $W(s,t) = k > 1$.

In order to compare the size of a Petri net with the size of the translated process, we sum up the numbers of places, transitions, arcs and tokens in the initial marking.

Definition 2.2.4 ($\| - \| : \mathcal{PN} \to \mathbb{N}$)
The *size* of a Petri net $\mathcal{N} = (S, T, W, M_0)$ is

$$\|\mathcal{N}\| := |S| + |T| + \Sigma_{s \in S} \Sigma_{t \in T} (W(s,t) + W(t,s)) + \Sigma_{s \in S} M_0(s).$$

♦

Let S' be a superset of the places S of Petri net $\mathcal{N}$. It will be helpful to consider those functions in $\mathbb{N}^{S'}$, which map elements outside S to zero, as markings of $\mathcal{N}$.

Convention 2.2.5
Consider $\mathcal{N} = (S, T, W, M_0)$ and a set S' with $S \subseteq S'$. There is a bijection between $\mathbb{N}^S$ and $\mathbb{N}^{S'}_{supp(S)} := \{M' \in \mathbb{N}^{S'} \mid M'(s') = 0 \text{ for all } s' \in S' \setminus S\}$. More precisely, the following function *res* is a bijection:

$$\begin{aligned} res : \mathbb{N}^{S'}_{supp(S)} &\to \mathbb{N}^S \\ M' &\mapsto res(M') \text{ with } res(M')(s) = M'(s) \text{ for all } s \in S. \end{aligned}$$

Hence, we can understand $M' \in \mathbb{N}^{S'}_{supp(S)}$ as marking $res(M')$ of $\mathcal{N}$. Furthermore, we say that M' *equals* a marking $M \in \mathbb{N}^S$, if $res(M') = M$ holds. ♦

Consider a Petri net $\mathcal{N} = (S, T, W, M_0)$ with $t \in T$. The set of places transition t consumes tokens from is the *preset* of t, ${}^\bullet t := \{s \in S \mid W(s,t) > 0\}$. The places t produces tokens on are in the *postset* of t, $t^\bullet := \{s \in S \mid W(t,s) > 0\}$. Pre- and postsets of places are defined similarly. *Communication-free* Petri nets are a subclass of nets where transitions do not synchronise. The syntactic restriction is that each transition has a single place in its preset, which is connected to the transition with an arc of weight one. Communication-free nets can be interpreted as (potentially unbounded) number of finite automata running concurrently. We shall translate closed processes to communication-free Petri nets.

Definition 2.2.6 (Communication-free Petri Net)
A Petri net $\mathcal{N} = (S, T, W, M_0)$ is *communication-free*, if for all $t \in T$ and all $s \in S$ the following holds: $|{}^\bullet t| \leq 1$ and $W(s,t) \leq 1$. ♦

The behaviour of a Petri net is given by a transition relation. It states how marking M changes to M' when executing transition t.

Definition 2.2.7 (Transition Relation)
Consider the Petri net $\mathcal{N} = (S, T, W, M_0)$. A transition $t \in T$ is *enabled* under a marking $M \in \mathbb{N}^S$, if $M(s) \geq W(s,t)$ for all $s \in {}^\bullet t$. The *transition relation*, also called *firing relation*, $[\rangle \subseteq \mathbb{N}^S \times T \times \mathbb{N}^S$ is defined by

$$\begin{aligned} M[t\rangle M' \text{ iff } & t \text{ is enabled under } M \text{ and} \\ & M'(s) = M(s) - W(s,t) + W(t,s), \text{ for all } s \in S. \end{aligned}$$

♦

When we compute the transition system of a Petri net, we omit the identity of transitions, i.e., we use the *unlabelled transition relation* $\to \subseteq \mathbb{N}^S \times \mathbb{N}^S$ defined by $M \to M'$ whenever $M[t\rangle M'$ for some $t \in T$. As usual, we denote the reflexive and transitive closure of $\to$ by $\to^*$. The *set of all markings reachable from a*

marking $M \in \mathbb{N}^S$ is $Reach(M) := \{M' \mid M \to^* M'\}$. By $Reach(\mathcal{N})$ we refer to the states reachable from the initial marking, $Reach(\mathcal{N}) := Reach(M_0)$.

Definition 2.2.8 (Transition System)
The *transition system* of $\mathcal{N} = (S, T, W, M_0)$ is created by the unlabelled transition relation with M_0 as initial state, i.e., $\mathcal{T}(\mathcal{N}) := (Reach(M_0), \to, M_0)$. ♦

It is worth noting that the semantics is a total function. In particular, the Petri net $\mathcal{N}_\emptyset = (\emptyset, \emptyset, \emptyset, \emptyset)$ is assigned the transition system where the empty function $\emptyset : \emptyset \to \emptyset$ is the initial marking, i.e., $\mathcal{T}(\mathcal{N}_\emptyset) = (\{\emptyset\}, \emptyset, \emptyset)$. Like for processes, we define termination of a Petri net.

Definition 2.2.9 (Terminating Petri net)
A Petri net $\mathcal{N} = (S, T, W, M_0)$ *terminates*, if every transition sequence $M_0 \to M_1 \to M_2 \to \ldots$ is finite. ♦

The state space of a Petri net is finite if and only if there is a natural number that bounds the numbers of tokens in all places. There are some boundedness restrictions commonly used in net theory, which define different classes of nets.

Definition 2.2.10 (Bounded and Safe Petri nets)
A Petri net $\mathcal{N} = (S, T, W, M_0)$ is *k-bounded* for $k \in \mathbb{N}$ if no reachable marking puts more than k tokens on a place, i.e., $\forall M \in Reach(\mathcal{N}) : \forall s \in S : M(s) \leq k$. A net is *bounded* if it is k-bounded for some $k \in \mathbb{N}$ and *safe* if it is 1-bounded. ♦

2.2.2 S-Invariants

S-Invariants are constraints on the distribution of tokens in all reachable markings of a Petri net. They may be interpreted as over-approximations of the state space. Technically, S-invariants are solutions to homogenous equations. Hence, their computation does not require costly state space explorations but relies on methods from linear algebra.

Definition 2.2.11 (Incidence Matrix and S-Invariant)
For a Petri net $\mathcal{N} = (S, T, W, M_0)$, the *incidence matrix* is $C : S \times T \to \mathbb{Z}$ with $C(s,t) := W(t,s) - W(s,t)$. An *S-Invariant* of $\mathcal{N}$ is a vector $I \in \mathbb{N}^S$ with $C^t \cdot I = 0$, where C^t is the transposed incidence matrix and $\_ \cdot \_$ is the matrix product. ♦

Consider the Petri net in Figure 2.2. We understand the function $I \in \mathbb{N}^S$ with $I(s_0) = 1 = I(s_1) = I(s_2) = I(s_3)$ and $I(s_4) = 2$ as vector $I = (1, 1, 1, 1, 2)^t$,

where entry i is $I(s_i)$. To show that it is an S-invariant, we compute the product of I with the incidence matrix, where we also let entry i, j be $C(s_i, t_j)$:

$$C^t \cdot I = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 & 1 \\ 1 & 1 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

An S-invariant assigns a weight to each place. In the example, every place is weighted by one except s_4, which is weighted by two. The fundamental property of S-Invariants is that the weighted sum of tokens is invariant under transitions.

Lemma 2.2.12 (Fundamental Property of S-Invariants)
Let M, M' be two markings of a Petri net with $M \rightarrow^* M'$, let I be an S-Invariant. Then $I^t \cdot M = I^t \cdot M'$ holds.

Given the initial marking $M_0 = (1, 1, 0, 0, 0)^t$ and the S-invariant I, we compute the weighted sum of tokens in all reachable markings of the client/server system:

$$I^t \cdot M_0 = (1, 1, 1, 1, 2) \cdot (1, 1, 0, 0, 0)^t = 2.$$

With Lemma 2.2.12, a token on s_4 implies the remaining places are empty. The example explains the use of S-invariants to prove mutual exclusion properties. We shall see another application of S-invariants in Chapter 6. Note that for these realistic case studies, S-invariants can no longer be computed by hand but we rely on mature tools like the Integrated Net Analyser developed in the group of Starke [Sta03].

2.2.3 Unfoldings

Unfoldings are efficient encodings of the linear-time behaviour of Petri nets. For the sake of brevity, we restrict ourselves to safe Petri nets in this section. In the literature, unfoldings have also been proposed for bounded [Kho03] and unbounded Petri nets [EH08]. We explain the idea of unfoldings on the client/server example in Figure 2.2. In the transition system, a local state change of the client from s_0 to s_2 yields a new global marking of the whole Petri net. In the unfolding in Figure 2.3, the state is distributed and the state change is recorded locally by transition e_0, which removes a token from c_0 and adds a token to c_2. The state c_1 of the server is not altered. Consequently, unfoldings alleviate the problem of diamonds in the transition systems of concurrent systems, which arise when components change their states independently. While the number of states in the transition system grows exponentially in the number of independent executions,

the unfolding's size increases linearly. Therefore, it is more efficient in terms of memory to compute the unfolding instead of the state space of a Petri net—we rely on the tool called PUNF that was developed by Victor Khomenko [Kho08]

The technical definition of unfoldings is rather involved. We only explain the basic concepts and refer to [EH08] for details. The *unfolding* of a Petri net $\mathcal{N}$ is an acyclic and safe Petri net $\mathcal{U}$. Different from standard Petri nets, the places of $\mathcal{U}$ are called *conditions* and are labelled by places of $\mathcal{N}$. The transitions in $\mathcal{U}$ are called *events* and labelled by transitions in $\mathcal{N}$. The terms are justified by the intuition that an event in the unfolding represents an occurrence of the transition it is labelled with in an execution of the Petri net. To compute $\mathcal{U}$ from $\mathcal{N}$, we iteratively fire all transitions in $\mathcal{N}$ and record their effect in the unfolding. Instead of formalising the procedure, we explain it on the example.

Example 2.2.13
Consider the Petri net in Figure 2.2. Initially, s_0 and s_1 are marked and we add conditions c_0 and c_1 labelled by s_0 and s_1 to the unfolding. In the initial marking, t_0 is enabled. Firing the transition removes the token from s_0 and adds a token to s_2. In the unfolding, we add an event e_0 labelled by t_0 with c_0 in its preset. For the token created on s_2, we add a new condition c_2 in the postset of e_0, which we label by s_2. Figure 2.3 shows some more events and conditions. The unfolding is not finite. ♦

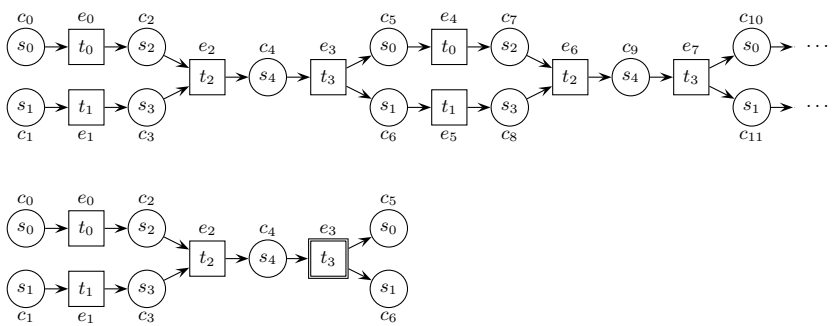


Figure 2.3:
The Petri net above is the unfolding of the client/server net in Figure 2.2. The names of places and transitions are written outside, their labels inside the circles and boxes. The Petri net below is a finite and complete prefix of the unfolding with e_3 as cut-off event.

The reachable markings of a Petri net are reflected by so-called *configurations* of its unfolding. A configuration C is a set of events that satisfies the following two conditions. (1) It is *causally closed*, i.e., if event f precedes e in the unfolding and e is in C, then also f is in C. (2) It is *conflict-free*, which means it does not contain two events $e \neq f$ that share a common condition in their presets. In the example, $C = \{e_0, e_1, e_2\}$ is a configuration, $\{e_2\}$ is not. A configuration of $\mathcal{U}$ yields at least one transition sequence in $\mathcal{N}$, called a *realisation*. All realisations yield the same marking. For example, t_0, t_1, t_2 and t_1, t_0, t_2 are the two possible realisations of $C = \{e_0, e_1, e_2\}$ and both put a single token on s_4. It can be shown that the reachable markings in the net $\mathcal{N}$ are precisely those that are reached by realisations of configurations of the unfolding.

If the Petri net has an infinite run, the unfolding is infinite. Since a safe Petri net reaches only finitely many markings, longer configurations start to repeat markings of shorter ones. The computation of the unfolding stops and returns a so-called *finite and complete prefix*, cf. Figure 2.3. The events that reproduce already known markings are called *cut-off*. It is guaranteed that the number of non-cut-off events is bounded by the number of reachable markings in the Petri net, and often finite and complete prefixes are significantly smaller than transition systems.

Esparza and Heljanko explain how to rephrase verification problems for Petri nets on finite and complete prefixes. In particular, properties in linear-time temporal logic can be restated as properties of the prefix. To establish properties of prefixes, we recall a technique due to Heljanko and Khomenko in Section 5.4.

2.2.4 Coverability Trees

Coverability trees are finite representations of the infinite state spaces of unbounded Petri nets, which allow for deciding verification problems like termination and infinity of states. As we shall never compute coverability trees explicitly in this thesis, we only explain the idea but do not define them formally. We decided to introduce them for they explain well the verification of infinite-state systems, they are fundamental to the computation of the structural semantics (cf. Section 3.5), and they serve us as intuition to finite reachability trees in Section 8.2.

The idea of coverability trees is to build the computation tree and detect situations that lead to an unbounded number of tokens. In this case, not the precise amount of tokens is recorded in the tree, but *generalised* markings M_ω, N_ω with ω as token count are used. An ω-entry indicates that there are computations where the number of tokens on that place exceeds any natural number.

Technically, the root of the coverability tree is a vertex labelled by the initial marking of the Petri net. For every vertex labelled by a marking M_ω, we compute all markings N_ω reachable by firing a transition $M_\omega[t\rangle N_\omega$. (The token count ω

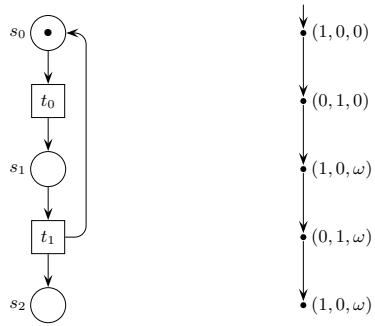


Figure 2.4:
An unbounded Petri net and its coverability tree. Firing t_1 in state $M_1 = (0,1,0)$ yields $M_2 = (1,0,1)$. Since for the initial marking $M_0 = (1,0,0) < (1,0,1) = M_2$ holds, we update the marking to $M_2' = (1,0,\omega)$.

always enables a transition and is never consumed.) Now for any vertex on the path from the root to M_ω it is checked whether it is labelled by $M_\omega' < N_\omega$. (Of course, ω is considered to be larger than any natural number.) In this case, marking N_ω is updated to N_ω', where an ω is introduced in any place s with $M_\omega'(s) < N_\omega(s)$. A new vertex labelled by N_ω' is added to the tree and connected with the vertex of M_ω. The computation stops when no more new markings are found. Figure 2.4 illustrate the procedure.

It can be shown that the coverability tree is always finite. To relate the states in a Petri net with the labels in the coverability tree, we observe that any reachable marking M in the net is covered by a label M_ω in the tree, i.e., $M \leq M_\omega$. Conversely, if there is an M_ω in the tree with entries ω for some places, then there are markings where the amount of tokens in these places grows unboundedly.

We conclude with a remark on what can be decided with the help of the coverability tree. Of course, finiteness of the state space and hence k-boundedness, and termination. Moreover, it is decidable whether a given marking M is coverable, i.e., a marking N is reachable with $M \leq N$. In the tool PETRUCHIO, we use this procedure to compute the places that are markable simultaneously—a crucial problem to be solved when computing the structural semantics (cf. Section 3.5).

In the literature, usually so called *coverability graphs* instead of coverability trees are found, where vertices with identical labels are stored only once. We chose coverability trees since they are easier to explain and fit better to the finite reachability trees in Section 8.2.

3 A Structural Semantics for the π-Calculus

Contents

We develop a new translation of π-Calculus processes into place/transition Petri nets. For process algebras the investigation of automata-theoretic models has a long standing tradition. The classical question was to find representations that reflect the concurrency of processes (cf. [Old91] for an overview). Our translation exploits the connections created by restricted names instead. We view a process not as a group of sequential programs running concurrently, but as a structured object, a graph where the references to restricted names connect processes. We therefore call our translation a *structural semantics* to distinguish it from classical concurrency semantics. As Chapter 4 shows, the benefit of taking the viewpoint of structure instead of concurrency are finite net representations for processes with unboundedly many restricted names and unbounded parallelism.

Technically, the definition of the structural semantics differs from semantics known from the literature as follows. Concurrency semantics decompose processes along the parallel composition operator, i.e., they represent the sequential processes by places in the Petri net. We transform the process into a normal form that we call *restricted form*. A process in restricted form is a parallel composition of *fragments*, groups of processes connected by restricted names. We decompose the process along the parallel composition of fragments, i.e., the fragments are the places in our Petri net semantics.

In the introduction, we discussed the quality aspects we believe a Petri net semantics has to satisfy to be useful for verification. In this section we establish *retrievability*, i.e., the state space of the Petri net reflects the behaviour of the process. More precisely, we show that the transition system of the process and that of its Petri net representation are isomorphic. We also give the inverse of the isomorphism. It reconstructs—up to structural congruence—from a Petri net marking the corresponding reachable process. Hence, the translation not only preserves the transition behaviour but also the term structure of the reachable processes. We call this result, which we state in Theorem 3.4.3, *full retrievability*. To preserve the term structure of the reachable processes allows one to reason about properties defined in spatial logics like [CC03] using our Petri net representation.

The more information a semantics preserves about a process, the more memory is needed to store it. We argue that our semantics preserves exactly the information required for model checking not only behavioural but also structural properties. The descriptive power of spatial logics often coincides with structural congruence, i.e., two processes satisfy the same formulas in a spatial logic if and only if they are structurally congruent [San01, Hir04, CL04]. Our *full abstraction* result in Proposition 3.4.4 shows that the structural semantics is as precise as structural congruence, i.e., two processes are structurally congruent if and only if they are mapped to the same Petri net. Combining both equivalences shows that two processes satisfy the same spatial logic formulas if and only if they are mapped to the same Petri net. Therefore we claim that our semantics is appropriate for model checking spatial logics.

The chapter also contributes to the quality criterion we called *analysability*. A semantics useful for verification should produce nets where required properties can be inferred efficiently. In Lemma 3.3.11 we observe that closed processes are mapped to *communication-free* Petri nets. The graph structure of these nets allows one to conclude about their behaviour [Esp97b].

An analysable semantics also needs an efficient translation algorithm. We comment on the implementation of the semantics by Tim Strazny [Str07]. While the definition of the semantics is declarative, the implementation uses a fixed point algorithm on the set of Petri nets.

To sum up, our contributions are as follows.

- We investigate the *restricted form* of processes, a new normal form under structural congruence. It characterises structural congruence with a strong equivalence relation called *restricted equivalence*.

- Based on the restricted form, we define the new semantical mapping of π-Calculus into Petri nets. It reflects the connection structure of processes. We prove full retrievability and full abstraction, and observe that closed processes are mapped to communication-free Petri nets.

- We explain the implementation of the semantics by Tim Strazny [Str07] in the tool PETRUCHIO [SM08].

The structure of the chapter is as follows. To begin with, we recall the interpretation of π-Calculus processes as graphs along the lines of [Mil99, SW01] and informally explain the idea of our semantics. We give it a rigorous formal definition in Section 3.3 based on the normal form for processes in Section 3.2. We prove full retrievability and full abstraction in Section 3.4, before we comment on the implementation in Section 3.5. Section 3.6, reviewing related work and pointing out points for future research concludes the chapter.

3.1 Idea of the Structural Semantics

The *graph interpretation of a π-Calculus process P* is a hypergraph $\mathcal{G}[\![P]\!]$, which makes the use of active restricted names in sequential processes explicit. Technically, a hypergraph is a graph where several vertices may be connected with one so-called hyperedge. We often refer to hypergraphs as graphs and to hyperedges as edges. The graph interpretation of a process is obtained as follows. We draw a vertex labelled by Q for every sequential process $Q = M^{\neq \mathbf{0}}$ or $Q = K\lfloor \tilde{a} \rfloor$ in $\mathcal{S}(P)$ and add a hyperedge labelled by a for every active restricted name νa. An arc is inserted between a vertex Q and an edge a if the name is free in the process, $a \in fn(Q)$. Due to name passing, process creation, and process destruction, this graph structure changes during system execution. As the graph interpretation only serves as intuition for the structural semantics, we illustrate it on an example but defer its definition until Chapter 7.

Example 3.1.1 (Graph Interpretation of a Process)
Consider the process

$$P = \nu a.(\overline{a}\langle a \rangle.\nu b.b(x) + \overline{c}\langle c \rangle \mid c(x).K\lfloor a \rfloor \mid \nu d.K\lfloor d \rfloor).$$

It contains the three sequential processes $\overline{a}\langle a \rangle.\nu b.b(x) + \overline{c}\langle c \rangle$, $c(x).K\lfloor a \rfloor$, and $K\lfloor d \rfloor$. The restricted names νa and νd are active while νb is covered by the prefix $\overline{a}\langle a \rangle$. This explains the vertices and hyperedges in the hypergraph $\mathcal{G}[\![P]\!]$ in Figure 3.1. The name a is free in $\overline{a}\langle a \rangle.\nu b.b(x) + \overline{c}\langle c \rangle$ and in $c(x).K\lfloor a \rfloor$, thus the corresponding vertices are connected with the hyperedge a. Similarly, the vertex for $K\lfloor d \rfloor$ is connected with the edge d. Note that the choice composition $\overline{a}\langle a \rangle.\nu b.b(x) + \overline{c}\langle c \rangle$ is represented by one vertex which is connected with a, although the alternative $\overline{c}\langle c \rangle$ does not contain a as a free name. The reason is that only the free names in a sequential process determine the connections of a vertex, not the operators. Furthermore, $\overline{a}\langle a \rangle.\nu b.b(x) + \overline{c}\langle c \rangle$ and $c(x).K\lfloor a \rfloor$ can communicate on channel c but there is no hyperedge c as the name is free in the process. ♦

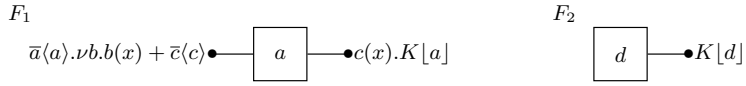


Figure 3.1: The graph interpretation $\mathcal{G}[\![P]\!]$ of process P in Example 3.1.1.

The example shows that several unconnected graphs, F_1 and F_2 in the example, represent one process. This means, DRS are modelled by independent parts communicating over public channels only.

The idea of our semantics is to represent each such graph by a place in a Petri net. We then obtain the overall process by putting tokens on the places, one for each occurrence of the graph in the current process. For the process in the previous example, we compute the Petri net depicted in Figure 3.2. It contains F_1 and F_2 as places and both are marked by a token since they are present in the initial process. The reaction

$$\nu a.(\overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor) \rightarrow \nu a.K\lfloor a\rfloor,$$

which transforms F_1 into the graph F_2 (up to α-conversion), is imitated by transition t in the net. The remainder of the net depends on the definition of K.

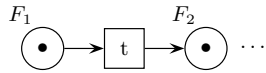


Figure 3.2:
Initial part of the structural semantics $\mathcal{N}[\![P]\!]$ of process P in Example 3.1.1. The remainder of the net is determined by the behaviour of F_2.

Technically, we do not work at graph level but show that every process can be assumed to be in *restricted form*, where so called *fragments* represent the unconnected graphs F_1 and F_2 in the above example. Normal forms similar to ours are defined in [EG99, EG04b], but with different aims. We discuss the relationship to the work of Engelfriet and Gelsema in Section 3.6.

3.2 Restricted Form

With the restricted form, we capture the notion of unconnected graphs introduced in the previous section. It serves us for the definition of our semantics and also permits the definition of the characteristic functions depth and breadth in Chapter 7. The idea of the restricted form is to minimise the scopes of active

restricted names. Processes congruent with $\mathbf{0}$ are removed. This results in a process, where the topmost parallel components are the unconnected graphs. We call them *fragments*. The decomposition function that we define in Section 3.3 then counts how often a fragment occurs in a process in restricted form and serves as marking in the the structural semantics.

Example 3.2.1 (Restricted Form)
Consider the process in Example 3.1.1:

$$\begin{aligned} & \nu a.\ \ (\ \ \overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor \mid \nu d.K\lfloor d\rfloor\ \ ) \\ \equiv\ & \nu a.\ \ (\ \ \overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor\ \ )\ \ \mid \nu d.K\lfloor d\rfloor. \end{aligned}$$

The latter process is in restricted form as the scopes of νa and νd are minimal. Fragment $\nu a.(\overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor)$ corresponds to the graph to the left in Figure 3.1, fragment $\nu d.K\lfloor d\rfloor$ is the graph depicted to the right. ♦

Formally, sequential processes, i.e., non-empty choices $M^{\neq\mathbf{0}}$ and calls to process identifiers $K\lfloor\tilde{a}\rfloor$, are fragments. On the graph level they are nodes. Restricting a name in a parallel composition of fragments, $\nu a.(F_1 \mid \ldots \mid F_n)$, is possible only if the name is free in all fragments, $a \in fn(F_i)$ for all i. This ensures the graphs of all fragments F_i are connected with the hyperedge for the restricted name νa, and so the graph of $\nu a.(F_1 \mid \ldots \mid F_n)$ is connected. A *process P^{rf} in restricted form* is a parallel composition of fragments, $P^{rf} = \Pi_{i\in I} F_i$. Since different fragments do not share restricted names, P^{rf} is represented by the graphs of the fragments F_i, which are not connected.

Definition 3.2.2 (Fragments and Restricted Form)
Fragments, typically denoted by F, G, H, are defined inductively by

$$F ::= M^{\neq\mathbf{0}} \ \mid\ K\lfloor\tilde{a}\rfloor \ \mid\ \nu a.(F_1 \mid \ldots \mid F_n),$$

where $a \in fn(F_i)$ for all $1 \le i \le n$. The *set of all fragments* is $\mathcal{P}_{\mathcal{F}}$. A process $P^{rf} = \Pi_{i\in I} F_i$ is *in restricted form*. The *set of all processes in restricted form* is $\mathcal{P}_{rf}$ with $\mathcal{P}_{\mathcal{F}} \subseteq \mathcal{P}_{rf}$. ♦

Convention 3.2.3
The fragments that are sequential processes, i.e., $F = K\lfloor\tilde{a}\rfloor$ or $F = M^{\neq\mathbf{0}}$, are called *elementary*. To indicate that a fragment F is assumed to be elementary, we denote it by F^e. ♦

Note that the index set I may be empty in a process in restricted form $P^{rf} = \Pi_{i\in I} F_i$. By Convention 2.1.2, this means $P^{rf} = \mathbf{0}$. So the stop process is in restricted form, $\mathbf{0} \in \mathcal{P}_{rf}$. We shall often need the set of fragments in a process in restricted form, it is computed by the function fg.

$$rf(M^{=\mathbf{0}}) := \mathbf{0} \qquad rf(M^{\neq\mathbf{0}}) = M^{\neq\mathbf{0}} \qquad rf(K\lfloor\tilde{a}\rfloor) := K\lfloor\tilde{a}\rfloor$$

$$rf(P \mid Q) := \begin{cases} \mathbf{0}, & \text{if } rf(P) = \mathbf{0} = rf(Q) \\ rf(P), & \text{if } rf(P) \neq \mathbf{0} = rf(Q) \\ rf(Q), & \text{if } rf(P) = \mathbf{0} \neq rf(Q) \\ rf(P) \mid rf(Q), & \text{if } rf(P) \neq \mathbf{0} \neq rf(Q) \end{cases}$$

$$rf(\nu a.P) := \begin{cases} rf(P), & \text{if } a \notin fn(P) \\ \nu a.rf(P), & \text{if } a \in fn(P) \text{ and (1)} \\ \nu a.\,(\Pi_{i\in I_a} F_i) \mid \Pi_{i\in I\setminus I_a} F_i, & \text{if } a \in fn(P) \text{ and (2)} \end{cases}$$

Table 3.1:
Definition of function rf. Let $rf(P) = \Pi_{i\in I\neq\emptyset} F_i$, condition (1) demands that $I_a = I$, (2) states that $I_a \neq I$.

Definition 3.2.4 ($fg : \mathcal{P}_{rf} \to \mathbb{P}(\mathcal{P}_{\mathcal{F}})$)
Let $P^{rf} \in \mathcal{P}_{rf}$ with $P^{rf} = \Pi_{i\in I} F_i$. The function $fg : \mathcal{P}_{rf} \to \mathbb{P}(\mathcal{P}_{\mathcal{F}})$, which returns the *set of fragments in* P^{rf}, is defined by $fg\left(P^{rf}\right) := \bigcup_{i\in I}\{F_i\}$. ♦

For a process $\Pi_{i\in I} F_i$, we often refer (1) to the fragments F_i that contain some name a and (2) to those that are structurally congruent with a given fragment F. To determine these fragments, we define subsets I_a and I_F of the index set I.

Definition 3.2.5 (I_a, I_F)
Consider the process $\Pi_{i\in I} F_i$ in $\mathcal{P}_{rf}$. For every name $a \in \mathcal{N}$, we define the index set $I_a \subseteq I$ by $i \in I_a$ if and only if $a \in fn(F_i)$. For every fragment $F \in \mathcal{P}_{\mathcal{F}}$, we define $I_F \subseteq I$ by $i \in I_F$ if and only if $F \equiv F_i$. ♦

To transform a process into a process in restricted form via structural congruence, we employ the recursive function $rf : \mathcal{P} \to \mathcal{P}_{rf}$. It uses the rule for scope extrusion to shrink the scopes of restrictions and removes parallel compositions of stop processes $\mathbf{0}$.

Definition 3.2.6 ($rf : \mathcal{P} \to \mathcal{P}_{rf}$)
The function rf in Table 3.1 computes for any process $P \in \mathcal{P}$ a process $rf(P)$ in restricted form, i.e., $rf(P) \in \mathcal{P}_{rf}$. We call $rf(P)$ the *restricted form of* P. ♦

The following lemma states that $rf(P)$ is in fact in restricted form and structurally congruent with P. Furthermore, the function does not change processes

in restricted form, i.e., $rf(P^{rf}) = P^{rf}$.

Lemma 3.2.7 (Properties of rf)
For a process $P \in \mathcal{P}$ we have $rf(P) \in \mathcal{P}_{rf}$, $rf(P) \equiv P$, $\mathcal{S}(rf(P)) = \mathcal{S}(P)$, and $rf(P\sigma) = rf(P)\sigma$. For $P^{rf} \in \mathcal{P}_{rf}$ even $rf(P^{rf}) = P^{rf}$ holds.

Proof

Properties of $rf(P)$ By an induction on the structure of P we verify that $rf(P)$ is in restricted form, structurally congruent with P, that the sequential processes are preserved, and that the function is compatible with the application of substitutions.

Base Cases Like for the standard form in the proof of Lemma 2.1.28.

Induction Step Assume $rf(P) \in \mathcal{P}_{rf}$ with $rf(P) \equiv P$, $\mathcal{S}(rf(P)) = \mathcal{S}(P)$, and $rf(P\sigma) = rf(P)\sigma$ and similar for Q.

Case $P \mid Q$ We assume that $rf(P) \neq \mathbf{0} \neq rf(Q)$, the remaining cases are similar. That $rf(P) \in \mathcal{P}_{rf}$ means $rf(P) = \Pi_{i \in I \neq \emptyset} F_i$. Similarly, we get $rf(Q) = \Pi_{j \in J \neq \emptyset} G_j$. To show $rf(P \mid Q) \in \mathcal{P}_{rf}$ we observe that

$$rf(P \mid Q) = rf(P) \mid rf(Q) = \Pi_{i \in I \neq \emptyset} F_i \mid \Pi_{j \in J \neq \emptyset} G_j \in \mathcal{P}_{rf},$$

where the first equation is the definition of rf and the second the assumption on the form of $rf(P)$ and $rf(Q)$. Structural congruence, i.e., $P \mid Q \equiv rf(P \mid Q)$, follows from the hypothesis and the definition of rf:

$$P \mid Q \equiv rf(P) \mid rf(Q) = rf(P \mid Q).$$

That the sequential processes are preserved holds with the definitions of rf and $\mathcal{S}$, as well as the hypothesis:

$$\mathcal{S}(rf(P \mid Q)) = \mathcal{S}(rf(P)) \cup \mathcal{S}(rf(Q)) = \mathcal{S}(P) \cup \mathcal{S}(Q) = \mathcal{S}(P \mid Q).$$

For the application of substitutions we get

$$\begin{aligned} & rf(P \mid Q)\sigma \\ (\text{ Def. } rf\text{, applic. } \sigma\ ) \quad &= \quad rf(P)\sigma \mid rf(Q)\sigma \\ (\text{ Hypothesis }) \quad &= \quad rf(P\sigma) \mid rf(Q\sigma) \\ (\ rf(P) \neq \mathbf{0} \text{ implies } rf(P\sigma) \neq \mathbf{0}\ ) \quad &= \quad rf(P\sigma \mid Q\sigma) \\ (\text{ Applic. } \sigma\ ) \quad &= \quad rf((P \mid Q)\sigma). \end{aligned}$$

Case $\nu a.P$ We distinguish between $a \in fn(P)$ and $a \notin fn(P)$. While the latter case is trivial, the former case requires another case distinction. Let $rf(P) =$

$\Pi_{i\in I\neq\emptyset}F_i$ so that $I = I_a$. This means, $a \in fn(F_i)$ for all $i \in I$. Since by the hypothesis all F_i are fragments, $\nu a.rf(P)$ is a fragment and thus in $\mathcal{P}_{rf}$. Structural congruence immediately follows from the definition of $rf(\nu a.P)$ and the hypothesis that $rf(P) \equiv P$:

$$rf(\nu a.P) = \nu a.rf(P) \equiv \nu a.P.$$

We need to show that the sets of sequential processes coincide. Again, this follows from the definition of rf, $\mathcal{S}$, and the hypothesis:

$$\mathcal{S}(rf(\nu a.P)) = \mathcal{S}(\nu a.rf(P)) = \mathcal{S}(rf(P)) = \mathcal{S}(P) = \mathcal{S}(\nu a.P).$$

The application of substitutions is similar. Note that by Convention 2.1.13 the name a is neither in the domain nor codomain of σ:

$$rf(\nu a.P)\sigma = (\nu a.rf(P))\sigma = \nu a.(rf(P)\sigma) = \nu a.rf(P\sigma) = rf((\nu a.P)\sigma).$$

Finally, we consider $rf(P) = \Pi_{i\in I\neq\emptyset}F_i$ so that $I \neq I_a$. Since we assume $a \in fn(P)$ we have $a \in fn(rf(P))$ with Lemma 2.1.19. Hence, there is a fragment F_i with $a \in fn(F_i)$ and thus $I_a \neq \emptyset$. Like in the case above, we observe that $\nu a.(\Pi_{i\in I_a}F_i)$ is a fragment. Moreover, the remaining F_i with $i \in I \setminus I_a$ are fragments. So $rf(\nu a.P) = \nu a.(\Pi_{i\in I_a}F_i) \mid \Pi_{i\in I\setminus I_a}F_i$ is a parallel composition of fragments, i.e., $rf(\nu a.P) \in \mathcal{P}_{rf}$. To show $rf(\nu a.P) \equiv \nu a.P$, we argue

$$\begin{aligned}
& & & rf(\nu a.P) \\
& (\text{ Def. } rf \text{ }) & = & \nu a.(\Pi_{i\in I_a}F_i) \mid \Pi_{i\in I\setminus I_a}F_i \\
& (\text{ Scope extr., } a \notin fn(F_i) \text{ with } i \in I \setminus I_a \text{ }) & \equiv & \nu a.(\Pi_{i\in I_a}F_i \mid \Pi_{i\in I\setminus I_a}F_i) \\
& (\text{ Commut. and assoc. } \mid, I_a \subseteq I \text{ }) & \equiv & \nu a.(\Pi_{i\in I}F_i) \\
& (\text{ } \Pi_{i\in I}F_i = rf(P) \text{ }) & = & \nu a.rf(P) \\
& (\text{ Hypothesis }) & \equiv & \nu a.P.
\end{aligned}$$

To show that the sequential processes are preserved, we compute

$$\begin{aligned}
& & & \mathcal{S}(rf(\nu a.P)) \\
& (\text{ Def. } rf \text{ }) & = & \mathcal{S}(\nu a.(\Pi_{i\in I_a}F_i) \mid \Pi_{i\in I\setminus I_a}F_i) \\
& (\text{ Def. } \mathcal{S} \text{ }) & = & \textstyle\bigcup_{i\in I_a}\mathcal{S}(F_i) \cup \bigcup_{i\in I\setminus I_a}\mathcal{S}(F_i) \\
& (\text{ Commut. and assoc. } \cup, I_a \subseteq I \text{ }) & \equiv & \textstyle\bigcup_{i\in I}\mathcal{S}(F_i) \\
& (\text{ Def. } \mathcal{S} \text{ }) & = & \mathcal{S}(\Pi_{i\in I}F_i) \\
& (\text{ } \Pi_{i\in I}F_i = rf(P) \text{ }) & = & \mathcal{S}(rf(P)) \\
& (\text{ Hypothesis }) & = & \mathcal{S}(P) \\
& (\text{ Def. } \mathcal{S} \text{ }) & \equiv & \mathcal{S}(\nu a.P).
\end{aligned}$$

We finally need to consider the application of substitutions:

$$\begin{aligned} & rf((\nu a.P)\sigma) \\ (\text{ Applic } \sigma) \quad &= \quad rf(\nu a.(P\sigma)) \\ (\text{ Explained below }) \quad &= \quad \nu a.(\Pi_{i\in I_a}(F_i\sigma)) \mid \Pi_{i\in I\setminus I_a}(F_i\sigma) \\ (\text{ Applic. } \sigma) \quad &= \quad (\nu a.(\Pi_{i\in I_a}F_i) \mid \Pi_{i\in I\setminus I_a}F_i)\sigma \\ (\text{ Def. } rf\text{, comment below }) \quad &= \quad rf(\nu a.P)\sigma. \end{aligned}$$

We have $rf(P\sigma) = rf(P)\sigma = \Pi_{i\in I}(F_i\sigma)$ by the hypothesis and the definition of substitution application. The equation then holds with the definition of rf. In the last equation, we exploit that $a \in fn(F_i)$ iff $a \in fn(F_i\sigma)$, which means that the index set I_a of fragments having a as free name is equal for $rf(P\sigma)$ and $rf(P)$.

Identity on P^{rf} We need to show that rf does not change processes P^{rf} in restricted form. It is sufficient to show that rf is the identity on fragments. For a process $P^{rf} = \Pi_{i\in I}F_i$ this implies

$$rf(P^{rf}) = rf(\Pi_{i\in I}F_i) = \Pi_{i\in I}rf(F_i) = \Pi_{i\in I}F_i = P^{rf}.$$

The base case where fragments are sequential processes is trivial. Assume the equality $rf(F_i) = F_i$ holds for $1 \leq i \leq n$ and consider $F = \nu a.(\Pi_{i=1}^n F_i)$. By definition, $rf(F)$ computes the restricted form of $\Pi_{i=1}^n F_i$. With the definition of rf and the hypothesis, this is $rf(\Pi_{i=1}^n F_i) = \Pi_{i=1}^n rf(F_i) = \Pi_{i=1}^n F_i$. We then compute the index set I_a. Since $\nu a.(\Pi_{i=1}^n F_i)$ is a fragment, $a \in fn(F_i)$ for all i. Thus, $I_a = \{1, \ldots, n\}$ and $rf(F) = \nu a.(\Pi_{i=1}^n F_i) = F$. ■

The restricted form is only invariant under structural congruence up to reordering and rewriting of fragments, i.e., $P \equiv Q$ does not imply $rf(P) = rf(Q)$ but it implies $rf(P) \equiv_{rf} rf(Q)$, where we define the relation $\equiv_{rf}$ as follows.

Definition 3.2.8 (Restricted Equivalence)
The *restricted equivalence relation* $\equiv_{rf} \subseteq \mathcal{P}_{rf} \times \mathcal{P}_{rf}$ is the smallest equivalence on processes in restricted form that satisfies commutativity and associativity of parallel compositions,

$$P_1^{rf} \mid P_2^{rf} \equiv_{rf} P_2^{rf} \mid P_1^{rf} \qquad P_1^{rf} \mid (P_2^{rf} \mid P_3^{rf}) \equiv_{rf} (P_1^{rf} \mid P_2^{rf}) \mid P_3^{rf},$$

and permits replacing fragments by structurally congruent ones,

$$F \mid P^{rf} \equiv_{rf} G \mid P^{rf},$$

where $F \equiv G$ and P^{rf} is optional. ♦

Since processes in restricted form do not contain parallel compositions of $\mathbf{0}$, the processes P^{rf} in each of the axioms above are different from $\mathbf{0}$. For example, $K\lfloor\tilde{a}\rfloor \mid \mathbf{0} \not\equiv_{rf} \mathbf{0} \mid K\lfloor\tilde{a}\rfloor$ as both processes are not in restricted form, i.e., not in $\mathcal{P}_{rf}$. By reflexivity, $\mathbf{0}$ is only equivalent to itself.

We illustrate the indicated relationship between $P \equiv Q$ and $rf(P)$ and $rf(Q)$ on an example.

Example 3.2.9 (Invariance of rf under $\equiv$ up to $\equiv_{rf}$)
Consider process P in Example 3.1.1 and a structurally congruent process Q:

$$\begin{aligned} P &= \nu a.(\overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor \mid \nu d.K\lfloor d\rfloor) \\ &\equiv \nu a.(c(x).K\lfloor a\rfloor \mid \overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid \nu d.K\lfloor d\rfloor) = Q. \end{aligned}$$

The restricted form of P is the process in Example 3.2.1. We compare it with the restricted form of Q:

$$\begin{aligned} rf(P) &= \nu a.(\overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor) \mid \nu d.K\lfloor d\rfloor \\ &\equiv_{rf} \nu a.(c(x).K\lfloor a\rfloor \mid \overline{a}\langle a\rangle.\nu b.b(x) + \overline{c}\langle c\rangle) \mid \nu d.K\lfloor d\rfloor = rf(Q). \end{aligned}$$

Since $\nu a.(\overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle \mid c(x).K\lfloor a\rfloor)$ and $\nu a.(c(x).K\lfloor a\rfloor \mid \overline{a}\langle a\rangle.\nu b.b(x)+\overline{c}\langle c\rangle)$ are not syntactically equal but structurally congruent fragments, $rf(P) \neq rf(Q)$ but $rf(P) \equiv_{rf} rf(Q)$ holds. ♦

Proposition 3.2.10 states the discussed invariance of the restricted form up to $\equiv_{rf}$, $P \equiv Q$ implies $rf(P) \equiv_{rf} rf(Q)$. In fact, the restricted equivalence relation characterises structural congruence, i.e., also $rf(P) \equiv_{rf} rf(Q)$ implies $P \equiv Q$.

Proposition 3.2.10 (Characterisation of $\equiv$ with $\equiv_{rf}$)
For $P, Q \in \mathcal{P}$ the following holds: $P \equiv Q$ if and only if $rf(P) \equiv_{rf} rf(Q)$.

Proof
$\Leftarrow$ To show the implication from right to left we observe that all rules making up the equivalence $\equiv_{rf}$ also hold for structural congruence. Thus, $rf(P) \equiv_{rf} rf(Q)$ implies $rf(P) \equiv rf(Q)$. Combined with $P \equiv rf(P)$ from Lemma 3.2.7, we get $P \equiv Q$ with the transitivity of structural congruence.

$\Rightarrow$ The proof of the reverse direction is by induction on the derivations of structural congruence. Since it contains several case distinctions (for $I_a = \emptyset$ or $I_a = I$), we remark that in all cases that are not considered here the proofs are either trivial or similar to the presented proofs.

Base Cases We prove that the restricted equivalence holds for the axioms.

Case α-conversion For input prefixes, the proof is trivial as α-conversion yields structurally congruent fragments. Consider $\nu a.P \equiv \nu b.(P\{b/a\})$, where b is a fresh name, i.e., $b \notin (fn(P) \cup bn(P))$. Let $a \in fn(P)$ and $rf(P) = \Pi_{i \in I \neq \emptyset} F_i$ with $I_a \neq I$.

$$\begin{array}{rcl} & & rf(P) \\ (\text{ Def. } rf \text{ }) & = & \nu a.(\Pi_{i \in I_a} F_i) \mid \Pi_{i \in I \setminus I_a} F_i \\ & \equiv_{rf} & \nu b.(\Pi_{i \in I_a} F_i\{b/a\}) \mid \Pi_{i \in I \setminus I_a} F_i \end{array}$$

With α-conversion, we have $\nu a.(\Pi_{i \in I_a} F_i) \equiv \nu b.(\Pi_{i \in I_a} F_i\{b/a\})$. Since $a \in fn(F_i)$, we have $b \in fn(F_i\{b/a\})$ for all $i \in I_a$. So $\nu b.(\Pi_{i \in I_a} F_i\{b/a\})$ is a fragment which we replace for the fragment $\nu a.(\Pi_{i \in I_a} F_i)$ with the last axiom in the definition of $\equiv_{rf}$. We continue the equation with $F_i\{b/a\} = F_i$ if $a \notin fn(F_i)$, which means $i \in I \setminus I_a$:

$$\begin{array}{rcl} (\ F_i\{b/a\} = F_i \text{ since } a \notin fn(F_i)\ ) & = & \nu b.(\Pi_{i \in I_a}(F_i\{b/a\})) \mid \Pi_{i \in I \setminus I_a}(F_i\{b/a\}) \\ (\ a \in fn(F_i) \text{ iff } b \in fn(F_i\{b/a\})\ ) & = & \nu b.(\Pi_{i \in I_b}(F_i\{b/a\})) \mid \Pi_{i \in I \setminus I_b}(F_i\{b/a\}) \\ & = & rf(\nu b.(P\{b/a\})). \end{array}$$

For the last equation, we argue $rf(P\{b/a\}) = rf(P)\{b/a\} = (\Pi_{i \in I} F_i)\{b/a\} = \Pi_{i \in I}(F_i\{b/a\})$ with Lemma 3.2.7 and the definition of substitution application.

Case $+$, $\mid$, and $\nu a.\mathbf{0} \equiv \mathbf{0}$ The proofs are straightforward.

Case $\nu a.\nu b.P \equiv \nu b.\nu a.P$ Let $rf(P) = \Pi_{i \in I \neq \emptyset} F_i$ with $I_a \neq \emptyset \neq I_b$. Let $I \neq I_a \cup I_b$, we distinguish between $I_a \cap I_b = \emptyset$ and $I_a \cap I_b \neq \emptyset$. In the first case, we get

$$\begin{array}{rcl} rf(\nu a.\nu b.P) & = & \nu a.\,(\Pi_{i \in I_a} F_i) \mid \nu b.\,(\Pi_{i \in I_b} F_i) \mid \Pi_{i \in I \setminus (I_a \cup I_b)} F_i \\ & \equiv_{rf} & \nu b.\,(\Pi_{i \in I_b} F_i) \mid \nu a.\,(\Pi_{i \in I_a} F_i) \mid \Pi_{i \in I \setminus (I_a \cup I_b)} F_i = rf(\nu b.\nu a.P). \end{array}$$

The equations holds because $I_a \cap I_b = \emptyset$. Consider $I_a \cap I_b \neq \emptyset$. By definition, $rf(\nu b.P) = \nu b.(\Pi_{i \in I_b} F_i) \mid \Pi_{i \in I \setminus I_b} F_i$. Thus, the restricted form of $\nu a.\nu b.P$ is

$$\begin{array}{rcl} rf(\nu a.\nu b.P) & = & \nu a.\,\big(\nu b.\,(\Pi_{i \in I_b} F_i) \mid \Pi_{i \in I_a \setminus I_b} F_i\big) \mid \Pi_{i \in I \setminus (I_a \cup I_b)} F_i \\ & \equiv_{rf} & \nu b.\,\big(\nu a.\,(\Pi_{i \in I_a} F_i) \mid \Pi_{i \in I_b \setminus I_a} F_i\big) \mid \Pi_{i \in I \setminus (I_a \cup I_b)} F_i \\ & = & rf(\nu b.\nu a.P). \end{array}$$

For $\equiv_{rf}$ we argue as follows:

$$\begin{array}{rcl} \nu a.\,\big(\nu b.\,(\Pi_{i \in I_b} F_i) \mid \Pi_{i \in I_a \setminus I_b} F_i\big) & \equiv & \nu a.\nu b.\,(\Pi_{i \in I_a \cup I_b} F_i) \\ & \equiv & \nu b.\,\big(\nu a.\,(\Pi_{i \in I_a} F_i) \mid \Pi_{i \in I_b \setminus I_a} F_i\big)\,. \end{array}$$

Since the first and last element are fragments, restricted equivalence holds.

Case Scope Extrusion We prove $rf(\nu a.(P \mid Q)) \equiv_{rf} rf(P \mid \nu a.Q)$, if $a \notin fn(P)$. Let $rf(P) = \Pi_{i \in I \neq \emptyset} F_i$ and $rf(Q) = \Pi_{j \in J \neq \emptyset} G_j$ with $\emptyset \neq J_a \neq J$.

$$\begin{aligned} & & & rf(\nu a.(P \mid Q)) \\ & (\ a \notin fn(P) \text{ implies } a \notin fn(F_i)\ ) & = & \nu a.(\Pi_{j \in J_a} G_j) \mid \Pi_{i \in I} F_i \mid \Pi_{j \in J \setminus J_a} G_j \\ & (\ \text{Commut. and assoc. } \mid \text{ in } \equiv_{rf}\ ) & \equiv_{rf} & \Pi_{i \in I} F_i \mid \nu a.(\Pi_{j \in J_a} G_j) \mid \Pi_{j \in J \setminus J_a} G_j \\ & (\ \text{Def. } rf\ ) & = & rf(P \mid \nu a.Q). \end{aligned}$$

Induction Step Assume that $P \equiv Q$ implies $rf(P) \equiv_{rf} rf(Q)$ and similar for $Q \equiv R$. It is immediate to see that $rf(Q) \equiv_{rf} rf(P)$ and $rf(P) \equiv_{rf} rf(R)$ since $\equiv_{rf}$ is an equivalence. Let $rf(P) = \Pi_{i \in I} F_i \equiv_{rf} \Pi_{j \in J} G_j = rf(Q)$. Note that $\equiv_{rf}$ implies $|I| = |J|$. Without loss of generality, we assume $I = J = \{1, \ldots, n\} \neq \emptyset$ and that both sets are ordered so that $F_i \equiv G_i$. If both restricted forms are $\mathbf{0}$, i.e., $I = J = \emptyset$, the following proofs become trivial.

Case $\pi.P + M \equiv \pi.Q + M$ By definition of rf, we get:

$$rf(\pi.P + M) = \pi.P + M \equiv \pi.Q + M = rf(\pi.Q + M).$$

Since both are fragments, $rf(\pi.P + M) \equiv_{rf} rf(\pi.Q + M)$ holds.

Case $P \mid R \equiv Q \mid R$ Let $rf(R) \neq \mathbf{0}$, the first and last of the following equations hold by definition of rf:

$$rf(P \mid R) = rf(P) \mid rf(R) \equiv_{rf} rf(Q) \mid rf(R) = rf(Q \mid R).$$

To show $\equiv_{rf}$, recall that we can assume I and J ordered so that $F_i \equiv G_i$:

$$\begin{aligned} & & & rf(P) \mid rf(R) \\ & (\ rf(P) = \Pi_{i=1}^{n} F_i\ ) & = & F_1 \mid \Pi_{i=2}^{n} F_i \mid rf(R) \\ & (\ F_i \equiv G_i, \text{ def. } \equiv_{rf}\ ) & \equiv_{rf} & G_1 \mid \Pi_{i=2}^{n} F_i \mid rf(R) \\ & (\ \text{Commut. and assoc. } \mid \text{ in } \equiv_{rf}\ ) & \equiv_{rf} & F_2 \mid G_1 \mid \Pi_{i=3}^{n} F_i \mid rf(R) \\ & (\ \text{Replace remaining } F_i\ ) & \equiv_{rf} & \Pi_{i=1}^{n} G_i \mid rf(R) \\ & (\ \Pi_{i=1}^{n} G_i = rf(Q)\ ) & = & rf(Q) \mid rf(R). \end{aligned}$$

Case $\nu a.P \equiv \nu a.Q$ We consider the case that $\emptyset \neq I_a \neq I = J$. Since free names are preserved by structural congruence, we have $a \in fn(F_i)$ if and only if $a \in fn(G_i)$. We compute:

$$rf(\nu a.P) = \nu a.(\Pi_{i \in I_a} F_i) \mid \Pi_{i \in I \setminus I_a} F_i \ \equiv_{rf}\ \nu a.(\Pi_{i \in I_a} G_i) \mid \Pi_{i \in I \setminus I_a} G_i = rf(\nu a.Q).$$

Since $F_i \equiv G_i$ for all $i \in I_a$, we get $\nu a.(\Pi_{i \in I_a} F_i) \equiv \nu a.(\Pi_{i \in I_a} G_i)$. Both are fragments, so $\equiv_{rf}$ holds. The remaining F_i are replaced by structurally congruent G_i like in the case of parallel composition. ■

We will often combine Lemma 3.2.7 and Proposition 3.2.10, to strengthen the relation $rf(P) \equiv rf(Q)$ to $rf(P) \equiv_{rf} rf(Q)$.

Corollary 3.2.11 ($\equiv$ and $\equiv_{rf}$ coincide on $\mathcal{P}_{rf}$)
For $P^{rf}, Q^{rf} \in \mathcal{P}_{rf}$ we have $P^{rf} \equiv Q^{rf}$ if and only if $P^{rf} \equiv_{rf} Q^{rf}$.

Proof
With Proposition 3.2.10, we have $P^{rf} \equiv Q^{rf}$ if and only if $rf(P^{rf}) \equiv_{rf} rf(Q^{rf})$. With Lemma 3.2.7, we get $rf(P^{rf}) = P^{rf}$ and similar for Q^{rf}. Hence $rf(P^{rf}) \equiv_{rf} rf(Q^{rf})$ if and only if $P^{rf} \equiv_{rf} Q^{rf}$. ■

3.3 Structural Semantics

In this section, we define a Petri net semantics for the π-Calculus, i.e., a mapping $\mathcal{N} : \mathcal{P} \to \mathcal{PN}$ that assigns to every process P a Petri net $\mathcal{N}[\![P]\!]$. Example 3.3.8, which we discuss later in this section, illustrates the translation. The places of the net are the fragments of all processes reachable from P by the reaction relation. More precisely, we deal with classes of fragments under structural congruence.

We use two disjoint sets of transitions. Transitions of the first kind are pairs $([F], [Q])$ of places $[F]$ and processes $[Q]$, with the condition that F reacts to Q. These transitions represent reactions inside fragments. The second set of transitions contains pairs $([F_1 \mid F_2], [Q])$ where $[F_1]$ and $[F_2]$ are places and $F_1 \mid F_2$ reacts to Q. These transitions represent communications between fragments using public channels.

There is an arc from place $[G]$ to transition $([F], [Q])$ if G is structurally congruent with F, i.e., $[G] = [F]$. If G is structurally congruent with F_1 and F_2, there is an arc weighted two from place $[G]$ to transition $([F_1 \mid F_2], [Q])$. In this case, fragment F_1 communicates with the structurally congruent fragment F_2 on a public channel. If G is structurally congruent with F_1 or F_2, there is an arc weighted one from place $[G]$ to transition $([F_1 \mid F_2], [Q])$. There is no arc, if G is neither structurally congruent with F_1 nor with F_2.

The number of arcs from $([F], [Q])$ to place $[G]$ (or from $([F_1 \mid F_2], [Q])$ to place $[G]$, respectively) is determined by the number of occurrences of G in the decomposition of Q. Similarly, the initial marking of the net is determined by the decomposition of the initial process P.

To formally capture the notion of a process decomposition, we define the function $dec(P^{rf})$. As was explained in Section 3.1, the structural semantics counts how often an unconnected graph occurs in the current process. Since unconnected graphs are represented by fragments $[F]$, the decomposition function $dec(P^{rf})$ counts how many fragments of class $[F]$ are present in process P^{rf}. For example,

for $P^{rf} = F \mid G \mid F'$ with $F \equiv F'$ and $F \not\equiv G$ we have $(dec(P^{rf}))([F]) = 2$, $(dec(P^{rf}))([G]) = 1$, and $(dec(P^{rf}))([H]) = 0$ with $F \not\equiv H \not\equiv G$.

Definition 3.3.1 ($dec : \mathcal{P}_{rf} \to \mathbb{N}^{\mathcal{P}_{\mathcal{F}}/\equiv}$)
Consider $P^{rf} = \Pi_{i \in I} F_i$. We assign to P^{rf} the function $dec(P^{rf}) : \mathcal{P}_{\mathcal{F}}/_{\equiv} \to \mathbb{N}$ via $(dec(P^{rf}))([F]) := |I_F|$. ♦

By definition, *dec* is compatible with parallel compositions. Furthermore, the support of $dec(P^{rf})$ equals the set of fragments in P^{rf}, factorised by structural congruence. Both properties are important in the proof of full retrievability.

Lemma 3.3.2 (Properties of *dec*)
Consider $P^{rf}, Q^{rf} \in \mathcal{P}_{rf}$. The equalities $dec(P^{rf} \mid Q^{rf}) = dec(P^{rf}) + dec(Q^{rf})$ and $supp(dec(P^{rf})) = fg\left(P^{rf}\right)/_{\equiv}$ hold.

Proof
Let $P^{rf} = \Pi_{i \in I} F_i$ and $Q^{rf} = \Pi_{j \in J} G_j$. We then have for any fragment $F \in \mathcal{P}_{\mathcal{F}}$:

$$\begin{aligned} & & (dec(P^{rf} \mid Q^{rf}))([F]) \\ (\text{ Def. } dec\text{, form of } P^{rf} \text{ and } Q^{rf}) & = & |I_F + J_F| \\ (\ I_F \cap J_F = \emptyset\ ) & = & |I_F| + |J_F| \\ (\text{ Def. } dec\text{, form of } P^{rf} \text{ and } Q^{rf}) & = & (dec(P^{rf}))([F]) + (dec(Q^{rf}))([F]). \end{aligned}$$

Hence, the two functions are equal. To show that the support and the factorised fragment set coincide, we observe

$$\begin{aligned} & & [F] \in supp(dec(P^{rf})) \\ (\text{ Def. } supp) & \Leftrightarrow & (dec(P^{rf}))([F]) > 0 \\ (\text{ Def. } (dec(P^{rf}))([F])) & \Leftrightarrow & |I_F| > 0 \\ (\text{ Def. } I_F) & \Leftrightarrow & P^{rf} = P_1^{rf} \mid G \mid P_2^{rf} \text{ with } G \equiv F \text{ and } P_i^{rf} \text{ optional} \\ (\text{ Def. } fg) & \Leftrightarrow & G \in fg\left(P^{rf}\right) \text{ with } G \equiv F \\ (\text{ Def. } -/_{\equiv}) & \Leftrightarrow & [F] = [G] \in fg\left(P^{rf}\right)/_{\equiv}. \end{aligned}$$

This proves the equality. ■

The second statement in the previous lemma implies that the support of $dec(P^{rf})$ is always finite. This ensures that the process

$$\Pi_{[H] \in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H$$

is defined.[1] Intuitively, we construct the term from a given process P^{rf} as follows. We choose a representative for each fragment and then rearrange the fragments so that the same representatives lie next to each other.

Example 3.3.3 (Elementary Equivalence)
For the process $P^{rf} = F \mid G \mid F'$ with $F \equiv F'$ and $F \not\equiv G$ we choose F as representative for $F \equiv F'$ and let G represent itself. We then rearrange $F \mid G \mid F$ to $F \mid F \mid G = \Pi^2 F \mid \Pi^1 G$. By definition of $dec(P^{rf})$, exactly $(dec(P^{rf}))([F]) = 2$ fragments F lie next to each other. The same holds for G. Hence, we have the equivalence

$$F \mid G \mid F' \equiv_{rf} \Pi^2 F \mid \Pi^1 G = \Pi^{(dec(P^{rf}))([F])} F \mid \Pi^{(dec(P^{rf}))([G])} G.$$

The proof of Lemma 3.3.4 shows that in fact all transformations preserve restricted equivalence. ♦

Lemma 3.3.4 (Elementary Equivalence)
For every process $P^{rf} \in \mathcal{P}_{rf}$ the equivalence

$$P^{rf} \equiv_{rf} \Pi_{[H] \in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H$$

holds.

Proof
Consider $P^{rf} = \Pi_{i \in I} F_i$ with $fg\left(P^{rf}\right)/_{\equiv} = \{[H_1], \ldots, [H_p]\}$ for some representatives $H_1, \ldots, H_p$. We replace every fragment F_i by its representative $H_{\phi(i)}$, i.e., we use a function $\phi : I \to \{1, \ldots, p\}$ with $F_i \equiv H_{\phi(i)}$. That this replacement is possible under restricted equivalence follows from the proof of Proposition 3.2.10. We then reorder the fragments H using associativity and commutativity of parallel composition. Since $I_H \subseteq I$ is defined by $i \in I_H$ iff $H \equiv F_i$, there are $|I_H|$ fragments F_i which are mapped to H. We compute:

$$\begin{aligned}
& & & P^{rf} \\
& (\text{ Form of } P^{rf} \text{ }) & = & \Pi_{i \in I} F_i \\
& (\text{ Replace } F_i \text{ by representative } H_{\phi(i)} \text{ }) & \equiv_{rf} & \Pi_{i \in I} H_{\phi(i)} \\
& (\text{ Ass. commut. } \mid, \text{ discussion }) & \equiv_{rf} & \Pi_{[H] \in fg\left(P^{rf}\right)/_{\equiv}} \Pi^{|I_H|} H \\
& (\text{ Def. } (dec(P^{rf}))([H]) \text{ }) & = & \Pi_{[H] \in fg\left(P^{rf}\right)/_{\equiv}} \Pi^{(dec(P^{rf}))([H])} H \\
& (\text{ } fg\left(P^{rf}\right)/_{\equiv} = supp(dec(P^{rf})) \text{ }) & = & \Pi_{[H] \in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H.
\end{aligned}$$

This concludes the proof, restricted equivalence holds. ■

[1] Technically, we also have to assume that the fragments in the support of $dec(P^{rf})$ are ordered. Since different orderings yield processes that are restricted equivalent, we omit this detail.

That *dec* is invariant under restricted equivalence, i.e., $P^{rf} \equiv_{rf} Q^{rf}$ implies $dec(P^{rf}) = dec(Q^{rf})$, ensures the structural semantics is well-defined. That it characterises restricted equivalence, $dec(P^{rf}) = dec(Q^{rf})$ also implies $P^{rf} \equiv_{rf} Q^{rf}$, is exploited in the proof of Theorem 3.4.3. We remark that the latter implication has a short and elegant proof which exploits the elementary equivalence in Lemma 3.3.4.

Lemma 3.3.5 (Characterisation of $\equiv_{rf}$ by $=$)
Consider $P^{rf}, Q^{rf} \in \mathcal{P}_{rf}$, then $P^{rf} \equiv_{rf} Q^{rf}$ if and only if $dec(P^{rf}) = dec(Q^{rf})$.

Before we establish the lemma, we state a corollary that allows us to switch freely between structural congruence of processes, restricted equivalence of the standard forms, and equality of the decomposition functions.

Corollary 3.3.6
The following statements are equivalent:

(1) $P \equiv Q$ (2) $rf(P) \equiv rf(Q)$
(3) $rf(P) \equiv_{rf} rf(Q)$ (4) $dec(rf(P)) = dec(rf(Q))$.

Proof
Equivalence of (1) and (3) is Proposition 3.2.10. That (3) and (2) are equivalent is Corollary 3.2.11. Equivalence between (3) and (4) is Lemma 3.3.5. ■

Proof (of Lemma 3.3.5)
We start with the direction from right to left:

$$\begin{aligned} & & & P^{rf} \\ & (\text{ Lemma 3.3.4 }) & \equiv_{rf} \quad & \Pi_{[H]\in supp(dec(P^{rf}))} \Pi^{(dec(P^{rf}))([H])} H \\ & (\ dec(P^{rf}) = dec(Q^{rf})\ ) & = \quad & \Pi_{[H]\in supp(dec(Q^{rf}))} \Pi^{(dec(Q^{rf}))([H])} H \\ & (\text{ Lemma 3.3.4 }) & \equiv_{rf} \quad & Q^{rf}. \end{aligned}$$

By transitivity $P^{rf} \equiv_{rf} Q^{rf}$ holds.

The direction from left to right requires an induction on the derivations of restricted equivalence. For commutativity and associativity of parallel composition, the proof is immediate with Lemma 3.3.2. We consider the rule $F \mid P^{rf} \equiv_{rf} G \mid P^{rf}$ with $F \equiv G$. Let $P^{rf} = \Pi_{i\in I} F_i$. For the class $[F]$, we get

$$(dec(F \mid P^{rf}))([F]) = 1 + |I_F| = (dec(G \mid P^{rf}))([F]).$$

For $G \not\equiv H \not\equiv F$, we have $(dec(F \mid P^{rf}))([H]) = |I_H| = (dec(G \mid P^{rf}))([H])$. Thus, $dec(F \mid P^{rf})$ and $dec(G \mid P^{rf})$ are equal. The induction step is trivial. ■

We are now prepared to define our structural semantics.

Definition 3.3.7 (Structural Semantics $\mathcal{N} : \mathcal{P} \to \mathcal{PN}$)
The *structural semantics* is a mapping $\mathcal{N} : \mathcal{P} \to \mathcal{PN}$ that yields a Petri net $\mathcal{N}[\![P]\!]$ for every process P as defined in Table 3.2. We call $\mathcal{N}[\![P]\!]$ the *structural semantics of process P*. ♦

$$
\begin{aligned}
S &:= \mathit{fg}\,(\mathit{rf}(\mathit{Reach}(P)))/_{\equiv} \\
T &:= \quad \{([F],[Q]) \in S \times \mathcal{P}/_{\equiv} \mid F \to Q\} \\
&\quad \cup \{([F_1 \mid F_2],[Q]) \in \mathcal{P}/_{\equiv} \times \mathcal{P}/_{\equiv} \mid [F_1],[F_2] \in S \text{ and } F_1 \mid F_2 \to Q\} \\
M_0 &:= \mathit{dec}(\mathit{rf}(P)).
\end{aligned}
$$

Consider place $[G] \in S$ and two transitions $([F],[Q]),([F_1 \mid F_2],[Q]) \in T$. The weight function W is defined as follows:

$$
\begin{aligned}
W([G],([F],[Q])) &:= (\mathit{dec}(F))([G]) \\
W([G],([F_1 \mid F_2],[Q])) &:= (\mathit{dec}(F_1 \mid F_2))([G]) \\
W(([F],[Q]),[G]) &:= (\mathit{dec}(\mathit{rf}(Q)))([G]) \\
W(([F_1 \mid F_2],[Q]),[G]) &:= (\mathit{dec}(\mathit{rf}(Q)))([G]).
\end{aligned}
$$

Table 3.2:
Definition of the structural semantics $\mathcal{N}[\![P]\!] = (S, T, W, M_0)$ of process P.

We briefly discuss the definition. The structural semantics has the reachable fragments of P as set of places, $S = \mathit{fg}\,(\mathit{rf}(\mathit{Reach}(P)))/_{\equiv}$. The set S is a subset of all fragment classes, $S \subseteq \mathcal{P}_{\mathcal{F}}/_{\equiv}$. Furthermore, $(\mathit{dec}(\mathit{rf}(P)))([F]) = 0$ for all fragments that are not in S. Hence, $\mathit{dec}(\mathit{rf}(P))$ is a correct initial marking of $\mathcal{N}[\![P]\!]$ by Convention 2.2.5.

The weight $W([G],([F],[Q]))$ is defined by

$$
(\mathit{dec}(F))([G]) = \begin{cases} 1, & \text{if } [F] = [G] \\ 0, & \text{otherwise.} \end{cases}
$$

This means, there is an arc weighted one from place $[G]$ to transition $([F],[Q])$ if the classes $[F]$ and $[G]$ coincide. In all other cases there is no arc. We prefer the technical definition to a case distinction because it is more convenient in the

proofs (cf. proof of Lemma 3.4.5). Similarly, $W([G],([F_1 \mid F_2],[Q]))$ is

$$(dec(F_1 \mid F_2))([G]) = \begin{cases} 2, & \text{if } [F_1] = [G] = [F_2] \\ 0, & \text{if } [F_1] \neq [G] \neq [F_2] \\ 1, & \text{otherwise.} \end{cases}$$

Since fragments are different from $\mathbf{0}$, a process $P \equiv \mathbf{0}$ is mapped to the empty net without places and transitions.

Consider a fragment F_1 that reacts to Q. This reaction is modelled by a transition $([F_1],[Q])$. But $F_1 \mid F_2$ also reacts to $Q \mid F_2$ for every fragment F_2. Thus, we additionally get a transition $([F_1 \mid F_2],[Q \mid F_2])$ for every reachable fragment $[F_2]$, cf. Figure 3.3. Since only a loop reproducing a token on place

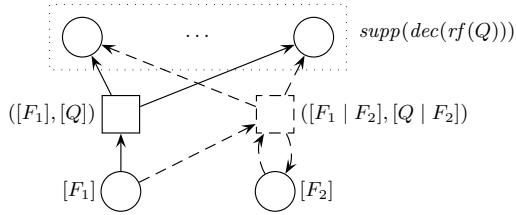


Figure 3.3:
Illustration of the transitions $([F_1],[Q])$ and $([F_1 \mid F_2],[Q \mid F_2])$. Dashed lines indicate that $([F_1 \mid F_2],[Q \mid F_2])$ (with the corresponding arcs) is not computed as it does not change the transition system.

$[F_2]$ differentiates $([F_1 \mid F_2],[Q \mid F_2])$ from $([F_1],[Q])$, the additional transitions do not change the transition system and we do not compute them. We do not exclude them by a side condition as this complicates the proof of Theorem 3.4.3.

Example 3.3.8 (Structural Semantics)
We illustrate our translation on a small example. Consider the process

$$P \quad = \quad \Pi^2 a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x)).$$

The semantics $\mathcal{N}[\![P]\!]$ is depicted in Figure 3.4. To begin with, we compute all reachable fragments. They are given by the reaction sequence

$$\begin{aligned} & \Pi^2 a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x)) \\ \rightarrow \quad & b(y).y(z).\overline{a}\langle d\rangle \mid \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x)) \\ \rightarrow \quad & \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x))) \\ \rightarrow \quad & \overline{a}\langle d\rangle \mid c(x) \mid c(x). \end{aligned}$$

All processes in this reaction sequence are in restricted form. We take the fragments as the set of places, cf. Figure 3.4. The transitions are computed as follows. Fragment F_1 reacts with a structurally congruent fragment to F_2. This yields $t_1 = ([F_1 \mid F_1], [F_2])$. Both processes, $F_1 \mid F_1$ and F_2, are in restricted form. Thus, the decompositions are $(dec(F_1 \mid F_1))([F_1]) = 2$ and $(dec(F_1 \mid F_1))([F]) = 0$ otherwise. Similarly, $(dec(F_2))([F_2]) = 1$ and $(dec(F_2))([F]) = 0$ otherwise. This explains the arc weights.

Fragment F_3 passes the restricted name h to F_2, which results in the fragment $F_4 = \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x)))$. The two processes $h(z).\overline{a}\langle d\rangle$ and $\overline{h}\langle b\rangle.(c(x) \mid c(x))$ inside F_4 share the restricted name h, so F_4 is in restricted form. Transition $t_2 = ([F_2 \mid F_3], [F_4])$ models the communication. It demonstrates how the scope of restricted names influences our Petri net semantics. A pair of processes is represented by one token on place $[F_4]$. All semantics known from the literature (cf. Section 3.6) represent the processes $h(z).\overline{a}\langle d\rangle$ and $\overline{h}\langle b\rangle.(c(x) \mid c(x))$ inside F_4 by separate places, each carrying a token.

Fragment F_4 performs an internal reaction. The two processes it consists of communicate on the restricted channel h. The fragment reacts to process $Q = \overline{a}\langle d\rangle \mid c(x) \mid c(x) = F_6 \mid F_5 \mid F_5$. By definition, we get the transition $t_3 = ([F_4], [Q])$. There is an arc weighted one from place $[F_4]$ to t_3. The process Q is in restricted form. Its decomposition is $dec(Q)$ with $(dec(Q))([F_5]) = 2$, $(dec(Q))([F_6]) = 1$, and $(dec(Q))([F]) = 0$ otherwise. The transition shows how fragments consisting of several processes break up when restricted names are forgotten.

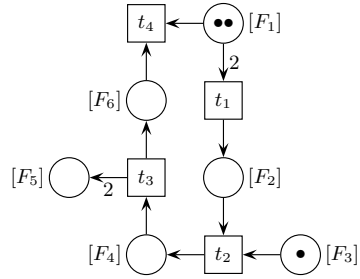
The definition of the set of transitions does not take the overall process behaviour into account. The Petri net may contain transitions that are never enabled. Transition t_4 illustrates this fact. The fragments F_1 and F_6 react to $G = d(y).y(z).\overline{a}\langle d\rangle$. This results in $t_4 = ([F_1 \mid F_6], [G])$. The transition is never executed since the reaction is not possible in the process P. Since G is no reachable fragment, $(dec(G))([F]) = 0$ for all places $[F]$, so transition t_4 has no places in its postset. Fragment G with $(dec(G))([G]) = 1$ is not considered.

The process P is in restricted form. The initial marking is given by the decomposition of P, which is defined by $(dec(P))([F_1]) = 2$, $(dec(P))([F_3]) = 1$, and $(dec(P))([F]) = 0$ otherwise. ♦

The example suggests the following rules of thumb for the structural semantics.

Remark 3.3.9 (Merging and Splitting of Fragments)
Passing of restricted names *merges* fragments. More precisely, if fragment F passes a restricted name νa to fragment G, this may result in a new fragment $\nu a.(F' \mid G')$ and we have a transition from places $[F]$ and $[G]$ to place $[\nu a.(F' \mid G')]$. Confer to transition t_2 in Example 3.3.8.

Oblivion of restricted names *splits* fragments. If fragment F forgets the restric-



$F_1 = a(x).x(y).y(z).\overline{a}\langle d\rangle + \overline{a}\langle b\rangle$ $\quad F_2 = b(y).y(z).\overline{a}\langle d\rangle$

$F_3 = \nu h.\overline{b}\langle h\rangle.\overline{h}\langle b\rangle.(c(x) \mid c(x))$ $\quad F_4 = \nu h.(h(z).\overline{a}\langle d\rangle \mid \overline{h}\langle b\rangle.(c(x) \mid c(x)))$

$F_5 = c(x)$ $\quad F_6 = \overline{a}\langle d\rangle$

Figure 3.4:
The structural semantics $\mathcal{N}[\![P]\!]$ of process P in Example 3.3.8. For the sake of readability, the fragments are named F_1 to F_6 as defined below the figure. The meaning of transitions is explained in the text.

ted name a when it evolves to F', fragment $\nu a.(F \mid G)$ reacts to $F' \mid \nu a.G$. This results in a transition with $[\nu a.(F \mid G)]$ in its preset and $[F']$ and $[\nu a.G]$ in its postset. Transition t_3 in Example 3.3.8 illustrates this behaviour. The transition also shows that splitting (and merging) of fragments can be more complicated if F' consist of several fragments or F and G both forget the name νa. ♦

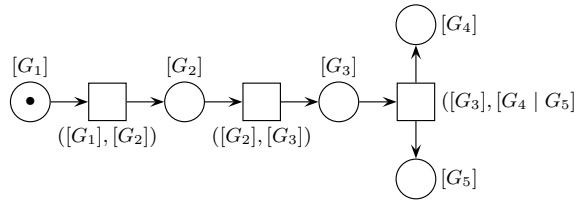
The example—in particular the computation of transition t_4—indicates that the set of places determines the size of the structural semantics. We defer the discussion of how the size of $\mathcal{N}[\![P]\!]$ is related to the size of process P until Chapter 4, where we investigate those processes that are finitely represented under the structural semantics.

Lemma 3.3.10 (Finiteness)
The Petri net $\mathcal{N}[\![P]\!]$ of a process $P \in \mathcal{P}$ is finite if and only if the set of places in $\mathcal{N}[\![P]\!]$ is finite.

Proof
We show that finiteness of the set of places implies finiteness of the set of transitions. Let the places be $[F_1], \ldots, [F_n]$. For every F and every pair F_1, F_2 there are up to structural congruence finitely many processes Q with $F \to Q$ and

$F_1 \mid F_2 \to Q$ because the reaction relation is image-finite up to structural congruence, Lemma 2.1.39. Thus, there are finitely many combinations $([F], [Q])$ and $([F_1 \mid F_2], [Q])$, respectively. The set of transitions is finite. ■

Example 3.3.8 also reveals that a communication between two fragments requires a public channel. The Petri net of a closed process without public names has transitions of the form $([F], [Q])$ only, which means it is communication-free. Figure 3.5 shows the structural semantics of a closed process. In fact, each transition has a single place in its preset and the arcs leading to the transitions are weighted by one. The arcs from transitions to places in the semantics of a closed process may be weighted arbitrarily.



$G_1 = \nu a, b, c, d.(F_1 \mid F_1 \mid F_3)$ $\quad$ $G_2 = \nu a, b, c, d.(F_2 \mid F_3)$

$G_3 = \nu a, b, c, d.F_4$ $\quad$ $G_4 = \nu c.(F_5 \mid F_5)$

$G_5 = \nu a, d.F_6$

Figure 3.5:
The structural semantics $\mathcal{N}[\![\nu a, b, c, d.P]\!]$ of the closed version of process P in Example 3.3.8. Transition $([G_3], [G_4 \mid G_5])$ shows that the number of tokens in the semantics of a closed process need not be constant or bounded.

Lemma 3.3.11
If $P \in \mathcal{P}$ is a closed process then $\mathcal{N}[\![P]\!]$ is a communication-free Petri net.

Proof
Consider the structural semantics $\mathcal{N}[\![P]\!] = (S, T, W, M_0)$ of a closed process P. Assume there is a transition of the form $([F_1 \mid F_2], Q) \in T$. From this fact we derive that there is a free name in F_1 and that F_1 is closed, a contradiction. Hence the assumption that such a transition exists has to be false.

By definition of T, we have $F_1 \mid F_2 \to Q$. Let $sf(F_1) = \nu\tilde{a}_1.P_1^{\neq\nu}$ and $sf(F_2) = \nu\tilde{a}_2.P_2^{\neq\nu}$. With Rule (Struct), $sf(F_1 \mid F_2) = \nu\tilde{a}_1.\nu\tilde{a}_2.(P_1^{\neq\nu} \mid P_2^{\neq\nu})$

reacts to Q. With Proposition 2.1.38, either a τ-action is consumed, a process identifier is called, or two sequential processes in $P_1^{\neq\nu} \mid P_2^{\neq\nu}$ communicate. Calls to process identifiers and τ-actions give rise to transitions $([F],[Q])$, similarly communications between two processes that are located in $P_1^{\neq\nu}$ or $P_2^{\neq\nu}$. Since we have a transition $([F_1 \mid F_2],[Q])$, we have a communication between a process $M_1 + \overline{x}\langle z\rangle.P_1 + N_1$ within $P_1^{\neq\nu}$ and a process $M_2 + x(y).P_2 + N_2$ within $P_2^{\neq\nu}$. Assume now that $x \in \tilde{a}_1$. Then x is bound in F_1, which means it is neither free nor bound in F_2 with Convention 2.1.11. Consequently x is not contained in $\nu\tilde{a}_2.P_2^{\neq\nu}$, a contradiction. Hence, $x \in fn(\nu\tilde{a}_1.P_1^{\neq\nu}) = fn(F_1)$.

We now show that at the same time $fn(F_1) = \emptyset$, which contradicts $x \in fn(F_1)$. Since $[F_1] \in S$ there is a reachable process Q with $[F_1] \in fg\,(rf(Q))/_{\equiv}$. Since P is closed, Q is closed by Lemma 2.1.37. With the definition of fn and the invariance of fn under structural congruence, we derive the following inclusion: $fn(F_1) \subseteq fn(rf(Q)) = fn(Q) = \emptyset$. Fragment F_1 is closed. ■

3.4 Full Retrievability and Full Abstraction

To ensure that our semantics is a suitable representation of π-Calculus processes, we show that we can retrieve all information about a process and its reactions from the semantics. To relate a marking in the Petri net $\mathcal{N}[\![P]\!]$ and a process, we define the function $retrieve : Reach(\mathcal{N}[\![P]\!]) \to \mathcal{P}/_{\equiv}$. It constructs a process from a marking by composing (1) the fragments that are marked in parallel (2) as often as demanded by the marking. This mimics the construction in the elementary equivalence. In fact, Lemma 3.3.4 is crucial in the proof of full retrievability.

Definition 3.4.1 ($retrieve : Reach(\mathcal{N}[\![P]\!]) \to \mathcal{P}/_{\equiv}$)
Given a process $P \in \mathcal{P}$, the function $retrieve : Reach(\mathcal{N}[\![P]\!]) \to \mathcal{P}/_{\equiv}$ associates with every marking reachable in the structural semantics, $M \in Reach(\mathcal{N}[\![P]\!])$, a process class $[Q] \in \mathcal{P}/_{\equiv}$ as follows:

$$retrieve\,(M) := [\Pi_{[H]\in supp(M)} \Pi^{M([H])} H].$$

♦

The support of the reachable marking M has to be finite to ensure $retrieve\,(M)$ is a process. Since the structural semantics $\mathcal{N}[\![P]\!]$ may be an infinite net, finiteness of the support is not obvious. We state it in the following lemma.

Lemma 3.4.2
For every process $P \in \mathcal{P}$ and every marking $M \in Reach(\mathcal{N}[\![P]\!])$ the support of M, $supp(M)$, is a finite set.

Proof
Consider an arbitrary process $P \in \mathcal{P}$. The initial marking is $M_0 = dec(P)$. With $supp(dec(P)) = fg(P)/_{\equiv}$ we conclude that the support of $dec(P)$ is finite.

Assume that the support of M_n is finite. We have $M_n \rightarrow M_{n+1}$ if there is a transition $t = ([F], [Q])$ or $t = ([F_1 \mid F_2], [Q])$ with $M_n[t\rangle M_{n+1}$. The postset of t are the fragments in the decomposition of Q, i.e., $t^\bullet = supp(dec(Q))$. Hence, the support of M_{n+1} is included in $supp(M) \cup supp(dec(Q))$, which is finite. ■

The transition systems of a π-Calculus process and its Petri net semantics are isomorphic. Furthermore, the states in both transition systems correspond using the retrieve function. For the process $P = F_1 \mid F_1 \mid F_3$ in Example 3.3.8, this relationship is illustrated in Figure 3.6.

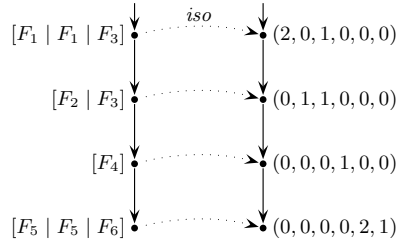


Figure 3.6:
Illustration of the transition system isomorphism in Theorem 3.4.3. The transition system $\mathcal{T}(P)$ of process P in Example 3.3.8 is depicted to the left, the transition system $\mathcal{T}(\mathcal{N}[\![P]\!])$ of the structural semantics is depicted to the right. The isomorphism $iso : Reach(P)/_{\equiv} \rightarrow Reach(\mathcal{N}[\![P]\!])$ is represented by dotted arrows, where the source is a process $[Q]$ and the target is the marking $iso([Q]) = dec(rf(Q))$. We denote $dec(rf(Q))$ as vector where entry i the value of the decomposition function for fragment F_i, i.e., $(dec(rf(Q)))([F_i])$.

Theorem 3.4.3 (Full Retrievability)
The transition systems of a process $P \in \mathcal{P}$ and that of its structural semantics $\mathcal{N}[\![P]\!]$ are isomorphic. The isomorphism $iso : Reach(P)/_{\equiv} \rightarrow Reach(\mathcal{N}[\![P]\!])$ maps $[Q]$ to $dec(rf(Q))$. The function *retrieve* is the inverse of *iso*, i.e., a process is reconstructed from a marking by $retrieve\,(iso([Q])) = [Q]$.

Before we turn to the proof of the theorem, we state our full abstraction result as a first pleasant consequence. According to Theorem 3.4.3, the structural semantics preserves up to structural congruence all information about a process, i.e., $\mathcal{N}[\![P]\!] = \mathcal{N}[\![Q]\!]$ implies $P \equiv Q$. In fact, it describes a process as precise

as structural congruence, i.e., also the reverse direction of the implication holds. This follows from the closure of the reaction relation under structural congruence and the definition of the set of places.

Proposition 3.4.4 (Full Abstraction)
Consider $P, Q \in \mathcal{P}$. Then $P \equiv Q$ if and only if $\mathcal{N}[\![P]\!] = \mathcal{N}[\![Q]\!]$.

Proof
From $P \equiv Q$ we have $Reach(P) = Reach(Q)$ with Rule (Struct). Thus, the reachable fragments coincide, $fg\,(rf(Reach(P))) = fg\,(rf(Reach(Q)))$. With this, $S_{\mathcal{N}[\![P]\!]} = fg\,(rf(Reach(P)))/_{\equiv}$ equals $S_{\mathcal{N}[\![Q]\!]} = fg\,(rf(Reach(Q)))/_{\equiv}$. Since the transition sets as well as the weight functions depend on S only, the equalities $T_{\mathcal{N}[\![P]\!]} = T_{\mathcal{N}[\![Q]\!]}$ and $W_{\mathcal{N}[\![P]\!]} = W_{\mathcal{N}[\![Q]\!]}$ hold. With Corollary 3.3.6, $P \equiv Q$ implies $dec(rf(P)) = dec(rf(Q))$. Thus, the initial markings coincide, the nets are equal.

The reverse holds with Theorem 3.4.3. Let $M_{0,P}{=}M_{0,Q}$ be the initial markings in $\mathcal{N}[\![P]\!]{=}\mathcal{N}[\![Q]\!]$. Then $[P] = retrieve\,(M_{0,P}) = retrieve\,(M_{0,Q}) = [Q]$. ■

We spend the remainder of the section proving Theorem 3.4.3. To show that *iso* is well-defined, we observe that $[Q] = [R]$ implies $iso([Q]) = dec(rf(Q)) = dec(rf(R)) = iso([R])$ with Corollary 3.3.6. Since the domain of the decomposition functions is $\mathcal{P}_{\mathcal{F}}/_{\equiv}$, we also need to ensure that for any reachable process $[Q]$ the function $iso([Q]) = dec(rf(Q))$ is a valid marking of $\mathcal{N}[\![P]\!]$ with Convention 2.2.5. This means, we have to prove $(dec(rf(Q)))([F]) = 0$ for all $[F] \notin S$. With Lemma 3.3.2, $supp(dec(rf(Q))) = fg\,(rf(Q))/_{\equiv}$. Since Q is reachable, we get $fg\,(rf(Q))/_{\equiv} \subseteq fg\,(rf(Reach(P)))/_{\equiv} = S$. Hence, $supp(dec(rf(Q))) \subseteq S$.

To prove the theorem we show that *retrieve* is the inverse of *iso* and that *iso* is an isomorphism between the transition systems, i.e., *iso* maps the initial process to the initial marking, *iso* is bijective, and *iso* is a strong graph homomorphism. A strong graph homomorphism demands that transition $[P_1] \to [P_2]$ exists in the transition system of process P if and only if transition $iso([P_1]) \to iso([P_2])$ exists in the transition system of the Petri net $\mathcal{N}[\![P]\!]$. Lemma 3.4.5 eases the proof of this equivalence as it shows that taking transitions in the Petri net is closely related with performing reactions in the process.

Lemma 3.4.5
Take $iso : Reach(P)/_{\equiv} \to Reach(\mathcal{N}[\![P]\!])$ from Theorem 3.4.3. For $M_1 = iso([P_1])$ and $M_2 = iso([P_2])$ the following (1) and (2) as well as (3) and (4) are equivalent:

(1) $\exists t = ([F], [Q]) \in T : M_1(s) \geq W(s,t)$, for all $s \in {}^\bullet t$
and $M_2(s) = M_1(s) - W(s,t) + W(t,s)$, for all $s \in S$

(2) $\exists F \in fg\,(rf(Reach(P)))\,, Q, P' \in \mathcal{P} : F \to Q$
and $P_1 \equiv F \mid P'$ and $P_2 \equiv Q \mid P'$

(3) $\exists t = ([F_1 \mid F_2], [Q]) \in T : M_1(s) \geq W(s,t)$, for all $s \in {}^\bullet t$
and $M_2(s) = M_1(s) - W(s,t) + W(t,s)$, for all $s \in S$

(4) $\exists F_1, F_2 \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F_1 \mid F_2 \to Q$
and $P_1 \equiv F_1 \mid F_2 \mid P'$ and $P_2 \equiv Q \mid P'$.

Proof
We prove equivalence of (1) and (2) directly via a chain of equivalences. The proof for (3) and (4) is similar:

$$\begin{aligned}
&\exists t = ([F], [Q]) \in T : M_1(s) \geq W(s,t), \text{ for all } s \in {}^\bullet t \\
&\quad \wedge\ M_2(s) = M_1(s) - W(s,t) + W(t,s), \text{ for all } s \in S \\
\Leftrightarrow\ & \text{(Def. } W \text{)} \\
&\exists t = ([F], [Q]) \in T : M_1([F]) \geq W([F], ([F], [Q])) = 1 \\
&\quad \wedge\ M_2(s) = M_1(s) - W(s,t) + W(t,s), \text{ for all } s \in S \\
\Leftrightarrow\ & \text{(Def. } iso,\ M_i = iso([P_i]);\ S = fg\left(Reach(rf(P))\right)/_{\equiv} \text{)} \\
&\exists ([F], [Q]) \in T : (dec(rf(P_1)))([F]) \geq 1 \\
&\quad \wedge\ (dec(rf(P_2)))([G]) = (dec(rf(P_1)))([G]) \\
&\qquad - W([G], ([F], [Q])) + W(([F], [Q]), [G]), \text{ for all } [G] \in S \\
\Leftrightarrow\ & \text{(Def. } W \text{)} \\
&\exists ([F], [Q]) \in T : (dec(rf(P_1)))([F]) \geq 1 \\
&\quad \wedge\ (dec(rf(P_2)))([G]) = (dec(rf(P_1)))([G]) \\
&\qquad - (dec(F))([G]) + (dec(rf(Q)))([G]), \text{ for all } [G] \in S \\
\Leftrightarrow\ & \text{(Def. } (dec(rf(P_1)))([F]) \text{)} \\
&\exists ([F], [Q]) \in T, P' \in \mathcal{P} : rf(P_1) \equiv F \mid P' \\
&\quad \wedge\ (dec(rf(P_2)))([G]) = (dec(rf(P_1)))([G]) \\
&\qquad - (dec(F))([G]) + (dec(rf(Q)))([G]), \text{ for all } [G] \in S \\
\Leftrightarrow\ & \text{(Def. } T \text{)} \\
&\exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \wedge rf(P_1) \equiv F \mid P' \\
&\quad \wedge\ (dec(rf(P_2)))([G]) = (dec(rf(P_1)))([G]) \\
&\qquad - (dec(F))([G]) + (dec(rf(Q)))([G]), \text{ for all } [G] \in S.
\end{aligned}$$

All fragments in F, $rf(Q)$, $rf(P_1)$, $rf(P_2)$ are in S. Hence, for all fragments

$[G] \notin S$ we get $(dec(F))([G]) = (dec(rf(Q)))([G]) = (dec(rf(P_i)))([G]) = 0$. Thus, the equality $(dec(rf(P_2)))([G]) = (dec(rf(P_1)))([G]) - (dec(F))([G]) + (dec(rf(Q)))([G])$ holds for all $[G] \in \mathcal{P}_{\mathcal{F}/\equiv}$, the functions are equal:

$$\begin{aligned} \Leftrightarrow \quad & \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \wedge rf(P_1) \equiv F \mid P' \\ & \wedge\ dec(rf(P_2)) = dec(rf(P_1)) - dec(F) + dec(rf(Q)). \end{aligned}$$

We now observe that $rf(P_1) \equiv F \mid P'$ implies $dec(rf(P_1)) = dec(rf(F \mid P'))$ with Corollary 3.3.6. Since we preserve the premise of the implication, i.e., the statement $rf(P_1) \equiv F \mid P'$, we can make use of the equality and still get an equivalent statement:

$$\begin{aligned} \Leftrightarrow \quad & \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \wedge rf(P_1) \equiv F \mid P' \\ & \wedge\ dec(rf(P_2)) = dec(rf(F \mid P')) - dec(F) + dec(rf(Q)) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow \quad & (\ dec(rf(F \mid P')) = dec(F) + dec(rf(P'))\ \text{explained below}\ ) \\ & \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \wedge rf(P_1) \equiv F \mid P' \\ & \wedge\ dec(rf(P_2)) = dec(F) + dec(rf(P')) - dec(F) + dec(rf(Q)) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow \quad & (\ dec(rf(P')) + dec(rf(Q)) = dec(rf(P' \mid Q))\ \text{explained below}\ ) \\ & \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \wedge rf(P_1) \equiv F \mid P' \\ & \wedge\ dec(rf(P_2)) = dec(rf(P' \mid Q)) \end{aligned}$$

$$\begin{aligned} \Leftrightarrow \quad & (\ \text{Corollary 3.3.6: } dec(rf(P_2)) = dec(rf(P' \mid Q))\ \text{iff } P_2 \equiv P' \mid Q\ ) \\ & \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \\ & \wedge\ P_1 \equiv F \mid P' \wedge P_2 \equiv P' \mid Q. \end{aligned}$$

The equations rely on the definition of rf and Lemma 3.3.2. They in particular hold for $rf(P') = \mathbf{0}$ or $rf(Q) = \mathbf{0}$. The last equivalence also needs $rf(P_1) \equiv P_1$ in Lemma 3.2.7. This concludes the proof. ■

Proof (of Theorem 3.4.3)

Initial States The initial states coincide by the definition of $\mathcal{N}$ and the definition of *iso* since $M_0 := dec(rf(P)) =: iso([P])$.

Strong Graph Homomorphism Let P_1 and P_2 be two reachable processes and $M_1 = iso([P_1])$, $M_2 = iso([P_2])$. We show $M_1 \to M_2$ if and only if $[P_1] \to [P_2]$. The proof amounts to applying Lemma 3.4.5 to switch from Petri net to process level. We then need the fact that at most two processes are involved in a reaction.

$$M_1 \to M_2$$

$$
\begin{aligned}
(\text{ Def. } \to) \quad &\Leftrightarrow \quad \exists t \in T : M_1[t\rangle M_2 \\
(\text{ Def. } [t\rangle) \quad &\Leftrightarrow \quad \exists t \in T : M_1(s) \geq W(s,t), \text{ for all } s \in {}^\bullet t \\
&\qquad \wedge\ M_2(s) = M_1(s) - W(s,t) + W(t,s), \text{ for all } s \in S \\
(\text{ Def. } T) \quad &\Leftrightarrow \quad \exists t = ([F],[Q]) \in T : M_1(s) \geq W(s,t), \text{ for all } s \in {}^\bullet t \\
&\qquad \wedge\ M_2(s) = M_1(s) - W(s,t) + W(t,s), \text{ for all } s \in S \\
&\vee \quad \exists t = ([F_1 \mid F_2],[Q]) \in T : M_1(s) \geq W(s,t), \text{ for all } s \in {}^\bullet t \\
&\qquad \wedge\ M_2(s) = M_1(s) - W(s,t) + W(t,s), \text{ for all } s \in S \\
(\text{ Lemma 3.4.5 }) \quad &\Leftrightarrow \quad \exists F \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F \to Q \\
&\qquad \wedge\ P_1 \equiv F \mid P' \wedge P_2 \equiv Q \mid P' \\
&\vee \quad \exists F_1, F_2 \in fg\left(rf(Reach(P))\right), Q, P' \in \mathcal{P} : F_1 \mid F_2 \to Q \\
&\qquad \wedge\ P_1 \equiv F_1 \mid F_2 \mid P' \wedge P_2 \equiv Q \mid P'.
\end{aligned}
$$

The following implication from left to right holds with the Rules (Par) and (Struct). The reverse direction is a consequence of Proposition 2.1.38, which reveals that at most two sequential processes are involved in a reaction. Since a sequential process is located in exactly one fragment, reactions require at most two fragments.

$$
\begin{aligned}
&\Leftrightarrow \quad P_1 \to P_2 \\
(\text{ Def. } \to_{\mathcal{T}}) \quad &\Leftrightarrow \quad [P_1] \to_{\mathcal{T}} [P_2].
\end{aligned}
$$

Hence, function *iso* is a strong graph homomorphism.

Codomain We already argued that the functions $iso([Q]) = dec(rf(Q))$ can be understood as markings in $\mathbb{N}^S$. We still need to ensure $iso([Q])$ is a reachable marking. This follows from the previous two statements. Technically, we do an induction on the reachable processes. The base case is $[P]$ with $iso([P]) = M_0 \in Reach(\mathcal{N}[\![P]\!])$.

Let $[P_n] \in Reach(P)/_{\equiv}$ with $iso([P_n]) \in Reach(\mathcal{N}[\![P]\!])$. Consider $[P_{n+1}]$ with $[P_n] \to_{\mathcal{T}} [P_{n+1}]$. Since $[P_{n+1}]$ is a reachable process, it is mapped by *iso* to a marking $iso([P_{n+1}])$. Since *iso* is a strong graph homomorphism, $[P_n] \to_{\mathcal{T}} [P_{n+1}]$ now implies $iso([P_n]) \to_{\mathcal{T}} iso([P_{n+1}])$ in $\mathcal{N}[\![P]\!]$. Since by the hypothesis $iso([P_n]) \in Reach(\mathcal{N}[\![P]\!])$, we conclude $iso([P_{n+1}]) \in Reach(\mathcal{N}[\![P]\!])$, which closes the induction.

Injectivity Let $[Q_1], [Q_2] \in Reach(P)/_{\equiv}$. With Corollary 3.3.6, $Q_1 \not\equiv Q_2$ implies $dec(rf(Q_1)) \neq dec(rf(Q_2))$. Since $iso([Q_1]) = dec(rf(Q_1)) \neq dec(rf(Q_2)) = iso([Q_2])$, function *iso* is injective.

Surjectivity Let M be a reachable marking. We have to show that a process $[Q]$ is reachable with $iso([Q]) = M$. We prove this by induction on the length of

the transition sequences. In the base case, we consider the empty sequence, i.e., we reach M_0 in the Petri net. We already observed $M_0 = iso([P])$.

Assume for M_n we have the reachable process $[P_n]$ with $iso([P_n]) = M_n$. Consider M_{n+1} with $M_n[t\rangle M_{n+1}$. We prove the existence of $[P_{n+1}]$ with $P_n \to P_{n+1}$ and $iso([P_{n+1}]) = M_{n+1}$. By definition of the transition relation for Petri nets we have

$$\begin{aligned} & M_n[t\rangle M_{n+1} \\ \Leftrightarrow\quad & M_n(s) \geq W(s,t), \text{ for all } s \in {}^\bullet t \text{ and} \\ & M_{n+1}(s) = M_n(s) - W(s,t) + W(t,s), \text{ for all } s \in S. \end{aligned}$$

Two kinds of transitions exist, $t = ([F],[Q])$ with $F \to Q$ and $t = ([F_1 \mid F_2],[Q])$ with $F_1 \mid F_2 \to Q$. We consider the first case, the second is similar. Since the places in S are classes of reachable fragments, we denote them by $[G]$:

$$\begin{aligned} & & & M_n([G]) \geq W([G],([F],[Q])), \text{ for all } [G] \in {}^\bullet([F],[Q]) \\ & (\text{ Def. } W\ ) & \Rightarrow\quad & M_n([F]) \geq W([F],([F],[Q])) = 1 \\ & (\ M_n = iso([P_n])\ ) & \Rightarrow\quad & (dec(rf(P_n)))([F]) \geq 1 \\ & (\text{ Def. } dec(rf(P_n))\ ) & \Rightarrow\quad & \exists P' \in \mathcal{P} : rf(P_n) \equiv F \mid P' \\ & (\text{ Def. } t\ ) & \Rightarrow\quad & \exists P' \in \mathcal{P} : P_n \equiv rf(P_n) \equiv F \mid P' \to Q \mid P'. \end{aligned}$$

This means $P_n \to Q \mid P'$. It remains to be shown that $[Q \mid P']$ is mapped to M_{n+1}, i.e., $iso([Q \mid P']) = dec(rf(Q \mid P')) = M_{n+1}$. The domain of the decomposition function is $\mathcal{P}_{\mathcal{F}/\equiv}$ while the domain of M_{n+1} is the subset $S = fg\,(rf(Reach(P)))/_\equiv$. Since $Q \mid P'$ is a reachable process, we already showed that it maps all fragments outside S to zero, hence it is correct to understand it as a marking in $\mathbb{N}^S$ with Convention 2.2.5. We now show that $dec(rf(Q \mid P'))$ and M_{n+1} coincide on the fragments $[G] \in S$:

$$\begin{aligned} & (dec(rf(Q \mid P')))([G]) \\ = \quad & (dec(rf(Q)))([G]) + (dec(rf(P')))([G]) \\ = \quad & (\ dec(rf(P')) = dec(rf(P_n)) - dec(F) \text{ explained below }\ ) \\ & (dec(rf(Q)))([G]) + (dec(rf(P_n)))([G]) - (dec(F))([G]) \\ = \quad & (\text{ Def. } W,\ dec(rf(P_n)) = iso([P_n]) = M_n\ ) \\ & W(([F],[Q]),[G]) + M_n([G]) - W([G],([F],[Q])) \\ = \quad & (\text{ Def. } M_n[([F],[Q])\rangle M_{n+1}\ ) \\ & M_{n+1}([G]). \end{aligned}$$

With Corollary 3.3.6, the congruence $P_n \equiv F \mid P'$ implies the first of the following equations: $dec(rf(P_n)) = dec(rf(F \mid P')) = dec(F) + dec(rf(P'))$. The second exploits the definition of rf and Lemma 3.3.2. It is immediate to check that it also holds in case $rf(P') = \mathbf{0}$. Rewriting the equation yields the equality $dec(rf(P')) = dec(rf(P_n)) - dec(F)$ used above and concludes the proof that $iso([Q \mid P']) = M_{n+1}$ in case $t = ([F], [Q])$. The proof for $t = ([F_1 \mid F_2], [Q])$ is similar. Function iso is surjective.

Retrievability We derive $retrieve\,(iso([Q])) = [Q]$ with the following equalities:

$$\begin{aligned} & [Q] \\ (\text{ Lemma 3.2.7 }) \quad &= \quad [rf(Q)] \\ (\text{ Lemma 3.3.4 }) \quad &= \quad [\Pi_{[H] \in supp(dec(rf(Q)))} \Pi^{(dec(rf(Q)))([H])} H] \\ (\text{ Def. } iso, \text{ def. } retrieve \text{ }) \quad &= \quad retrieve\,(iso([Q]))\,. \end{aligned}$$

This concludes the proof of Theorem 3.4.3. ■

The definition of the structural semantics is declarative as it refers to the set of all reachable fragments and adds transitions where appropriate. In the following section, we comment on the implementation.

3.5 Implementation Issues

We implemented the translation in the tool PETRUCHIO [Str07, SM08]. Since the set of places in the Petri net $\mathcal{N}[\![P]\!]$ is based on the set of reachable fragments $fg\,(rf(Reach(P)))$ and since the set of reachable processes $Reach(P)$ is defined inductively, our algorithm computes the Petri net $\mathcal{N}[\![P]\!]$ inductively as follows. We determine a sequence of nets $N_0, N_1, N_2, \ldots$ with $N_k = (S_k, T_k, W_k, M_0)$. The initial net is $N_0 = (fg\,(rf(P))/_{\equiv}, \emptyset, \emptyset, dec(rf(P)))$, i.e., the places are the fragments in the initial process, $S_0 = fg\,(rf(P))/_{\equiv}$, the initial transition set is empty, $T_0 = \emptyset$, and so is the initial weight function, $W_0 = \emptyset$. The initial marking is $M_0 = dec(rf(P))$.

Assume we computed the net $N_k = (S_k, T_k, W_k, M_0)$. For every place $[F] \in S_k$ that has an internal reaction, i.e., $F \to Q$, we add a transition $([F], [Q])$. Similarly, for two places $[F_1], [F_2] \in S_k$ that are (1) simultaneously markable and (2) able to communicate, i.e., $F_1 \mid F_2 \to Q$, we add a transition $([F_1 \mid F_2], [Q])$. This yields the new transition set T_{k+1}. The postset of a transition is the set of fragments in the restricted form of process Q, $fg\,(rf(Q))/_{\equiv}$. The new fragments are added to S_k giving the new set of places S_{k+1}. We get the new weight function W_{k+1} from W_k by adding arcs between the new transitions and the (old and new) places according to Definition 3.3.7. The initial marking M_0 does not change.

The computation stops if there are no more transitions to add, i.e., $N_{k+1} = N_k$. We then have $N_k = \mathcal{N}[\![P]\!]$ as we computed exactly the reachable fragments.[2]

The critical issue in the implementation of the semantics is to determine the places $[F_1], [F_2] \in S_k$, which can be marked simultaneously. We solve the problem by computing the coverability graph $Cov(N_k)$ of the nets N_k. To avoid recomputing $Cov(N_k)$ for every net N_k, Tim Strazny showed that the coverability graph $Cov(N_{k+1})$ is an extension of $Cov(N_k)$ [Str07].

To make use of today's multi-core computers, the compiler is implemented in a dual-threaded software architecture. The first thread updates the coverability graph and the second thread computes the Petri net as described above.

Note that it is not necessary to compute the coverability graph if P is a closed process. By Lemma 3.3.11, $\mathcal{N}[\![P]\!]$ is a communication-free net, i.e., it only contains transitions $([F], [Q])$, which depend on a single place that is markable.

Due to memory limitations, the coverability graph of a subnet of $\mathcal{N}[\![P]\!]$ may not be computable in practice. In this case, we add the transition $([F_1 \mid F_2], [Q])$ if the places $[F_1]$ and $[F_2]$ can communicate, i.e., $F_1 \mid F_2 \to Q$, regardless of whether they can be marked simultaneously. This results in a Petri net $\mathcal{N}_{NoCov}[\![P]\!]$ which subsumes $\mathcal{N}[\![P]\!]$, i.e., $\mathcal{N}[\![P]\!] = (S, T, W, M_0)$ and $\mathcal{N}_{NoCov}[\![P]\!] = (S', T', W', M_0')$ with $S \subseteq S'$, $T \subseteq T'$, $W \subseteq W'$, and $M_0 = M_0'$. The transition systems of $\mathcal{N}[\![P]\!]$ and $\mathcal{N}_{NoCov}[\![P]\!]$ are still isomorphic. The reason is that a transition $([F_1 \mid F_2], [Q])$, which is added although the places $[F_1]$ and $[F_2]$ are not simultaneously markable, is never enabled. A negative effect of this inaccuracy is that $\mathcal{N}_{NoCov}[\![P]\!]$ is often much larger than $\mathcal{N}[\![P]\!]$ and may even become infinite although $\mathcal{N}[\![P]\!]$ is finite.

3.6 Related Work and Conclusion

We first review the operational semantics of the π-Calculus, then we discuss the relationship with work on structural congruence relations, which led to normal forms related to ours.

Operational Semantics To begin with, we discuss the automata-theoretic semantics that reflect process behaviour. All of them explicitly represent the concurrency of sequential processes in the following sense. A state (place) of the associated automaton (or Petri net) represents a derivative of a sequential process [Eng96, BG95, AM02, DKK06a, MP01]. Thus, processes are split along the parallel composition operator in contrast with our semantics decomposing along fragments, i.e., along substructures induced by the scopes of restricted names. Subsequently, we discuss a semantics that also represents structural information [MP95b].

[2] More precisely, N_k equals $\mathcal{N}[\![P]\!]$ without dead transitions like t_4 in Example 3.3.8.

In [Eng96], a Petri net semantics is defined for the *small* π-Calculus, which does not contain choice compositions and uses replication instead of recursion. The proposed semantics reflects the reaction relation. Name creation is handled by using fresh global names, bound names in input prefixes are replaced by de Bruijn indices, and replication is modelled by countably infinite union. In subsequent papers [EG99, EG04b], Engelfriet and Gelsema show that the discriminating power of their semantics corresponds to extended and decidable versions of structural congruence. Due to their focus on structural congruence, the authors are not concerned with finiteness of the resulting Petri nets. In fact, the semantics immediately yields (1) infinite nets with (2) arcs that have countably infinite weights (denoted by ω), and (3) infinite markings (ω-tokens are allowed), which makes it unusable for automatic verification of the system behaviour.

A translation of the π-Calculus into potentially infinite place/transition Petri nets with inhibitor arcs is presented in [BG95]. It models the *early* transition relation. Inhibitor arcs are employed in two ways. They check for the presence of restrictions on names and allow for replacing choices by parallel compositions. In a recent paper, the authors show that for so-called *finite net processes* the resulting inhibitor nets are finite and primitive [BG09]. Primitive inhibitor nets enjoy the property that an inhibiting place can never be emptied, if a marking exceeds a certain bound. A result by Busi [Bus02] shows that important properties of finite primitive inhibitor nets—including model checking of linear-time μ-Calculus—are decidable. The crucial characteristics of finite net processes is that they generate a bounded number of restricted names. We shall study this class of processes in Chapter 9 under the name of *restriction-bounded* processes. We show that they also have a finite place/transition Petri net representation.

The main contribution in [BG09] is the study of non-interleaving and causal semantics for the π-Calculus. The authors argue that it is impossible to give a semantics in terms of standard place/transition Petri nets, which reflects the intended causality of processes. As our structural semantics reflects the interleaving behaviour of processes, it is coarser than the mentioned causal semantics and so their impossibility result does not apply here.

The control reachability problem (CRP) asks for the reachability of a process containing a given process identifier [AM02]. The authors prove decidability for input bounded systems with unique receiver. A system is input bounded if the continuation of a name generating process is uniquely determined. The unique receiver condition demands that only the creating process listens on a restricted name. Combined, the conditions give an upper bound on the restricted names actually used for communication. Dead restricted names are replaced by generic ones. Decidability is obtained via a reduction to the coverability problem of Petri nets with transfer. We observe that CRP is decidable for the structurally stationary processes we study in the following chapter. The relationship of input bounded and unique receiver processes with our work will become clear in Chapter 9.

In [DKK06a, DKK06b, DKK08], a translation into finite high-level Petri nets with read arcs is presented. The transitions of the net reflect strongly bisimilar the *indexed* transition system of the π-Calculus. A process is first translated into a context-based representation which removes restrictions. From this representation the high-level nets are obtained compositionally, i.e., for all remaining operators there is a corresponding net operator. Different instances of recursive processes are distinguished via invocation trails. Invocation itself is handled via marking equivalence. The authors aim at using the translation for automatic verification and report on successful first experiments [KKN06].

History-Dependent automata (HD-automata) [MP95a, Pis99, MP01] handle restricted names explicitly by associating to each state a set of names. Functions on transitions relate the names in label, source, and target states. HD-automata come equipped with bisimulation theory and techniques for computing irredundant and minimal representations. The *ground* and *early* π-Calculus semantics are translated into HD-automata. Depending on the translation, bisimilarity on the automata coincides with the corresponding bisimilarity on processes. Furthermore, bisimilarity on the automata coincides with isomorphism of the minimal representations. Finite HD-automata are gained for *finitary processes*. This covers the well-known *finite control processes* [Dam96], where parallel composition is not used within recursions. In the following chapter, we show that finitary processes are also finitely represented under our structural semantics.

The only semantics reflecting a notion of structure as discussed in Section 3.1 is the graph rewriting semantics in [MP95b]. A hypergraph is constructed by mapping names to vertices and sequential processes to hyperedges. An arc indicates that a name is free in a process. The *early* transition relation of the π-Calculus is mimicked by a finite number of graph rewriting rules. The authors investigate observational equivalences based on their semantics.

Normal Forms Engelfriet and Gelsema use normal forms similar to ours (but for the *small* π-Calculus) in the proofs of their decidability and correspondence results [EG99, EG04b]. In the *subconnected normal form* replications are moved inwards. Like our restricted form, the *cell normal form* strives for innermost restrictions, but also under prefixes. Our work differs in the way the normal form is exploited. While Engelfriet and Gelsema use it as technical tool, we use fragments as places in our semantics, i.e., we exploit the structure to finitely represent the process behaviour.

Recently, the authors investigated the structural congruence in [Mil99] but for the small π-Calculus. For the class of *replication restricted processes*, they prove decidability via a reduction to linear equations with natural coefficients [EG04a]. It is based on a normal form that finds the connected subprocesses. These *webs* differ from our fragments in that they have outermost restrictions. In [EG07], the authors show decidability for restriction-free processes by a translation to

reversible Petri nets. The transitions of the net model applications of the replication law $!P \equiv P \mid !P$. This semantics does not reflect any behavioural relation of processes.

Conclusion and Future Work We presented a Petri net semantics that reflects the reaction relation of the monadic π-Calculus with recursion. It is based on a normal form, which computes groups of connected process that we call *fragments*. Since fragments only depend on the restricted names that sequential processes share, we note that the extension of the so-called *restricted form* and thus of the semantics to the polyadic π-Calculus is straightforward [Str07].

Theorem 3.4.3 shows that the structural semantics satisfies the request for *retrievability*: the transition systems of the Petri net and the process are isomorphic and the structure of the reachable processes is preserved by the markings. As a consequence, verification results obtained for the Petri net carry over to the process under study. We also argued that the semantics is *analysable*: Lemma 3.3.11 shows that closed processes are translated to communication-free Petri nets, where behavioural properties can be inferred efficiently [Esp97b].

We implemented the semantics in our tool PETRUCHIO [Str07, SM08]. The implementation relies on the coverability graph to detect the places that are markable simultaneously. As the size of the coverability graph is not bounded by a primitive recursive function in the size of the net, we plan to use more compact structures like unfolding prefixes to decide simultaneous reachability.

4 Structural Stationarity

Contents

In the previous chapter, we proposed a Petri net semantics for the π-Calculus that highlights the connection structure of processes. With this new viewpoint, processes are finitely represented if and only if there are finitely many substructures (or fragments) every reachable process consists of.[1] In particular, unbounded name creation and unbounded process creation do not necessarily imply infinite automata-theoretic representations. Recall that *finiteness* is one of the quality aspects a Petri net semantics for verification has to satisfy, since all automated verification methods rely on this constraint.

Processes that reach finitely many fragments are called *structurally stationary*. Since the definition refers to the precise form of fragments, it is hard to prove that a given process is structurally stationary. We present a complete characterisation, which refers to the parallel composition operator: a process is structurally stationary if and only if the number of sequential processes in all reachable fragments is bounded. The corresponding Theorem 4.3.2 is a powerful tool to prove structural stationarity and thus our semantics is suitable to finitely represent

[1] Since the π-Calculus is Turing complete but finite place/transition Petri nets are not, the semantics has to yield infinite nets for some processes.

the process. For example, it shows that structurally stationary processes unify the classes of finitary processes (that are finitely represented by HD-automata) [Dam96, Cai04, Pis99, MP01, FGMP03] and restriction-free processes (which are finitely represented by different concurrency semantics [BG95, AM02, BG09]).

The systems we aim to verify typically have a client-server architecture, and we design the syntactic class of *finite handler processes* to model them. An application of the characterisation shows that finite handler processes are structurally stationary. Combined with the unification of the process classes, this demonstrates the *expressiveness* of structurally stationary processes, i.e., the structural semantics is usable for the verification of a wide range of systems.

Finally, we show that Petri nets can be represented by structurally stationary processes. This translation reveals complexity-theoretic aspects of the structural semantics. It shows that the size is in general not bounded by a primitive recursive function in the size of the process. Moreover, it indicates undecidability or decidability but EXPSpace-hardness of several verification problems for structurally stationary processes.

To sum up, our contributions are as follows.

- We define the semantical property of structural stationarity and show that it completely characterises the processes that are mapped to finite Petri nets by the structural semantics.
- We give a complete characterisation of structural stationarity. It reveals that important classes of processes in the literature are structurally stationary and implies a new method for the computation of the structural semantics.
- We design a rich syntactic class, called finite handler processes, that we prove to be structurally stationary.
- We investigate the representation of Petri nets as structurally stationary processes and draw complexity- as well as decidability-theoretic conclusions.

The chapter is organised as follows. Section 4.1 defines structural stationarity and characterises finiteness of the structural semantics. Section 4.2 defines derivatives, the technical tool for the proof of the complete characterisation of structural stationarity in Section 4.3. We exploit the characterisation in Section 4.4, where we show that the syntactic class of finite handler processes is structurally stationary. In Section 4.5, we investigate the representation of Petri nets as structurally stationary processes, before we discuss related work and points of future research in Section 4.6.

4.1 Structural Stationarity and Finiteness

Intuitively, a process is structurally stationary if there is a finite number of types of fragments in the system. Technically, there are finitely many fragments so that the restricted form of all reachable processes is a parallel composition of those fragments.

Definition 4.1.1 (Structural Stationarity)
A process $P \in \mathcal{P}$ is *structurally stationary* if there is a finite set of fragments such that the fragments of all reachable processes are up to structural congruence included in that set:

$$\exists\{F_1, \ldots, F_n\} : \forall Q \in \mathit{Reach}(P) : \forall F \in \mathit{fg}\,(\mathit{rf}(Q)) : \exists i : F \equiv F_i.$$

The *set of all structurally stationary processes is* $\mathcal{P}_{FG<\infty}$. ♦

Lemma 4.1.2 states the equivalence between finiteness of the structural semantics and structural stationarity mentioned in the introduction.

Lemma 4.1.2 (Finiteness)
The structural semantics $\mathcal{N}[\![P]\!]$ is finite if and only if the process P is structurally stationary, i.e., $P \in \mathcal{P}_{FG<\infty}$.

Proof
With Lemma 3.3.10, the finiteness of $\mathcal{N}[\![P]\!]$ is equivalent to the finiteness of the set of places, $\mathit{fg}\,(\mathit{rf}(\mathit{Reach}(P)))/_{\equiv}$. The finiteness of $\mathit{fg}\,(\mathit{rf}(\mathit{Reach}(P)))/_{\equiv}$ is equivalent to structural stationarity. ■

With Lemma 4.1.2, the computation of the structural semantics in our tool PETRUCHIO terminates exactly if the process is structurally stationary. For the user, an algorithm would be desirable that checks before the compilation whether the process is structurally stationary. Unfortunately, the property is undecidable. We state the result here but delay the proof until Chapter 8, where we study several undecidability results.

Lemma 4.1.3
For a process $P \in \mathcal{P}$ it is undecidable whether P is structurally stationary, i.e., whether $P \in \mathcal{P}_{FG<\infty}$ holds.

Since we cannot rely on an algorithm to decide termination of our translation, it is important to know in advance that a process of interest is structurally stationary. To prove structural stationarity means to come up with a concrete

set of fragments and then to show that it includes all reachable fragments. For terminating processes, we can just take the set of all reachable fragments, which is a finite by König's lemma.[2]

Lemma 4.1.4 (Terminating Processes are Structurally Stationary)
If $P \in \mathcal{P}$ terminates then it is structurally stationary, i.e., $P \in \mathcal{P}_{FG<\infty}$.

For arbitrary processes, finding a suitable set of fragments is far from trivial. The characterisation in Section 4.3 reduces this task to finding a bound on the number of sequential processes in every reachable fragment. To establish completeness of this characterisation, i.e., to prove structural stationarity from boundedness, we in fact have to construct a finite set of fragments. The benefit is that we do this construction once when proving Theorem 4.3.2. After it is established, we simply apply the characterisation whenever we show structural stationarity.

Technically, the construction of the reachable fragments relies on the notion of *derivatives*, a finite set of processes computed from a given process P. The main result in the following Section 4.2 shows that all processes reachable from P are created from elements in *derivatives*(P) via parallel composition, restriction, and substitution. The corresponding Proposition 4.2.2 is crucial in the proof of Theorem 4.3.2.

4.2 Derivatives

The idea to construct the set of *derivatives* of process P is to recursively remove all prefixes as if they were consumed in communications. If a process identifier K is called, directly in P or indirectly in a defining equation, we also add the derivatives of the process defining K. So, for the process $b(y).\overline{y}\langle b\rangle.K\lfloor a\rfloor$ with $K(x) := \overline{x}\langle x\rangle$ we get the derivatives $b(y).\overline{y}\langle b\rangle.K\lfloor a\rfloor$, $\overline{y}\langle b\rangle.K\lfloor a\rfloor$, $K\lfloor a\rfloor$, and $\overline{x}\langle x\rangle$.

Definition 4.2.1 (*derivatives* : $\mathcal{P} \to \mathbb{P}(\mathcal{P})$)
To define the derivatives of a process we need the function $der : \mathcal{P} \to \mathbb{P}(\mathcal{P})$:

$$\begin{aligned} der(\mathbf{0}) &:= \emptyset & der(K\lfloor\tilde{a}\rfloor) &:= \{K\lfloor\tilde{a}\rfloor\} \\ der(\pi.P) &:= \{\pi.P\} \cup der(P) & der(M+N) &:= \{M+N\} \cup der(M) \cup der(N) \\ der(P \mid Q) &:= der(P) \cup der(Q) & der(\nu a.P) &:= der(P). \end{aligned}$$

Consider $P \in \mathcal{P}$. The set of *derivatives* of P, denoted by *derivatives*(P), is the smallest set so that (1) $der(P) \subseteq$ *derivatives*(P) and (2) if $K\lfloor\tilde{a}\rfloor \in$ *derivatives*(P) then $der(Q) \subseteq$ *derivatives*(P), where $K(\tilde{x}) := Q$. ♦

[2] König's lemma says that every tree of finite degree is either finite or contains an infinite path.

There are two differences between the derivatives and the processes obtained with the reaction relation. Names y that are replaced by received names when an action $b(y)$ is consumed remain unchanged in the derivatives. Parameters $\tilde{x}$ that are replaced by $\tilde{a}$ when an identifier $K\lfloor\tilde{a}\rfloor$ is called are not replaced in the derivatives. Both shortcomings are corrected by substitutions applied to the free names in the derivatives. Proposition 4.2.2 shows that this yields all reachable processes.

Proposition 4.2.2
Let $P \in \mathcal{P}$. Every $Q \in \mathit{Reach}(P)$ and every $F \in \mathit{fg}\,(\mathit{rf}(Q))$ is structurally congruent with a process $\nu\tilde{a}.Q^{\neq\nu}$ in standard form so that $Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i$ with $Q_i \in \mathit{derivatives}(P)$ and $\sigma_i : \mathit{fn}(Q_i) \to \mathit{fn}(P) \cup \tilde{a}$.

Example 4.2.3 gives an intuitive understanding to this technical statement.

Example 4.2.3 (Derivatives)
Consider $P = \nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a, y\rfloor$, where $K(a, y) := \overline{y}\langle a\rangle$. The only reaction sequence is

$$\nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a, y\rfloor \to \nu b.(b(x) \mid K\lfloor a, b\rfloor) \to \nu b.(b(x) \mid \overline{b}\langle a\rangle) \to \mathbf{0}.$$

We compute the set of derivatives:

$$\mathit{derivatives}(P) = \{\overline{a}\langle b\rangle.b(x), b(x), a(y).K\lfloor a, y\rfloor, K\lfloor a, y\rfloor, \overline{y}\langle a\rangle\}.$$

The following congruences show that every reachable fragment can be constructed from the derivatives as stated in Proposition 4.2.2. The reachable fragments are depicted to the left, the constructed processes to the right:

$$\begin{aligned}
\nu b.\overline{a}\langle b\rangle.b(x) &\equiv \nu b.((\overline{a}\langle b\rangle.b(x))\{a, b/a, b\}) \\
a(y).K\lfloor a, y\rfloor &\equiv (a(y).K\lfloor a, y\rfloor)\{a/a\} \\
\nu b.(b(x) \mid K\lfloor a, b\rfloor) &\equiv \nu b.(b(x)\{b/b\} \mid K\lfloor a, y\rfloor\{a, b/a, y\}) \\
\nu b.(b(x) \mid \overline{b}\langle a\rangle) &\equiv \nu b.(b(x)\{b/b\} \mid \overline{y}\langle a\rangle\{b, a/y, a\}).
\end{aligned}$$

The stop process $\mathbf{0}$ is represented by a parallel composition with empty index set, $\Pi_{i\in\emptyset} P_i$. ♦

We now turn to the proof of Proposition 4.2.2. It exploits two important inclusions. First, $\mathit{der}(P)$ always includes the sequential processes in P. Second, if we have a process Q in the derivatives of P then we already know that the set $\mathit{der}(Q)$ is included in the derivatives of P. The second inclusion relies on the fact that $Q \in \mathit{der}(P)$ implies $\mathit{der}(Q) \subseteq \mathit{der}(P)$.

Lemma 4.2.4
For two process $P, Q \in \mathcal{P}$ the following statements hold: (1) $\mathcal{S}(P) \subseteq der(P)$, (2) $Q \in der(P)$ implies $der(Q) \subseteq der(P)$, and (3) $Q \in derivatives(P)$ implies $der(Q) \subseteq derivatives(P)$.

Proof
The proof of the first property is a straightforward induction on the structure of P.

Property (2) We use induction on the structure of process P, i.e., we show for all $Q \in der(P)$ that $der(Q) \subseteq der(P)$ holds.

Base Cases The base cases are $P = \mathbf{0}$ for which the proof is trivial and $P = K\lfloor\tilde{a}\rfloor$. In this case, $Q \in der(P) = \{K\lfloor\tilde{a}\rfloor\}$ implies $Q = K\lfloor\tilde{a}\rfloor$. Thus, $der(Q) = der(P)$ holds.

Induction Step Assume the property holds for $P \in \mathcal{P}$. We only show the case $\pi.P$, the remaining cases $M + N$, $P \mid Q$, and $\nu a.P$ are similar. If $Q \in der(\pi.P) = \{\pi.P\} \cup der(P)$ then either $Q = \pi.P$ and the property is trivial or $Q \in der(P)$. In this case, the hypothesis yields $der(Q) \subseteq der(P) \subseteq der(\pi.P)$.

Property (3) The third property follows immediately from the second. If we have $R \in derivatives(P)$, then either $R \in der(P) \subseteq derivatives(P)$ or there is an identifier $K\lfloor\tilde{a}\rfloor \in derivatives(P)$ with $K(\tilde{x}) := Q$ and $R \in der(Q) \subseteq derivatives(P)$. In both cases, the second statement yields $der(R) \subseteq der(P)$ and $der(R) \subseteq der(Q)$, which ensures inclusion of $der(R)$ in $derivatives(P)$ and concludes the proof. ■

Proof (of Proposition 4.2.2)
If we can show that for a reachable process $Q \in Reach(P)$ we have $Q \equiv \nu\tilde{a}.Q^{\neq\nu}$ with $Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i$, then the scope of the substitutions σ_i has to be $fn(P) \cup \tilde{a}$. To see this, it suffices to show $fn(Q_i\sigma_i) \subseteq fn(P) \cup \tilde{a}$ with Lemma 2.1.15. Consider a name $a \in fn(Q_i\sigma_i)$, which is not in $\tilde{a}$. Then $a \in fn(\nu\tilde{a}.Q^{\neq\nu})$ by definition of fn. By the invariance of fn under structural congruence, we get

$$fn(\nu\tilde{a}.Q^{\neq\nu}) = fn(Q) \subseteq fn(P).$$

The inclusion holds with Lemma 2.1.37. A similar argumentation holds for the reachable fragments. We now prove the congruence for $Q \in Reach(P)$ via an induction on the length of the reaction sequences. The statement for F follows as a corollary.

Base Case The base case is the empty sequence, i.e., $Q_0 = P$. By Lemma 2.1.28, $P \equiv sf(P)$. For $sf(P) = \mathbf{0}$, the claim is trivial. Let $sf(P) = \nu\tilde{a}.P^{\neq\nu}$ where

$P^{\neq\nu} = \Pi_{i\in I} P_i = \Pi_{i\in I} P_i id$ and id is the identity function. To see that the P_i are in $derivatives(P)$, we check

$$\begin{aligned} P_i &\in \bigcup_{i\in I}\{P_i\} \\ (\text{ Def. } \mathcal{S},\ sf(P) = \nu\tilde{a}.(\Pi_{i\in I} P_i)\ ) &= \mathcal{S}(sf(P)) \\ (\text{ Lemma 2.1.28 }) &= \mathcal{S}(P) \\ (\text{ Lemma 4.2.4 (1) }) &\subseteq der(P) \\ (\text{ Def. } derivatives\ ) &\subseteq derivatives(P). \end{aligned}$$

Induction Step Let $Q_n \equiv \nu\tilde{a}.R^{\neq\nu}$ in standard form so that $R^{\neq\nu} = \Pi_{i\in I} R_i\sigma_i$ with $R_i \in derivatives(P)$ and $\sigma_i : fn(R_i) \to fn(P) \cup \tilde{a}$. If $Q_n \to Q_{n+1}$, then we derive the reaction $\nu\tilde{a}.R^{\neq\nu} \to Q_{n+1}$ with Rule (Struct). According to Proposition 2.1.38, there are three possible reactions for $\nu\tilde{a}.R^{\neq\nu}$. Either a communication takes place between two processes, say $R_1\sigma_1$ and $R_2\sigma_2$, a τ-prefix is consumed in a process like $R_1\sigma_1 = M_1\sigma_1 + \tau.(R'_1\sigma_1) + N_1\sigma_1$, or a process identifier is called, which means $R_1\sigma_1 = K\lfloor\tilde{x}\rfloor\sigma_1$. We consider the first case, the remaining proofs are similar. Let $\nu\tilde{a}.R^{\neq\nu} = \nu\tilde{a}.(R_1\sigma_1 \mid R_2\sigma_2 \mid R^{\neq\nu}_{rem})$, where

$$\begin{aligned} R_1\sigma_1 &= M_1\sigma_1 + \overline{x_1\sigma_1}\langle y_1\sigma_1\rangle.(R'_1\sigma_1) + N_1\sigma_1 = M_1\sigma_1 + \overline{a}\langle b\rangle.(R'_1\sigma_1) + N_1\sigma_1 \\ R_2\sigma_2 &= M_2\sigma_2 + x_2\sigma_2(y_2).(R'_2\sigma_2) + N_2\sigma_2 = M_2\sigma_2 + a(y_2).(R'_2\sigma_2) + N_2\sigma_2. \end{aligned}$$

The index *rem* stands for *remainder* and denotes the remaining processes in the parallel composition $R^{\neq\nu}$. With Proposition 2.1.38, we have

$$Q_{n+1} \equiv \nu\tilde{a}.(R'_1\sigma_1 \mid R'_2\sigma_2\{y_1\sigma_1/y_2\} \mid R^{\neq\nu}_{rem}).$$

We transform the latter process into the required form. With Lemma 2.1.28, we compute $R'_2\sigma_2\{y_1\sigma_1/y_2\} \equiv sf(R'_2\sigma_2\{y_1\sigma_1/y_2\})$. If the standard form is $\mathbf{0}$, the proposition immediately follows. We consider $sf(R'_2\sigma_2\{y_1\sigma_1/y_2\}) = \nu\tilde{a}_2.R_2^{\neq\nu}$ where $R_2^{\neq\nu} = \Pi_{j\in J} R_{2,j}$ and $\tilde{a}_2 \subseteq arn(R'_2\sigma_2\{y_1\sigma_1/y_2\})$. Since we assume the bound names to be disjoint from the free names in Convention 2.1.11, the inclusion implies $\tilde{a}_2 \cap (fn(R'_1\sigma_1) \cup fn(R^{\neq\nu}_{rem})) = \emptyset$. We need this disjointness later. To see that the $R_{2,j}$ are processes in $derivatives(P)$ to which substitutions are applied, we observe:

$$\begin{aligned} R_{2,j} &\in \bigcup_{j\in J}\{R_{2,j}\} \\ (\text{ Def. } \mathcal{S},\ sf(R'_2\sigma_2\{y_1\sigma_1/y_2\}) = \nu\tilde{a}_2.R_2^{\neq\nu}\ ) &= \mathcal{S}(sf(R'_2\sigma_2\{y_1\sigma_1/y_2\})) \\ (\text{ Properties } sf, \text{ Lemma 2.1.28 }) &= \mathcal{S}(R'_2\sigma_2\{y_1\sigma_1/y_2\}) \\ (\text{ Compatible } \mathcal{S} \text{ and } \sigma, \text{ Lemma 2.1.21 }) &= \mathcal{S}(R'_2\sigma_2)\{y_1\sigma_1/y_2\} \\ (\text{ Compatible } \mathcal{S} \text{ and } \sigma, \text{ Lemma 2.1.21 }) &= \mathcal{S}(R'_2)\sigma_2\{y_1\sigma_1/y_2\} \\ (\text{ Lemma 4.2.4 (1) }) &\subseteq der(R'_2)\sigma_2\{y_1\sigma_1/y_2\} \\ (\ R'_2 \in der(R_2), \text{ Lemma 4.2.4 (2) }) &\subseteq der(R_2)\sigma_2\{y_1\sigma_1/y_2\} \end{aligned}$$

$$(\ R_2 \in \mathit{derivatives}(P), \text{ Lemma 4.2.4 (3) }) \quad \subseteq \quad \mathit{derivatives}(P)\sigma_2\{y_1\sigma_1/y_2\},$$

where in the last step $R_2 \in \mathit{derivatives}(P)$ holds by the hypothesis. With this argumentation, we have $R_{2,j} \in \mathit{derivatives}(P)\sigma_2\{y_1\sigma_1/y_2\}$, i.e., there is a process $P_{2,j} \in \mathit{derivatives}(P)$ with $R_{2,j} = P_{2,j}\sigma_2\{y_1\sigma_1/y_2\}$. To sum up, we now have the following congruences:

$$\begin{aligned}
& & Q_{n+1} \\
(\text{ Proposition 2.1.38 }) & \equiv & \nu\tilde{a}.(R_1'\sigma_1 \mid R_2'\sigma_2\{y_1\sigma_1/y_2\} \mid R_{rem}^{\neq\nu}) \\
(\text{ Standard form }) & \equiv & \nu\tilde{a}.(R_1'\sigma_1 \mid \nu\tilde{a}_2.(\Pi_{j\in J}R_{2,j}) \mid R_{rem}^{\neq\nu}) \\
(\ R_{2,j} = P_{2,j}\sigma_2\{y_1\sigma_1/y_2\}\ ) & = & \nu\tilde{a}.(R_1'\sigma_1 \mid \nu\tilde{a}_2.(\Pi_{j\in J}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\}) \mid R_{rem}^{\neq\nu}) \\
(\text{ Scope ext., disjoint }) & \equiv & \nu\tilde{a},\tilde{a}_2.(R_1'\sigma_1 \mid \Pi_{j\in J}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\} \mid R_{rem}^{\neq\nu}) \\
(\ \nu a.P \equiv P \text{ if } a \notin \mathit{fn}(P)\ ) & \equiv & \nu\tilde{a}',\tilde{a}_2.(R_1'\sigma_1 \mid \Pi_{j\in J}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\} \mid R_{rem}^{\neq\nu}).
\end{aligned}$$

In the last step, we remove unused names from $\tilde{a}$, i.e.,

$$\tilde{a}' := \tilde{a} \cap \mathit{fn}(R_1'\sigma_1 \mid \Pi_{j\in J}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\} \mid R_{rem}^{\neq\nu}).$$

By construction, we have $\tilde{a}_2 \subseteq \mathit{fn}(\Pi_{j\in J}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\})$, and we do not remove names from this set. We handle $R_1\sigma_1$ similarly and get the following process that is structurally congruent with Q_{n+1} and of the desired form:

$$\nu\tilde{a}',\tilde{a}_1,\tilde{a}_2.(\Pi_{j\in J_1}P_{1,j}\sigma_1 \mid \Pi_{j\in J_2}P_{2,j}\sigma_2\{y_1\sigma_1/y_2\} \mid R_{rem}^{\neq\nu}).$$

Statement for F Consider $F \in \mathit{fg}(\mathit{rf}(Q))$. The first part gives $Q \equiv \nu\tilde{a}.Q^{\neq\nu}$ so that $Q^{\neq\nu} = \Pi_{i\in I}Q_i\sigma_i$ with $Q_i \in \mathit{derivatives}(P)$ and $\sigma_i : \mathit{fn}(Q_i) \to \mathit{fn}(P) \cup \tilde{a}$. With Proposition 3.2.10, we have $\mathit{rf}(Q) \equiv_{rf} \mathit{rf}(\nu\tilde{a}.Q^{\neq\nu})$, i.e., $F \equiv G$ where G is a fragment in $\mathit{fg}\left(\mathit{rf}(\nu\tilde{a}.Q^{\neq\nu})\right)$. We observe that

$$\mathcal{S}(G) \subseteq \mathcal{S}(\mathit{rf}(\nu\tilde{a}.Q^{\neq\nu})) = \mathcal{S}(\nu\tilde{a}.Q^{\neq\nu}) = \textstyle\bigcup_{i\in I}\{Q_i\sigma_i\}$$

by definition of $\mathcal{S}$, Lemma 3.2.7, and the form of $Q^{\neq\nu}$. Computing the standard form with Lemma 2.1.28 yields $G \equiv \mathit{sf}(G) = \nu\tilde{a}'.(\Pi_{j\in J}R_j)$ with $R_j \in \mathcal{S}(G) \subseteq \bigcup_{i\in I}\{Q_i\sigma_i\}$. With $F \equiv G$ we derive $F \equiv \nu\tilde{a}'.(\Pi_{j\in J}Q_j\sigma_j)$ with $Q_j \in \mathit{derivatives}(P)$. ■

In the proof of Theorem 4.3.2, finiteness of the set of derivatives is important. There we construct for a given process P a finite set of fragments FG using $\mathit{derivatives}(P)$. An application of Proposition 4.2.2 then shows that every fragment reachable from P is structurally congruent with a fragment in FG.

Lemma 4.2.5
The set $\mathit{derivatives}(P)$ is finite for all $P \in \mathcal{P}$.

Proof
An induction on the structure of processes shows that $der(P)$ is finite. Since every process relies on finitely many defining equations, finiteness of $derivatives(P)$ follows immediately. ■

4.3 A First Characterisation of Structural Stationarity

In this section, we provide the characterisation of structural stationarity mentioned above: structural stationarity is equivalent to boundedness of all reachable fragments in the number of sequential processes. As the name indicates, we restrict the use of the parallel operator to compose sequential processes. In Chapter 7, we prove a second characterisation of structural stationarity, which restricts the use of the operator ν instead. While this characterisation gives a handle to establish structural stationarity, the second characterisation explains which processes fail to be structurally stationary.

Definition 4.3.1 ($\mathcal{P}_{\mathcal{S}<\infty}$)
A process $P \in \mathcal{P}$ is *bounded in the sequential processes*, if there is a bound on the number of sequential processes in all reachable fragments, i.e.,

$$\exists k_{\mathcal{S}} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf(Q)) : \|F\|_{\mathcal{S}} \leq k_{\mathcal{S}}.$$

The *set of all processes that are bounded in the sequential processes* is $\mathcal{P}_{\mathcal{S}<\infty}$. ♦

We state the characterisation via boundedness of $\| - \|_{\mathcal{S}}$ in Theorem 4.3.2. Several well-known subclasses of π-Calculus are immediately shown to be structurally stationary with an application of this result. Furthermore, the proofs of Theorem 4.4.8 and Theorem 7.2.8 in this thesis underline its importance.

Theorem 4.3.2 (Characterisation of Structural Stationarity via $|$)
$\mathcal{P}_{FG<\infty} = \mathcal{P}_{\mathcal{S}<\infty}$.

Boundedness follows immediately from structural stationarity. To establish completeness, i.e., to show structural stationarity from boundedness in $\| - \|_{\mathcal{S}}$, we construct a finite set of fragments FG, which includes up to structural congruence every reachable fragment. More precisely, we show that for every $Q \in Reach(P)$ and every fragment $F \in fg\,(rf(Q))$ there is a fragment $G \in FG$ with $G \equiv F$.

We first explain the idea underlying the construction of the fragments in FG and then turn to the technicalities. The set FG is the union of sets FG_i that contain fragments with i sequential processes. To build the fragments in FG_i, we

consider processes $\nu\tilde{a}.(\Pi_{j=1}^{i} Q_j\sigma'_j)$ of the form in Proposition 4.2.2, i.e., the Q_j are derivatives of P and the σ'_j are substitutions mapping $fn(Q_j)$ into $fn(P) \cup \tilde{a}$. We rename the names $\tilde{a}$ to a bounded set of unique names $\tilde{u}_i$, parameterised by the number of processes i. This ensures we only need to consider finitely many substitutions σ_j for every derivative Q_j. We add the restricted form $rf(\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j\sigma_j))$ to FG_i, if it is a fragment.

Let $k_{\mathcal{S}} \in \mathbb{N}$ be a bound on the number of sequential processes in all reachable fragments. Lemma 4.2.5 gives the finiteness of $derivatives(P)$. Thus the maximum $maxFN := max\{|fn(Q)| \mid Q \in derivatives(P)\}$ on the number of free names in derivatives exists. Let $\tilde{u}_i := u_1, \ldots, u_{i \cdot maxFN}$ be unique names distinct from the free names in P. For every $i \in \mathbb{N}$, we define

$$FG_i := \big\{ rf(\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j\sigma_j)) \mid Q_j \in derivatives(P),\ \sigma_j : fn(Q_j) \to fn(P) \cup \tilde{u}_i, \text{ and } rf(\nu\tilde{u}_i.(\Pi_{j=1}^{i} Q_j\sigma_j)) \text{ is a fragment} \big\}.$$

The set FG is the union of all sets FG_i with $i \leq k_{\mathcal{S}}$. This means, the fragments in FG have at most $k_{\mathcal{S}}$ sequential processes:

$$FG := FG_1 \cup \ldots \cup FG_{k_{\mathcal{S}}}.$$

With Proposition 4.2.2, it is easy to show that every reachable fragment F is structurally congruent with a fragment $rf(\nu\tilde{u}_{|I|}.(\Pi_{i \in I} Q_i\sigma_i))$ in $FG_{|I|}$. To prove the theorem, the inclusion $FG_{|I|} \subseteq FG$ remains to be shown. This follows with $|I| = \|F\|_{\mathcal{S}} \leq k_{\mathcal{S}}$, which holds with the invariance of $\| - \|_{\mathcal{S}}$ under structural congruence, Lemma 2.1.24, and the assumption. We explain the construction of FG on an example and then give the full proof that we just sketched.

Example 4.3.3 (FG)
Consider $P = \nu b.\overline{a}\langle b\rangle.b(x) \mid a(y).K\lfloor a, y\rfloor$ with $K(a, y) := \overline{y}\langle a\rangle$. In Example 4.2.3, we computed the reachable fragments:

$$\nu b.\overline{a}\langle b\rangle.b(x), \quad a(y).K\lfloor a, y\rfloor, \quad \nu b.(b(x) \mid K\lfloor a, b\rfloor), \quad \nu b.(b(x) \mid \overline{b}\langle a\rangle).$$

The number of sequential processes in all reachable fragments is bounded by $k_{\mathcal{S}} = 2$, which equals, e.g. $\|\nu b.(b(x) \mid K\lfloor a, b\rfloor)\|_{\mathcal{S}}$. The set FG is therefore defined by $FG = FG_1 \cup FG_2$. The set of derivatives is

$$derivatives(P) = \{\overline{a}\langle b\rangle.b(x), b(x), a(y).K\lfloor a, y\rfloor, K\lfloor a, y\rfloor, \overline{y}\langle a\rangle\}.$$

The maximal number of free names in derivatives is $maxFN = 2$, e.g. given by $|fn(\overline{a}\langle b\rangle.b(x))|$. Thus, FG_1 and FG_2 contain fragments

$$rf(\nu u_1, u_2.(Q\sigma)) \text{ and } rf(\nu u_1, \ldots, u_4.(Q_1\sigma_1 \mid Q_2\sigma_2)),$$

where $Q \in \mathit{derivatives}(P)$ with $\sigma : \mathit{fn}(Q) \to \{u_1, u_2, a\}$ and $Q_j \in \mathit{derivatives}(P)$ with $\sigma_j : \mathit{fn}(Q_j) \to \{u_1, \ldots, u_4, a\}$, for $j = 1, 2$. As an example, consider process $Q = \overline{a}\langle b\rangle.b(x) \in \mathit{derivatives}(P)$. Applying all substitutions $\sigma : \{a, b\} \to \{u_1, u_2, a\}$ yields the following fragments in FG_1, where structurally congruent ones are omitted:

$$\begin{aligned}
\overline{a}\langle b\rangle.b(x)\{a, a/a, b\} &= \overline{a}\langle a\rangle.a(x)\\
\nu u_1.((\overline{a}\langle b\rangle.b(x))\{u_1, u_1/a, b\}) &= \nu u_1.\overline{u_1}\langle u_1\rangle.u_1(x)\\
\nu u_1.((\overline{a}\langle b\rangle.b(x))\{a, u_1/a, b\}) &= \nu u_1.\overline{a}\langle u_1\rangle.u_1(x)\\
\nu u_1.((\overline{a}\langle b\rangle.b(x))\{u_1, a/a, b\}) &= \nu u_1.\overline{u_1}\langle a\rangle.a(x)\\
\nu u_1, u_2.((\overline{a}\langle b\rangle.b(x))\{u_1, u_2/a, b\}) &= \nu u_1, u_2.\overline{u_1}\langle u_2\rangle.u_2(x).
\end{aligned}$$

The reachable fragment $\nu b.\overline{a}\langle b\rangle.b(x)$ is structurally congruent with the element $\nu u_1.\overline{a}\langle u_1\rangle.u_1(x) \in FG_1 \subseteq FG$. ♦

Proof (of Theorem 4.3.2)
$\Leftarrow$ Consider $P \in \mathcal{P}_{\mathcal{S}<\infty}$, where $k_{\mathcal{S}} \in \mathbb{N}$ is a bound on the number of sequential processes in all reachable fragments. We show structural stationarity, $P \in \mathcal{P}_{FG<\infty}$.

To begin with, we argue that FG_j is finite for every $j \in \mathbb{N}$. The finiteness of FG follows immediately. There are finitely many derivatives $P' \in \mathit{derivatives}(P)$ by Lemma 4.2.5. Every process has finitely many free names, thus $\mathit{fn}(P')$ and $\mathit{fn}(P)$ are finite. The set $\tilde{u}_j = u_1, \ldots, u_{j \cdot \mathit{maxFN}}$ also is finite. This shows there are finitely many mappings $\sigma : \mathit{fn}(P') \to \mathit{fn}(P) \cup \tilde{u}_j$ for every P'. We conclude that FG_j is finite. It remains to be shown that up to structural congruence every reachable fragment is included in FG.

Consider $Q \in \mathit{Reach}(P)$ and $F \in \mathit{fg}\,(\mathit{rf}(Q))$. With Proposition 4.2.2, fragment F is structurally congruent with a process $\nu\tilde{a}.Q^{\neq\nu}$ so that $Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i$ with $Q_i \in \mathit{derivatives}(P)$ and $\sigma_i : \mathit{fn}(Q_i) \to \mathit{fn}(P) \cup \tilde{a}$. We show that

(1) F is structurally congruent with a fragment in $FG_{|I|}$ and

(2) $|I| \leq k_{\mathcal{S}}$.

This proves the theorem as it shows that F is structurally congruent with a fragment in FG. Since $\nu\tilde{a}.Q^{\neq\nu}$ is in standard form, we have $\tilde{a} \subseteq \mathit{fn}(Q^{\neq\nu})$. We need that the number of elements in $\tilde{a}$ is bounded by $|I| \cdot \mathit{maxFN}$:

$$\begin{aligned}
& & |\tilde{a}|\\
(\ \tilde{a} \subseteq \mathit{fn}(Q^{\neq\nu})\ ) & \leq & |\mathit{fn}(Q^{\neq\nu})|\\
(\ \text{Def. } \mathit{fn},\ Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i\ ) & = & |\textstyle\bigcup_{i\in I} \mathit{fn}(Q_i\sigma_i)|\\
& \leq & \Sigma_{i\in I} |\mathit{fn}(Q_i\sigma_i)|\\
(\ \sigma_i \text{ may identify names}\ ) & \leq & \Sigma_{i\in I} |\mathit{fn}(Q_i)|
\end{aligned}$$

$$\begin{aligned} (\text{ Def. } maxFN\ ) &\leq \Sigma_{i\in I} maxFN \\ &= |I| \cdot maxFN. \end{aligned}$$

With α-conversion we rename the names in $\tilde{a}$ to names in $\tilde{u}_{|I|}$:

$$\begin{aligned} &\nu\tilde{a}.(\Pi_{i\in I} Q_i\sigma_i) \\ (\ \tilde{a} = a_1,\ldots,a_l\ ) &\equiv \nu u_1,\ldots,u_l.(\Pi_{i\in I} Q_i\sigma_i\{u_l/a_l\}\ldots\{u_1/a_1\}). \end{aligned}$$

Define σ_i' by $a\sigma_i' := a\sigma_i\{u_l/a_l\}\ldots\{u_1/a_1\}$. Since $|\tilde{a}| \leq |I| \cdot maxFN$, we add the missing names in $\tilde{u}_{|I|}$, exploiting the congruence $\nu a.P \equiv P$ if $a \notin fn(P)$:

$$\begin{aligned} (\text{ Def. } \sigma'\ ) &= \nu u_1,\ldots,u_l.(\Pi_{i\in I} Q_i\sigma_i') \\ (\text{ Add missing names }) &\equiv \nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i\sigma_i'). \end{aligned}$$

We now have $F \equiv \nu\tilde{a}.Q^{\neq\nu} \equiv \nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i\sigma_i')$. With Lemma 3.2.7 and Proposition 3.2.10 it follows that

$$F = rf(F) \equiv_{rf}\ rf(\nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i\sigma_i')).$$

As F is a fragment, restricted equivalence implies $rf(\nu\tilde{u}_{|I|}.(\Pi_{i\in I} Q_i\sigma_i'))$ is a fragment and thus in $FG_{|I|}$. With $|I| \leq k_{\mathcal{S}}$ the inclusion $FG_{|I|} \subseteq FG$ holds:

$$\begin{aligned} & & & k_{\mathcal{S}} \\ (\text{ Assumption } F \in \mathcal{P}_{\mathcal{S}<\infty}\ ) & & \geq & \|F\|_{\mathcal{S}} \\ (\ \|-\|_{\mathcal{S}} \text{ invariant under } \equiv\ ) & & = & \|\nu\tilde{a}.Q^{\neq\nu}\|_{\mathcal{S}} \\ (\text{ Def. } \|-\|_{\mathcal{S}},\ Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i\ ) & & = & \Sigma_{i\in I}\|Q_i\sigma_i\|_{\mathcal{S}} \\ (\ \|-\|_{\mathcal{S}} \text{ invariant under } \sigma\ ) & & = & \Sigma_{i\in I}\|Q_i\|_{\mathcal{S}} \\ (\ Q_i \in derivatives(P)\ ) & & = & |I|. \end{aligned}$$

$\Rightarrow$ Conversely, if P is structurally stationary, all reachable processes are made up of finitely many fragments $F_1,\ldots,F_n$. Thus, the number of sequential processes in all reachable fragments is bounded by

$$max\{\|F_i\|_{\mathcal{S}}\ \mid\ 1 \leq i \leq n\}.$$

This concludes the proof. ■

Although the characterisation of structural stationarity in Theorem 4.3.2 is semantical in the sense that it refers to all reachable fragments, it has important implications. The first and unconventional application of the theorem is an algorithm to compute the structural semantics without using the coverability graph. The idea is to compute a Petri net $\mathcal{N}_{FG}[\![P]\!]$, which has the set FG in the proof of Theorem 4.3.2 as places. Transitions, arcs, and the initial marking

are added according to Definition 3.3.7. Since FG contains all reachable fragments, the Petri net $\mathcal{N}_{FG}[\![P]\!]$ subsumes $\mathcal{N}[\![P]\!]$. Like for $\mathcal{N}_{NoCov}[\![P]\!]$ in Section 3.5, the transition systems of $\mathcal{N}_{FG}[\![P]\!]$ and $\mathcal{N}[\![P]\!]$ are isomorphic but $\mathcal{N}_{FG}[\![P]\!]$ may contain places which are not markable, i.e., which are not in $\mathcal{N}[\![P]\!]$. We plan a prototypical implementation of $\mathcal{N}_{FG}$ in our tool PETRUCHIO. The crucial issue in the implementation is to limit the size of the set FG. Static analysis techniques like [BDNN98] may be helpful to solve this problem. We defer the discussion of the work of Bodei et. al. until Section 4.6.

As second application, Theorem 4.3.2 yields structural stationarity of important classes of processes known from the literature, starting with the syntactic class of *restriction-free processes* [AM02]. Although Lemma 4.3.4 follows from our work on finite handler processes in the next section, we give a direct proof here to illustrate the application of Theorem 4.3.2.

Lemma 4.3.4 (Restriction-free Processes are Structurally Stationary)
If $P \in \mathcal{P}$ is restriction-free, then $P \in \mathcal{P}_{FG<\infty}$.

Proof
If process $P \in \mathcal{P}$ is defined without the restriction operator (cf. Definition 2.1.3), all reachable fragments are sequential processes. Thus, $\|F\|_{\mathcal{S}} \leq 1$ and therefore $P \in \mathcal{P}_{\mathcal{S}<\infty} = \mathcal{P}_{FG<\infty}$ with Theorem 4.3.2. ■

Also the syntactic class of *finite control processes* (FCPs) [Dam96] is a subclass of $\mathcal{P}_{FG<\infty}$. Consider the FCP $\nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$, where the processes P_i do not use the parallel composition operator (cf. Definition 2.1.5). It is immediate to show that the number of sequential processes in all reachable fragments is bounded by n, the degree of parallelism in the initial process. Finite control processes are generalised by the semantic class of *finitary processes* defined by Montanari and Pistore. We establish the stronger result that all finitary processes are structurally stationary.

Definition 4.3.5 (Finitary Processes [MP95a, Pis99, MP01])
A process $P \in \mathcal{P}$ is *finitary*, if there is a bound on the number of sequential processes in every reachable process, i.e.,

$$\exists k_{\mathcal{S}} : \forall Q \in \mathit{Reach}(P) : \|Q\|_{\mathcal{S}} \leq k_{\mathcal{S}}.$$

♦

Lemma 4.3.6 (Finitary Processes are Structurally Stationary)
If $P \in \mathcal{P}$ is a finitary process then $P \in \mathcal{P}_{FG<\infty}$.

Proof
Consider the finitary process $P \in \mathcal{P}$, where $k_{\mathcal{S}}$ bounds the number of sequential processes in all reachable processes Q. We show that $k_{\mathcal{S}}$ bounds the number of sequential processes in all reachable fragments as well. Consider $F \in fg\,(rf(Q))$, by definition $rf(Q)$ is a parallel composition $rf(Q) = \Pi_{i\in I} F_i \mid F \mid \Pi_{j\in J} F_j$, where both index sets may be empty. We check that

$$\|F\|_{\mathcal{S}} \leq \|rf(Q)\|_{\mathcal{S}} = \|Q\|_{\mathcal{S}} \leq k_{\mathcal{S}}$$

by definition of $\|-\|_{\mathcal{S}}$, invariance of $\|-\|_{\mathcal{S}}$ under structural congruence, and the assumption that P is finitary. Thus, the inclusion $P \in \mathcal{P}_{\mathcal{S}<\infty} = \mathcal{P}_{FG<\infty}$ holds with Theorem 4.3.2. ■

Remark 4.3.7
The structural semantics is—to the best of our knowledge—the first automata-theoretic translation that finitely represents both, the class of restriction-free processes and the class of finitary processes. This is even more surprising, as the proofs of Lemma 4.3.4 and Lemma 4.3.6 are simple and straightforward. ♦

Example 4.3.8 (Structurally Stationary Processes)
We present three examples that separate the process classes.

1. Consider $K_1\lfloor a \rfloor$ with $K_1(x) := K_1\lfloor x \rfloor \mid K_1\lfloor x \rfloor$. The only reaction sequence of the process is

   $$K_1\lfloor a \rfloor \to K_1\lfloor a \rfloor \mid K_1\lfloor a \rfloor \to \Pi^3 K_1\lfloor a \rfloor \to \ldots$$

   Clearly, the process is restriction-free but not finitary as the number of sequential processes grows unboundedly. As every fragment consists of a single process $K_1\lfloor a \rfloor$, the process is structurally stationary.

2. The process $\nu a.K_2\lfloor a \rfloor$ with $K_2(x) := \nu b.K_2\lfloor b \rfloor$ is not restriction-free but finitary as each of the following processes consists of a single sequential process:

   $$\nu a.K_2\lfloor a \rfloor \to \nu b.K_2\lfloor b \rfloor \to \nu c.K_2\lfloor c \rfloor \to \ldots$$

   The number of sequential processes in fragments is bounded by one as well, so $\nu a.K_2\lfloor a \rfloor$ is structurally stationary.

3. Consider $\nu a.K_3\lfloor a \rfloor$ with $K_3(x) := K_3\lfloor x \rfloor \mid \nu b.K_3\lfloor b \rfloor$ and the reaction sequence

   $$\nu a.K_3\lfloor a \rfloor \to \nu a.K_3\lfloor a \rfloor \mid \nu b.K_3\lfloor b \rfloor \to \nu a.K_3\lfloor a \rfloor \mid \nu b.K_3\lfloor b \rfloor \mid \nu c.K_3\lfloor c \rfloor \to \ldots$$

   Of course, $\nu a.K_3\lfloor a \rfloor$ is neither restriction-free nor finitary. Again, the number of sequential processes in fragments is bounded by one, i.e., the process is structurally stationary with Theorem 4.3.2.

This illustrates the differences between restriction-free and finitary processes, and shows that they are generalised by structurally stationary processes. Processes at the borderline of structural stationarity are subject to Chapter 7, 8, and 9. ♦

The present section identified the properties that lead to finiteness and infinity of the structural semantics. In the following Section 4.4, we apply these insights to design a syntactic class of processes for modelling client-server architectures. The main theorem states that all processes in the new class are structurally stationary, i.e., finitely represented under the structural semantics. Again, Theorem 4.3.2 is a helpful tool in the proof.

4.4 Finite Handler Processes

We start with a sketch of the idea underlying our syntactic class of *finite handler processes*, the explanation is illustrated in Figure 4.1. Afterwards, we turn to the definition. A *handler* process listens on a set of channels represented by *distinguished public names*. The analogy is a server located at an IP address listening on a set of ports, part (a) in the figure. To register at a handler, *participant* processes send a restricted name over one of the distinguished channels. This mimics a client that passes its own IP address when contacting a server, (b) and (c). The handler introduces the participants to each other and by this creates fragments, Figure 4.1 (d). No protocol is specified for finite handler processes. The crucial restriction in the communication is that handler processes receive finitely many messages from participants per session. In particular, only finitely many participants can register at a handler. At some point, the handler loses connection to the fragment and restarts listening on the distinguished channels, Figure 4.1 (e). This ensures boundedness of the fragments in the number of processes.

The set of *distinguished public names* is $\mathcal{N}_\mathcal{P} \subseteq \mathcal{N}$. We use the letter p for names in $\mathcal{N}_\mathcal{P}$, whereas a, b, x, y refer to names in $\mathcal{N} \setminus \mathcal{N}_\mathcal{P}$. To syntactically detect the use of distinguished public names we define *receiving prefixes* π^R. They differ from prefixes π in the ability to receive on distinguished names, $\pi^R = p(x)$. An arbitrary prefix π^{RS} also allows for *sending* on distinguished names, $\pi^{RS} = \overline{p}\langle x\rangle$:

$$\pi ::= \overline{x}\langle y\rangle \mid x(y) \mid \tau \qquad \pi^R ::= \pi \mid p(x) \qquad \pi^{RS} := \pi^R \mid \overline{p}\langle x\rangle.$$

The formalisation of *participants* requires some explanation. A participant is a process of the form

$$\nu\tilde{a}.M^{PT} \qquad \text{with} \qquad M^{PT} = \overline{p_1}\langle a_1\rangle.SQ_1 + \ldots + \overline{p_n}\langle a_n\rangle.SQ_n,$$

which satisfies $fn(\nu\tilde{a}.M^{PT}) \subseteq \mathcal{N}_\mathcal{P}$. To register at a handler process, a participant sends a name $a_i \in \mathcal{N} \setminus \mathcal{N}_\mathcal{P}$ over a distinguished channel $p_i \in \mathcal{N}_\mathcal{P}$ using the choice

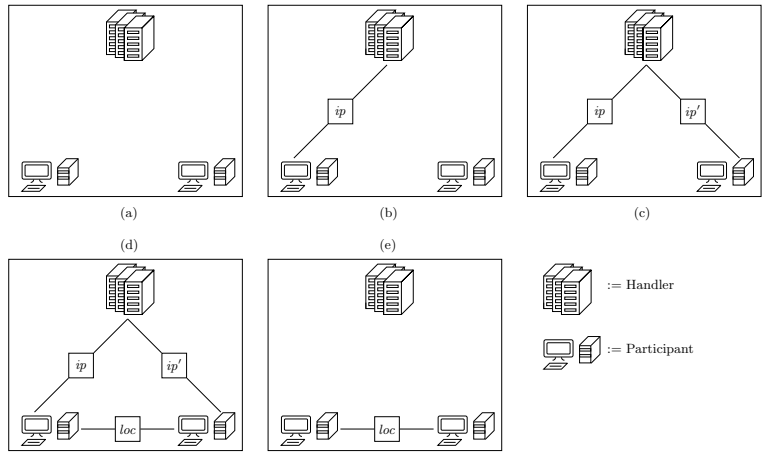


Figure 4.1: Illustration of the behaviour of finite handler processes.

composition M^{PT}. The side condition that all free names are in $\mathcal{N}_\mathcal{P}$ ensures a_i is restricted, i.e., $a_i \in \tilde{a}$. After the registration, the participant becomes a process SQ, which is sequential in the sense that it does not contain the parallel composition operator. Hence, communications with participants do not increase the number of sequential processes within fragments. Furthermore, SQ processes do not use distinguished channels. So a participant registers at precisely one handler process.

Definition 4.4.1 (Participants)
To define participants, we first require processes SQ that do not contain the parallel composition operator and do not use distinguished names in $\mathcal{N}_\mathcal{P}$:

$$\begin{array}{rcl} M^{SQ} & ::= & \mathbf{0} \mid \pi.SQ \mid M_1^{SQ} + M_2^{SQ} \\ SQ & ::= & M^{SQ} \mid K^{SQ}\lfloor \tilde{a} \cup \tilde{p} \rfloor \mid K^{PT}\lfloor \tilde{p} \rfloor \mid \nu a.SQ. \end{array}$$

An identifier K^{SQ} is defined by an SQ process and an identifiers K^{PT} by a process PT below. For $K^{SQ}\lfloor \tilde{a} \cup \tilde{p} \rfloor$ with $K^{SQ}(\tilde{x}_a \cup \tilde{x}_p) := SQ$ we additionally require that for all $x \in fn(SQ)$ we have $x \in \tilde{x}_p$ if and only if $x \in \mathcal{N}_\mathcal{P}$.

The *set of participants* $\mathcal{P}_{\mathcal{PT}}$ with typical elements PT is defined by

$$\begin{array}{rcl} M^{PT} & ::= & \mathbf{0} \mid \overline{p}\langle a \rangle.SQ \mid M_1^{PT} + M_2^{PT} \\ PT & ::= & \nu\tilde{a}.M^{PT} \mid K^{PT}\lfloor \tilde{p} \rfloor \mid PT_1 \mid PT_2, \end{array}$$

where $fn(\nu\tilde{a}.M^{PT}) \subseteq \mathcal{N}_\mathcal{P}$. ♦

The side condition on the defining equation of $K^{SQ}\lfloor \tilde{a} \cup \tilde{p} \rfloor$ ensures that the parameters $\tilde{p}$ serve as parameters of further calls $K^{SQ}\lfloor \tilde{a} \cup \tilde{p} \rfloor$ or calls $K^{PT}\lfloor \tilde{p} \rfloor$ but are not used anywhere else.

Example 4.4.2 (Participant)
The client in Figure 4.1 may be formalised by $C\lfloor url \rfloor$ with

$$C(url) \quad := \quad \nu ip.\overline{url}\langle ip \rangle.ip(loc).T\lfloor loc, url \rfloor.$$

The identifier T stands for *talk* and is assumed to be defined by an SQ process.

If we choose $\mathcal{N}_{\mathcal{P}} = \{url\}$ as distinguished names, the process $C\lfloor url \rfloor$ is a participant in $\mathcal{P}_{\mathcal{PT}}$. It corresponds to $K^{PT}\lfloor \tilde{p} \rfloor$ in Definition 4.4.1. To see this, we check that the defining process is in $\mathcal{P}_{\mathcal{PT}}$. The only free name is url. The send action $\overline{url}\langle ip \rangle$ is the registration at the handler, denoted by $M^{PT} = M_1^{PT} + \overline{p}\langle a \rangle.SQ + M_2^{PT}$ in Definition 4.4.1. Then the free agent becomes an SQ process as it avoids the parallel composition. ♦

A *handler* process is a parallel composition of *connector processes* CN. To enable participants to register, CN contains receiving prefixes $\pi^R = p(x)$. Following the explanation above, the connector should receive messages from finitely many participants. Therefore, we exclude recursive calls between CN processes, i.e., there is no process identifier K^{CN} defined by a CN process (cf. process identifiers K^{SQ} in Definition 4.4.1). To make sure a CN process communicates with registered participants only, we add the side condition that the free names in CN are in $\mathcal{N}_{\mathcal{P}}$. This implies that all names—except those for registration purposes—are bound.

Definition 4.4.3 (Handler)
The *set of handler processes* $\mathcal{P}_{\mathcal{HD}}$ with $HD \in \mathcal{P}_{\mathcal{HD}}$ is defined inductively:

$$\begin{array}{rcl} M^{CN} & ::= & \mathbf{0} \mid \pi^R.CN \mid M_1^{CN} + M_2^{CN} \\ CN & ::= & M^{CN} \mid K^{HD}\lfloor \tilde{p} \rfloor \mid \nu a.CN \\ HD & ::= & CN^{\mathcal{N}_{\mathcal{P}}} \mid K^{HD}\lfloor \tilde{p} \rfloor \mid HD_1 \mid HD_2, \end{array}$$

where $CN^{\mathcal{N}_{\mathcal{P}}}$ is a connector process CN that satisfies $fn(CN^{\mathcal{N}_{\mathcal{P}}}) \subseteq \mathcal{N}_{\mathcal{P}}$. A process identifier K^{HD} is defined by a handler process $HD \in \mathcal{P}_{\mathcal{HD}}$. ♦

Example 4.4.4 (Handler)
The server in the explanation above is $S\lfloor url \rfloor$ with

$$S\lfloor url \rfloor \quad := \quad url(x).url(y).\nu loc.\overline{x}\langle loc \rangle.\overline{y}\langle loc \rangle.S\lfloor url \rfloor.$$

Let again $\mathcal{N}_{\mathcal{P}} = \{url\}$. To see that $S\lfloor url \rfloor$ is a handler process of the form $K^{HD}\lfloor \tilde{p} \rfloor$, consider the process defining S. It is a CN process that receives twice

on the distinguished name *url*. All other names are bound, i.e., the side condition that the free names are in $\mathcal{N}_\mathcal{P}$ holds. Hence, $S\lfloor url \rfloor$ is a process in $\mathcal{P}_{\mathcal{HD}}$. ♦

Finite handler processes, denoted by FH, are built from participants and handler processes using choice composition, $M_1^{FH} + M_2^{FH}$, and parallel composition, $FH_1 \mid FH_2$. Restricted names occur in participants and handler processes only. Therefore, their use is well-controlled as a participant sends restricted names to a handler who distributes them among the other participants that are registered.

Definition 4.4.5 (Finite Handler Process)
The *set of finite handler processes* $\mathcal{P}_{\mathcal{FH}}$ with elements $FH \in \mathcal{P}_{\mathcal{FH}}$ is defined by

$$\begin{array}{rcl} M^{FH} & ::= & \mathbf{0} \mid \pi.FH \mid M_1^{FH} + M_2^{FH} \\ FH & ::= & PT \mid HD \mid M^{FH} \mid K^{FH}\lfloor \tilde{a} \cup \tilde{p} \rfloor \mid FH_1 \mid FH_2, \end{array}$$

where $PT \in \mathcal{P}_{\mathcal{PT}}$, $HD \in \mathcal{P}_{\mathcal{HD}}$, and K^{FH} is defined by $K^{FH}(\tilde{x}_a, \tilde{x}_p) := FH$. Again, we require for $x \in fn(FH)$ that $x \in \tilde{x}_p$ if and only if $x \in \mathcal{N}_\mathcal{P}$. ♦

Example 4.4.6 concludes the explanation that the client-server system is a finite handler process. The case study shows that finite handler processes may have an infinite number of reachable processes. Furthermore, there is no bound on the number of restricted names.

Example 4.4.6 (Finite Handler Process)
Let $\mathcal{N}_\mathcal{P} = \{url\}$. We already observed that $C\lfloor url \rfloor \in \mathcal{P}_{\mathcal{PT}}$ and $S\lfloor url \rfloor \in \mathcal{P}_{\mathcal{HD}}$ holds. Let the environment process $ENV\lfloor url \rfloor$ generate clients, i.e., we define $ENV(url) := ENV\lfloor url \rfloor \mid C\lfloor url \rfloor$. The process is of the form $K^{FH}\lfloor \tilde{a} \cup \tilde{p} \rfloor \in \mathcal{P}_{\mathcal{FH}}$, so we conclude that $ENV\lfloor url \rfloor \mid S\lfloor url \rfloor$ is a finite handler process in $\mathcal{P}_{\mathcal{FH}}$. ♦

Although the class of finite handler processes may seem tailored towards client-server applications, the syntactic restrictions are not severe. Finite handler processes truly generalise the class of restriction-free processes (cf. Definition 2.1.3). These correspond to finite handler processes without handlers and participants.

Lemma 4.4.7 (Restriction-free processes are Finite Handler Processes)
If $P \in \mathcal{P}$ is restriction-free then $P \in \mathcal{P}_{\mathcal{FH}}$.

We now turn to our main result that finite handler processes are finitely represented under the structural semantics.

Theorem 4.4.8 (Finite Handler Processes are Structurally Stationary)
$\mathcal{P}_{\mathcal{FH}} \subseteq \mathcal{P}_{FG<\infty}$.

To establish the theorem, we show that all fragments reachable from a finite handler process consist of a bounded number of sequential processes. The maximal nesting of prefixes that receive on distinguished channels $\|P\|_{\mathcal{N}_\mathcal{P}}$ serves as the bound. For example, the function yields $\|p(x).a(y).p(z) + \overline{p}\langle a\rangle\|_{\mathcal{N}_\mathcal{P}} = max\{2,0\} = 2$.

Definition 4.4.9 ($\|-\|_{\mathcal{N}_\mathcal{P}} : \mathcal{P} \to \mathbb{N}$)
For every process $P \in \mathcal{P}$, the *maximal nesting of prefixes that receive on distinguished channels*, i.e., prefixes $p(x)$, is given by $\|P\|_{\mathcal{N}_\mathcal{P}}$:

$$\|\pi^{RS}\|_{\mathcal{N}_\mathcal{P}} := \begin{cases} 1, & \text{if } \pi^{RS} = p(x) \\ 0, & \text{otherwise} \end{cases} \qquad \|K\lfloor\tilde{a}\rfloor\|_{\mathcal{N}_\mathcal{P}} := 0$$

$$\|\pi^{RS}.P\|_{\mathcal{N}_\mathcal{P}} := \|\pi^{RS}\|_{\mathcal{N}_\mathcal{P}} + \|P\|_{\mathcal{N}_\mathcal{P}} \qquad \|M + N\|_{\mathcal{N}_\mathcal{P}} := max\{\|M\|_{\mathcal{N}_\mathcal{P}}, \|N\|_{\mathcal{N}_\mathcal{P}}\}$$

$$\|P \mid Q\|_{\mathcal{N}_\mathcal{P}} := max\{\|P\|_{\mathcal{N}_\mathcal{P}}, \|Q\|_{\mathcal{N}_\mathcal{P}}\} \qquad \|\nu a.P\|_{\mathcal{N}_\mathcal{P}} := \|P\|_{\mathcal{N}_\mathcal{P}}.$$

Let P rely on the n defining equations $K_i(\tilde{x}_i) := P_i$. Taking them into account, we define $\|P\|_{max_{\mathcal{N}_\mathcal{P}}} := max\{\|P\|_{\mathcal{N}_\mathcal{P}}, \|P_1\|_{\mathcal{N}_\mathcal{P}}, \ldots, \|P_n\|_{\mathcal{N}_\mathcal{P}}\}$. ♦

To prove the indicated boundedness requires a deeper understanding of the behaviour of finite handler processes. An induction on the reaction sequences reveals that every fragment reachable from $FH \in \mathcal{P}_{\mathcal{FH}}$ satisfies the constraints given by Definition 4.4.10. It is a process identifier, a participant $\nu\tilde{a}.M^{PT}$ ready to register at a connector, or a finite handler process M^{FH} communicating on public channels. In any other case, the fragment is a parallel composition of SQ processes with at most one CN process, $F \equiv \nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$. Moreover, the number of sequential processes in this fragment satisfies $\|F\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \leq \|FH\|_{max_{\mathcal{N}_\mathcal{P}}} + 1$. The inequality is crucial in the proof of Theorem 4.4.8, we explain it after the definition.

Definition 4.4.10 (Finite Handler Form)
A fragment $F \in \mathcal{P}_\mathcal{F}$ reachable from $FH \in \mathcal{P}_{\mathcal{FH}}$ is in *finite handler form (fhf)* if

(1) either F is structurally congruent with one of the following fragments:

$$K^{PT}\lfloor\tilde{p}\rfloor \;\mid\; K^{HD}\lfloor\tilde{p}\rfloor \;\mid\; K^{FH}\lfloor\tilde{a} \cup \tilde{p}\rfloor \;\mid\; M^{FH} \;\mid\; \nu\tilde{a}.M^{PT}$$

with $fn(\nu\tilde{a}.M^{PT}) \subseteq \mathcal{N}_\mathcal{P}$,

(2) or $F \equiv \nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$ in standard form where the parallel composition of either the SQ_i or CN may be missing, $fn(F) \subseteq \mathcal{N}_\mathcal{P}$, and

$$\|F\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \leq \|FH\|_{max_{\mathcal{N}_\mathcal{P}}} + 1.$$

Note that M^{PT} as well as M^{FH} are non-empty and that not both, the SQ_i and CN, may be omitted since F is a fragment. ♦

Consider fragment $F \equiv \nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$ created by connector CN. Since neither the connector CN nor registered participants SQ_i contain parallel compositions, $\|F\|_{\mathcal{S}}$ does not increase by internal communications. Registered participants and connector communicate on restricted names. The free names of the fragment are the input prefixes of CN, which are in $\mathcal{N}_{\mathcal{P}}$. As input prefixes do not match, the fragment does not merge with a fragment or a connector. Sequential processes M^{FH} do not use names in $\mathcal{N}_{\mathcal{P}}$. Hence, the only way to increase the number of processes in F is a communication between CN and a participant $\nu\tilde{a}.M^{PT}$.

The connector process receives finitely many participants it may add to F. Every such communication decreases $\|CN\|_{\mathcal{N}_{\mathcal{P}}} = \|F\|_{\mathcal{N}_{\mathcal{P}}}$ by one and increases $\|F\|_{\mathcal{S}}$ by at most one. Initially, $\|CN\|_{\mathcal{N}_{\mathcal{P}}} \leq \|FH\|_{max_{\mathcal{N}_{\mathcal{P}}}}$ and all fragments consist of one process, i.e., $\|F\|_{\mathcal{S}} = 1$. We conclude $\|F\|_{\mathcal{N}_{\mathcal{P}}} + \|F\|_{\mathcal{S}} \leq \|FH\|_{max_{\mathcal{N}_{\mathcal{P}}}} + 1$.

Lemma 4.4.11
If $FH \in \mathcal{P}_{\mathcal{FH}}$, then every reachable fragment is in finite handler form, i.e.,

$$\forall Q \in \mathit{Reach}(FH) : \forall F \in \mathit{fg}\,(\mathit{rf}(Q)) : F \text{ is in } \mathit{fhf}.$$

Before we continue with the proof of Lemma 4.4.11, we prove Theorem 4.4.8 as a corollary of the lemma and Theorem 4.3.2.

Proof (of Theorem 4.4.8)
Consider $FH \in \mathcal{P}_{\mathcal{FH}}$ and let F be a reachable fragment. With Lemma 4.4.11, we get the inequality $\|F\|_{\mathcal{S}} + \|F\|_{\mathcal{N}_{\mathcal{P}}} \leq \|FH\|_{max_{\mathcal{N}_{\mathcal{P}}}} + 1$. Thus, $\|F\|_{\mathcal{S}} \leq \|FH\|_{max_{\mathcal{N}_{\mathcal{P}}}} + 1$, which means FH is bounded in the sequential processes, $FH \in \mathcal{P}_{\mathcal{S}<\infty}$. Theorem 4.3.2 yields $FH \in \mathcal{P}_{FG<\infty}$. ■

To establish Lemma 4.4.11, we do an induction on the length of the reaction sequence leading to the reachable process Q. We have to show that the fragments in $\mathit{fg}\,(\mathit{rf}(Q))$ are in fhf. The following Lemma 4.4.12 handles the base case, i.e., $Q_0 = FH$. It is also used in the induction step when a process identifier is called or processes in $\mathcal{P}_{\mathcal{FH}}$ communicate.

Lemma 4.4.12
For every participant $PT \in \mathcal{P}_{\mathcal{PT}}$, handler $HD \in \mathcal{P}_{\mathcal{HD}}$, and finite handler process $FH \in \mathcal{P}_{\mathcal{FH}}$ the fragments of the restricted form are in fhf, i.e., every fragment $F \in \mathit{fg}\,(\mathit{rf}(PT))$, $F \in \mathit{fg}\,(\mathit{rf}(HD))$, and $F \in \mathit{fg}\,(\mathit{rf}(FH))$ is in fhf.

Proof
We establish the statement for $PT \in \mathcal{P}_{\mathcal{PT}}$, the proofs for $HD \in \mathcal{P}_{\mathcal{HD}}$ and $FH \in \mathcal{P}_{\mathcal{FH}}$ are similar.

Base Cases Consider $K^{PT}\lfloor\tilde{p}\rfloor$ and $\nu\tilde{a}.M^{PT}$ with $fn(\nu\tilde{a}.M^{PT}) \subseteq \mathcal{N}_{\mathcal{P}}$. The restricted form does not change process identifiers, i.e., $rf(K^{PT}\lfloor\tilde{p}\rfloor) = K^{PT}\lfloor\tilde{p}\rfloor$. Thus, $fg\left(rf(K^{PT}\lfloor\tilde{p}\rfloor)\right) = \{K^{PT}\lfloor\tilde{p}\rfloor\}$ which is a fragment in *fhf*. For an empty choice composition, the proof is trivial as $rf(\nu\tilde{a}.M^{PT}) = \mathbf{0}$ does not contain any fragments. If the choice is non-empty, the restricted form removes those names from $\tilde{a}$ that are not in $fn(M^{PT})$. We get $fg\left(rf(\nu\tilde{a}.M^{PT})\right) = \{\nu\tilde{a}'.M^{PT}\}$ with $\tilde{a}' \subseteq \tilde{a}$. As only superfluous restrictions are removed, the free names remain in $\mathcal{N}_{\mathcal{P}}$ and we have a fragment in *fhf*.

Induction Step Assume that all $F \in fg\left(rf(PT_i)\right)$ are in *fhf* with $i = 1, 2$. Without loss of generality let $rf(PT_i) \neq \mathbf{0}$. By definition, $rf(PT_1 \mid PT_2) = rf(PT_1) \mid rf(PT_2)$. Thus, $F \in fg\left(rf(PT_1 \mid PT_2)\right)$ if and only if $F \in fg\left(rf(PT_1)\right)$ or $F \in fg\left(rf(PT_2)\right)$. In both cases, the hypothesis shows that F is in *fhf*. ■

Lemma 4.4.13 handles the induction step in the proof of Lemma 4.4.11 in case a participant registers at a connector process or a fragment consisting of several processes performs a reaction.

Lemma 4.4.13
Consider $FH \in \mathcal{P}_{\mathcal{FH}}$ with $F \equiv \nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$ and $G \equiv \nu\tilde{a}_G.M^{PT}$ reachable fragments in *fhf*. Let $F \to R$ and $F \mid G \to R$. In either case, the restricted form of R is a parallel composition of fragments in *fhf*: $rf(R) = \Pi_{i\in I} F_i$ with F_i in *fhf* for all $i \in I$.

Proof
Statement for F Since $F \equiv \nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$, the reaction $F \to R$ implies $\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN) \to R$ with Rule (Struct). Without loss of generality, we assume that SQ_i as well as CN are present. According to Proposition 2.1.38, there are three possible reactions for $\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$. A process $SQ_1 = M_1 + \tau.SQ_1' + N_1$ or the process CN consumes a τ-prefix, a process $K^{SQ}\lfloor\tilde{a} \cup \tilde{p}\rfloor$ calls its defining equation, or two processes, say SQ_1 and SQ_2 or process SQ_1 and CN, communicate. We consider the first case where $SQ_1 = M_1 + \tau.SQ_1' + N_1$ consumes a τ-prefix. The remaining cases are similar but notationally less convenient as the require dealing with substitutions. With Proposition 2.1.38, we get

$$R \equiv \nu\tilde{a}_F.(SQ_1' \mid \Pi_{i=2}^{n} SQ_i \mid CN).$$

Since the free names in M_1 and N_1 are lost, we have $fn(SQ_1') \subseteq fn(SQ_1)$. Con-

sequently, the restricted form of $\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN)$ may consist of several fragments:

$$rf(\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN)) = \Pi_{i\in I} F_i.$$

Case $SQ'_1 = K^{PT}\lfloor\tilde{p}\rfloor$ Since $\tilde{a}_F \subseteq \mathcal{N} \setminus \mathcal{N}_\mathcal{P}$, we have $\tilde{p} \cap \tilde{a}_F = \emptyset$. When we now compute the restricted from of $\nu\tilde{a}_F.(K^{PT}\lfloor\tilde{p}\rfloor \mid \Pi_{i=2}^n SQ_i \mid CN)$, the process $K^{PT}\lfloor\tilde{p}\rfloor$ forms a fragment on its own, i.e., there is a fragment F_i in $\Pi_{i\in I} F_i$ with $F_i = SQ'_1 = K^{PT}\lfloor\tilde{p}\rfloor$. This fragment is obviously in *fhf*.

Case $SQ'_1 \neq K^{PT}\lfloor\tilde{p}\rfloor$ and remaining fragments Consider a fragment F_i in the restricted form $\Pi_{i\in I} F_i = rf(\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN))$. By definition of rf, the standard form of F_i is a parallel composition of processes SQ_j with at most one process CN,

$$F_i \equiv sf(F_i) = \nu\tilde{a}.(SQ_{j_1} \mid \ldots \mid SQ_{j_n} \mid CN).$$

Either the SQ_j or CN may be missing. We check the free names as follows:

$$fn(F_i) \subseteq fn(\Pi_{i\in I} F_i) = fn(R) \subseteq fn(F) \subseteq \mathcal{N}_\mathcal{P}.$$

The first inclusion holds by definition of fn, the following equation with $R \equiv \Pi_{i\in I} F_i$ and the invariance of fn under structural congruence, the next inclusion holds with Lemma 2.1.37, and the last is the assumption that F is in *fhf*. We prove the inequality to conclude F_i is in *fhf*. Note that $\|F_i\|_{\mathcal{N}_\mathcal{P}} = 0$ if CN is not in F_i and $\|F_i\|_{\mathcal{N}_\mathcal{P}} = \|F\|_{\mathcal{N}_\mathcal{P}}$ otherwise:

$$\begin{aligned}
& & \|F_i\|_\mathcal{S} + \|F_i\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Explained above }) & \leq & \|F_i\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \\
(\ \|F_i\|_\mathcal{S} \leq \|\Pi_{i\in I} F_i\|_\mathcal{S}\ ) & \leq & \|rf(\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN))\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Invariance } \|-\|_\mathcal{S} \text{ under } \equiv) & = & \|\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN)\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Case } SQ'_1 = \mathbf{0}) & \leq & \|\nu\tilde{a}_F.(SQ_1 \mid \Pi_{i=2}^n SQ_i \mid CN)\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Invariance } \|-\|_\mathcal{S} \text{ under } \equiv) & = & \|F\|_\mathcal{S} + \|F\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Assumption } F \text{ in } \textit{fhf}) & \leq & \|FH\|_{max_{\mathcal{N}_\mathcal{P}}} + 1.
\end{aligned}$$

We now have $R \equiv \nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN)$, which implies restricted equivalence with Proposition 3.2.10:

$$rf(R) \equiv_{rf} rf(\nu\tilde{a}_F.(SQ'_1 \mid \Pi_{i=2}^n SQ_i \mid CN)) = \Pi_{i\in I} F_i.$$

By definition of restricted equivalence, for every fragment G in $rf(R)$ there is a fragment F_i in $\Pi_{i\in I} F_i$ so that $G \equiv F_i$. We just showed that all F_i are in finite handler form, hence G is in *fhf* as well.

Statement for $F \mid G$ Since $F \mid G \to R$, with Rule (Struct) also the standard form of process $\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN) \mid \nu\tilde{a}_G.M^{PT}$ reacts to R:

$$\nu\tilde{a}_F.\nu\tilde{a}_G.(SQ_1 \mid \ldots \mid SQ_n \mid CN \mid M^{PT}) \to R.$$

We already handled reactions within F in the previous statement. Therefore, we now only consider a reaction between $M^{PT} = M_1^{PT} + \overline{p}\langle a\rangle.SQ + M_2^{PT}$ and the process $CN = M_1^{CN} + p(x).CN' + M_2^{CN}$. Proposition 2.1.38 yields

$$R \equiv \nu\tilde{a}_F.\nu\tilde{a}_G.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\} \mid SQ).$$

We compute the restricted form

$$rf(\nu\tilde{a}_F.\nu\tilde{a}_G.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\} \mid SQ)) = \Pi_{i\in I} F_i. \tag{4.1}$$

Like in the previous case, we check that

$$F_i \equiv sf(F_i) = \nu\tilde{a}.(SQ_{j_1} \mid \ldots \mid SQ_{j_n} \mid SQ \mid CN'\{a/x\}),$$

where we assume that the fragment contains SQ as well as $CN'\{a/x\}$. The remaining cases follow from this one. Note that we have to do case distinctions for $SQ = K^{PT}\lfloor\tilde{p}\rfloor$ and additionally for $CN' = K^{HD}\lfloor\tilde{p}\rfloor$, but both are similar to the previous case. We now check the free Names:

$$fn(F_i) \subseteq fn(R) \subseteq fn(F \mid G) = fn(F) \cup fn(G) \subseteq \mathcal{N}_\mathcal{P}.$$

The first inclusion holds by definition of *fn* and $R \equiv \Pi_{i\in I} F_i$, the second follows with $F \mid G \to R$ and Lemma 2.1.37, the next equation is again the definition of *fn*, and the final inclusion holds with the assumption that F and G are in *fhf*, and hence $fn(F) \subseteq \mathcal{N}_\mathcal{P} \supseteq fn(G)$. We still need to show the inequality:

$$\begin{aligned}
& & \|F_i\|_\mathcal{S} + \|F_i\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Def. } \|-\|_{\mathcal{N}_\mathcal{P}}) & = & \|F_i\|_\mathcal{S} + \|CN'\{a/x\}\|_{\mathcal{N}_\mathcal{P}} \\
(\text{ Def. } \|-\|_{\mathcal{N}_\mathcal{P}}) & \leq & \|F_i\|_\mathcal{S} + \|CN\|_{\mathcal{N}_\mathcal{P}} - 1 \\
(\text{ See below }) & \leq & \|\nu\tilde{a}_F.\nu\tilde{a}_G.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\} \mid SQ)\|_\mathcal{S} + \\
& & \|CN\|_{\mathcal{N}_\mathcal{P}} - 1 \\
(\text{ See below }) & \leq & \|F\|_\mathcal{S} + 1 + \|CN\|_{\mathcal{N}_\mathcal{P}} - 1 \\
(\text{ } F \text{ in } \mathit{fhf} \text{ }) & \leq & \|FH\|_{max_{\mathcal{N}_\mathcal{P}}} + 1.
\end{aligned}$$

The second inequality requires some consideration. Since F_i is a fragment in $\Pi_{i\in I} F_i$, we have $\|F_i\|_\mathcal{S} \leq \Sigma_{i\in I}\|F_i\|_\mathcal{S} = \|\Pi_{i\in I} F_i\|_\mathcal{S}$. With Equation (4.1), $rf(P) \equiv P$, and the invariance of $\|-\|_\mathcal{S}$ under structural congruence we get the stated inequality. For the third inequality we compute

$$\|\nu\tilde{a}_F.\nu\tilde{a}_G.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\} \mid SQ)\|_\mathcal{S}$$

$$\begin{aligned}
(\text{ Def. } \|-\|_{\mathcal{S}}) \quad &= \quad \|\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\} \mid SQ)\|_{\mathcal{S}} \\
(\text{ } SQ \text{ may be } \mathbf{0}) \quad &\leq \quad \|\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN'\{a/x\})\|_{\mathcal{S}} + 1 \\
(\text{ } CN'\{a/x\} \text{ may be } \mathbf{0}) \quad &\leq \quad \|\nu\tilde{a}_F.(SQ_1 \mid \ldots \mid SQ_n \mid CN)\|_{\mathcal{S}} + 1 \\
&\leq \quad \|F\|_{\mathcal{S}} + 1.
\end{aligned}$$

The last inequality holds with the invariance of $\|-\|_{\mathcal{S}}$ under structural congruence and concludes the proof. ■

In the proof of Lemma 4.4.11, the induction hypothesis assumes that all fragments in $fg\,(rf(Q_n))$ are in *fhf*. We distinguish all possible fragments in Definition 4.4.10 and all reactions $Q_n \to Q_{n+1}$. Lemma 4.4.12 and Lemma 4.4.13 show that the resulting fragments in $fg\,(rf(Q_{n+1}))$ are in *fhf*.

Proof (of Lemma 4.4.11)
Let $FH \in \mathcal{P}_{\mathcal{FH}}$. We proceed by induction on the length of the reaction sequences. For the base case $Q_0 = FH$, all fragments in $fg\,(rf(Q_0))$ are in *fhf* by Lemma 4.4.12.

Induction Step Assume that all fragments in $fg\,(rf(Q_n))$ are in *fhf*, where Q_n is reachable from FH with $n \in \mathbb{N}$ reactions. We have

$$\begin{aligned}
Q_n \to Q_{n+1} \quad \Leftrightarrow \quad & \exists F \in fg\,(rf(Q_n)), R, Q'_n \in \mathcal{P} : F \to R \\
& \wedge\, Q_n \equiv F \mid Q'_n \wedge Q_{n+1} \equiv R \mid Q'_n \\
\vee \quad & \exists F_1, F_2 \in fg\,(rf(Q_n)), R, Q'_n \in \mathcal{P} : F_1 \mid F_2 \to R \\
& \wedge\, Q_n \equiv F_1 \mid F_2 \mid Q'_n \wedge Q_{n+1} \equiv R \mid Q'_n.
\end{aligned}$$

We distinguish the cases and show that the fragments $F \in fg\,(rf(R))$ are in *fhf*. This proves the induction step as $F \in fg\,(rf(Q'_n))$ is in *fhf* by the hypothesis. We begin with reactions $F \to R$.

Case $K^{PT}\lfloor\tilde{p}\rfloor \triangleright K^{HD}\lfloor\tilde{p}\rfloor \triangleright K^{FH}\lfloor\tilde{a} \cup \tilde{p}\rfloor$ Lemma 4.4.12 shows that the fragments in $rf(R)$ are in *fhf*.

Case M^{FH} A process $M^{FH} = M_1^{FH} + \pi.FH + M_2^{FH}$ reacts to FH if and only if $\pi = \tau$. Lemma 4.4.12 shows that the fragments in $fg\,(rf(FH))$ are in *fhf*.

Case $\nu\tilde{a}.M^{PT}$ A process $\nu\tilde{a}.M^{PT} = \nu\tilde{a}.(M_1^{PT} + \overline{p}\langle a\rangle.SQ + M_2^{PT})$ has no reactions.

Case $\nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$ By the hypothesis, F is in *fhf*. Lemma 4.4.13 shows that the restricted form of R is a parallel composition of fragments in *fhf*.

We now turn to reactions between fragments F_1 and F_2, i.e., $F_1 \mid F_2 \to R$.

Case $F_1 \equiv K^{PT}\lfloor\tilde{p}\rfloor$, $F_1 \equiv K^{HD}\lfloor\tilde{p}\rfloor$, $F_1 \equiv K^{FH}\lfloor\tilde{a} \cup \tilde{p}\rfloor$ Calls to process identifiers do not react with other fragments.

Case $F_1 \equiv \nu\tilde{a}.M^{PT} = \nu\tilde{a}.(M_1^{PT} + \overline{p}\langle a\rangle.SQ + M_2^{PT})$ This fragments sends on a distinguished channel. The only fragments that listen on these channels are fragments $F_2 \equiv \nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$, which contain a connector process CN. Since by the hypothesis F_1 and F_2 are in *fhf*, Lemma 4.4.13 shows that the restricted form of R is a parallel composition of fragments in *fhf*.

Case $F_1 \equiv \nu\tilde{a}.(SQ_1 \mid \ldots \mid SQ_n \mid CN)$ Since $fn(F_1) \subseteq \mathcal{N}_\mathcal{P}$, the fragment only listens on distinguished channels. Since only fragments $\nu\tilde{a}.M^{PT}$ send on these channels, we are back in the previous case.

Case $F_1 \equiv M^{FH}$ A fragment $M^{FH} = M_1^{FH} + \overline{a}\langle b\rangle.FH_1 + N_1^{FH}$ sends on a public channel. Only fragments of the same form listen on public channels. Thus, we have a reaction $M_1^{FH} + \overline{a}\langle b\rangle.FH_1 + N_1^{FH} \mid M_2^{FH} + a(x).FH_2 + N_2^{FH} \to R$. With Proposition 2.1.38, we get

$$R \equiv FH_1 \mid FH_2\{b/x\}.$$

For the resulting fragments, it holds $F \in fg\,(rf(FH_1 \mid FH_2\{b/x\}))$ if and only if $F \in fg\,(rf(FH_1))$ or $F \in fg\,(rf(FH_2\{b/x\}))$. In both cases, Lemma 4.4.12 states that F is in *fhf*. ■

The client-server example indicates that finite handler processes naturally arise when modelling systems with central control units. Also the larger car platoon case study we investigate in Chapter 6 is a finite handler process and demonstrates the applicability of this system class. Theorem 4.4.8 ensures the property of structural stationarity holds for finite handler systems. We also observed in Section 4.3 that various and important subclasses of the π-Calculus known from the literature are structurally stationary. This implies that the constraint in Theorem 4.3.2 is frequently satisfied. Structural stationarity is a common property of π-Calculus processes. We continue with an investigation of how Petri nets relate to structurally stationary systems.

4.5 Complexity- and Decidability-theoretic Aspects

We investigate the translation of Petri nets back into structurally stationary processes. We start with an adaptation of a translation of Amadio and Meyssonnier [AM02], which yields a restriction-free process $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ that is linear in the size

of the translated net $\mathcal{N}$. The transition systems of the Petri net and the process are isomorphic. This relation strongly indicates that model checking structurally stationary processes against branching-time logics is undecidable, as it is undecidable for Petri nets. On the other hand, the relation shows decidability but EXPSPACE-hardness of reachability and model checking linear-time action-based logics for structurally stationary processes.

We then change the translation to $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}$, which has the following property. In $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$, every reachable process—which represents a reachable marking of $\mathcal{N}$—is a *single fragment*. Hence, the number of reachable fragments in $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$, and consequently the size of the structural semantics $\mathcal{N}[\![\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]]\!]$, is determined by the number of reachable markings of $\mathcal{N}$. It is well-known that this number is not bounded by a primitive recursive function [MM81] in the size of the net. Hence, the size of our structural semantics is not bounded by a primitive recursive function in the size of the process.

The section is written in semi-formal style. The aim is to give strong indications for the mentioned undecidability results without formally defining the respective equivalences and logics.

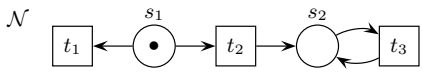
4.5.1 From Petri nets to Structural Stationarity

We define a translation of Petri nets into processes along the lines of [AM02]. Consider the Petri net $\mathcal{N} = (S, T, W, M_0)$, the idea to represent it by a process $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ is as follows. The places $s \in S$ are modelled by public names $s \in \mathcal{N}$. A marking with k tokens on s is reflected by the parallel composition of k processes $\overline{s}$. While Petri net transitions read from arbitrarily many places at once, at most two processes communicate in a reaction. The solution to this problem is to serialise the reading of tokens. Let transition t take three tokens from place s_1 and one token from s_3, and produce two tokens on s_2. We represent it by a process identifier $act.K_t\lfloor act, s_1, s_3, s_2\rfloor$ that is defined by

$$K_t(act, s_1, s_3, s_2) \quad := \quad s_1.s_1.s_1.s_3.(\overline{act} \mid act.K_t\lfloor act, s_1, s_2, s_3\rfloor \mid \overline{s_2} \mid \overline{s_2}).$$

The action act, which has to be received before the series of actions $s_1.s_1.s_1.s_3$ can be communicated, ensures that the firing of transitions does not interfere. This action is missing in [AM02] and facilitates the proof that the transition systems are isomorphic modulo deadlocks, a result that does not hold in [AM02]. Only with this isomorphism one can conclude about undecidability of model checking. The translation is illustrated in Figure 4.2.

Definition 4.5.1 ($\mathcal{P}_{\mathcal{PN}} : \mathcal{PN} \to \mathcal{P}_{FG<\infty}$)
Consider a Petri net $\mathcal{N} = (S, T, W, M_0)$. The *$\pi$-Calculus semantics of the Petri*



$$\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!] = \overline{act} \mid act.K_{t_1}\lfloor act, s_1\rfloor \mid act.K_{t_2}\lfloor act, s_1, s_2\rfloor \mid act.K_{t_3}\lfloor act, s_2, s_2\rfloor \mid \overline{s_1}$$

$$K_{t_1}(act, s_1) := s_1.(\overline{act} \mid act.K_{t_1}\lfloor act, s_1\rfloor)$$
$$K_{t_2}(act, s_1, s_2) := s_1.(\overline{act} \mid act.K_{t_2}\lfloor act, s_1, s_2\rfloor \mid \overline{s_2})$$
$$K_{t_3}(act, s_2, s_2') := s_2.(\overline{act} \mid act.K_{t_3}\lfloor act, s_2, s_2'\rfloor \mid \overline{s_2'}).$$

Figure 4.2:
Translation of a Petri net $\mathcal{N}$ into the restriction-free process $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$. Note that s_2 and s_2' in the defining equation of K_{t_3} are instantiated by the same free name s_2, so the term is correct.

net is the process

$$\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!] \quad := \quad \overline{act} \mid \Pi_{t\in T}\, act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{s\in supp(M_0)}\Pi^{M_0(s)}\overline{s}.$$

The defining equation of K_t with ${}^\bullet t = \{s_1, \ldots, s_n\}$ is

$$K_t(act, {}^\bullet t, t^\bullet) := \bullet s_1 \ldots \bullet s_n.(\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{s\in t^\bullet}\Pi^{W(t,s)}\overline{s}).$$

Here, $\bullet s$ abbreviates a sequence of $W(s,t)$ receive actions s, $\bullet s := s \ldots s$. ♦

We observe that the size of the process is linear in the size of the Petri net. This becomes important in the following section where we investigate the size of the structural semantics.

Lemma 4.5.2 (Size of $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$)
For every $\mathcal{N} \in \mathcal{PN}$ we have $\|\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]\| \leq 15\|\mathcal{N}\| + 3$.

Proof
Consider $\mathcal{N} = (S, T, W, M_0)$. We start with the length of the main process:

$$\begin{aligned} & len(\overline{act}.\mathbf{0} \mid \Pi_{t\in T}\, act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{s\in supp(M_0)}\Pi^{M_0(s)}\overline{s}.\mathbf{0}) \\ = \quad & 3 + 5|T| + \Sigma_{t\in T}(|{}^\bullet t| + |t^\bullet|) + 4(\Sigma_{s\in supp(M_0)} M_0(s)) \\ = \quad & 3 + 5|T| + \Sigma_{t\in T}(|{}^\bullet t| + |t^\bullet|) + 4(\Sigma_{s\in S} M_0(s)). \end{aligned}$$

The last equation holds since $s \notin supp(M_0)$ means $M_0(s) = 0$. We now consider the defining equation of every transition $t \in T$:

$$len(\quad K_t(act, {}^\bullet t, t^\bullet) := \bullet s_1 \ldots \bullet s_n.$$

$$
\begin{aligned}
& (\overline{act}.\mathbf{0} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{s\in t^\bullet}\Pi^{W(t,s)}\overline{s}.\mathbf{0}) \quad) \\
= \;& 10 + 2|{}^\bullet t| + 2|t^\bullet| + 2(\Sigma_{s\in {}^\bullet t}W(s,t)) + 4(\Sigma_{s\in t^\bullet}W(t,s)) \\
= \;& 10 + \Sigma_{s\in {}^\bullet t}(2W(s,t)+2) + \Sigma_{s\in t^\bullet}(4W(t,s)+2).
\end{aligned}
$$

In the last equation, we join the terms $2|{}^\bullet t|$ and $2(\Sigma_{s\in {}^\bullet t}W(s,t))$ as follows:

$$2|{}^\bullet t| + 2(\Sigma_{s\in {}^\bullet t}W(s,t)) = (\Sigma_{s\in {}^\bullet t}2) + \Sigma_{s\in {}^\bullet t}2W(s,t) = \Sigma_{s\in {}^\bullet t}(2W(s,t)+2).$$

Summing up the length of the main process and the length of all defining equations yields

$$
\begin{aligned}
& 3 + 5|T| + \Sigma_{t\in T}(|{}^\bullet t| + |t^\bullet|) + 4(\Sigma_{s\in S}M_0(s)) + \\
& \qquad \Sigma_{t\in T}\big(10 + \Sigma_{s\in {}^\bullet t}(2W(s,t)+2) + \Sigma_{s\in t^\bullet}(4W(t,s)+2)\big) \\
= \;& 3 + 15|T| + 4(\Sigma_{s\in S}M_0(s)) + \\
& \qquad \Sigma_{t\in T}\big(\Sigma_{s\in {}^\bullet t}(2W(s,t)+3) + \Sigma_{s\in t^\bullet}(4W(t,s)+3)\big).
\end{aligned}
$$

We now approximate this term with the size of $\mathcal{N}$. We first observe that

$$\Sigma_{s\in {}^\bullet t}(2W(s,t)+3) \leq \Sigma_{s\in {}^\bullet t}5W(s,t),$$

since $s \in {}^\bullet t$ implies $W(s,t) \geq 1$. We apply this approximation also to the sum over the places in the postset of t, which yields the following inequality:

$$
\begin{aligned}
\leq \;& 3 + 15|T| + 4(\Sigma_{s\in S}M_0(s)) + \Sigma_{t\in T}\big(\Sigma_{s\in {}^\bullet t}5W(s,t) + \Sigma_{s\in t^\bullet}7W(t,s)\big) \\
\leq \;& 3 + 15|T| + 4(\Sigma_{s\in S}M_0(s)) + 7\big(\Sigma_{t\in T}\big(\Sigma_{s\in {}^\bullet t}W(s,t) + \Sigma_{s\in t^\bullet}W(t,s)\big)\big) \\
= \;& 3 + 15|T| + 4(\Sigma_{s\in S}M_0(s)) + 7\big(\Sigma_{t\in T}\Sigma_{s\in S}\big(W(s,t) + W(t,s)\big)\big).
\end{aligned}
$$

The last equation exploits that $W(s,t) = 0$ if $s \notin {}^\bullet t$ and similar for $W(t,s)$. We raise all coefficients to their maximum $max\{15,4,7\} = 15$ and factor it out:

$$
\begin{aligned}
\leq \;& 3 + 15\big(|T| + \Sigma_{s\in S}M_0(s) + \Sigma_{t\in T}\Sigma_{s\in S}\big(W(s,t) + W(t,s)\big)\big) \\
\leq \;& 15\|\mathcal{N}\| + 3.
\end{aligned}
$$

In the last step, we have an inequality since the definition of $\|\mathcal{N}\|$ also has a term $|S|$. This concludes the proof. ■

The transition systems of the Petri net and that of its process coincide in the following sense. If the Petri net fires a transition from one marking to another, the process representation will do several reactions from one process that contains the send action $\overline{act}$ to another process with this action. We call these processes *valid* and denote the *set of all valid reachable processes* by $Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$.

Conversely, in a valid process P the process identifier K_t of any transition may attempt to execute its transition. It removes the $\overline{act}$ action to block the other transitions and starts to consume send actions according to its defining

equation (cf. Definition 4.5.1). If there are enough send actions, it reaches another valid process. In Petri net terms, transition t is enabled in the marking that corresponds to P. If there are not enough send actions, process P deadlocks after a few steps and does not reach a valid state. Hence, one valid process reacts to another one with a call to process identifier K_t if and only if transition t is enabled in the corresponding marking. We explain the deadlock behaviour on an example.

Example 4.5.3 (Deadlocks in $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$)
Consider the Petri net $\mathcal{N}$ in Figure 4.2. Transition t_3 is not enabled in the initial marking. We show that a call to K_{t_3} leads to a deadlock in the process $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$:

$$\begin{aligned}
&\overline{act} \mid act.K_{t_1}\lfloor act, s_1\rfloor \mid act.K_{t_2}\lfloor act, s_1, s_2\rfloor \mid act.K_{t_3}\lfloor act, s_2, s_2\rfloor \mid \overline{s_1}\\
\rightarrow\ &act.K_{t_1}\lfloor act, s_1\rfloor \mid act.K_{t_2}\lfloor act, s_1, s_2\rfloor \mid K_{t_3}\lfloor act, s_2, s_2\rfloor \mid \overline{s_1}\\
\rightarrow\ &act.K_{t_1}\lfloor act, s_1\rfloor \mid act.K_{t_2}\lfloor act, s_1, s_2\rfloor \mid s_2.(\overline{act} \mid act.K_{t_3}\lfloor act, s_2, s_2\rfloor \mid \overline{s_2}) \mid \overline{s_1}.
\end{aligned}$$

The process is in a deadlock, since K_{t_3} tries to consume $\overline{s_2}$ but fails as the action is not present and since the remaining transitions are blocked. ⧫

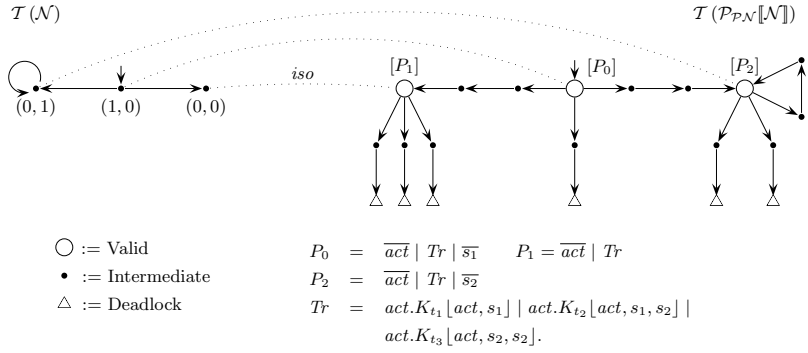


Figure 4.3:
Illustration of the isomorphism between the transition systems of the Petri net $\mathcal{N}$ in Figure 4.2 and the transition system of $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$. Note that *iso* relates markings to valid processes, only.

The explanation above indicates that the transition systems of the Petri net and that of its process representation are isomorphic, if we contract the several steps leading from one valid process to another one, cf. Figure 4.3. Technically, the contraction is done by the relation $\rightarrow_{\mathcal{T}}^{V} \subseteq Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv} \times$

$Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}$. It is defined by

$$[P] \rightarrow_{\mathcal{T}}^{V} [Q] \text{ iff } [P] \rightarrow_{\mathcal{T}} [P_1] \rightarrow_{\mathcal{T}} \ldots \rightarrow_{\mathcal{T}} [P_n] \rightarrow_{\mathcal{T}} [Q],$$

where all intermediate processes P_i are not valid. If we define the *valid process transition system* to be $\mathcal{T}_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]) := (Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}, \rightarrow_{\mathcal{T}}^{V}, \mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$, we can formulate the first proposition in this section. The transition system of the Petri net $\mathcal{N}$ coincides with the valid process transition system of $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$.

Proposition 4.5.4
For every Petri net $\mathcal{N} \in \mathcal{PN}$, the transition systems $\mathcal{T}(\mathcal{N})$ and $\mathcal{T}_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$ are isomorphic.

Proof
The isomorphism $iso : Reach(\mathcal{N}) \rightarrow Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$ maps M to

$$[\overline{act} \mid \Pi_{t \in T}\, act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in supp(M)} \Pi^{M(s)} \overline{s}].$$

We do not check every detail, but just consider the crucial point that *iso* is a strong graph homomorphism. Let $M \rightarrow M'$ by firing transition t. We show that we also have a reaction $iso(M) \rightarrow_{\mathcal{T}}^{V} iso(M')$. With the shortcut

$$rem := \Pi_{t' \in T \setminus \{t\}}\, act.K_{t'}\lfloor act, {}^\bullet t', t'^\bullet \rfloor \mid \Pi_{s \in supp(M)} \Pi^{M(s)} \overline{s},$$

we compute the following reaction:

$$\begin{aligned} & \overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid rem \\ \rightarrow \quad & K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid rem \\ \rightarrow \quad & \bullet s_1 \ldots \bullet s_n.(\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in t^\bullet} \Pi^{W(t,s)} \overline{s}) \mid rem. \end{aligned}$$

That t is enabled in M means $M(s) \geq W(s,t)$ for all $s \in {}^\bullet t$. Hence, there are more than $W(s,t)$ actions $\overline{s}$ composed in parallel in $\Pi_{s \in supp(M)} \Pi^{M(s)} \overline{s}$ and we can communicate $\bullet s_1 \ldots \bullet s_n$. We also resolve *rem* and integrate $act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor$ into the parallel composition of all transitions:

$$\begin{aligned} \rightarrow^* \quad & \overline{act} \mid \Pi_{t \in T}\, act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in supp(M)} \Pi^{M(s) - W(s,t)} \overline{s} \mid \Pi_{s \in t^\bullet} \Pi^{W(t,s)} \overline{s} \\ \equiv \quad & iso(M'). \end{aligned}$$

The last congruence holds with the equation $M'(s) = M(s) + W(s,t) + W(t,s)$ for all $s \in S$ in the definition of the Petri net transition relation. Since all intermediate processes are not valid, we have $iso(M) \rightarrow_{\mathcal{T}}^{V} iso(M')$.

The reverse direction, i.e., $iso(M) \rightarrow iso(M')$ implies $M \rightarrow M'$, is simpler. Since we can consume the sequence of receive actions $\bullet s_1 \ldots \bullet s_n$, we know that $M(s) \geq W(s,t)$ for all $s \in {}^\bullet t$. Hence, t is enabled and fires to a marking M'' for which $M''(s) = M(s) - W(s,t) + W(t,s)$ holds. This means $M'' = M'$. ■

The relationship in Proposition 4.5.4 has several complexity- and decidability-theoretic consequences.

Remark 4.5.5 (Complexity- and Decidability-theoretic Conclusions) Consider a structurally stationary process $P \in \mathcal{P}_{FG<\infty}$.

Reachability is decidable but EXPSPACE-hard for structurally stationary processes. To decide whether R is reachable from P, it is sufficient to construct $\mathcal{N}[\![P]\!]$ and decide whether $dec(rf(R))$ is a reachable marking, Theorem 3.4.3.

To show EXPSPACE-hardness we reduce the reachability problem for Petri nets, using the translation of Petri nets into processes in the present section. With Proposition 4.5.4, a marking M is reachable in $\mathcal{N}$ if and only if process

$$[\overline{act} \mid \Pi_{t \in T} act.K_t \lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in supp(M)} \Pi^{M(s)} \overline{s}]$$

is reachable in $\mathcal{T}$ ($\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$). Since reachability is EXPSPACE-hard for Petri nets and since the translation $\mathcal{P}_{\mathcal{PN}}$ is linear, reachability in structurally stationary processes is EXPSPACE-hard.

Model checking action-based linear-time logics should be decidable but EXPSPACE-hard. Consider the problem whether P satisfies a formula $\varphi_{\mathcal{P}}$ in an action-based linear-time logic. To argue why this problem should be decidable, we have to make the notion of *actions* precise. We assume that actions are communications between sequential processes $M^{\neq \mathbf{0}} \mid N^{\neq \mathbf{0}} \to R$ or between fragments $F \mid G \to R$. We first compute the structural semantics $\mathcal{N}[\![P]\!]$. From the π-Calculus formula $\varphi_{\mathcal{P}}$, we compute a formula $\theta_{\mathcal{PN}}(\varphi_{\mathcal{P}})$ in a linear-time logic for Petri nets, where the actions are transition names. To reduce the model checking problem from processes to Petri nets, the translation $\theta_{\mathcal{PN}}$ has to satisfy the relationship

$$P \models \varphi_{\mathcal{P}} \text{ if and only if } \mathcal{N}[\![P]\!] \models \theta_{\mathcal{PN}}(\varphi_{\mathcal{P}}).$$

We briefly comment on how to define $\theta_{\mathcal{PN}}$ to achieve this. Of course, $\theta_{\mathcal{PN}}$ depends on whether we choose fragments or sequential processes as actions. If the actions in $\varphi_{\mathcal{P}}$ are fragment communications $F \mid G \to R$, we just replace the action by the transition name $([F \mid G], [R])$. If we consider communications $M^{\neq \mathbf{0}} \mid N^{\neq \mathbf{0}} \to R$, the translation $\theta_{\mathcal{PN}}$ has to consider each transition $([F], [Q])$ where F contains $M^{\neq \mathbf{0}}$ and $N^{\neq \mathbf{0}}$ and every transition $([F_1 \mid F_2], [Q])$ where F_1 contains $M^{\neq \mathbf{0}}$ and F_2 contains $N^{\neq \mathbf{0}}$. In both cases, Q should contain R. Since there are finitely many transitions in the structural semantics, $\theta_{\mathcal{PN}}(\phi_{\mathcal{P}})$ will be a finite formula. Even for unbounded Petri nets, it is decidable whether $\mathcal{N}[\![P]\!] \models \theta_{\mathcal{PN}}(\varphi_{\mathcal{P}})$ holds [Esp94]. With the equivalence above, this decides the model checking problem for structurally stationary processes.

Model checking unbounded Petri nets against action-based linear-time logics is EXPSPACE-hard [Hab97] in the size of the net. We observe that $\mathcal{P}_{\mathcal{PN}}[\![P]\!]$ reflects isomorphically even the *labelled* transition system of a Petri net. If M fires transition t to marking M', i.e., $M[t\rangle M'$, then the process

$$\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{t' \in T\setminus\{t\}} act.K_{t'}\lfloor act, {}^\bullet t', t'^\bullet\rfloor \mid \Pi_{s\in supp(M)}\Pi^{M(s)}\overline{s}$$

that corresponds to M chooses the communication

$$\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \rightarrow K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor.$$

From $K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor$ we deterministically reach a process corresponding to M'. A net formula $\varphi_{\mathcal{PN}}$ over transition names t is now translated into a π-Calculus formula $\theta_{\mathcal{P}}(\varphi_{\mathcal{PN}})$ over reactions $\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \rightarrow K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor$ so that

$$\mathcal{N} \models \varphi_{\mathcal{PN}} \text{ if and only if } \mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!] \models \theta_{\mathcal{P}}(\varphi_{\mathcal{PN}}).$$

Since the translation of Petri net and formula are linear, model checking structurally stationary processes against action-based linear-time logics should be EXPSPACE-hard.

Model checking branching-time action-based logics should be undecidable for structurally stationary processes. Esparza showed that even the weakest action-based branching-time logics are undecidable for Petri nets [Esp97a]. The above reduction to show EXPSPACE-hardness for model checking linear-time logics still holds for branching-time logics and yields the undecidability result. The reason is that transition system isomorphism also preserves satisfaction of branching-time formulas.

Model checking state-based temporal logics should also be undecidable for the same reason. ♦

4.5.2 Size of the Structural Semantics

If the Petri net terminates, we can adapt the translation to processes so that every reachable process is a single fragment. This changed translation, which we denote by $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}$, shows that the size of the structural semantics $\mathcal{N}[\![P]\!]$ is in general not bounded by a primitive recursive function in the size of the process P—the main result in this section. The only difference in $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}$ is that act is restricted. To ensure that also send actions are connected with the name act, we replace $\overline{s}$ by $\overline{s}\langle act\rangle$:

$$\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!] \quad := \quad \nu act.(\overline{act} \mid \Pi_{t\in T}\, act.K_t\lfloor act, {}^\bullet t, t^\bullet\rfloor \mid \Pi_{s\in supp(M_0)}\Pi^{M_0(s)}\overline{s}\langle act\rangle).$$

The defining equations are of K_t are changed accordingly

$$K_t(act, {}^\bullet t, t^\bullet) \quad := \quad \bullet s_1(y_1) \ldots \bullet s_n(y_n). \\ (\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in t^\bullet} \Pi^{W(t,s)} \overline{s}\langle act \rangle).$$

It should be clear that the reachable processes in $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ and $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ can be identified. A process P reachable from $\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ corresponds to $\nu act.P'$ reachable from $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$, where all send actions $\overline{s}$ in P are replaced by $\overline{s}\langle act \rangle$ in P' and similar for receive actions.

Of course, the size of $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ is bounded by $15\|\mathcal{N}\| + 4$. The prefix νact increases the constant, but remember that $\overline{s}$ stands for $\overline{s}\langle s \rangle$, hence $\overline{s}$ and $\overline{s}\langle act \rangle$ both have size two. We now formally state that every reachable process is a single fragment.

Lemma 4.5.6
Consider a Petri net $\mathcal{N} \in \mathcal{PN}$. The restricted form of every $Q \in Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!])$ is a single fragment, i.e., $rf(Q) \in \mathcal{P}_{\mathcal{F}}$.

Proof
By induction on the length of the reaction sequence, we show that every reachable process is either of the form $\nu act.P'$ where P' is valid, or an intermediate process

$$\nu act.(\Pi_{t' \in T \setminus \{t\}} act.K_{t'}\lfloor act, {}^\bullet t', t'^\bullet \rfloor \mid \Pi_{i \in I} \overline{s}\langle act \rangle \mid R)$$

where R is of the following form:

$$s(x_1) \ldots s(x_m) \bullet s_i(y_i) \ldots \bullet s_n(y_n). \\ (\overline{act} \mid act.K_t\lfloor act, {}^\bullet t, t^\bullet \rfloor \mid \Pi_{s \in t^\bullet} \Pi^{W(s,t)} \overline{s}\langle act \rangle).$$

In both cases, computing the restricted form yields the process itself, which is a fragment. ■

We did not introduce $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}$ in the previous section, since it does not yield structurally stationary processes for arbitrary Petri nets, but only for bounded ones (with a finite state space). This is explained by the following lemma. It shows that the number of reachable fragments in $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]$ is larger than the number of reachable states in $\mathcal{N}$. Hence any Petri net with an infinite state space has infinitely many reachable fragments, and is therefore not structurally stationary.

Lemma 4.5.7
For any net $\mathcal{N} \in \mathcal{PN}$, we get $|Reach(\mathcal{N})| \leq |fg\left(rf(Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]))\right)/_{\equiv}|$.

Proof
With Proposition 4.5.4, there is an isomorphism between $\mathcal{T}(\mathcal{N})$ and $\mathcal{T}_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$. Hence, we have

$$|Reach(\mathcal{N})| = |Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}|.$$

Consider two different markings $M \neq M'$ in $Reach(\mathcal{N})$. By definition of function equality, this means $M(s) \neq M'(s)$ for at least one place s. Consider the two classes $iso(M) = [P] \neq [Q] = iso(M')$ in $Reach_V(\mathcal{N})/_{\equiv}$. We show that $[P] \neq [Q]$ give rise to different classes in $Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}$. By definition of iso, the number of send actions $\overline{s}$ is different in P and Q. We observed that P is represented by $\nu act.P'$ in $Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!])$ and Q corresponds to $\nu act.Q'$, where the send actions $\overline{s}$ are changed to $\overline{s}\langle act\rangle$ in P' and Q'. As these are the only prefixes sending on the public channel s, $\nu act.P'$ and $\nu act.Q'$ have a different number of prefixes sending on the free name s. Since this number is invariant under structural congruence, we get $\nu act.P' \not\equiv \nu act.Q'$ and conclude

$$|Reach_V(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}| \leq |Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv}|.$$

With Lemma 3.2.7 and Lemma 4.5.6 above, we have

$$Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!])/_{\equiv} = rf(Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]))/_{\equiv} = fg\left(rf(Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}]\!]))\right)/_{\equiv}.$$

This proves the inequality. ■

We now state our main result. In the class of structurally stationary processes, there is a sequence of processes $P_1, P_2, \ldots$ where the size $\|P_i\|$ grows linearly but the size of the structural semantics $\|\mathcal{N}[\![P_i]\!]\|$ grows faster than any primitive recursive function.

Theorem 4.5.8 (Size of the Structural Semantics)
For terminating processes $P \in \mathcal{P}_{FG<\infty}$ the size of $\mathcal{N}[\![P]\!]$ is not bounded by a primitive recursive function in the size of P.

Proof
In [MM81], Mayr and Meyer define a sequence of terminating nets $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \ldots$ where the size increases linearly, but the number of reachable states $Reach(\mathcal{N}_k)$ exceeds the value $A(k)$ of the following function $A : \mathbb{N} \to \mathbb{N}$. Consider for every $n \in \mathbb{N}$ the auxiliary functions $A_n := \mathbb{N} \to \mathbb{N}$ defined by

$$A_0(x) := 2x + 1 \qquad A_{n+1}(0) := 1 \qquad A_{n+1}(x+1) := A_n(A_{n+1}(x)).$$

Then, function A is defined by $A(n) := A_n(2)$. It can be shown that it dominates the primitive recursive functions, i.e., for every primitive recursive function f there is an index $n_0 \in \mathbb{N}$ so that $A(n) > f(n)$ for all $n \geq n_0$.

The function $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}$ is also linear in the size of the net. With the cited result, we obtain a sequence of processes $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_1]\!], \mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_2]\!], \mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_3]\!], \ldots$ where the size increases linearly and additionally

$$A(k) \leq |Reach(\mathcal{N}_k)| \leq |fg\left(rf(Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_k]\!]))\right)/_{\equiv}|$$

holds with Lemma 4.5.7. Since the processes $\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_n]\!]$ terminate, they are structurally stationary with Lemma 4.1.4. The reachable fragments form the places in the structural semantics, which yields the following inequality

$$|fg\left(rf(Reach(\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_n]\!]))\right)/_{\equiv}| \leq \|\mathcal{N}[\![\mathcal{P}^{\mathcal{F}}_{\mathcal{PN}}[\![\mathcal{N}_n]\!]]\!]\|.$$

We conclude that the size of the structural semantics $\|\mathcal{N}[\![P]\!]\|$ of terminating processes $P \in \mathcal{P}_{FG<\infty}$ can not be bounded by a primitive recursive function in the size of the process. ■

One may be tempted to argue that this is a negative result for the structural semantics, which shows that our translation is not usable for practical verification purposes. But this argumentation is not justified. Theorem 4.5.8 proves that in the large class of structurally stationary processes *there are* processes for which the translation behaves badly. This statement of existence does not imply that the structural semantics yields in general large Petri nets. Stated positively, the theorem demonstrates that processes with complex behaviour are still structurally stationary. It would be interesting to see how the size of the structural semantics behaves for subclasses of structurally stationary processes. We leave this as a point for future work.

4.6 Related Work and Conclusion

The structural semantics maps π-Calculus processes into place/transition Petri nets. For automatic verification purposes, *finiteness* of the semantic image is a prerequisite. In Lemma 4.1.2, we prove that exactly the structurally stationary processes have a finite Petri net representation under the structural semantics.

For the applicability of our semantics, it is important that the class of structurally stationary processes is *expressive*. Theorem 4.3.2 completely characterises structural stationarity and shows that important classes of processes have this property. For example, an application of the theorem immediately yields structural stationarity of *finite control processes* [Dam96], *finitary processes* [MP95a, Pis99, MP01], and *restriction-free processes* [AM02]. We stress that the structural semantics is the first translation that finitely represents both, finitary and restriction-free processes.

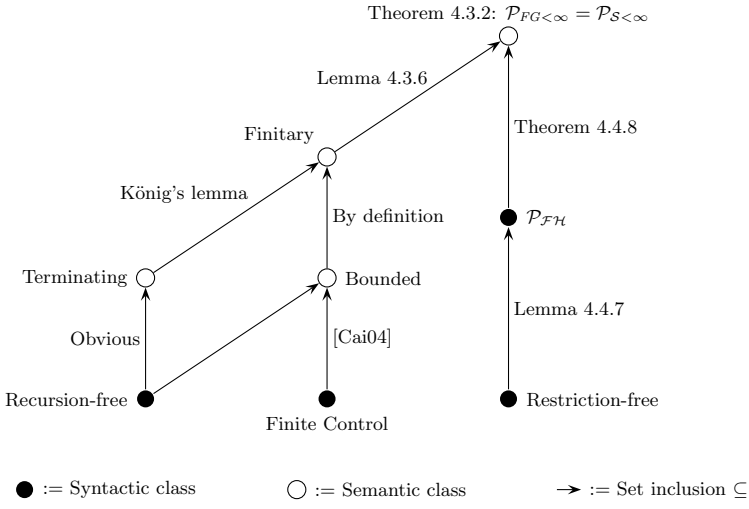


Figure 4.4: A first hierarchy of processes.

Figure 4.4 summarises the relationships between the process classes. For completeness reasons, we added the class of *bounded processes* defined by Caires in [Cai04]. A process is bounded if the state space of every syntactic subprocess is finite. Caires shows that Dam's finite control processes satisfy this constraint. We observe that by definition bounded processes are finitary. The restriction-bounded processes of Busi and Gorrieri [BG09], which we discussed in the previous chapter, are missing in Figure 4.4. It turns out that they are incomparable with structurally stationary processes and that both classes can be unified. We elaborate on this relationship in Chapter 9 and update the picture accordingly.

With the aim of modelling client-server systems, we defined the class of *finite handler processes*. Its usefulness is demonstrated in Chapter 6, where we model a larger case study from the traffic control domain. Again Theorem 4.3.2 proves structural stationarity for finite handler processes (cf. Theorem 4.4.8).

Astonishingly, the theorem also suggests a different technique for computing the structural semantics (cf. Section 3.5). If the bound $k_{\mathcal{S}}$ on the number of sequential processes in fragments is known, the set FG can be used as places in a Petri net $\mathcal{N}_{FG}[\![P]\!]$, which subsumes the structural semantics $\mathcal{N}[\![P]\!]$. To compute $\mathcal{N}_{FG}[\![P]\!]$ efficiently, precise approximations to the bound $k_{\mathcal{S}}$ and to the set of substitutions, which are applied to derivatives, need to be computed statically from the process P. In [BDNN98], a control flow analysis for the π-Calculus has been proposed that over-approximates (1) for ever channel the set of names

that may be sent on it and (2) for every input prefix the channels which it may receive. It should be possible to adapt the technique to compute the required approximations.

A translation of Petri nets back into structurally stationary processes proved that both models are computationally equivalent. The main finding is that for terminating Petri nets, a structurally stationary process can be constructed that has one fragment for every reachable marking. A classical result from Petri net theory now shows that the size of the structural semantics is not bounded by any primitive recursive function in the size of the process. Although this is a negative result, one should keep in mind that structurally stationary processes are a very general class. For restriction-free processes, for example, we conjecture that the structural semantics is polynomial and similarly, for finite handler and finite control processes it should be exponential, but this is future work.

Our translation of Petri nets into processes is inspired by a proposal of Amadio and Meyssonnier [AM02], which in turn is closely related to [BRdS86]. The motivation of Boudol et. al. was to study algebraic operators to construct Petri nets in a systematic way, and may be compared with the achievements for the Petri Box Calculus of Best et. al. [BDK01]. Motivated by the research on structural subclasses of Petri nets, a different translation into processes was proposed by Dietz and Schreiber [DS94]. They decompose a net (with binary synchronisation) into communication-free parts that synchronise. Inspired by the idea of unfoldings (cf. Section 2.2.3), the process translation highlights the flow of tokens in these subnets.

To conclude, we remark that structural stationarity is a non-compositional property. If P and Q are structurally stationary, then $\pi.P$ is structurally stationary but $P \mid Q$ and $\nu a.P$ need not be. To enlarge the class of structurally stationary processes, it would be interesting to find compositional subclasses.

Part II

Reasoning in Structural Stationarity

5 Unfolding-based Model Checking of Finite Control Processes

Contents

In this chapter, we contribute to the *analysability* of structurally stationary processes. For safe Petri nets particularly efficient model checking techniques have been developed. To exploit these algorithms for π-Calculus verification, we sacrifice a part of the modelling power of structurally stationary processes and investigate the translation of syntactic subclasses with the structural semantics. The result is that *finite control processes (FCPs)* [Dam96], which have an acceptably high modelling power, admit a translation into safe Petri nets. We proceed in two steps.

We first prove a general boundedness result for the Petri nets obtained from the translation of FCPs. More precisely, we show that the structural semantics of an FCP is a bounded Petri net, and we develop a technique for computing a non-trivial bound by static analysis of the process term.

The boundedness result suggests the definition of a syntactic subclass of FCPs, so-called *safe processes* that are translated into safe Petri nets. The second main result is that every FCP can be translated into a safe process of at most quadratic size. Combined with the structural semantics, this gives a procedure for translating an FCP into a safe Petri net.

We present a concrete model checking technique for safe Petri nets due to Heljanko [Hel02] and Khomenko, Koutny, and Yakovlev [KKY04]. It first computes

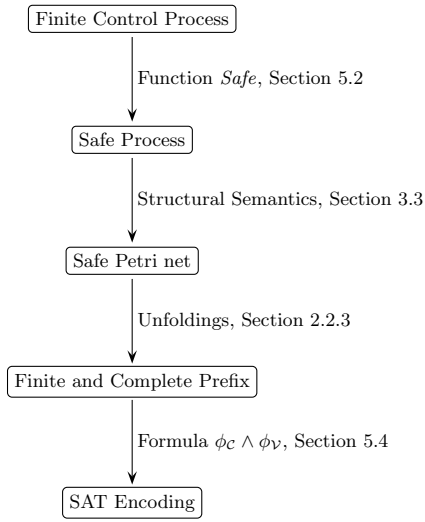


Figure 5.1: Illustration of our approach to verification of FCPs.

the finite and complete prefix (cf. Section 2.2.3) and then encodes it together with the verification problem at hand into a Boolean satisfiability problem. The encoding ensures that any satisfying assignment provides a counterexample to the property. Figure 5.1 sums up our verification approach.

To demonstrate the applicability of our approach, we verify a number of benchmark case studies from the literature for deadlock freedom—the common denominator of all tools for π-Calculus verification. The experiments show that it has a significant advantage over other existing tools in terms of memory consumption and runtime. In brief, the contributions in this chapter are as follows:

- We prove a general boundedness result for the structural semantics of FCPs.
- We define safe processes, a syntactic subclass of FCPs which are translated into safe Petri nets. The main result is that every FCP can be translated into a bisimilar safe process. We show optimality of this translation.
- We recall an efficient verification technique from the literature. The resulting tool chain in Figure 5.1 is applied to the verification of a number of case studies. Our approach outperforms existing tools by orders of magnitude.

The chapter is organised as follows. In Section 5.1, we prove the general boundedness result for the Petri nets resulting from the translation of FCPs. Based on

this insight, we define safe processes in Section 5.2 and show that every FCP can be translated into a safe process. We prove optimality of this translation in Section 5.3. In Section 5.4, we introduce a particularly efficient verification approach for unfoldings, which we apply in Section 5.5. Section 5.6 concludes the chapter. We illustrate the developed theory on a client-server system.

Example 5.0.1 (Client/Server System)
Consider the process $C\lfloor url\rfloor \mid C\lfloor url\rfloor \mid S\lfloor url\rfloor$ modelling two clients and a sequential server, with the corresponding process identifiers defined as

$$C(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url\rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url\rfloor.$$

The server is located at some URL, $S\lfloor url\rfloor$. To contact it, a client sends its *ip* address on the channel *url*. The server receives the IP address and—to establish a private connection with the client—creates a temporary session νses, which it passes to the client, $\overline{y}\langle ses\rangle$. Upon reception of the session, $ip(s)$, client and server continue to interact, which is not modelled explicitly. At some point, the server decides that the session has expired. It sends the session object itself to the client, $\overline{ses}\langle ses\rangle$, and becomes a server again, $S\lfloor url\rfloor$. The client receives the message, $s(x)$, and calls its recursive definition to be able to contact the server once more, $C\lfloor url\rfloor$. The model can contain several clients (two in our case), but the server is engaged with one client at a time. ♦

5.1 Boundedness of Finite Control Process Nets

We investigate the translation of *finite control processes (FCPs)* [Dam96]. Recall from Definition 2.1.5, that FCPs are of the form $\nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ where the P_i do not use the parallel composition operator. Without loss of generality, we assume that either $\nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ is $\mathbf{0}$ or none of the P_i contains $\mathbf{0}$. This can always be achieved by replacing a process $\pi.\mathbf{0}$ by $\pi.K_{\mathbf{0}}\lfloor - \rfloor$ with $K_{\mathbf{0}}(-) := \mathbf{0}$. To indicate that a process is finite control, we denote it by $P_{\mathcal{FC}}$.

The main result in this section states that the Petri net $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ is bounded, and a non-trivial bound can be derived syntactically from the structure of $P_{\mathcal{FC}}$. Such a bound follows from the intuitive idea is that k tokens on a place $[F]$ require at least k processes P_i in $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ sharing some process identifiers.

To make the notion of sharing process identifiers precise we define *orbits*. The *orbit* of a process P_i consists of the identifiers P_i calls, both directly and indirectly. With this definition, we rephrase the above idea: if there are at most k orbits in $P_{\mathcal{FC}}$ whose intersection is non-empty then the net $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ is k-bounded.

The result states that *the bound of* $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ *is small.* If $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ then $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ is trivially n-bounded, as the total number of orbits is n. Often, our method yields bounds which are better than n. This limits the state space in our translation and makes such nets relatively easy to model check.

The intuitive idea of the *orbit function* is to collect all process identifiers syntactically reachable from a given process. To collect the process identifiers in a single process, we employ the function *ident*.

Definition 5.1.1 (*ident*, $orb : \mathcal{P} \to \mathbb{P}(\mathcal{ID})$)
The function $ident : \mathcal{P} \to \mathbb{P}(\mathcal{ID})$ computes the set of process identifiers $ident(P)$ that are in the process $P \in \mathcal{P}$:

$$\begin{aligned} ident(\mathbf{0}) &:= \emptyset & ident(K\lfloor\tilde{a}\rfloor) &:= \{K\} \\ ident(\pi.P) &:= ident(P) & ident(M+N) &:= ident(M) \cup ident(N) \\ ident(P \mid Q) &:= ident(P) \cup ident(Q) & ident(\nu a.P) &:= ident(P). \end{aligned}$$

The *orbit* of a process P, denoted by $orb(P)$, is the smallest set so that (1) $ident(P) \subseteq orb(P)$ and (2) if a process identifier K with a defining equation $K(\tilde{x}) := Q$ is in $orb(P)$ then $ident(Q) \subseteq orb(P)$. ♦

By induction on the derivations of structural congruence, it can be shown that the set of identifiers in a process is invariant under structural congruence.

Lemma 5.1.2 (Invariance of *ident* under $\equiv$)
For all $P, Q \in \mathcal{P}$ the congruence $P \equiv Q$ implies $ident(P) = ident(Q)$.

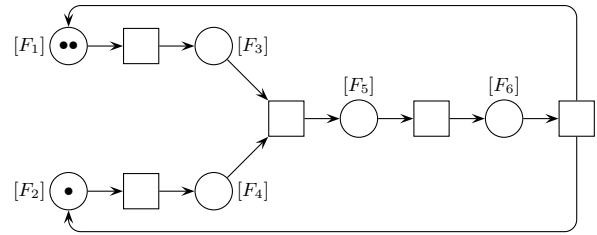
To formally state the boundedness result, we define the *maximal number of intersecting orbits* of a process $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ to be

$$\|P_{\mathcal{FC}}\|_{\cap} := max\left\{|I| \mid I \subseteq \{1,\ldots,n\} \text{ and } \textstyle\bigcap_{i\in I} orb(P_i) \neq \emptyset\right\}.$$

Theorem 5.1.3 (Boundedness Result for Finite Control Process Nets)
$\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ is $\|P_{\mathcal{FC}}\|_{\cap}$-bounded.

Example 5.1.4 (Application of Theorem 5.1.3)
Consider process $P_{\mathcal{FC}} = C\lfloor url\rfloor \mid C\lfloor url\rfloor \mid S\lfloor url\rfloor$ in Example 5.0.1. We compute $orb(S\lfloor url\rfloor) = \{S\}$ and $orb(C\lfloor url\rfloor) = \{C\}$ for both clients. Thus, $\|P_{\mathcal{FC}}\|_{\cap} = 2$ and so the structural semantics $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ in Figure 5.2 is 2-bounded. This is an improvement on the trivial bound of 3, the number of concurrent processes. ♦

We spend the rest of the section proving Theorem 5.1.3. The Petri net $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ is k-bounded iff in every reachable process $Q \in Reach(P_{\mathcal{FC}})$ there are at most k



$F_1 = C\lfloor url \rfloor$ $\qquad$ $F_3 = \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url \rfloor$

$F_2 = S\lfloor url \rfloor$ $\qquad$ $F_4 = url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor$

$F_5 = \nu ip.(ip(s).s(x).C\lfloor url \rfloor \mid \nu ses.\overline{ip}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor)$

$F_6 = \nu ses.(ses(x).C\lfloor url \rfloor \mid \overline{ses}\langle ses\rangle.S\lfloor url \rfloor).$

Figure 5.2:
The structural semantics $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ of the client/server system in Example 5.0.1. Transition names are omitted for the sake of readability. The Petri net is in fact bounded by $\|P_{\mathcal{FC}}\|_\cap = 2$, which is the optimal bound in this example.

fragments that are structurally congruent. Thus, we need to show that the number of structurally congruent fragments is bounded by $\|P_{\mathcal{FC}}\|_\cap$ in every reachable process. To do so, we assume there are k fragments $F_1 \equiv \ldots \equiv F_k$ in Q and conclude that there are at least k intersecting orbits in $P_{\mathcal{FC}}$, i.e., $\|P_{\mathcal{FC}}\|_\cap \geq k$. We argue as follows. From structural congruence we know that the identifiers in all F_i are equal, Lemma 5.1.2. We now show that the identifiers of the F_i are already contained in the orbits of different P_i in $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$. Thus, the intersection $orb(P_1) \cap \ldots \cap orb(P_k)$ is not empty. This means that we found k intersecting orbits, so $\|P_{\mathcal{FC}}\|_\cap \geq k$ holds.

To show $ident(F_i) \subseteq orb(P_i)$ we relate the processes in every reachable fragment with the initial process $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$. To achieve this, we again employ the theory of derivatives presented in Section 4.2. We prove that every reachable process is as a parallel composition of derivatives of the processes P_i in $P_{\mathcal{FC}}$.

Lemma 5.1.5
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$. Then every $Q \in Reach(P_{\mathcal{FC}})$ is structurally congruent with $\nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m)$ so that there is an injective function $inj : \{1,\ldots,m\} \to \{1,\ldots,n\}$ for which $Q_i \in derivatives(P_{inj(i)})$ and $\sigma_i : fn(Q_i) \to \tilde{c} \cup fn(P_{\mathcal{FC}})$ hold.

Proof
The proof is similar to that of Proposition 4.2.2 in Section 4.2. ■

The lemma below states that the identifiers of any derivative Q of P are in the orbit of P. Combined with the previous lemma, this relates the identifiers in a reachable fragment and the orbits in the initial process.

Lemma 5.1.6
For every process $P \in \mathcal{P}$ and every $Q \in \mathit{derivatives}(P)$ the inclusion $\mathit{ident}(Q) \subseteq \mathit{orb}(P)$ holds.

The proof of Lemma 5.1.6 requires the following observation. If $Q \in \mathit{der}(P)$ then the identifiers of Q are among those of P. This can be shown by induction on the structure of P.

Lemma 5.1.7
For every process $P \in \mathcal{P}$ and every $Q \in \mathit{der}(P)$ we have $\mathit{ident}(Q) \subseteq \mathit{ident}(P)$.

The statement in Lemma 5.1.6 now follows by induction on the structure of $\mathit{derivatives}(P)$.

Proof (of Lemma 5.1.6)
In the base case, we consider $Q \in \mathit{der}(P)$. By Lemma 5.1.7 and the definition of the orbit function, we get $\mathit{ident}(Q) \subseteq \mathit{ident}(P) \subseteq \mathit{orb}(P)$.

Consider $K\lfloor\tilde{a}\rfloor \in \mathit{derivatives}(P)$ with $K(\tilde{x}) := Q$ and assume we already proved the inclusion $\mathit{ident}(K\lfloor\tilde{a}\rfloor) = \{K\} \subseteq \mathit{orb}(P)$. For every $R \in \mathit{der}(Q)$ we establish $\mathit{ident}(R) \subseteq \mathit{orb}(P)$. By Lemma 5.1.7, $\mathit{ident}(R) \subseteq \mathit{ident}(Q)$. As $K \in \mathit{orb}(P)$ by the hypothesis, the definition of the orbit function implies $\mathit{ident}(Q) \subseteq \mathit{orb}(P)$. This concludes the proof. ■

We return to the argumentation on Theorem 5.1.3. Consider a reachable process $Q \equiv \Pi^k F \mid Q'$. By Lemma 5.1.5, Q is structurally congruent with a process $\nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m)$ that satisfies $Q_i \in \mathit{derivatives}(P_{\mathit{inj}(i)})$. By transitivity, also $\Pi^k F \mid Q'$ is structurally congruent with this process. Lemma 3.2.7 and Proposition 3.2.10 relate the restricted forms of both processes by restricted equivalence:

$$\mathit{rf}(\Pi^k F \mid Q') = \Pi^k F \mid \mathit{rf}(Q') \equiv_{\mathit{rf}} \Pi_{i\in I} G_i = \mathit{rf}(\nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m)),$$

for some fragments G_i. By definition of restricted equivalence, k of the G_i are structurally congruent. As identifiers are preserved by structural congruence, these G_i have the same identifiers. Each G_i is a parallel composition of processes $Q_i\sigma_i$. Since every G_i consists of different $Q_i\sigma_i$, there are k processes Q_i that share

process identifiers. With Lemma 5.1.6, the identifiers of every Q_i are in the orbit of $P_{inj(i)}$. Since *inj* is injective, we have k processes $P_{inj(i)}$ with intersecting orbits, and so Theorem 5.1.3 holds. We now turn this argumentation into a formal proof.

Proof (of Theorem 5.1.3)
Consider a reachable marking of $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ with k tokens on place $[F]$. We show that $\|P_{\mathcal{FC}}\|_\cap \geq k$. According to Theorem 3.4.3, the reachable marking corresponds to a process $Q \in \mathit{Reach}(P_{\mathcal{FC}})$ with $Q \equiv \Pi^k F \mid Q'$ for some $Q' \in \mathcal{P}$.

According to Lemma 5.1.5, $\Pi^k F \mid Q' \equiv Q \equiv \nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m)$. With Proposition 3.2.10 it follows that $rf(\Pi^k F \mid Q') \equiv_{rf} rf(\nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m))$. Let the restricted form $rf(\nu\tilde{c}.(Q_1\sigma_1 \mid \ldots \mid Q_m\sigma_m))$ be the parallel composition $\Pi_{i\in I}G_i$ for some fragments G_i. Lemma 3.2.7 yields $rf(F) = F$, and so

$$\Pi^k F \mid rf(Q') \equiv_{rf} \Pi_{i\in I}G_i.$$

Restricted equivalence ensures that for every F there is a G_i so that $F \equiv G_i$. Without loss of generality, let these G_i be $G_1, \ldots, G_k$. Since all F are structurally congruent, we have $G_1 \equiv \ldots \equiv G_k$.

By definition of restricted and standard form, every G_i is structurally congruent with $\nu\tilde{c}_i.(\Pi_{i\in I_i}Q_i\sigma_i)$, where $\emptyset \neq I_i \subseteq \{1, \ldots, m\}$ and $I_i \cap I_j = \emptyset$ for $i \neq j$. Non-emptiness follows from the fact that fragments are not structurally congruent to $\mathbf{0}$, disjointness from the fact that every sequential process $Q_i\sigma_i$ belongs to exactly one fragment. The identifiers are preserved by structural congruence according to Lemma 5.1.2:

$$\mathit{ident}(G_i) = \mathit{ident}(\nu\tilde{c}_i.(\Pi_{i\in I_i}Q_i\sigma_i)) = \bigcup_{i\in I_i}\mathit{ident}(Q_i\sigma_i) = \bigcup_{i\in I_i}\mathit{ident}(Q_i).$$

The second equality holds by the definition of *ident*, and the third because of the invariance of *ident* under substitution. Again, as $G_1 \equiv \ldots \equiv G_k$ and since the identifiers are preserved by structural congruence, we have $\mathit{ident}(G_1) = \ldots = \mathit{ident}(G_k)$, which means

$$\bigcup_{i\in I_1}\mathit{ident}(Q_i) = \ldots = \bigcup_{i\in I_k}\mathit{ident}(Q_i). \tag{5.1}$$

Consider a process Q_{i_1} where $i_1 \in I_1$. As derivatives are different from $\mathbf{0}$, it contains an identifier $K \in \mathit{ident}(Q_{i_1})$. With Equality (5.1), there are processes Q_{i_2} where $i_2 \in I_2$ up to Q_{i_k} where $i_k \in I_k$ so that $K \in \mathit{ident}(Q_{i_j})$ for all $2 \leq j \leq k$. By Lemma 5.1.6, the inclusion $\mathit{ident}(Q_{i_j}) \subseteq \mathit{orb}(P_{inj(i_j)})$ holds for all j. Of course, *inj* is the injection that exists due to Lemma 5.1.5. Combining both arguments, we conclude $\mathit{orb}(P_{inj(i_1)}) \cap \ldots \cap \mathit{orb}(P_{inj(i_k)}) \neq \emptyset$. As the index sets I_i are pairwise disjoint and as *inj* is injective, all $P_{inj(i_j)}$ are distinct. So we found k processes with intersecting orbits, which means $\|P_{\mathcal{FC}}\|_\cap \geq k$. ■

In case the orbits of all P_i in $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ are disjoint, Theorem 5.1.3 implies safeness of the structural semantics $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$. We now show that every FCP can be translated into a bisimilar process with disjoint orbits.

5.2 From Finite Control to Safe Processes

Safe nets are a prerequisite to apply particularly efficient unfolding-based verification techniques. According to Theorem 5.1.3, the reason for non-safeness of the nets of arbitrary FCPs is the intersection of orbits. In this section we investigate a translation of FCPs into their syntactic subclass called *safe processes*, where the sequential processes comprising an FCP have pairwise disjoint orbits. The idea of translating $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ to the safe process $\mathit{Safe}(P_{\mathcal{FC}})$ is to create copies of the process identifiers that are shared among several P_i, i.e., of those that belong to several orbits. The corresponding defining equations are duplicated as well. The intuition is that every P_i gets its own set of process identifiers which it can call during system execution. Hence, due to Theorem 5.1.3, the resulting safe processes are mapped to safe Petri nets.

The main result in this section states that the processes $P_{\mathcal{FC}}$ and $\mathit{Safe}(P_{\mathcal{FC}})$ are bisimilar, and, moreover, that the fragments are preserved. Furthermore, the size of the specification $\mathit{Safe}(P_{\mathcal{FC}})$ is at most quadratic in the size of $P_{\mathcal{FC}}$. In Section 5.3, we show that this translation is optimal.

Definition 5.2.1 (Safe Process)
An FCP $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ is a *safe process* if the orbits are pairwise disjoint, i.e., for all $i, j \in \{1, \ldots, n\}$: if $i \neq j$ then $\mathit{orb}(P_i) \cap \mathit{orb}(P_j) = \emptyset$. ♦

To translate an FCP $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ into a safe process $\mathit{Safe}(P_{\mathcal{FC}})$, we choose a unique number for every sequential process, say i for P_i. We then rename every process identifier K in the orbit of P_i to a fresh identifier K^i. Technically, we use the functions $\mathit{ren}_k : \mathcal{P} \to \mathcal{P}$.

Definition 5.2.2 ($\mathit{ren}_k : \mathcal{P} \to \mathcal{P}$)
For every $k \in \mathbb{N}$, the function $\mathit{ren}_k : \mathcal{P} \to \mathcal{P}$ maps a process $P \in \mathcal{P}$ to the process $\mathit{ren}_k(P)$ as follows:

$$\begin{aligned}
\mathit{ren}_k(\mathbf{0}) &:= \mathbf{0} & \mathit{ren}_k(K) &:= K^k \\
\mathit{ren}_k(K\lfloor\tilde{a}\rfloor) &:= \mathit{ren}_k(K)\lfloor\tilde{a}\rfloor & \mathit{ren}_k(\pi.P) &:= \pi.\mathit{ren}_k(P) \\
\mathit{ren}_k(M+N) &:= \mathit{ren}_k(M) + \mathit{ren}_k(N) & \mathit{ren}_k(P \mid Q) &:= \mathit{ren}_k(P) \mid \mathit{ren}_k(Q) \\
\mathit{ren}_k(\nu a.P) &:= \nu a.\mathit{ren}_k(P).
\end{aligned}$$

The defining equation of K^K is defined to be $K^k(\tilde{x}) := \mathit{ren}_k(Q)$ if $K(\tilde{x}) := Q$. ♦

We shall need that renaming is compatible with the computation of process identifiers and the application of substitutions. The proof is by induction on the structure of processes.

Lemma 5.2.3
For every $P \in \mathcal{P}$ we get $ident(ren_k(P)) = ren_k(ident(P))$ and $ren_k(P)\sigma = ren_k(P\sigma)$.

With the ren_k functions, an FCP is translated into a safe process as follows:

Definition 5.2.4
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$, then $Safe(P_{\mathcal{FC}}) := \nu\tilde{a}.(ren_1(P_1) \mid \ldots \mid ren_n(P_n))$. Note that the defining equation of K^k is $K^k(\tilde{x}) := ren_k(Q)$ if $K(\tilde{x}) := Q$. The original equations $K(\tilde{x}) := Q$ are removed. ♦

Example 5.2.5 (Translation to Safe Processes)
Consider the FCP $P_{\mathcal{FC}} = C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor$ in Example 5.0.1. The translation is $Safe(P_{\mathcal{FC}}) = C^1\lfloor url \rfloor \mid C^2\lfloor url \rfloor \mid S^3\lfloor url \rfloor$, where

$$C^1(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^1\lfloor url \rfloor$$
$$C^2(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^2\lfloor url \rfloor$$
$$S^3(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S^3\lfloor url \rfloor.$$

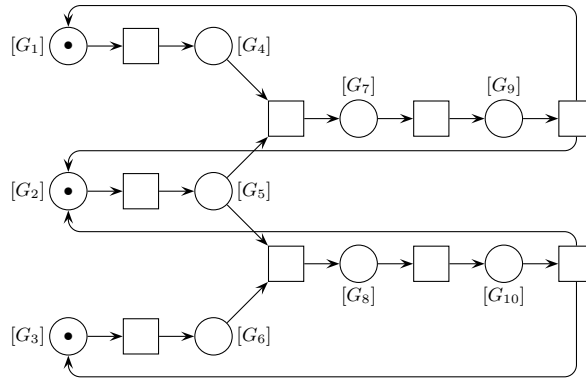
The equations for C and S are removed. The structural semantics $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ is depicted in Figure 5.3. The Petri net is safe. ♦

In the example, we just created another copy of the equation defining a client. The following result shows that the size of the translated system is always at most quadratic in the size of the original specification.

Proposition 5.2.6 (Size)
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ be an FCP. Then $\|Safe(P_{\mathcal{FC}})\| \leq n \cdot \|P_{\mathcal{FC}}\|$.

Proof
The process $Safe(P_{\mathcal{FC}})$ is defined to be $\nu\tilde{a}.(ren_1(P_1) \mid \ldots \mid ren_1(P_n))$. Consider the renaming of P_i to $ren_i(P_i)$. In the worst case, all definitions for process identifiers in $P_{\mathcal{FC}}$ are copied. This results in a specification of size at most $\|P_{\mathcal{FC}}\|$ for every process P_i. When we translate $P_{\mathcal{FC}}$ to $Safe(P_{\mathcal{FC}})$, we rename n processes P_i. Therefore, the size of $Safe(P_{\mathcal{FC}})$ is bounded by $n \cdot \|P_{\mathcal{FC}}\|$. ■

Note that since $n \leq \|P_{\mathcal{FC}}\|$, the result shows that the size of $Safe(P_{\mathcal{FC}})$ is at most quadratic in the size of $P_{\mathcal{FC}}$. We now show that $Safe(P_{\mathcal{FC}})$ is in fact a



$$G_1 = C^1\lfloor url \rfloor \qquad G_4 = \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^1\lfloor url\rfloor$$
$$G_2 = S^3\lfloor url \rfloor \qquad G_5 = url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S^3\lfloor url\rfloor$$
$$G_3 = C^2\lfloor url \rfloor \qquad G_6 = \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^2\lfloor url\rfloor$$
$$G_7 = \nu ip.(ip(s).s(x).C^1\lfloor url\rfloor \mid \nu ses.\overline{ip}\langle ses\rangle.\overline{ses}\langle ses\rangle.S^3\lfloor url\rfloor)$$
$$G_8 = \nu ip.(ip(s).s(x).C^2\lfloor url\rfloor \mid \nu ses.\overline{ip}\langle ses\rangle.\overline{ses}\langle ses\rangle.S^3\lfloor url\rfloor)$$
$$G_9 = \nu ses.(ses(x).C^1\lfloor url\rfloor \mid \overline{ses}\langle ses\rangle.S^3\lfloor url\rfloor)$$
$$G_{10} = \nu ses.(ses(x).C^2\lfloor url\rfloor \mid \overline{ses}\langle ses\rangle.S^3\lfloor url\rfloor).$$

Figure 5.3:
Structural semantics of the safe process $Safe(P_{\mathcal{FC}})$ in Example 5.2.5. The Petri net is safe and bisimilar with the structural semantics of the client/server system $P_{\mathcal{FC}}$ in Figure 5.2.

safe process. This follows from the compatibility of the renaming with the orbit function in Lemma 5.2.7.

Lemma 5.2.7
For every $k \in \mathbb{N}$ the equality $orb(ren_k(P)) = ren_k(orb(P))$ holds.

Proof
Inclusion $\subseteq$ We show $orb(ren_k(P)) \subseteq ren_k(orb(P))$ by induction on the structure of $orb(ren_k(P))$. Let $K^k \in ident(ren_k(P))$. The compatibility of $ident$ and ren_k in Lemma 5.2.3 yields $K^k \in ren_k(ident(P)) \subseteq ren_k(orb(P))$. The inclusion holds as $ident(P) \subseteq orb(P)$ by definition of the orbit function.

In the induction step, assume the inclusion $K^k \in \mathit{ren}_k(\mathit{orb}(P))$ holds for $K^k \in \mathit{orb}(\mathit{ren}_k(P))$. Let the defining equation be $K^k(\tilde{x}) := \mathit{ren}_k(Q)$. We have to show that $\mathit{ident}(\mathit{ren}_k(Q)) \subseteq \mathit{ren}_k(\mathit{orb}(P))$. The hypothesis reveals that $K \in \mathit{orb}(P)$ with $K(\tilde{x}) := Q$ by definition of ren_k. By Lemma 5.2.3, we have $\mathit{ident}(\mathit{ren}_k(Q)) = \mathit{ren}_k(\mathit{ident}(Q))$. Since $K \in \mathit{orb}(P)$, the definition of orbit yields $\mathit{ident}(Q) \subseteq \mathit{orb}(P)$. We conclude $\mathit{ren}_k(\mathit{ident}(Q)) \subseteq \mathit{ren}_k(\mathit{orb}(P))$.

Inclusion $\supseteq$ The reverse inclusion is shown similarly by induction on $\mathit{orb}(P)$. ■

Proposition 5.2.8 (Safeness)
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ be an FCP. Then $\mathit{Safe}(P_{\mathcal{FC}})$ is a safe process.

Proof
Function *Safe* is defined by $\mathit{Safe}(P_{\mathcal{FC}}) = \nu\tilde{a}.(\mathit{ren}_1(P_1) \mid \ldots \mid \mathit{ren}_n(P_n))$. We have to show that the orbits are pairwise disjoint. Take distinct indices i, j. We observe that $\mathit{orb}(\mathit{ren}_i(P_i)) = \mathit{ren}_i(\mathit{orb}(P_i))$ and $\mathit{orb}(\mathit{ren}_j(P_j)) = \mathit{ren}_j(\mathit{orb}(P_j))$ with Lemma 5.2.7. As identifiers in $\mathit{ren}_i(\mathit{orb}(P_i))$ have i as superscript while those in $\mathit{ren}_j(\mathit{orb}(P_j))$ have j, we get $\mathit{ren}_i(\mathit{orb}(P_i)) \cap \mathit{ren}_j(\mathit{orb}(P_j)) = \emptyset$. ■

The translation of $P_{\mathcal{FC}}$ into $\mathit{Safe}(P_{\mathcal{FC}})$ does not alter the behaviour of the process: both processes are bisimilar with a meaningful bisimulation relation. This relation shows that the processes reachable from $P_{\mathcal{FC}}$ and $\mathit{Safe}(P_{\mathcal{FC}})$ coincide up to the renaming of process identifiers. Thus, not only the behaviour of $P_{\mathcal{FC}}$ is preserved by $\mathit{Safe}(P_{\mathcal{FC}})$, but also the structure of the reachable process terms, in particular their fragments. Recall that two transition systems $(S, \rightsquigarrow, s_0)$, $(S', \rightsquigarrow', s_0')$ are *bisimilar*, denoted by $\approx$, if there is a bisimulation relation $\mathcal{R} \subseteq S \times S'$ that relates the initial states, i.e., $(s_0, s_0') \in \mathcal{R}$. A relation is a *bisimulation* if for all $(s, s') \in \mathcal{R}$ the following two implications hold.

(1) For all $t \in S$ with $s \rightsquigarrow t$ there is $t' \in S'$ with $s' \rightsquigarrow' t'$ and $(t, t') \in \mathcal{R}$.

(2) For all $t' \in S'$ with $s' \rightsquigarrow' t'$ there is $t \in S$ with $s \rightsquigarrow t$ and $(t, t') \in \mathcal{R}$.

Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ and $P'_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid \mathit{ren}_i(P_i) \mid \ldots \mid P_n)$. We define the relation $\mathcal{R}_i$ by $(P, Q) \in \mathcal{R}_i$ if there are $P^{sf} \equiv P$ and $Q^{sf} \equiv Q$ in standard form with

$$P^{sf} = \nu\tilde{a}.(P_1^{\neq\nu} \mid P_i' \mid P_2^{\neq\nu}) \quad \text{and} \quad Q^{sf} = \nu\tilde{a}.(P_1^{\neq\nu} \mid \mathit{ren}_i(P_i') \mid P_2^{\neq\nu}).$$

Intuitively, two processes are related if their standard forms coincide up to the identifiers in process P_i. While process P_i in $P_{\mathcal{FC}}$ uses the original identifiers, $\mathit{ren}_i(P_i)$ in $P'_{\mathcal{FC}}$ has identifiers with i as superscript.

Proposition 5.2.9
For any $i \in \{1, \ldots, n\}$, the relation $\mathcal{R}_i$ is a bisimulation relating $P_{\mathcal{FC}}$ and $P'_{\mathcal{FC}}$.

Proof
Without loss of generality, assume that process P_2 is renamed, i.e., we have the relation $\mathcal{R}_2$. Consider a pair $(P, Q) \in \mathcal{R}_2$.

Case $P \to P'$ We have to show that the reaction can be imitated by Q, i.e., $Q \to Q'$ so that P' and Q' are related by $\mathcal{R}_2$. As P is structurally congruent with P^{sf}, we have $P^{sf} \to P'$ by Rule (Struct).

Proposition 2.1.38 shows that there are three possible reactions for P^{sf}: a process performs a τ action, some process identifier calls its defining equation, or two processes communicate. We consider the latter case, where we assume that the first two processes communicate, i.e.,

$$\begin{aligned} P^{sf} &= \nu\tilde{a}.(M_1 + \overline{a}\langle b\rangle.P_1' + N_1 \mid M_2 + a(x).P_2' + N_2 \mid P_{rem}^{\neq\nu}) \\ P' &\equiv \nu\tilde{a}.(P_1' \mid P_2'\{b/x\} \mid P_{rem}^{\neq\nu}). \end{aligned}$$

We have to show that $Q \to Q'$ so that $(P', Q') \in \mathcal{R}_2$. By definition of $\mathcal{R}_2$, we have $Q \equiv Q^{sf}$ with

$$\begin{aligned} Q^{sf} &= \nu\tilde{a}.(M_1 + \overline{a}\langle b\rangle.P_1' + N_1 \mid ren_2(M_2 + a(x).P_2' + N_2) \mid P_{rem}^{\neq\nu}) \\ &= \nu\tilde{a}.(M_1 + \overline{a}\langle b\rangle.P_1' + N_1 \mid ren_2(M_2) + a(x).ren_2(P_2') + ren_2(N_2) \mid P_{rem}^{\neq\nu}) \\ &\to \nu\tilde{a}.(P_1' \mid ren_2(P_2')\{b/x\} \mid P_{rem}^{\neq\nu}). \end{aligned}$$

The second equality holds by definition of ren_2. Lemma 5.2.3 shows that renaming and applications of substitutions are compatible. Thus, we can push the substitution inside the renaming to get the equality:

$$\nu\tilde{a}.(P_1' \mid ren_2(P_2')\{b/x\} \mid P_{rem}^{\neq\nu}) = \nu\tilde{a}.(P_1' \mid ren_2(P_2'\{b/x\}) \mid P_{rem}^{\neq\nu}) =: Q'.$$

To relate P' and Q' by $\mathcal{R}_2$, the processes

$$\nu\tilde{a}.(P_1' \mid P_2'\{b/x\} \mid P_{rem}^{\neq\nu}) \quad \text{and} \quad \nu\tilde{a}.(P_1' \mid ren_2(P_2'\{b/x\}) \mid P_{rem}^{\neq\nu})$$

need to be in standard form, i.e., the topmost operators of P_1' and $P_2'\{b/x\}$ need to be different from restriction. Assume $P_1' = \nu\tilde{a}_1.P_1''$, where either $P_1'' = K\lfloor\tilde{a}\rfloor$ or $P_1'' = M^{\neq\mathbf{0}}$. (Recall that in FCPs the processes P_i do not use the parallel composition operator.) Similarly, let $P_2' = \nu\tilde{a}_2.P_2''$. By disjointness of bound and free names in Convention 2.1.11, we can extrude the scopes of $\tilde{a}_1$ and $\tilde{a}_2$. With $\nu a.P \equiv P$ if $a \notin fn(P)$, we remove those names from $\tilde{a}, \tilde{a}_1$, and $\tilde{a}_2$ that are not free in $P_1'' \mid P_2''\{b/x\} \mid P_{rem}^{\neq\nu}$. This results in $\tilde{a}', \tilde{a}_1', \tilde{a}_2'$:

$$\begin{aligned} & & \nu\tilde{a}.(P_1' \mid P_2'\{b/x\} \mid P_{rem}^{\neq\nu}) \\ &(\text{ Scope extrusion }) & \equiv \nu\tilde{a}.\tilde{a}_1.\tilde{a}_2.(P_1'' \mid P_2''\{b/x\} \mid P_{rem}^{\neq\nu}) \\ &(\ \nu a.P \equiv P \text{ if } a \notin fn(P)\ ) & \equiv \nu\tilde{a}'.\tilde{a}_1'.\tilde{a}_2'.(P_1'' \mid P_2''\{b/x\} \mid P_{rem}^{\neq\nu}). \end{aligned}$$

We treat $\nu\tilde{a}.(P_1' \mid ren_2(P_2'\{b/x\}) \mid P_{rem}^{\neq\nu})$ similarly to get

$$\begin{aligned} & \nu\tilde{a}.(P_1' \mid ren_2(P_2'\{b/x\}) \mid P_{rem}^{\neq\nu}) \\ \equiv \quad & \nu\tilde{a}'.\tilde{a}_1'.\tilde{a}_2'.(P_1'' \mid ren_2(P_2''\{b/x\}) \mid P_{rem}^{\neq\nu}). \end{aligned}$$

Since $P' \equiv \nu\tilde{a}'.\tilde{a}_1'.\tilde{a}_2'.(P_1'' \mid P_2''\{b/x\} \mid P_{rem}^{\neq\nu})$ and similarly process Q' is structurally congruent with $\nu\tilde{a}'.\tilde{a}_1'.\tilde{a}_2'.(P_1'' \mid ren_2(P_2''\{b/x\}) \mid P_{rem}^{\neq\nu})$ and since these processes are in standard form, we conclude $(P', Q') \in \mathcal{R}_2$.

The imitation of reactions $Q \to Q'$ by P can be proved in a similar way. Also the proof that the initial processes $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ and $P'_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid ren_2(P_2) \mid \ldots \mid P_n)$ are related by $\mathcal{R}_2$ is similar to the latter part of this proof, but requires scope extrusion for all processes P_1 to P_n. ■

By transitivity of bisimilarity, Proposition 5.2.9 allows for renaming several P_i and still getting a bisimilar process. In particular, renaming all n processes in $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ yields the result for the safe system $Safe(P_{\mathcal{FC}})$.

Theorem 5.2.10 (Bisimilarity)
For every finite control process $P_{\mathcal{FC}}$, the transition systems of $P_{\mathcal{FC}}$ and $Safe(P_{\mathcal{FC}})$ are bisimilar, $\mathcal{T}(P_{\mathcal{FC}}) \approx \mathcal{T}(Safe(P_{\mathcal{FC}}))$.

The transition systems of $Safe(P_{\mathcal{FC}})$ and $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ are isomorphic. Hence, by transitivity of bisimilarity, the following corollary of Theorem 5.2.10 and Theorem 3.4.3 holds.

Corollary 5.2.11
The transition systems of a finite control process $P_{\mathcal{FC}}$ and the structural semantics $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ are bisimilar, $\mathcal{T}(P_{\mathcal{FC}}) \approx \mathcal{T}(\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!])$.

Example 5.2.12 (Bisimilarity)
As the transition systems of $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ and $P_{\mathcal{FC}}$ are isomorphic and those of $P_{\mathcal{FC}}$ and $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ bisimilar, by transitivity of bisimilarity the transition systems of $\mathcal{N}[\![P_{\mathcal{FC}}]\!]$ in Figure 5.2 and $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ in Figure 5.3 are bisimilar. ♦

The corollary shows that one can reason about the *behaviour* of $P_{\mathcal{FC}}$ using $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$. Consider a process Q reachable from $P_{\mathcal{FC}}$. We argue that also the structure of Q is preserved, first by the translation of $P_{\mathcal{FC}}$ to the safe process $Safe(P_{\mathcal{FC}})$, and then by the translation of $Safe(P_{\mathcal{FC}})$ to the safe Petri net $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$. With this result we can also reason about the *structure of all processes reachable from* $P_{\mathcal{FC}}$ using $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$.

With Theorem 5.2.10, $P_{\mathcal{FC}}$ and $Safe(P_{\mathcal{FC}})$ are bisimilar via the relation $\mathcal{R}_1 \circ \cdots \circ \mathcal{R}_n$, e.g. a process $Q = \nu a.\nu b.(K\lfloor a \rfloor \mid K\lfloor a \rfloor \mid L\lfloor b \rfloor)$ reachable from $P_{\mathcal{FC}}$ corresponds to $Q' = \nu a.\nu b.(K^1\lfloor a \rfloor \mid K^2\lfloor a \rfloor \mid L^3\lfloor b \rfloor)$ reachable from $Safe(P_{\mathcal{FC}})$.

Hence, one can reconstruct the fragments of Q form those of Q'. For example, computing the restricted forms for the processes above yields:

$$\begin{aligned} rf(Q) &= \nu a.(K\lfloor a\rfloor \mid K\lfloor a\rfloor) \mid \nu b.L\lfloor b\rfloor \\ rf(Q') &= \nu a.(K^1\lfloor a\rfloor \mid K^2\lfloor a\rfloor) \mid \nu b.L^3\lfloor b\rfloor. \end{aligned}$$

Dropping the superscripts in $rf(Q')$ yields the fragments in $rf(Q)$, since only the restricted names influence the restricted form, not the process identifiers.

The transition systems of $Safe(P_{\mathcal{FC}})$ and $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ are isomorphic with Theorem 3.4.3, e.g. Q' corresponds to marking $M([\nu a.(K^1\lfloor a\rfloor \mid K^2\lfloor a\rfloor)]) = 1$, $M([\nu b.L^3\lfloor b\rfloor]) = 1$, and $M([F]) = 0$ otherwise. Thus, from a marking of the net $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ one can obtain the restricted form of a reachable process in $Safe(P_{\mathcal{FC}})$, which in turn corresponds to the restricted form in $P_{\mathcal{FC}}$ (when the superscripts of process identifiers are dropped).

5.3 Optimality of the Translation

We discuss our choice to rename all P_i in $\nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ to gain a safe process. One might be tempted to improve our translation by renaming only a subset of processes P_i whose orbits intersect with many others, in hope to get a smaller specification than $Safe(P_{\mathcal{FC}})$. We show that this idea does not work, and the resulting specification will be of the same size, i.e., our definition of $Safe(P_{\mathcal{FC}})$ is *optimal*. First, we illustrate this issue on an example.

Example 5.3.1
Let $P = \tau.K\lfloor\tilde{a}\rfloor + \tau.L\lfloor\tilde{a}\rfloor$, $R = K\lfloor\tilde{a}\rfloor$, and $S = L\lfloor\tilde{a}\rfloor$, where $K(\tilde{x}) := Def_1$ and $L(\tilde{x}) := Def_2$. Consider the process $P \mid R \mid S$. The orbits of P and R as well as P and S intersect.

Renaming of P yields $ren_1(P) \mid R \mid S = \tau.K^1\lfloor\tilde{a}\rfloor + \tau.L^1\lfloor\tilde{a}\rfloor \mid R \mid S$, where $K^1(\tilde{x}) := ren_1(Def_1)$ and $L^1(\tilde{x}) := ren_1(Def_2)$. This means we create additional copies of the shared identifiers K and L.

The renaming of R and S yields $P \mid ren_1(R) \mid ren_2(S) = P \mid K^1\lfloor\tilde{a}\rfloor \mid L^2\lfloor\tilde{a}\rfloor$, where we create new defining equations for the identifiers K^1 and L^2. The size of the translation is the same. ⧫

This illustrates that any renaming of processes P_i where the orbits overlap results in a specification of the same size. To render this intuition precisely, we call $K^k(\tilde{x}) := ren_k(Q)$ a *copy of the equation* $K(\tilde{x}) := Q$, for any $k \in \mathbb{N}$. We also count $K(\tilde{x}) := Q$ as a copy of itself.

Proposition 5.3.2 (Necessary Condition for Safeness)
The number of copies of an equation $K(\tilde{x}) := Q$ necessary to get a safe process from $P_{\mathcal{FC}}$ equals to the number of orbits that contain K.

Proof
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$. If there are less copies of the equation $K(\tilde{x}) := Q$ than orbits containing K, by the pigeonhole principle there are at least two orbits $ren_k(P_i)$ and $ren_k(P_j)$ sharing one identifier K^k. By definition, the resulting process is not safe. If there are more copies than intersecting orbits, some identifiers do not belong to any orbit. Every process $ren_i(P_i)$ only calls the identifiers in its orbit, i.e., $K^i \in orb(ren_i(P_i))$. The remaining K^j that do not belong to any orbit are never used. The corresponding equations $K^j(\tilde{x}) := ren_j(Q)$ can be removed from the specification. ■

Now we show that our translation provides precisely this minimal number of copies of defining equations for every identifier, i.e., that it is optimal.

Proposition 5.3.3 (Optimality of Our Translation)
Our translation $Safe(P_{\mathcal{FC}})$ provides as many copies of an equation $K(\tilde{x}) := Q$ as there are orbits containing K.

Proof
Let $P_{\mathcal{FC}} = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$ and let the identifier K be contained in k orbits, without loss of generality $K \in orb(P_1) \cap \ldots \cap orb(P_k)$. By definition of *Safe*, we rename P_i to $ren_i(P_i)$. As K is in the orbit of P_i, the new identifier K^i is in the orbit of $ren_i(P_i)$ for all $1 \leq i \leq k$. The defining equations are $K^i(\tilde{x}) := ren_i(Q)$ where $K(\tilde{x}) := Q$. As K is not contained in any other orbit, no further copies of $K(\tilde{x}) := Q$ are added. Thus, we have $k + 1$ copies of the defining equation $K(\tilde{x}) := Q$. By definition, we remove the original equation and get the desired equality. ■

Remark 5.3.4
Note that one can, in general, optimise the translation by performing dynamic rather than syntactic analyses, and produce a smaller process whose corresponding Petri net is safe. However, since our notion of a safe process is syntactic the resulting process will not be safe according to our definition. ♦

5.4 Unfolding-based Model Checking

In the previous sections, we showed how to translate FCPs into safe Petri nets. For safe Petri nets, efficient algorithms are known to compute the finite and complete prefix of the unfolding (cf. Section 2.2.3 and [EH08]). In this section, we present a state-of-the-art procedure to establish properties for a safe Petri net with help of the finite and complete prefix. The idea is to reduce the verification task to a Boolean satisfiability (SAT) problem (cf. Figure 5.1). Off-the-shelf SAT solvers can then be used in black-box fashion to solve the problem. We remark that the technique presented in this section is due to Heljanko [Hel02] and Khomenko et. al. [KKY04] and is also described in [EH08].

Given a finite and complete prefix and a property to be established, the encoding into SAT proceeds in two steps. First, a so-called *configuration constraint* $\phi_{\mathcal{C}}$ is computed. It contains a variable x_e for every event e. The configuration constraint ensures that a satisfying assignment of the variables yields a configuration of the prefix. More precisely, the set $\{e \mid x_e = \mathit{true}\}$ is a configuration.

The *violation constraint* $\phi_{\mathcal{V}}$ expresses all violations of the property on the given prefix, i.e., it depends on property as well as prefix. The formula to be checked for satisfiability is the conjunction $\phi_{\mathcal{C}} \wedge \phi_{\mathcal{V}}$. It is unsatisfiable if and only if the property of interest holds. If a satisfying assignment is found, then the configuration $\{e \mid x_e = \mathit{true}\}$ yields a run in the Petri net which violates the property.

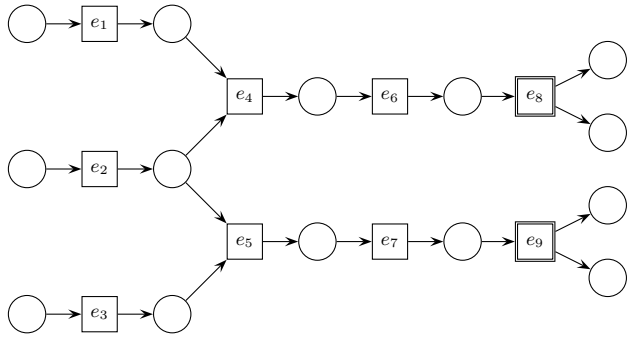
Technically, the configuration constraint $\phi_{\mathcal{C}}$ is a conjunction of

$$\bigwedge_{e \in E \setminus E_{cut}} \bigwedge_{f \in {}^{\bullet\bullet}e} (\neg x_e \vee x_f) \qquad \text{and} \qquad \bigwedge_{e \in E \setminus E_{cut}} \bigwedge_{f \in \mathit{Confl}_e} (\neg x_e \vee \neg x_f),$$

where $\mathit{Confl}_e := \{(({}^{\bullet}e)^{\bullet} \setminus \{e\}) \setminus E_{cut}\}$ is the set of non-cut-off events which are in the direct conflict relation with e. The first formula is a set of implications $x_e \Rightarrow x_f$. It ensures that a satisfying assignment to the variables yields a causally closed set of events, i.e., if it contains event e then it also contains the immediate predecessors $f \in {}^{\bullet\bullet}e$. The second formula consists of a number of implications $x_e \Rightarrow \neg x_f$, which ensures that the resulting set of events is conflict-free. Note that $\phi_{\mathcal{C}}$ is given in *conjunctive normal form*, the input format of most SAT solvers.

A Petri net is in a deadlock, if no transition is enabled. In the prefix, there are two reasons for which an event e cannot be executed. Either some predecessor $f \in {}^{\bullet\bullet}e$ has not fired or some event consuming tokens from ${}^{\bullet}e$ has been executed. These considerations lead to the following violation constraint $\phi_{\mathcal{V}}$ for deadlock-freedom:

$$\bigwedge_{e \in E} \Big(\bigvee_{f \in {}^{\bullet\bullet}e} \neg x_f \vee \bigvee_{f \in ({}^{\bullet}e)^{\bullet} \setminus E_{cut}} x_f \Big).$$



(a)

$$(\neg x_{e_4} \vee x_{e_1}) \wedge (\neg x_{e_4} \vee x_{e_2}) \wedge (\neg x_{e_5} \vee x_{e_2}) \wedge (\neg x_{e_5} \vee x_{e_3}) \wedge (\neg x_{e_6} \vee x_{e_4}) \wedge (\neg x_{e_7} \vee x_{e_5}) \wedge (\neg x_{e_4} \vee \neg x_{e_5})$$

(b)

$$x_{e_1} \wedge x_{e_2} \wedge x_{e_3} \wedge (\neg x_{e_1} \vee \neg x_{e_2} \vee x_{e_4} \vee x_{e_5}) \wedge (\neg x_{e_2} \vee \neg x_{e_3} \vee x_{e_4} \vee x_{e_5}) \wedge (\neg x_{e_4} \vee x_{e_6}) \wedge (\neg x_{e_5} \vee x_{e_7}) \wedge \neg x_{e_6} \wedge \neg x_{e_7}$$

(c)

Figure 5.4:
A finite and complete prefix of the structural semantics in Figure 5.3 and its SAT encoding as configuration constraint $\phi_{\mathcal{C}}$ (b). Part (c) is a violation constraint $\phi_{\mathcal{V}}$ for the prefix, which represents all possible deadlock situations. As only event names are important for the encodings, conditions and labels are omitted.

Example 5.4.1 ($\phi_{\mathcal{C}} \wedge \phi_{\mathcal{V}}$)
Consider the safe Petri net $\mathcal{N}[\![Safe(P_{\mathcal{FC}})]\!]$ of the client/server system in Figure 5.3. A finite and complete prefix of the net is depicted in part (a) of Figure 5.4. The configuration constraint $\phi_{\mathcal{C}}$ for the prefix is part (b), the violation constraint part (c) of the picture. ⧫

The presented approach is not limited to verification of deadlock-freedom but applies to any property which can be rephrased in terms of (un)reachability. To check for reachability of markings, also variables for conditions are introduced. For details we refer to [Hel02, KKY04, EH08].

5.5 Experimental Results

To demonstrate the practicality of our approach, Tim Strazny implemented the translation of FCPs to safe processes in the tool PETRUCHIO [SM08]. In this section, we present the results from applying our tool chain to check three series of benchmarks for deadlocks. We compare the efficiency of our unfolding-based verification approach with other well-known approaches and tools for π-Calculus verification. The tables in this section are taken from [MKS09]. The corresponding verification experiments have been conducted by Tim Strazny.

The *NESS (Newcastle E-Learning Support System)* example models an electronic course work submission system. This series of benchmarks is taken from [KKN06], where the only other unfolding-based verification technique for the π-Calculus is presented. The approach described in [KKN06] is limited to recursion-free processes (cf. Definition 2.1.4). It translates a process into a high-level Petri net using the results in [DKK06a] and model checks the latter. As discussed in Section 3.6, the translation to Petri nets in [DKK06a, KKN06] is very different from our approach, and a high-level net unfolder is used there for verification, while our technique uses the standard unfolding procedure for safe low-level nets. Moreover, our technique is not limited to recursion-free processes.

The model consists of a teacher process T composed in parallel with k students S (the system can be scaled up by increasing the number of students) and an environment process ENV. Every student has its own local channel for communication, h_i, and all students share the channel h:

$$\nu h, h_1, \ldots, h_k.(T\lfloor nessc, h_1, \ldots, h_k\rfloor \mid \Pi_{i=1}^{k} S\lfloor h, h_i\rfloor \mid ENV\lfloor nessc\rfloor).$$

The students are supposed to submit their work for assessment to *NESS*. The teacher passes the channel *nessc* of the system to all students, $\overline{h_i}\langle nessc\rangle$, and then waits for the confirmation that they have finished working on the assignment, $h_i(x)$. After receiving the ness channel, $h_i(nsc)$, students organise themselves in pairs. To do so, they send their local channel h_i on h and at the same time listen on h to receive a partner, $\overline{h}\langle h_i\rangle \ldots + h(x) \ldots$ When they finish, exactly one student of each pair sends two channels to the support system, $\overline{nsc}\langle h_i\rangle.\overline{nsc}\langle x\rangle$, which give access to their completed joint work. These channels are received by the ENV process. The students finally notify the teacher about completion of their work, $\overline{h_i}\langle fin\rangle$. Thus, the system is modelled by:

$$\begin{aligned} T(nessc, h_1, \ldots, h_k) &:= \Pi_{i=1}^{k} \overline{h_i}\langle nessc\rangle.h_i(x_i) \\ S(h, h_i) &:= h_i(nsc).(\overline{h}\langle h_i\rangle.\overline{h_i}\langle fin\rangle + h(x).\overline{nsc}\langle h_i\rangle.\overline{nsc}\langle x\rangle.\overline{h_i}\langle fin\rangle) \\ ENV(nessc) &:= nessc(y_1) \ldots nessc(y_k). \end{aligned}$$

In the following Tables 5.1 and 5.2, the row **ns**k gives the verification results for the *NESS* system with $k \in \mathbb{N}$ students. The property we verified was whether all

processes successfully terminate by reaching the end of their individual code (as distinguished from a deadlock, where some processes are stuck in the middle of their intended behaviour, waiting for a communication to occur). Obviously, the system successfully terminates iff the number of students is even, i.e., they can be organised into pairs. The **dns**k entries refer to a refined *NESS* model where the pairing of students is deterministic; thus the number of students is even, and these benchmarks are deadlock-free.

Model	FCP Size	HLNet |P|	HLNet |T|	Model Checking unf	Model Checking |B|	Model Checking |E*|	Model Checking sat	mwb dl	hal π2fc
dns4	84	1433	511	6	10429	181	< 1	10	93
dns6	123	3083	1257	46	28166	342	< 1	—	—
dns8	162	5357	2475	354	58863	551	< 1	—	—
dns10	201	8255	4273	—	—	—	—	—	—
dns12	240	11777	6791	—	—	—	—	—	—
ns2	61	157	200	1	5553	127	< 1	< 1	< 1
ns3	88	319	415	7	22222	366	< 1	1	8
ns4	115	537	724	69	101005	1299	1	577	382
ns5	142	811	1139	532	388818	4078	58	—	—
ns6	169	1141	1672	—	—	—	—	—	—
ns7	196	1527	2335	—	—	—	—	—	—

Table 5.1:
Experimental results for verification of the NESS benchmarks with the approach presented in [KKN06], the MWB [VM94], and HAL [FGMP03].

The second example is a client-server system similar to our running example. For a more realistic model, we extend the server to spawn separate sessions that handle the clients' requests. We change the server process in Example 5.0.1 to a more concurrent *CONCS* by adding separate session processes:

$$\begin{aligned} CONCS(url, getses) &:= url(y).getses(s).\overline{y}\langle s\rangle.CONCS\lfloor url, getses\rfloor \\ SES(getses) &:= \nu ses.\overline{getses}\langle ses\rangle.\overline{ses}\langle ses\rangle.SES\lfloor getses\rfloor. \end{aligned}$$

On a client's request, the server creates a new session object using the *getses* channel, $getses(s)$. A session object is modelled by an *SES* process. It sends its private channel νses along the *getses* channel to the server. The server forwards the session to the client, $\overline{y}\langle s\rangle$, which establishes the private session, and becomes available for further requests. This case study uses recursion and is scalable in the number of clients and the number of sessions. In Table 5.3, e.g., the entry 5s5c gives the verification results for the system with five *SES* processes, five *C* processes and one server. All these benchmarks are deadlock-free.

The last example is the well-known specification of the handover procedure in the GSM Public Land Mobile Network. We use the standard π-Calculus model with one mobile station, two base stations, and one mobile switching centre presented by Orava and Parrow in [OP92].

Model	FCP Size	Struct |P|	Struct |T|	Struct B	Safe Size	Struct |P|	Struct |T|	Model Checking unf	Model Checking |B|	Model Checking |E*|	Model Checking sat
dns4	99	22	47	8	115	32	50	< 1	113	38	< 1
dns6	145	32	94	12	169	48	99	< 1	632	159	< 1
dns8	191	42	157	16	233	64	164	< 1	3763	745	< 1
dns10	237	52	236	20	277	80	239	1	22202	3656	2
dns12	283	62	331	24	331	96	286	56	128295	18192	62
ns2	73	18	28	4	81	26	40	< 1	61	27	< 1
ns3	105	37	91	6	117	56	141	< 1	446	153	< 1
ns4	137	68	229	8	153	102	364	< 1	5480	1656	< 1
ns5	169	119	511	10	189	172	815	17	36865	7832	3
ns6	201	206	1087	12	225	282	1722	1518	377920	65008	84
ns7	233	361	2297	14	261	646	3605	—	—	—	—
ns2-r	72	16	24	4	80	24	36	< 1	51	22	< 1
ns3-r	104	29	70	6	116	48	117	< 1	292	99	< 1
ns4-r	134	45	123	8	150	79	216	< 1	1257	392	< 1
ns5-r	166	66	241	10	186	119	435	2	10890	2635	1
ns6-r	198	91	418	12	222	167	768	123	107507	19892	31
ns7-r	230	120	666	14	258	223	1239	—	—	—	—

Table 5.2:
Experimental results for verification of the NESS benchmarks with the approach presented in this chapter.

We compare our results with three other techniques for π-Calculus verification: the mentioned approach in [KKN06], the verification kit HAL [FGMP03], and the *mobility workbench* (MWB) [VM94]. HAL translates a π-Calculus process into an HD-automaton [Pis99]. This in turn is translated into a finite automaton which is checked using standard tools. The MWB does not use any automata translation, but builds the state space on the fly. These tools can verify various properties (cf. Section 5.6), but we perform our experiments for deadlock checking as it is the common denominator.

We briefly comment on the role of the models with the suffix $-r$ in Table 5.2. One can observe that parallel compositions inside a fragment lead to interleaving diamonds in our Petri net representation. Thus, restricted names that are known to a large number of processes can make the size of our Petri net translation grow dramatically (cf. Section 4.5). We demonstrate this effect by verifying some of the *NESS* benchmarks with and without (suffix $-r$ in the table) the restrictions on such critical names. Even with the critical restrictions our approach outperforms the other tools. But when such restrictions are removed, it becomes orders of magnitude faster. (Removing such critical restrictions does not alter the process behaviour: $\nu a.P$ reacts to $\nu a.P'$ iff P reacts to P'. Thus, one can replace $\nu a.P$ by P for model checking purposes. Note that this holds only for active restrictions in the initial process, not for those within recursive definitions.)

The columns in Tables 5.1, 5.2, and 5.3 are organised as follows. `FCP Size` gives the size of the process as defined in Section 2.1.1. The following two columns, `HLNet` and `Model Checking` (present only in Table 5.1), are the verification results when the approach in [KKN06] is applied. In the former column, $|P|$ and $|T|$ state

Model	FCP Size	mwb dl	hal π2fc	Struct |P|	Struct |T|	Struct B	Safe Size	Struct |P|	Struct |T|	Model Checking unf	Model Checking |B|	Model Checking |E*|	Model Checking sat
gsm	214	—	18	374	138	1	286	148	344	< 1	345	147	< 1
gsm-r	213	n/a	n/a	60	72	1	285	75	110	< 1	150	72	< 1
1s1c	48	—	< 1	11	13	1	48	12	15	< 1	17	9	< 1
1s2c	52	—	6	12	15	2	63	22	30	< 1	35	17	< 1
2s1c	52	—	2	20	31	2	61	22	35	< 1	37	18	< 1
2s2c	56	—	138	31	59	2	76	40	66	< 1	73	33	< 1
3s2c	60	—	—	68	159	3	89	66	128	< 1	137	57	< 1
3s3c	64	—	—	85	217	3	104	100	194	< 1	216	87	< 1
4s4c	72	—	—	362	1202	4	132	216	484	< 1	537	195	< 1
5s5c	80	—	—	980	3818	5	160	434	1132	< 1	1238	403	< 1

Table 5.3: Experimental results for GSM and client-server benchmarks.

the number of places and transitions in the high-level Petri net. The following column `unf` gives the time to compute the unfolding prefix of this net. We measure all runtimes in seconds. For this prefix, $|B|$ is the number of conditions, and $|E^*|$ is the number of events (excluding cut-offs). Like our technique, [KKN06] employs a SAT solver whose runtime is given in the `sat` column. The following two columns, `mwb dl` and `hal π2fc`, give the runtimes for the deadlock checking algorithm in MWB and for converting a π-Calculus process into a finite automaton (via HD-automata). The latter includes the translation of a π-Calculus process into an HD-automaton, minimisation of this HD-automaton, and the conversion of the minimised HD-automaton into a finite automaton [FGMP03]. The entries in Table 5.2 are the results of applying our model checking procedure. The column `Struct` gives the numbers of places and transitions and the bounds of the Petri nets corresponding to a direct translation of the FCPs. These nets are given only for comparison, and are not used for model checking. `Safe Size` gives the size of the safe process computed by the function *Safe* described in Section 5.2, and the next column gives the numbers of places and transitions of the corresponding safe Petri nets. Note that these nets, unlike those in [KKN06], are the usual low-level Petri nets. The following columns give the unfolding times, the prefix sizes, and the times for checking deadlocks on the prefixes using a SAT solver. A '−' in the tables indicates the corresponding tool did not produce an output within 30 minutes, and an 'n/a' means the technique was not applicable to the example.

Tables 5.1 and 5.2 illustrate the results for checking the *NESS* example with the different techniques. As the MWB requires processes where all names are restricted, we cannot check the '$-r$' versions of the case studies. Our runtimes are orders of magnitude smaller in comparison with HAL and MWB, and are much better compared with the approach in [KKN06]. Furthermore, they dramatically improve when the critical names are removed (the '$-r$' models).

The approach in [KKN06] only applies to recursion-free processes, so one cannot check the client-server or the GSM benchmarks with that technique.

Table 5.3 shows that the proposed technique dramatically outperforms MWB and HAL, and handles the benchmark with five sessions and clients in a second.

5.6 Related Work and Conclusion

We have proposed a practical approach for verification of finite control processes. It works by first translating the given FCP into a safe process, and then translating the latter with the structural semantics into a safe Petri net, for which unfolding-based model checking is performed. The translation to safe processes exploits a general boundedness result for FCP nets based on the developed theory of orbits. Our experiments show that this approach has significant advantages over other existing tools for verification of dynamically reconfigurable systems in terms of memory consumption and runtime. We summarise the outcomes of research on automatic verification tools for the π-Calculus and identify potential directions for future research on our verification approach.

Based on the theory of HD-automata, model and bisimulation checking tools for the π-Calculus are implemented in the HAL toolkit [FGMP03]. Finite HD-automata are translated into ordinary finite automata, which makes finite state verification tools applicable. The π-logic, defined for model checking, is capable of referring to the identities of names. *Strong early* bisimilarity checking is performed by checking bisimilarity of the finite automata [MP95a].

An extension of the modal μ-Calculus to cope with name creation and passing is proposed in [Dam96], together with a sound and complete proof system and a tableau-based model checking algorithm for finite control processes. The algorithm is integrated in the MWB—initially designed to decide the *open* bisimilarities [SW01] between finite control processes [VM94].

The spatial logic of Caires and Cardelli [CC03] specifies structural as well as behavioural properties of processes. It contains operators to refer to subprocesses, environments, and the freshness of names. A model checking algorithm, which is complete for the class of bounded processes, is available [Cai04].

Our approach to verification of π-Calculus using unfoldings outperforms the established tools on the property of deadlock-freedom and there is hope that it will also speed up the verification of the mentioned properties. In particular, we started to investigate a spatial logic inspired by [CC03] that can be compiled down to a temporal logic for Petri nets, i.e., we reduce the model checking problem whether process P satisfies a spatial logic formula $\phi_{\mathcal{P}}$ to whether $\mathcal{N}[\![P]\!]$ satisfies $\theta_{\mathcal{PN}}(\phi_{\mathcal{P}})$ in an classical temporal logic for nets [Lin08]. The advantage of this translation-based approach is that the existing results in Petri net theory help us judge hardness and decidability of the model checking problem [Esp97a]. We plan to compare its efficiency with the direct model checking algorithm in the Spatial Logic Model Checker [Cai04].

After the translation into a safe process, some fragments differ only by the replicated process identifiers. Such fragments are equivalent in the sense that they react in the same way and generate equivalent places in the postsets of the transitions. Hence, it should be possible to optimise the computation of the structural semantics, because many structural congruence checks can be omitted and several computations of enabled reactions become unnecessary. Moreover, this observation allows one to use a weaker (compared with marking equality) equivalence on configurations in the unfolding procedure [Kho03]. This would produce cut-off events more often and hence reduce the size of the unfolding prefix.

It seems to be possible to generalise our translation to safe processes to a wider subclass of structurally stationary processes. For example, consider the process $S\lfloor url\rfloor \mid C\lfloor url\rfloor \mid C\lfloor url\rfloor$ modelling a *concurrent* server and two clients, with the corresponding process identifiers defined as

$$\begin{aligned} S(url) &:= url(y).(\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle \mid S\lfloor url\rfloor) \\ C(url) &:= \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url\rfloor. \end{aligned}$$

Intuitively, when contacted by a client, the server spawns a new session and is ready to serve another client, i.e., several clients can be served in parallel. Though this specification is not an FCP, it still results in a 2-bounded Petri net very similar to the one in Figure 5.2. Our method can still be used to convert it into a safe Petri net for subsequent verification.

6 Case Studies

Contents

To further evaluate the usefulness of the structural semantics for automatic verification, we consider two realistic case studies. Our Petri net translation facilitates the semi-automatic verification of non-trivial properties of various kinds, ranging from occurrence number properties over topological properties asking for connections between processes to temporal properties.

As opposed to the fully automated model checking in the previous section, the verification in this chapter is *computer-aided*. We decompose global correctness properties by hand into lemmas, which we then establish with the help of tools. This approach is more efficient than uninformed model checking as it often considers system parts instead of the full system or simpler properties. Hence, it scales better with the size of the model.

For a wide range of properties, efficient verification algorithms that only inspect the graph structure of the net are sufficient. They avoid costly state space computations and so circumvent the state explosion problem, the main drawback

of model checking. These algorithms exploit the fact that the Petri net of interest is generated from a π-Calculus process, i.e., they rely on additional knowledge about the structural semantics (e.g. that the places yield all private connections between processes).

To begin with, we verify a simplified model of a highway control system [HESV91]. Since its size is limited, it allows us to explain the arguments that prove a property. The second case study is taken from the automated manufacturing domain [BR01]. As the model comprises 195 lines of π-Calculus code, we only sketch it and report on the conducted verification. Note that without tool support, the verification of the second case study would not have been possible. To conclude the chapter, we discuss our verification approach and compare it with related work on the verification of both case studies.

6.1 Car Platooning

The highway control system we consider has been developed in the PATH project at the University of California, Berkeley [PAT86]. It has been chosen as one of the benchmark case studies for verification techniques developed in the Transregional Collaborative Research Center on *Automatic Verification and Analysis of Complex Systems (AVACS)* [AVA04], hence offering related approaches to verification we shall compare our results with. The scenario is as follows [HESV91].

A vehicle enters a highway and communicates its goal location to a central device. The device responds by sending a path the vehicle has to follow. To do so, it chooses one of three elementary manoeuvres: change the current lane, merge with preceding vehicles into so-called platoons, and split platoons. Thanks to computer-supported control, cars in platoons drive close to each other. This gives hope for better utilisation of highways and shorter travelling times when the system is installed.

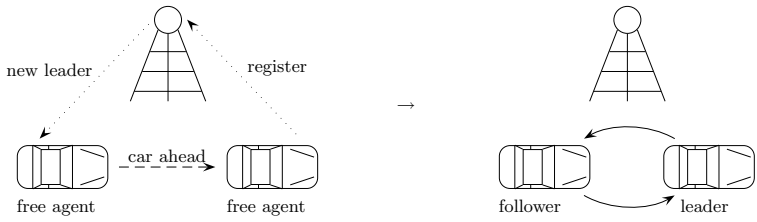


Figure 6.1: Illustration of the merge manoeuvre.

We focus on the merge manoeuvre as illustrated in Figure 6.1. For its descrip-

tion we call single vehicles *free agents*. When a free agent detects another one driving in front, depicted by a *car ahead* message, it contacts that vehicle with the request to merge. If the leading vehicle agrees, the now called *follower* speeds up to drive closely behind the *leader*. They maintain a permanent connection, indicated by the two arrows between follower and leader in Figure 6.1. The two cars form a *car platoon*. In our model, we only merge free agents whereas the original case study also merges platoons up to a given size. The theory of structural stationarity allows us to verify those systems with known bounds as well. We simplified the case study for the sake of brevity and clarity.

The verification in [HESV91] relies on proper connections between the vehicles. Therefore, the results achieved are only valid modulo this assumption. We present a model where the connections are built up appropriately with the help of central control units managing sections of the highway. These units are represented by the masts in Figure 6.1. Free agents entering a section register at the unit and receive the channel for contacting a preceding vehicle from the unit. To limit the number of messages in our case study, we let the central control unit send the car ahead message with the new leader as a parameter instead of sending two messages.

6.1.1 Modelling the Case Study

We consider the example of two free agents merging into a platoon. This case study is specified by the π-Calculus process $ENV\lfloor cfa \rfloor \mid MRG\lfloor cfa \rfloor$ in Table 6.1. The environment process ENV recursively generates new free agents. These free agents have an id, a channel ca, and a channel rq. The channel ca models a car ahead message, the request channel rq is used by a second free agent to issue a request to merge into a platoon. Free agents register at a MRG process. They pass their id to establish a private connection and then send the ca and rq channels. At some point, the MRG process sends a ca message to the second free agent that registered. It contains the rq channel of the first free agent. The agent receives this message and becomes a request process RQ. It contacts the other free agent on the rq channel and, if this agent accepts the request to merge, turns into a follower FL. The first car becomes a leader LD. This finishes the merge manoeuvre.

Remark 6.1.1
The car platoon system $ENV\lfloor cfa \rfloor \mid MRG\lfloor cfa \rfloor$ is a finite handler process as defined in Section 4.4. The distinguished public channel is cfa, free agents are participants, i.e., the call $FA\lfloor cfa \rfloor$ is of the form $K^{PT}\lfloor \tilde{p} \rfloor$ in $\mathcal{P}_{\mathcal{PT}}$, and the merge process $MRG\lfloor cfa \rfloor$ is a handler process $K^{HD}\lfloor \tilde{p} \rfloor$ in $\mathcal{P}_{\mathcal{HD}}$. Hence, by Theorem 4.4.8, the car platoon model $ENV\lfloor cfa \rfloor \mid MRG\lfloor cfa \rfloor$ is structurally stationary.

We remark that also the extended system where platoons of several cars merge can be modelled as a finite handler process. ♦

$$
\begin{array}{rcl}
ENV(cfa) & := & FA\lfloor cfa\rfloor \mid ENV\lfloor cfa\rfloor \\
MRG(cfa) & := & cfa(id_x).id_x(ca_x).id_x(rq_x). \\
 & & \quad cfa(id_y).id_y(ca_y).id_y(rq_y).\overline{ca_y}\langle rq_x\rangle.MRG\lfloor cfa\rfloor \\
FA(cfa) & := & \nu id, ca, rq.\overline{cfa}\langle id\rangle.\overline{id}\langle ca\rangle.\overline{id}\langle rq\rangle. \\
 & & \quad (\ \ ca(rqnl).RQ\lfloor id, rqnl\rfloor + rq(nf).\overline{nf}\langle id\rangle.LD\lfloor id, nf\rfloor\ \ ) \\
RQ(id, rqnl) & := & \overline{rqnl}\langle id\rangle.id(nl).FL\lfloor id, nl\rfloor.
\end{array}
$$

Table 6.1: π-Calculus model of the merge manoeuvre.

The structural semantics of $ENV\lfloor cfa\rfloor \mid MRG\lfloor cfa\rfloor$ is depicted in Figure 6.2. We explain the meanings of places and transitions. Initially, the processes $F_1 = ENV\lfloor cfa\rfloor$ and $F_4 = MRG\lfloor cfa\rfloor$ are present and so the corresponding places are marked. With transition t_1, the environment process $ENV\lfloor cfa\rfloor$ generates free agents $F_2 = FA\lfloor cfa\rfloor$. Transition t_2 represents a call to the process identifier FA, which yields

$$F_3 \quad = \quad \nu id, ca, rq.\overline{cfa}\langle id\rangle.\overline{id}\langle ca\rangle.\overline{id}\langle rq\rangle.choice.$$

We use *choice* as a shortcut, which is replaced by the choice composition

$$ca(rqnl).RQ\lfloor id, rqnl\rfloor + rq(nf).\overline{nf}\langle id\rangle.LD\lfloor id, nf\rfloor$$

to obtain the full definition of F_3. Shortcuts improve the readability of processes, they are not part of the π-Calculus syntax. The call $MRG\lfloor cfa\rfloor$, represented by transition t_3, gives

$$F_5 \quad = \quad cfa(id_x).id_x(ca_x).id_x(rq_x).reg_y.\overline{ca_y}\langle rq_x\rangle.MRG\lfloor cfa\rfloor,$$

where reg_y abbreviates

$$cfa(id_y).id_y(ca_y).id_y(rq_y).$$

With t_4 the first free agent passes its *id* to the MRG process. The transition consumes a token for a free agent F_3, and the token for the MRG process F_5, and produces a fragment that contains a free agent and the MRG process, F_6. With transitions t_5 and t_6, the free agent continues to pass its *ca* and *rq* channels, resulting in fragments F_7 and F_8:

$$
\begin{array}{rcl}
F_6 & = & \nu id.\big(\nu ca, rq.\overline{id}\langle ca\rangle.\overline{id}\langle rq\rangle.choice \\
 & & \quad \mid id(ca_x).id(rq_x).reg_y.\overline{ca_y}\langle rq_x\rangle.MRG\lfloor cfa\rfloor\big)
\end{array}
$$

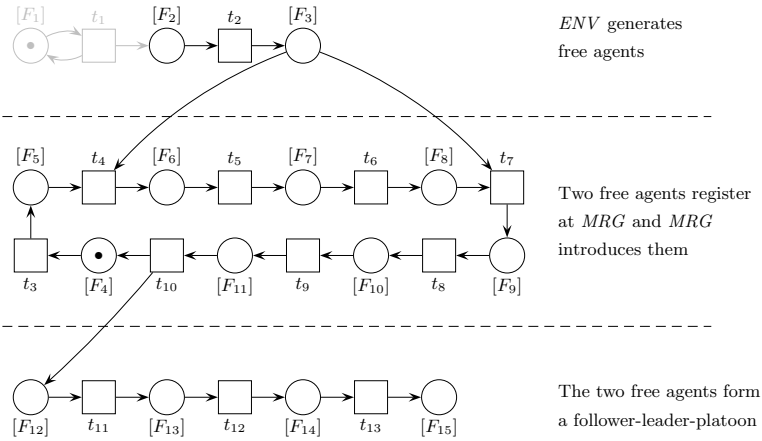


Figure 6.2:
Structural semantics $\mathcal{N}[\![ENV\lfloor cfa\rfloor \mid MRG\lfloor cfa\rfloor]\!]$ of the merge manoeuvre. The fragment definitions are given in the text. Cutting away the gray part yields a subnet where the number of cars stays constant.

$$F_7 = \nu id.\big(\nu ca, rq.\overline{id}\langle rq\rangle.choice \mid id(rq_x).reg_y.\overline{ca_y}\langle rq_x\rangle.MRG\lfloor cfa\rfloor\big)$$
$$F_8 = \nu rq.\big(\nu id, ca.choice \mid reg_y.\overline{ca_y}\langle rq\rangle.MRG\lfloor cfa\rfloor\big).$$

The registration of the second free agent, which yields F_9, F_{10}, and F_{11} given below, is similar. We use α-conversion to rename the id, ca, and rq channels of the first free agent to id_1, ca_1, and rq_1. The shortcuts $choice_1$ and $choice_2$ correspond to $choice$ with those names changed accordingly:

$$\begin{aligned}
F_9 &= \nu id_2.\big(\ \nu rq_1.(\nu id_1, ca_1.choice_1 \mid id_2(ca_y).id_2(rq_y).\overline{ca_y}\langle rq_1\rangle.MRG\lfloor cfa\rfloor) \\
&\qquad \mid \nu ca_2, rq_2.\overline{id_2}\langle ca_2\rangle.\overline{id_2}\langle rq_2\rangle.choice_2\ \big) \\
F_{10} &= \nu ca_2, id_2.\big(\ \nu rq_1.(\nu id_1, ca_1.choice_1 \mid id_2(rq_y).\overline{ca_2}\langle rq_1\rangle.MRG\lfloor cfa\rfloor) \\
&\qquad \mid \nu rq_2.\overline{id_2}\langle rq_2\rangle.choice_2\ \big) \\
F_{11} &= \nu ca_2.\big(\ \nu rq_1.(\nu id_1, ca_1.choice_1 \mid \overline{ca_2}\langle rq_1\rangle.MRG\lfloor cfa\rfloor) \\
&\qquad \mid \nu id_2, rq_2.(ca_2(rqnl).RQ\lfloor id_2, rqnl\rfloor + \ldots)\ \big).
\end{aligned}$$

In F_{11}, the MRG process is ready to pass the rq_1 channel of the leading car to the second free agent. Since it uses the ca_2 channel, we say that the MRG process *sends a car ahead message*. Afterwards, the process $MRG\lfloor cfa\rfloor$ forgets the restricted names of both free agents and the fragment is split up. The free

agent that receives the car ahead message becomes an $RQ\lfloor id_2, rq_1 \rfloor$ process in F_{12}. With transition t_{11} from F_{12} to F_{13}, RQ is replaced by its defining process:

$$\begin{aligned} F_{12} &= \nu rq_1.\big(\nu id_1, ca_1.choice_1 \mid \nu id_2.RQ\lfloor id_2, rq_1 \rfloor\big) \\ F_{13} &= \nu rq_1.\big(\nu id_1, ca_1.(\ldots + rq_1(nf).\overline{nf}\langle id_1\rangle.LD\lfloor id_1, nf \rfloor) \\ &\qquad \mid \nu id_2.\overline{rq_1}\langle id_2\rangle.id_2(nl).FL\lfloor id_2, nl \rfloor \big). \end{aligned}$$

In F_{13}, the second free agent issues a request to merge with the leading car. Transition t_{12} from F_{13} to F_{14} models the acceptance of this request as it reflects the communication of both cars on the rq_1 channel. With t_{13}, the now leader passes its id_1 channel to the follower, which yields the car platoon in F_{15}.

$$\begin{aligned} F_{14} &= \nu id_2.\big(\nu id_1.\overline{id_2}\langle id_1\rangle.LD\lfloor id_1, id_2 \rfloor \mid id_2(nl).FL\lfloor id_2, nl \rfloor\big) \\ F_{15} &= \nu id_1, id_2.\big(LD\lfloor id_1, id_2 \rfloor \mid FL\lfloor id_2, id_1 \rfloor\big). \end{aligned}$$

We continue with the investigation of the occurrence numbers of processes in the car platoon system.

6.1.2 Occurrence Number Properties

If we consider a reaction sequence where the number of cars stays constant, we expect a linear relationship between the number of free agents and the number of follower-leader-platoons. Each follower-leader-platoon created in the execution should correspond to two free agents beforehand. This relationship exists, we prove it using S-Invariants (cf. Section 2.2.2).

Consider the subnet of the structural semantics of the car platoon system, which is obtained by removing the environment process $F_1 = ENV\lfloor cfa \rfloor$ (cf. gray part in Figure 6.2). Since only the environment generates free agents, the number of cars stays constant. To relate the number of free agents and the number of follower-leader-platoons, we construct a relation between the markings of $[F_2]$ and $[F_{15}]$. We use the following S-invariant I, written as labelled vector to improve readability:

$$I^t = \begin{pmatrix} F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 & F_9 & F_{10} & F_{11} & F_{12} & F_{13} & F_{14} & F_{15} \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}.$$

Recall that we can rely on the INA toolkit to compute S-invariants [Sta03]. By the fundamental property of S-invariants in Lemma 2.2.12, for all markings M and M' of the subnet with $M \to^* M'$ it holds

$$\begin{aligned} & I^t \cdot M' = I^t \cdot M \\ \Leftrightarrow \quad & \Sigma_{i=2}^{15} I([F_i])M'([F_i]) = \Sigma_{i=2}^{15} I([F_i])M([F_i]). \end{aligned}$$

If we assume that all merging activities are finished in M and in M', i.e., there are only tokens on the places $[F_2]$, $[F_{15}]$, and $[F_4]$ with $F_4 = MRG\lfloor cfa \rfloor$ then the equation implies

$$\begin{aligned} & M'([F_2]) + 0M'([F_4]) + 2M'([F_{15}]) = M([F_2]) + 0M([F_4]) + 2M([F_{15}]) \\ \Leftrightarrow \quad & \Delta_{M,M'}([F_{15}]) = -1/2\Delta_{M,M'}([F_2]), \end{aligned}$$

where $\Delta_{M,M'}(x) := M'(x) - M(x)$. This means for every token added on $[F_{15}]$ two free agents $FA\lfloor cfa \rfloor$ are removed from $[F_2]$. Since F_{15} consists of two processes $\nu id_1, id_2.(LD\lfloor id_1, id_2 \rfloor \mid FL\lfloor id_2, id_1 \rfloor)$ and no processes are created, we conclude at process level that every free agent removed in a reaction sequence from process P to P' is a follower or a leader in P'.

To sum up, consider $P \in Reach(ENV\lfloor cfa \rfloor \mid MRG\lfloor cfa \rfloor)$ and $P' \in Reach(P)$, where the restricted forms of P and P' consist of fragments $FA\lfloor cfa \rfloor$, $MRG\lfloor cfa \rfloor$, and $\nu id_1, id_2.(LD\lfloor id_1, id_2 \rfloor \mid FL\lfloor id_2, id_1 \rfloor)$ only. Let P' be reachable without the reaction $ENV\lfloor cfa \rfloor \rightarrow FA\lfloor cfa \rfloor \mid ENV\lfloor cfa \rfloor$. Then the following result holds.

Result 6.1.2
The number of follower-leader-platoons added in P' is half the number of free agents removed in P'. Every free agent removed is a follower or a leader.

6.1.3 Topological Properties

In Section 3.1, we discussed the interpretation of π-Calculus processes as hypergraphs, where restricted names connect the sequential processes that share them. Inspired by this graph interpretation, we consider *connectedness properties*. We say that $Q, Q' \in \mathcal{S}(P)$ are *directly connected* in process P if they share a free name, i.e., $fn(Q) \cap fn(Q') \neq \emptyset$. Assume Q occurs only in fragments where its free names are restricted. Then the direct connectedness property can be established for all reachable processes by inspecting the places in the Petri net, without taking behavioural information into account. Note that the assumption always holds for closed processes.

Result 6.1.3
In every reachable process, a follower is directly connected with a leader, i.e., a process that contains $FL\lfloor id_x, id_y \rfloor$ also contains $LD\lfloor id_y, id_z \rfloor$.

Proof
Only fragment $F_{15} = \nu id_1, id_2.(LD\lfloor id_1, id_2 \rfloor \mid FL\lfloor id_2, id_1 \rfloor)$ contains a follower $FL\lfloor id_2, id_1 \rfloor$ and id_1 is the identifier of a leader. ■

The proof shows more. There is no situation, in which a follower knows a leader but the leader does not know the follower, i.e., $FL\lfloor id_2, id_1 \rfloor$ implies $LD\lfloor id_1, id_z \rfloor$ with $id_z = id_2$. With the argument that a leader is only present in the mentioned fragment we conclude that also a leader is always properly connected, which means directly connected with a follower, not with another leader or a free agent.

We say that $Q, Q' \in \mathcal{S}(P)$ are *connected* in process P, if they are in the transitive closure of the direct connection relation, i.e., there are $Q_1, \ldots, Q_n \in \mathcal{S}(P)$ so that $fn(Q) \cap fn(Q_1) \neq \emptyset$, $fn(Q_i) \cap fn(Q_{i+1}) \neq \emptyset$, and $fn(Q_n) \cap fn(Q') \neq \emptyset$. A connection is necessary for an interaction between Q and Q'. The following situation demonstrates that connections are critical. In fragment F_{11}, the free agent $ca_2(rqnl).RQ\lfloor id_2, rqnl \rfloor + \ldots$ waits for the request channel of its new leader. At the same time it is connected (via $\overline{ca_2}\langle rq_1 \rangle.MRG\lfloor cfa \rfloor$) with another free agent $\nu id, ca, rq.\overline{cfa}\langle id \rangle.\overline{id}\langle ca \rangle.\overline{id}\langle rq \rangle.choice$. The second free agent could send false information to the first.

Result 6.1.4
In $ENV\lfloor cfa \rfloor \mid MRG\lfloor cfa \rfloor$ a process is reachable where $ca_2(rqnl).RQ\lfloor id_2, rqnl \rfloor + \ldots$ and $\nu id, ca, rq.\overline{cfa}\langle id \rangle.\overline{id}\langle ca \rangle.\overline{id}\langle rq \rangle.choice$ are connected.

Proof
The process $ca_2(rqnl).RQ\lfloor id_2, rqnl \rfloor + \ldots$ only shares its ca_2 channel with the process $\overline{ca_2}\langle rq_1 \rangle.MRG\lfloor cfa \rfloor$ in F_{11}. The latter process additionally shares the channel cfa with the fragment $F_3 = \nu id, ca, rq.\overline{cfa}\langle id \rangle.\overline{id}\langle ca \rangle.\overline{id}\langle rq \rangle.choice$. A process is reachable where all three subprocesses occur in parallel composition iff a marking M is reachable with $M([F_3]) > 0$ and $M([F_{11}]) > 0$. The coverability tree shows that this is possible. ■

6.1.4 Temporal Properties

The topological properties verified in the last section are invariants, i.e., they hold in every reachable process. The property considered in this section talks about more elaborate temporal behaviour. The verification is purely graph theoretic, i.e., it solely relies on the net structure. Consider fragment

$$\begin{aligned} F_{11} \quad = \quad & \nu ca_2.\big(\quad \nu rq_1.\big(\nu id_1, ca_1.choice_1 \mid \overline{ca_2}\langle rq_1 \rangle.MRG\lfloor cfa \rfloor\big) \\ & \qquad \mid \nu id_2, rq_2.(ca_2(rqnl).RQ\lfloor id_2, rqnl \rfloor + \ldots) \ \big), \end{aligned}$$

in which process $\overline{ca_2}\langle rq_1 \rangle.MRG\lfloor cfa \rfloor$ sends a car ahead message with a new leader to the free agent $ca_2(rqnl).RQ\lfloor id_2, rqnl \rfloor + \ldots$

Result 6.1.5
For every free agent the following holds in every reachable process: if the agent receives a car ahead message, it will never receive a car ahead message again.

We briefly discuss the formalisation of the property in a temporal logic. The temporal behaviour (a car ahead message is received at most once) is specified relative to a free agent. Formulas in standard temporal logics like CTL^* refer to a finite set of atomic propositions, which is not suitable for reasoning about an unbounded number of free agents. Instead, the property requires a universal quantifier for restricted names, which covers temporal operators. Logics that support name quantification are presented in [Dam96, CC03, FGMP03].

The proof of Result 6.1.5 applies the following more general observation. Consider a non-empty (denoted by +) sequence

$$F \mid P \rightarrow^{+} F' \mid P' \rightarrow Q.$$

If the first and last reaction use the restricted name a (i.e., a prefix $\overline{a}\langle b\rangle$ is consumed in the reaction) and a is not renamed via α-conversion in the meantime, we say there are *two reactions in the sequence using one restricted name.*

Observation 6.1.6
Consider a reaction sequence where two reactions use one restricted name. Let F and F' be the fragments performing the reactions. Then there is a directed path in the Petri net from place $[F]$ to $[F']$ so that every fragment on the path has at least one restricted name. ■

The idea underlying the observation is that a restricted name a, which is used by F and F' with $F \mid P \rightarrow P_1 \rightarrow \ldots \rightarrow P_n \rightarrow F' \mid P'$, is remembered in all intermediate processes P_i. When the restricted forms $rf(P_i) \equiv \Pi_{j \in J_i} G_j$ are computed, one fragment G_j contains the restriction νa. These fragments form the path in the Petri net.

Proof (of Result 6.1.5)
To send a car ahead message to the same agent twice, a directed path in the net is needed from the first fragment sending the message to the second. The only fragment sending a car ahead message is F_{11}. The only cycle starting in $[F_{11}]$ is $[F_{11}].[F_4].[F_5].[F_6].[F_7].[F_8].[F_9].[F_{10}]$ (cf. Figure 6.2). Since place $[F_4]$ with $F_4 = MRG\lfloor cfa \rfloor$ on this cycle does not have a restricted name, no car ahead message is sent repeatedly to the same free agent. ■

6.2 Autonomous Transport

We report on the results of verifying an industrial case study from the automated manufacturing domain, which have been obtained by Philipp Gringel [Gri07] and Tim Strazny [MKS09]. The case study is taken from a project on integrating the design of mechanical systems and their control software funded by the German Research Council as Priority Program *Integration of Software Specification Techniques for Applications in Engineering* [EDD+04]. We briefly describe the case study and refer to [BR01] for details.

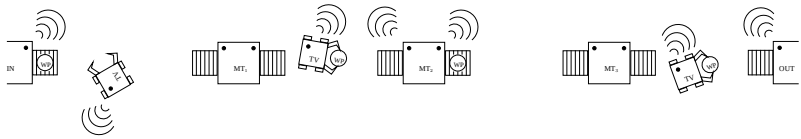


Figure 6.3: Sketch of the transportation system.

As part of a motor production process workpieces are drilled and cut, and afterwards washed. The workpieces arrive in an input storage, are brought to a drill and a cutter, finally washed and deposited to an output storage. The focus of the case study is on the transportation system. It is realised by autonomous vehicles that communicate by radio. Figure 6.3 illustrates the scenario. A key requirement of the system is flexibility in the range of workpieces that have to be processed. To ensure this, workpieces carry information about the ordering in which they have to pass the tools. For example, the motor parts first visit the drill, denoted by MT_1, or the cutter MT_2, and afterwards the washing machine MT_3. Hence, MT_1, MT_2, MT_3 as well as MT_2, MT_1, MT_3 are correct sequences through the system, any other path is forbidden. Our transportation system is highly flexible as it can be easily adapted to arbitrary sequences of processing steps.

6.2.1 Modelling the Case Study

The π-Calculus process representing the production system has been developed by Philipp Gringel in his Bachelor's thesis [Gri07]. We restrict ourselves to explaining the main components and remark that the model is a finite control process. Hence, it is structurally stationary with Lemma 4.3.6 and moreover, the verification approach from Chapter 5 is applicable to establish temporal properties.

Basic Model

The system is the parallel composition of the processes for input and output storages, machine tools, transportation vehicles, and workpieces. The model is scalable in the number of machine tools (of each type), transportation vehicles, and workpieces, as indicated by the iterated parallel compositions (Π) below:

$$IN \mid OUT \mid \Pi MT_1 \mid \Pi MT_2 \mid \Pi MT_3 \mid \Pi TV \mid \Pi WP.$$

Formally, all of the processes above are calls to process identifiers. We omitted the long parameter lists and just remark that all processes except transportation vehicles TV have unique identifiers. Moreover, there is a channel tv on which vehicles and machines communicate.

Workpieces We follow a workpiece through the production process. Upon creation, it sends its ID on channel *new* to the input storage, which then requests a vehicle to transport it. The workpiece non-deterministically decides in which order it has to be processed by machine tools. However, as explained above, the washing machine MT_3 is the last to be visited before the workpiece is put into the output storage. To announce the machine it has to visit next to the transportation vehicle, the workpiece sends the machine type on its ID channel wp.[1] For example, a processing sequence MT_1, MT_2, MT_3 gives rise to the communication sequence $\overline{wp}\langle mt_1\rangle.\overline{wp}\langle mt_2\rangle.\overline{wp}\langle mt_3\rangle$. When the workpiece arrives in the output storage, it finally communicates on the channel representing its ID and terminates, vanishing from the system's view:

$$\begin{aligned} WP(wp, new, mt_1, mt_2, mt_3, out) := \\ \overline{new}\langle wp\rangle.(\quad & \overline{wp}\langle mt_1\rangle.\overline{wp}\langle mt_2\rangle.\overline{wp}\langle mt_3\rangle.\overline{wp}\langle out\rangle.wp \\ + \ & \overline{wp}\langle mt_2\rangle.\overline{wp}\langle mt_1\rangle.\overline{wp}\langle mt_3\rangle.\overline{wp}\langle out\rangle.wp \ ). \end{aligned}$$

Input storage Upon reception of a new workpiece wp, the input storage calls for a transportation vehicle by sending its address *store* on the channel tv. When the vehicle arrives, the storage hands over the workpiece by sending wp on channel *store*. With a recursive call, the input storage is ready to handle the next workpiece:

$$IN(new, store, tv) \quad := \quad new(wp).\overline{tv}\langle store\rangle.\overline{store}\langle wp\rangle.IN\lfloor new, store, tv\rfloor.$$

Transportation vehicles A vehicle listens on channel tv for a transportation request. The sender of the request—a machine tool or the input storage—is stored in the variable *source*. The vehicle moves to the sender, modelled by a

[1] Note that the number of machines of each type is parametric. The workpiece decides only on the machine type, the actual machine is chosen non-deterministically.

τ-step, and receives the workpiece *wp* on channel *source*. As explained above, the workpiece sends the machine type it has to visit next. Since there may be several machine tools of the same type, the vehicle requests the identifier of an idle machine with the receive action $type(dest)$. It then moves to the machine that answered and delivers the workpiece by sending *wp* on channel *dest*. Having completed a task, the vehicle becomes ready for the next transportation:

$$TV(tv) \quad := \quad tv(source).\tau.source(wp).wp(type).type(dest).\tau.\overline{dest}\langle wp\rangle.TV\lfloor tv\rfloor.$$

Machine tools The definition of a machine tool process is more complicated—it is a realistic design consisting of seven subprocesses. We decided only to sketch its structure and behaviour rather than explain it in full detail. After a workpiece has been deposited on the input buffer of a machine tool, the buffer's handler informs the machine about the presence of a workpiece. The machine moves the workpiece from its input buffer to the processing unit and starts working. When the processing is done, it puts the workpiece to its output buffer, which subsequently calls for a transportation vehicle.

Processing of a workpiece, i.e., drilling, cutting, or washing, is performed in the process *WORK*. It receives a workpiece *wp* from the input buffer *in*, performs its operation with a τ-action, and calls the *PUT* process to deposit the workpiece to the output buffer:

$$WORK(full, empty, in) \quad := \quad in(wp).\tau.PUT\lfloor full, empty, wp\rfloor.$$

Output storage Finally, the workpiece arrives in the output storage. It is modelled like a working machine and therefore has a type and an ID. To accept a workpiece, it sends its ID *store* on its type channel and then awaits a workpiece on *store*. We already explained that a workpiece terminates when it arrives in the output storage and communicates on channel *wp*. The storage then forgets about the workpiece and is ready to receive the next one:

$$OUT(type, store) \quad := \quad \overline{type}\langle store\rangle.store(wp).\overline{wp}.OUT\lfloor type, store\rfloor.$$

This finishes the walk through the case study model.

Refined Model with Malfunctions

Braatz and Ritter introduce the case study in two levels of abstraction [BR01]. The basic model discussed above defines the communication infrastructure. The refined model accounts for failures in machine tools or transportation vehicles and uses an additional process modelling a mechanic to cope with malfunctions.

We assume malfunctions in vehicles to occur only during the transportation of workpieces. The process part that represents the transportation is split up into three steps,

$$\tau.source(wp).wp(type). \quad type(dest). \quad \tau.\overline{dest}\langle wp\rangle,$$

before each of which a vehicle may break down. Technically, a malfunction is a non-deterministic choice, similar to the interrupt mechanism in CSP [Hoa85].

When a malfunction occurs, a vehicle calls on an emergency channel for another vehicle to take over its transportation task. After the workpiece is handed over to this rescue vehicle, the broken vehicle issues a request to the mechanic. When it is repaired, the vehicle is ready to take over transportation jobs again.

The definition of a vehicle is generalised so that it can also act as a rescue vehicle as follows. It accepts requests on the emergency channel, which contain the status of the transportation task. More precisely, the broken vehicle sends its point of failure, which gives information about the steps of the transportation that have been finished. The rescue vehicle takes over the workpiece (or the transportation task) and performs the remaining part of the transportation. Rescue vehicles are assumed not to malfunction. After the transportation is finished, a rescue vehicle behaves like a standard vehicle again.

A machine tool only malfunctions with a workpiece inside its processing unit. Since a drill may get stuck or break within the workpiece, the machine tool is not immediately able to hand over the workpiece to its output buffer. Instead, it calls for the mechanic and waits until the problem is fixed. After being repaired, the machine tool outputs the processed workpiece and calls for a transportation vehicle as usual.

We now turn to the verification, first of temporal properties with the approach in Chapter 5, and afterwards of topological properties with the technique in Section 6.1.3.

6.2.2 Temporal Properties

Braatz and Ritter suggest a number of correctness criteria for the transportation system. We settle two of them using our unfolding-based verification technique (cf. Figure 5.1). We show that the system is free from deadlocks and that workpieces are processed in a correct order, i.e., they are only washed after having been drilled and cut. Both requirements are considered key in [BR01]. As the first step we establish correctness of the basic model. When we tried to verify the refined model, we discovered a subtle deadlock. A counterexample produced by our tool chain helped us understand the problem and change the design to eliminate the deadlock. To conclude the section, we consider a requirement not mentioned by Braatz and Ritter. We show that in certain situations a broken machine tool or transportation vehicle may never be repaired by the mechanic, if there are enough operational vehicles to compensate for it. (Note that there are no fairness requirements in the model.)

All results are supported by tables that give information about the sizes of the models and the corresponding verification times. They are taken from [MKS08], where the verification has been conducted by Tim Strazny. Unlike Chapter 5, the

tables do not give runtimes for the alternative model checkers MWB, HAL, or the unfolding-based approach in [KKN06]. The former two failed for the smallest instance of the case study model, while the latter cannot handle recursive π-Calculus definitions, which are necessary for our case study.

The benchmarks were run on a core of a 2.5 GHz Athlon 64 X2 with 4 GB memory. The columns in Table 6.2 are organised as follows. `Property` states the property we checked. We refer to these in the following subsections. A row labelled with Xp Ym Zv in the `Instance` column stands for the instance of the model with X workpieces, Y machine tools (in total), and Z transportation vehicles.[2] The meanings of the remaining columns are like in Table 5.2. Except for a few rows, we give the data only for instances with maximal size we were able to verify with our tools.

Deadlocks

Workpiece processes terminate when they arrive in the output storage. Hence, with a fixed number of workpieces, every run of the system is finite. To distinguish deadlocks from proper system terminations, we consider final states of the maximal system runs. If every workpiece is consumed, the system has terminated properly, else the final state contains a derivative of some workpiece process and so this state is a deadlock.

To check for deadlocks, we modified the definition of workpieces so that they are transferred back from output to input storage. Hence, the properly terminating runs become infinite, while the remaining maximal runs are still deadlocks. Technically, instead of terminating upon a message from the output storage (on their ID channel *wp*), workpieces call their recursive definition, so that they are ready to be processed again by sending their IDs on channel *new*. Rows with property *dl* in Table 6.2 show the sizes and verification times for different instances of the model. All basic models are deadlock-free.

Removing Deadlocks from the Refined Model We verified different instances of the refined model and found a subtle bug in case there are at least as many workpieces as there are vehicles.[3] The problem is in the communication protocol used by a broken vehicle. It hands over its transportation job to a rescue vehicle before calling the mechanic. Consider the following scenario. Every vehicle fetches a workpiece. Then, at some moment during transportation, all vehicles malfunction. They call for rescue vehicles to take over their transportation tasks, but since every vehicle is broken, none of the calls can be answered. As the mechanic is contacted by a vehicle only when its job has been handed over, he is never called, and the system is in a deadlock.

[2] The π-Calculus code for various instances can be found at [SM08].

[3] Instances of the model with more vehicles than workpieces are deadlock-free.

Property	Instance	FCP Size	Struct |P|	Struct |T|	B	Safe Size	Struct |P|	Struct |T|	Model Checking unf	Model Checking |B|	Model Checking |E*|	Model Checking sat
dl	1p 3m 6v basic	445	197	152	6	1133	312	347	<1	1942	1285	<1
dl	2p 3m 6v basic	455	277	243	6	1208	457	593	235	83898	56935	37
dl	3p 3m 3v basic	453	357	334	3	1223	455	536	22392	722603	481845	3491
dl	1p 6m 6v basic	550	356	275	6	2027	531	590	28	40138	27780	28
dl	2p 6m 3v basic	548	487	432	3	2042	595	656	27168	823960	560147	233
dl (♣)	1p 3m 1v refined	811	292	297	1	1475	292	297	<1	1427	1011	<1
dl	1p 3m 6v refined	861	797	1389	1	3200	797	1389	1289	199528	157316	1561
dl (♠)	2p 3m 2v refined	831	593	761	1	1895	593	761	1863	211066	159793	336
dl	2p 3m 3v refined	841	756	1101	1	2240	756	1101	24170	830328	647837	701
dl	1p 6m 4v refined	949	937	1371	1	3491	937	1371	28876	923236	721991	1182
flow	1p 3m 6v basic	848	245	185	6	1338	360	380	<1	2212	1452	<1
flow	2p 3m 1v basic	1130	379	334	1	1616	349	338	17989	625464	400094	139
flow	1p 6m 4v basic	945	417	324	4	2317	540	551	1416	182710	125557	327
flow	1p 3m 6v refined	1370	824	1401	1	3460	824	1401	1773	211908	167083	1063
wrong flow	1p 3m 1v basic	1120	240	204	1	1538	244	208	<1	974	635	<1
wrong flow	1p 3m 1v refined	1613	334	345	1	2031	338	349	<1	1606	1176	<1
malf reach	2p 3m 2v refined	831	593	761	1	1895	593	761	1863	211066	159793	<1

Table 6.2: Experimental results for the transportation system.

To solve this problem, we let transportation vehicles non-deterministically decide either to call for a rescue vehicle and then get repaired or to call for the mechanic immediately, without handing over the transportation job. In the latter case, a repaired vehicle is assumed to finish its job without malfunction. The rows for property *dl* in Table 6.2 give the data only for corrected versions of the refined model—they are deadlock-free. Uncorrected versions with deadlocks are not listed.

Correctness and Completeness of Workpiece Flows

We first prove that each workpiece specified above visits the machine tools in one of the correct orders, either MT_1, MT_2, MT_3 or MT_2, MT_1, MT_3. We then show that any other workpiece definition leads to forbidden sequences of machine tools. Hence, the workpiece model is complete with respect to the allowed paths. Both, correctness and completeness are established with deadlock detection methods by adapting our system model.

To establish correctness, we change the model in two respects. We first let a workpiece loop back from output to input storage (cf. checking for deadlocks above), so that the system does not terminate. Then, we restrict the behaviour of machine tools and output storage. They check whether a workpiece has visited the tools in the correct order, and deadlock if this is not the case. Hence, the system is deadlock-free iff the workpiece followed a correct path through the system.

A sequence of machine tools is incorrect if (1) a tool is visited more than once; (2) the workpiece is washed although it has not been drilled or cut, or (3) the workpiece has not visited any of the machines. The control mechanisms work as follows.

To ensure that a machine type $mt \in \{mt_1, mt_2, mt_3\}$ is visited at most once by a workpiece *wp*, messages *mt_fresh_wp* are generated when the workpiece arrives in the input storage, i.e., after it has sent its ID on channel *new*. The message states that the workpiece has not yet been processed by a machine of the corresponding type. The *WORK* process of machine tools is now modified to consume such a message before it starts working. Since we verify instances of the system with a single workpiece, the system deadlocks iff the workpiece is sent twice to a machine of the same type.

To ensure workpiece *wp* visits machine tools of type one and two before type three, we let each machine of type $mt \in \{mt_1, mt_2, mt_3\}$ spawn a message *mt_done_wp*. It states that workpiece *wp* has been processed by a tool of the given type. The *WORK* process of machine tools of type three is now modified to consume messages mt_1*_done_wp* and mt_2*_done_wp* when it receives workpiece *wp*. The process gets stuck, if the workpiece has not visited machine tools of type one and two. The control mechanism for the output storage is similar. Hence, machine tools of type three have to recreate the messages they consumed.

It is not obvious that spawning messages *mt_fresh_wp* (similar for *mt_done_wp*) can be modelled in the restricted syntax of finite control processes. The idea is to compose processes $FRESH_{mt,wp}$ in parallel with the main process, one process for each type of machine tools and every workpiece. When a workpiece arrives in the input storage, it sends a message *mt_forkfresh_wp* to the $FRESH_{mt,wp}$ process, which in response provides the *mt_fresh_wp* message.

The results for checking the modified system for deadlocks are given in the rows *flow* in Table 6.2. The system is deadlock-free and this, together with the above argumentation, proves that workpieces visit machine tools in a correct order.

To establish completeness, we alter the model in a different way. Workpieces terminate when they arrive in the output storage. Unlike workpieces in the original model, they non-deterministically choose a path through the system, i.e., a non-deterministic choice is used each time a workpiece is asked to announce its next destination. The process is designed so that the two valid paths above are excluded.

If one of the machine tools finds constraints (1), (2), or (3) on the correct order violated, it starts an infinite loop. Consequently, the system has a deadlock if and only if there is a path through the system, which is different from those in the original model. We verified the absence of deadlocks, which shows that any other path leads to a forbidden sequence of machine tools. The verification times are given in the *wrong flow* rows in Table 6.2.

Broken Vehicles Get Repaired

We now demonstrate that the fairness property that a broken transportation vehicle is eventually repaired is violated as long as there are enough vehicles to compensate for it. Note that in Table 6.2 neither the instance with one workpiece, three machine tools and one vehicle in row *dl* (♣) nor the instance with two workpieces, three machine tools, and two vehicles in row *dl* (♠) has deadlocks.

System (♠) is the parallel composition of system (♣), another workpiece process, and another vehicle process. Hence all runs that are possible in system (♣) are also possible in system (♠). We now verified that a state is reachable where the second transportation vehicle malfunctions and system (♣) is in its initial state. In row *malf reach* of Table 6.2 we give the sizes of the net, its prefix, and the running times. Since the malfunction of the second vehicle is reachable and system (♣) is deadlock-free, the second vehicle does not have to be repaired. There exists a run where the second vehicle fetches a workpiece, malfunctions, and issues a request for a rescue vehicle which is never answered. Since system (♣) is deadlock-free the remaining vehicle can always perform a step without communicating with the broken vehicle.

6.2.3 Topological Properties

We investigate the connections between workpieces, machine tools, and transportation vehicles. In contrast to Section 6.1.3, the definition of *direct connections* is refined towards restricted channels, i.e., $Q, Q' \in \mathcal{S}(P)$ are directly connected in P $fn(Q) \cap fn(Q') \cap fn(P) \neq \emptyset$. We discuss the verification of the following property.

Result 6.2.1
In every reachable state of the transportation system, a workpiece is directly connected either with a storage, a machine tool, or a transport vehicle, but never with two of them.

Since Braatz and Ritter abstract from the implementation of the communication infrastructure, these properties are not considered in [BR01]. However, their violation points to serious bugs in the implementation of the transport system. For example, if a workpiece is shared by a transport vehicle and a machine tool, one of them has a dangling reference to the workpiece and the memory management should be reconsidered.

Note that any process that is directly connected to a workpiece belongs to the same fragment as the workpiece, since direct connections rely on restricted channels. Since the places in the structural semantics represent all reachable fragments, it is sufficient to inspect all places for more than one direct connection to a workpiece. We consider an instance of the system where only the channel *wp* of a single workpiece is restricted. The characteristics of our model are listed in the following table, where |**PT**| (and |**TP**|) denotes the number of arcs from places to transitions (and vice versa) and **Compile** is the compile time in seconds:

Property	**Instance**	**Size**	**|P|**	**|T|**	**|PT|**	**|TP|**	**Compile**
con	1p 6m 6v refined	979	2084	3744	6233	6268	21,3

Due to the size of the model (2084 places), tool support is required to inspect the places. The idea is to load the file of the Petri net (in the `ll_net` format that is plain text) into a text editor and use regular expressions to find all places with at least three sequential processes. We used the following query

```
new [^(”] \( ([^|”]*\|){2}.
```

It first searches for a restriction operator, new, followed by the restricted name, [^(”]. Then it reads the opening bracket of the fragment, \(. The expression ([^|”]*\|) denotes a group of a sequential process [^|”]* followed by a parallel composition operator \|. Finding this group twice, {2}, means the fragment contains at least three sequential processes as it does not end with a parallel composition.

Since the query had no match, there is no situation in which a workpiece is directly connected with more than one tool, vehicle, or storage. As a sanity check, we changed the definition of vehicles to store the restricted *wp* channel and immediately found hits for the query above. In the following section we discuss our verification techniques.

6.3 Discussion of the Verification Approach

In Section 6.1.2, we pruned the net $\mathcal{N}[\![ENV \lfloor cfa \rfloor \mid MRG \lfloor cfa \rfloor]\!]$ by hand to exclude an unbounded generation of free agents by $ENV \lfloor cfa \rfloor$. Since our Petri net semantics is non-compositional, excluding the process $ENV \lfloor cfa \rfloor$ and computing the Petri net $\mathcal{N}[\![MRG \lfloor cfa \rfloor]\!]$ does not yield the subnet in Figure 6.2. Nevertheless, automating our pruning method should be possible for Petri nets $\mathcal{N}[\![P]\!]$, where P uses an environment process like $ENV \lfloor cfa \rfloor$ to create new processes.

Our verification algorithms exploit the fact that the places in the Petri net $\mathcal{N}[\![P]\!]$ are the reachable fragments of P. Most notably, we verified topological invariants with the help of a regular expression finder in a text editor. This ease of verification comes at the expense of a complicated computation of the semantics that determines precisely the reachable fragments. More syntactical Petri net translations like [Eng96, BG95, DKK06a] can be computed more efficiently (cf. Section 3.6 for a discussion of these semantics). But they ask for more expensive analyses. A topological verification problem like the correct connection of a workpiece requires to solve a complicated coverability problem in these translations. In general, such a coverability problem needs to be solved in every analysis while our semantics requires a complicated computation once and then eases the verification. Moreover, we already discussed that our translation still yields finite place/transition Petri nets where related approaches yield either infinite nets or Turing complete models.

In Section 3.5, we mentioned that we may not be able to determine the precise set of reachable fragments in the Petri net $\mathcal{N}[\![P]\!]$ due to memory limitations. In this case, we compute an over-approximating Petri net $\mathcal{N}_{NoCov}[\![P]\!]$, which subsumes $\mathcal{N}[\![P]\!]$. Our verification techniques that rely on the knowledge of the reachable fragments are still applicable to $\mathcal{N}_{NoCov}[\![P]\!]$ as follows. If we check for example the correct connection between a follower and a leader, we inspect all places in $\mathcal{N}_{NoCov}[\![P]\!]$. If the processes are properly connected in $\mathcal{N}_{NoCov}[\![P]\!]$ we can conclude that they are properly connected in $\mathcal{N}[\![P]\!]$. If we find a place $[F]$ in $\mathcal{N}_{NoCov}[\![P]\!]$ where follower and leader are not properly connected, we have to check whether $[F]$ is markable in $\mathcal{N}_{NoCov}[\![P]\!]$. It is markable in $\mathcal{N}_{NoCov}[\![P]\!]$ if and only if it is contained in $\mathcal{N}[\![P]\!]$. Thus, if it is markable there is an incorrect connection in $\mathcal{N}[\![P]\!]$ and the property does not hold for the process P. If it is not markable, the place is not contained in $\mathcal{N}[\![P]\!]$. We remove it from $\mathcal{N}_{NoCov}[\![P]\!]$

and repeat the analysis. This approach is a variant of counterexample-guided refinement of an imprecise system representation as proposed in [CGJ+00].

The temporal property verified in Section 6.1.4 was stated informally. In [Dam96, CC03, FGMP03] temporal logics are proposed to formalise the correct behaviour of π-Calculus processes. These logics are able to specify the way names flow through a system. To establish such properties on the structural semantics, we need to keep track of the identity of names. Following [MP95a, Pis99, MP01], transitions could be equipped with labels relating the names in pre- and postset. It deserves further investigation whether decidability results can be obtained for model checking linear-time variants of these logics.

In Chapter 5 and Section 6.2.2, we demonstrated that our Petri net semantics works well with efficient standard verification techniques for Petri nets. The performance of Petri net verification tools highly depends on the size of the nets. Our translation yields small nets when the number of processes inside fragments is small or the processes inside fragments tightly interact. We found out that the size of our translation is particularly sensitive to independent reactions inside a fragment. As an example, consider the fragment $\nu a.(\tau.\overline{a}\langle a_1\rangle \mid \ldots \mid \tau.\overline{a}\langle a_n\rangle)$, which yields 2^n places in our translation. Partial-order and Petri net reduction techniques [CGP99, ES01] are helpful to limit the size of the Petri net, some of which are already implemented in PETRUCHIO.

6.4 Related Work

In the AVACS project, three related verification techniques for the car platoon system have been developed [Bau06, Wes08, Tob08]. Unlike our technique, which is a decision procedure, the related approaches are abstraction-based, i.e., they construct a finite abstraction of the infinite state space of a DRS and verify properties on this abstraction. The benefit is that these techniques can handle arbitrary models, also Turing complete classes, while ours is restricted to structurally stationary systems. The drawback is that they are semi-decision procedures, i.e., not guaranteed to verify a property.

In [Bau06], graph transformation models of the car platoon system are proposed to demonstrate the expressiveness of the newly developed partner graph grammars and to evaluate the usefulness of the novel partner abstraction. Partner abstraction is used within the abstract-interpretation framework, i.e., the reachable system states (which are graphs) of the car platoon system are abstracted to finitely many instances, which constitute an invariant for all reachable connection structures. Bauer's work is closely related to our verification of topological properties, where the places form all (concrete) reachable fragments. Different from his technique, we also reflect the behaviour of the car platoon system in the net and can thus handle occurrence number and temporal properties.

Westphal models the car platoon system in the language of dynamic communicating systems [Wes08], the semantics of which are transition systems with graph-labelled states. He proposes a logic to specify temporal properties that refer to the identities of processes in the system. To prove a property, Westphal computes a finite abstraction of the transition system with the so-called spotlight abstraction technique. He succeeds in proving temporal liveness properties and topological invariants, but cannot handle occurrence number properties. As was explained in the previous section, the structural semantics needs to be extended with assignments on transitions to handle temporal properties that refer to identities.

The approach of Westphal has been extended in [Tob08] to an abstraction-refinement cycle [CGJ$^+$00]. With a coarse abstraction of the transition system, Toben tries to verify temporal properties that again talk about system entities. If a counterexample is produced that is not feasible in the concrete system, the abstraction is refined and the verification is repeated.

Several models of the production system by Braatz and Ritter [BR01] exist in the literature. We now discuss the related work on verification of this case study.

In [Weh00], the manufacturing system is modelled in the formal language *CSP-OZ*, which combines *CSP* for the description of behavioural and *Object-Z* for data aspects of the case study. With the model checker *FDR*, deadlock freedom and correctness of the workpiece flow were established. Unlike our model, positions of vehicles are modelled and a transportation job is given to the closest vehicle, while malfunctions and mechanic are omitted. We also remark that workpieces are modelled there as data rather than processes in the system. Compared to our approach, only small instances of the case study with two vehicles and three machine tools are verified.

In [MORW04], the case study illustrates the integration of the formal language *CSP-OZ* into a software development process. These authors create a *UML* model of the case study and translate it automatically into a *CSP-OZ* model. Then JAVA interfaces with assertions are generated from the formal model. A runtime checker monitors the execution of any program implementing the interfaces. If the checker detects an assertion violation, it terminates the execution of the program with an exception.

A model of the basic manufacturing system without malfunctions is considered in [FMPR01]. With focus on the timing behaviour, Flake et. al. use the language *MFERT* and then translate their model into the input format of the *RAVEN* model checker. Due to the limitations of the toolkit, the movement of workpieces is imitated by signals, in contrast with our model that creates channels dynamically. Correctness properties are specified in *Clocked CTL*, e.g. these authors show that a machine tool is not idle for more than 400 time units. They also analyse the timing behaviour, e.g. they evaluate the time a workpiece waits in the input storage for being picked up.

Also [RWKR04] uses the *RAVEN* model checker to establish time-dependent

properties. Their focus is on the positioning and movement of vehicles, therefore they explicitly include a primitive path finding algorithm in their model. They show that all workpieces eventually arrive at the output storage and that vehicles never collide. Different from our model, failures of machines are not considered.

Part III

Beyond Structural Stationarity

7 Depth and Breadth

Contents

In Chapter 4, we proved that the structural semantics $\mathcal{N}[\![P]\!]$ is finite exactly if process P is structurally stationary, i.e., it reaches only finitely many different fragments, Lemma 4.1.2. A main result was the complete characterisation of structural stationarity in Theorem 4.3.2: a process is structurally stationary if and only if the number of sequential processes in every reachable fragment is bounded. We applied the theorem to establish structural stationarity for a variety of well-known process classes. What is yet missing is an intuitive understanding of the processes that fail to be structurally stationary.

The main result in this chapter is the definition of two functions on π-Calculus processes called *depth* and *breadth* so that the following complete characterisation holds. A process is structurally stationary if and only if it is bounded in depth and bounded in breadth. The function *depth* measures the interdependence of restricted names in a process term, while the function *breadth* measures the distribution of restricted names. Hence, this second characterisation of structural stationarity refers to the restriction operator, in contrast to Theorem 4.3.2 relying on the parallel composition operator—a difference that is interesting from a

theoretical point of view. From a practical point of view, the contraposition of this equivalence answers the question raised above. A process fails to be structurally stationary if it is not bounded in breadth or not bounded in depth. So, infinity of the structural semantics has two different sources.

Unfortunately, the definitions of depth and breadth are difficult to understand as the functions refer to all processes in a congruence class. Therefore, we present intuitive characterisations that make use of the interpretation of processes as hypergraphs [MPW92, Mil99, SW01], which we introduced informally in Section 3.1. We establish two results. (1) A process is bounded in depth if and only if the length of the simple paths (i.e., without repetition of hyperedges) in the hypergraphs is bounded. (2) The breadth of a process equals the degree of the corresponding hypergraph. We demonstrate the application of our results by judging well-known modelling constructs in process algebras for boundedness in depth or breadth.

To sum up, this chapter provides an *intuitive understanding* of the processes that are (not) mapped to finite Petri nets using our structural semantics. The contributions are as follows:

- We define the novel characteristic functions *depth* and *breadth* on π-Calculus processes. We prove that boundedness in depth and breadth completely characterises structural stationarity.

- We show that boundedness in depth is equivalent to boundedness in the simple paths. The main technical contribution in the proof is a restrictive normal form for processes. Every process can be rewritten as a so-called *anchored fragment* using structural congruence.

- We establish equality between the breadth of a process and the degree of its graph. In the proof, we construct the process in the congruence class where the number of fragments under a restriction is maximal. By definition, the process exists. We give a construction for it.

- We apply the results to judge whether standard constructs in process algebras are bounded in depth or breadth.

The chapter is organised as follows. In Section 7.1, we recall the basic definitions of hypergraphs and formally define the interpretation of π-Calculus processes. The definitions of depth and breadth as well as the characterisation of structural stationarity are provided in Section 7.2. The theory of anchored fragments is introduced in Section 7.3. It is applied in Section 7.4, where we present the intuitive characterisation of boundedness in depth over the simple paths. Section 7.5 is devoted to the equality between breadth and hypergraph degree. In Section 7.6, we apply our results to judge well-known modelling constructs for boundedness before Section 7.7 discusses related work and concludes the chapter.

Throughout the chapter, we use the elementary fragments F^e introduced in Convention 3.2.3 as shortcut for sequential processes $M^{\neq \mathbf{0}}$ and $K\lfloor \tilde{a} \rfloor$. Since both processes are treated in the same way by the functions in this chapter, the use of F^e safes some case distinctions in definitions and proofs.

7.1 From Processes to Hypergraphs

We introduce the basic definitions of hypergraphs and then formally define the interpretation of π-Calculus processes.

7.1.1 Hypergraphs

Hypergraphs extend graphs by the ability to connect an arbitrary number of vertices with one hyperedge. We only introduce the basic definitions, deeper results on hypergraphs and hypergraph models of reconfigurable systems can be found in, e.g. [Hab92].

Definition 7.1.1 (Hypergraph)
Let $\mathcal{L}$ be a set of *vertex labels*. A *(vertex-labelled) hypergraph* is a tuple $\mathcal{G} = (V, E, l, inc)$, where

V is a finite set of *vertices*,

E is a finite set of *hyperedges*,

$l : V \to \mathcal{L}$ is a *vertex labelling function* that assigns a label $l(v) \in \mathcal{L}$ to every vertex $v \in V$, and

$inc : E \to \mathbb{P}(V)$ is an *incidence function*. It gives the set of vertices $inc(e) \subseteq V$ that are connected with $e \in E$.

The *set of all hypergraphs* is $\mathcal{H}$. In our setting, vertices are labelled by processes and edges are names, i.e., we have $\mathcal{L} = \mathcal{P}$ and $E \subseteq \mathcal{N}$. We sometimes refer to hypergraphs as *graphs* and to hyperedges as *edges*. If the naming is unambiguous, we refer to a vertex by its label. ⧫

As introduced in Section 3.1, a vertex $v \in V$ is drawn by a dot labelled by $l(v)$, an edge $e \in E$ is drawn by a box labelled by e. There is an arc between the vertex v and the edge e, if $v \in inc(e)$.

We say that two hypergraphs $\mathcal{G}_1 = (V_1, E_1, l_1, inc_1)$ and $\mathcal{G}_2 = (V_2, E_2, l_2, inc_2)$ are *equal*, denoted by $\mathcal{G}_1 = \mathcal{G}_2$, if $E_1 = E_2$ and there is a bijection between the sets of vertices that is compatible with the labelling and the incidence functions.

More precisely, there is a bijection $f : V_1 \to V_2$ so that $l_1(v) = l_2(f(v))$ and $f(inc_1(e)) = inc_2(e)$ for all $v \in V_1$ and $e \in E_1 = E_2$. With this definition, the identity of vertices is not important and we can always assume $V_1 \cap V_2 = \emptyset$. As indicated in the introduction, the *paths* in a hypergraph will play a particular role in this chapter.

Definition 7.1.2 (Paths)
A *path* in $\mathcal{G} = (V, E, l, inc)$ is a finite sequence $p = (v_1, e_1, \ldots, v_n, e_n, v_{n+1})$ of vertices and edges so that e_i connects v_i and v_{i+1}, i.e., $v_i, v_{i+1} \in inc(e_i)$ for all i. The *length* of p, $length(p)$, is the number of edges in p, e.g., $length(p) = n$ above. By $fe(p)$ we refer to v_1, the *first element in* p. A path is *simple*, if it does not repeat edges, i.e., $e_i \neq e_j$ for all $i \neq j$. By $lsp(\mathcal{G})$ we denote the *length of the longest simple path* in $\mathcal{G}$. The *set of all paths* in $\mathcal{G}$ is $Paths(\mathcal{G})$. ♦

While we need paths to characterise boundedness in depth, the breadth of a process is related to the degree of the graph. It is the maximal number of vertices connected with a hyperedge.

Definition 7.1.3 (Degree)
In the hypergraph $\mathcal{G} = (V, E, l, inc)$, the *degree of a hyperedge* $e \in E$ is the number of vertices e is connected with, i.e., $deg(e) := |inc(e)|$. The *degree of* $\mathcal{G}$ is the maximal degree of the hyperedges, $deg(\mathcal{G}) := max\{deg(e) \mid e \in E\}$. ♦

To define the interpretation of processes as graphs compositionally, we require two operations on graphs. The *disjoint union* of two hypergraphs $\mathcal{G}_1$ and $\mathcal{G}_2$, where $E_1 \cap E_2 = \emptyset$, puts both graphs side by side as illustrated in Figure 7.1 (a).

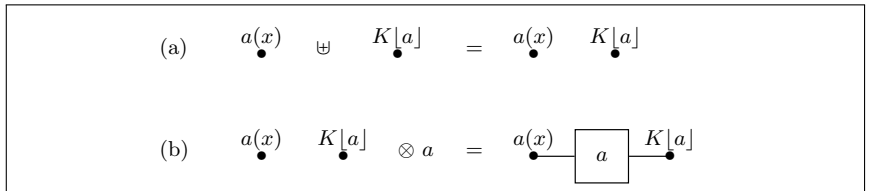


Figure 7.1:
Illustration of the graph operations $\mathcal{G}_1 \uplus \mathcal{G}_2$ in (a) and $\mathcal{G} \otimes a$ in (b).

Definition 7.1.4 (Disjoint union)
Let $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{H}$ with $\mathcal{G}_i = (V_i, E_i, l_i, inc_i)$ and $E_1 \cap E_2 = \emptyset$. The *disjoint union* of $\mathcal{G}_1$ and $\mathcal{G}_2$ is the graph $\mathcal{G}_1 \uplus \mathcal{G}_2 := (V_1 \uplus V_2, E_1 \uplus E_2, l_1 \uplus l_2, inc_1 \uplus inc_2)$. ♦

To lift the restriction operator to graphs, we define the *connect operator*. It takes a graph $\mathcal{G}$ and a name a which is not in the set of edges, $a \notin E$. An application of connect yields the graph $\mathcal{G} \otimes a$, where edge a is added to E. It connects the processes that have a as a free name. Figure 7.1 (b) illustrates the operator.

Definition 7.1.5 (Connect)
Let $\mathcal{G} = (V, E, l, inc)$ and $a \in \mathcal{N}$ with $a \notin E$. The graph $\mathcal{G}$ *connect* a is $\mathcal{G} \otimes a := (V, E \uplus \{a\}, l, inc \uplus \{(a, V_a)\})$, where $V_a \subseteq V$ with $v \in V_a$ iff $a \in fn(l(v))$. ♦

Disjoint union of graphs is commutative and associative. Also for the connect operator a form of commutativity holds. Both laws will be helpful in the proof of Lemma 7.1.10 in the following section.

Lemma 7.1.6
The following equalities hold:

$$\mathcal{G}_1 \uplus \mathcal{G}_2 = \mathcal{G}_2 \uplus \mathcal{G}_1 \qquad \mathcal{G}_1 \uplus (\mathcal{G}_2 \uplus \mathcal{G}_3) = (\mathcal{G}_1 \uplus \mathcal{G}_2) \uplus \mathcal{G}_3$$
$$(\mathcal{G} \otimes a) \otimes b = (\mathcal{G} \otimes b) \otimes a.$$

7.1.2 Graph Interpretation of Processes

Every π-Calculus process can be understood as a hypergraph by (1) creating a vertex for every sequential process, (2) taking the active restrictions as set of hyperedges, and (3) inserting an arc where a name is free in a process. The following function makes this idea precise. Note that active restrictions $\nu a.P$ that are not free in P do not yield hyperedges. This ensures the structurally congruent processes $\nu a.P$ and P have the same graph interpretation.

Definition 7.1.7 ($\mathcal{G} : \mathcal{P} \to \mathcal{H}$)
The *graph-theoretic interpretation* $\mathcal{G} : \mathcal{P} \to \mathcal{H}$ maps a given π-Calculus process P to a hypergraph $\mathcal{G}[\![P]\!]$ as follows:

$$\mathcal{G}[\![M^{=\mathbf{0}}]\!] := (\emptyset, \emptyset, \emptyset, \emptyset) \qquad \mathcal{G}[\![F^e]\!] := (\{v\}, \emptyset, \{(v, F^e)\}, \emptyset)$$
$$\mathcal{G}[\![P \mid Q]\!] := \mathcal{G}[\![P]\!] \uplus \mathcal{G}[\![Q]\!] \qquad \mathcal{G}[\![\nu a.P]\!] := \begin{cases} \mathcal{G}[\![P]\!] \otimes a, & \text{if } a \in fn(P) \\ \mathcal{G}[\![P]\!], & \text{if } a \notin fn(P). \end{cases}$$

♦

We briefly comment on the well-definedness of $\mathcal{G}[\![-]\!]$. By an induction on the structure of processes, we observe that $a \notin arn(P)$ implies $a \notin E_P$, the edge set

of $\mathcal{G}[\![P]\!]$. Since the name a is bound at most once in $\nu a.P$, it is not bound in P. Thus, $a \notin E_P$ and $\mathcal{G}[\![P]\!] \otimes a$ is well-defined. Similarly, we derive that the edge sets in $\mathcal{G}[\![P]\!]$ and $\mathcal{G}[\![Q]\!]$ are disjoint as required by $\mathcal{G}[\![P]\!] \uplus \mathcal{G}[\![Q]\!]$.

Structurally congruent processes $P \equiv Q$ are not mapped to the same hypergraph, i.e., $\mathcal{G}[\![P]\!] \neq \mathcal{G}[\![Q]\!]$. Vertex labels may be replaced by structurally congruent processes or hyperedges together with the attached processes may be renamed. Figure 7.2 illustrates the relationship.

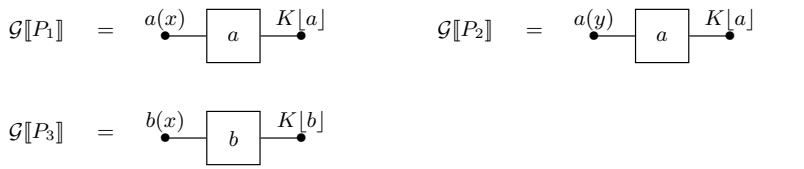


Figure 7.2:
Consider the processes $P_1 = \nu a.(a(x) \mid K\lfloor a \rfloor)$, $P_2 = \nu a.(a(y) \mid K\lfloor a \rfloor)$, and $P_3 = \nu b.(b(x) \mid K\lfloor b \rfloor)$. The processes are structurally congruent, $P_1 \equiv P_2 \equiv P_3$, but the graph interpretations are not equal. They are related by the graph equivalence in Definition 7.1.8.

Similar to standard and restricted equivalence on processes, we define a suitable equivalence relation on graphs of processes. It allows for precisely the two mentioned modifications. We then show that the function $\mathcal{G}[\![-]\!]$ is invariant under structural congruence up to this graph equivalence.

Definition 7.1.8 (Graph equivalence)
Let $\mathcal{G}[\![\mathcal{P}]\!] \subseteq \mathcal{H}$ be the codomain of $\mathcal{G}[\![-]\!]$. We define $\equiv_{\mathcal{G}} \subseteq \mathcal{G}[\![\mathcal{P}]\!] \times \mathcal{G}[\![\mathcal{P}]\!]$ as the smallest equivalence relation on $\mathcal{G}[\![\mathcal{P}]\!]$ satisfying the following two axioms, where all processes are assumed to be in standard form:

$$\mathcal{G}[\![\nu a.P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]$$

with $\{b\} \cap (fn(P^{sf}) \cup bn(P^{sf})) = \emptyset$ and

$$\mathcal{G}[\![\nu \tilde{a}.(M^{\neq \mathbf{0}} \mid P^{\neq \nu})]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![\nu \tilde{a}.(N^{\neq \mathbf{0}} \mid P^{\neq \nu})]\!],$$

where $M^{\neq \mathbf{0}} \equiv N^{\neq \mathbf{0}}$. ♦

Proposition 7.1.9 states the indicated invariance of $\mathcal{G}[\![-]\!]$. That also graph equivalence implies structural congruence means we found another complete characterisation of structural congruence, which relies on only two axioms.

Proposition 7.1.9 (Characterisation of $\equiv$ with $\equiv_{\mathcal{G}}$)
For all processes $P, Q \in \mathcal{P}$ we have $P \equiv Q$ if and only if $\mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!]$.

The definition of graph equivalence resembles the definition of standard equivalence, Definition 2.1.30.[1] The proof of Proposition 7.1.9 benefits from this similarity. Instead of proving the proposition directly, we first show that graph equivalence characterises standard equivalence.

Lemma 7.1.10 (Characterisation of $\equiv_{sf}$ by $\equiv_{\mathcal{G}}$)
For all $P^{sf}, Q^{sf} \in \mathcal{P}_{sf}$ we have $P^{sf} \equiv_{sf} Q^{sf}$ if and only if $\mathcal{G}[\![P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q^{sf}]\!]$.

The proposition then follows with the observation that process P and its standard form $sf(P)$ have the same graph.

Lemma 7.1.11 (Invariance of $\mathcal{G}[\![-]\!]$ under sf)
For all $P \in \mathcal{P}$ the equality $\mathcal{G}[\![P]\!] = \mathcal{G}[\![sf(P)]\!]$ holds.

Before we prove the auxiliary lemmas, we establish Proposition 7.1.9.

Proof (of Proposition 7.1.9)
Consider $P, Q \in \mathcal{P}$. We observe that Lemma 7.1.11, combined with reflexivity and transitivity of $\equiv_{\mathcal{G}}$, justifies the first of the following equivalences:

$$\begin{array}{rcl} & & \mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!] \\ (\text{ Comment above }) & \Leftrightarrow & \mathcal{G}[\![sf(P)]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![sf(Q)]\!] \\ (\text{ Lemma 7.1.10 }) & \Leftrightarrow & sf(P) \equiv_{sf} sf(Q) \\ (\text{ Proposition 2.1.31 }) & \Leftrightarrow & P \equiv Q. \end{array}$$

This proves the characterisation of $P \equiv Q$ via $\mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!]$. ■

Proof (of Lemma 7.1.10)
$\Rightarrow$ We do an induction on the derivations of $\equiv_{sf}$. The base cases are the axioms. For α-conversion of restricted names and for replacing non-empty choices by structurally congruent ones, graph equivalence holds by definition. Consider commutativity of parallel composition:

$$\begin{array}{rcl} & & \mathcal{G}[\![\nu\tilde{a}.(P_1^{\neq\nu} \mid P_2^{\neq\nu})]\!] \\ (\text{ Definition of } \mathcal{G}[\![-]\!]) & = & (\mathcal{G}[\![P_1^{\neq\nu}]\!] \uplus \mathcal{G}[\![P_2^{\neq\nu}]\!]) \otimes \tilde{a} \\ (\text{ Commutativity of } \uplus, \text{ Lemma 7.1.6 }) & = & (\mathcal{G}[\![P_2^{\neq\nu}]\!] \uplus \mathcal{G}[\![P_1^{\neq\nu}]\!]) \otimes \tilde{a} \end{array}$$

[1] Note that it does not require associativity and commutativity of parallel composition and commutativity of restriction.

$$(\text{ Definition of } \mathcal{G}[\![-]\!] \;) \quad = \quad \mathcal{G}[\![\nu\tilde{a}.(P_2^{\neq\nu} \mid P_1^{\neq\nu})]\!].$$

The proofs for associativity of parallel composition and commutativity of restriction are similar and use the remaining two equalities in Lemma 7.1.6.

In the induction step, we assume that $P^{sf} \equiv_{sf} Q^{sf}$ implies $\mathcal{G}[\![P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q^{sf}]\!]$ and similar for $Q^{sf} \equiv_{sf} R^{sf}$. We have to consider $Q^{sf} \equiv_{sf} P^{sf}$ and $P^{sf} \equiv_{sf} R^{sf}$. The graph equivalences $\mathcal{G}[\![Q^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![P^{sf}]\!]$ and $\mathcal{G}[\![P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![R^{sf}]\!]$ immediately follow with the hypothesis and the fact that $\equiv_{\mathcal{G}}$ is an equivalence relation.

$\Leftarrow$ We conduct an induction on the derivations of $\equiv_{\mathcal{G}}$. For the two axioms of graph equivalence in Definition 7.1.8, there is nothing to prove as the processes are standard equivalent by definition. Since $\equiv_{\mathcal{G}}$ is an equivalence relation, we have to consider the graphs that are equivalent by reflexivity, i.e., $\mathcal{G}[\![P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q^{sf}]\!]$ where $\mathcal{G}[\![P^{sf}]\!] = \mathcal{G}[\![Q^{sf}]\!]$. Let $P^{sf} = \nu\tilde{a}.P^{\neq\nu}$ with $P^{\neq\nu} = \Pi_{i\in I} P_i$ and let $Q^{sf} = \nu\tilde{b}.Q^{\neq\nu}$ with $Q^{\neq\nu} = \Pi_{j\in J} Q_j$. We compute the graph interpretations:

$$\mathcal{G}[\![P^{sf}]\!] = (\{v_i \mid i \in I\}, \tilde{a}, l, inc) \text{ with } l(v_i) = P_i$$
$$\mathcal{G}[\![Q^{sf}]\!] = (\{w_j \mid j \in J\}, \tilde{b}, l', inc') \text{ with } l'(w_j) = Q_j.$$

The incidence functions are not important. Since the graphs are equal, we have $\tilde{a} = \tilde{b}$. With commutativity of restriction in $\equiv_{sf}$, we reorder the names $\tilde{b}$ and get $\nu\tilde{b}.Q^{\neq\nu} \equiv_{sf} \nu\tilde{a}.Q^{\neq\nu}$. Equality of the graphs also gives a bijection between the sets of vertices $f : \{v_i \mid i \in I\} \to \{w_j \mid j \in J\}$, which is compatible with the labelling function. With $f(v_i) = w_j$ this means

$$P_i = l(v_i) = l'(f(v_i)) = l'(w_j) = Q_j.$$

Via associativity and commutativity of parallel composition in $\equiv_{sf}$, we reorder the processes Q_j so that $\nu\tilde{a}.(\Pi_{j\in J} Q_j) \equiv_{sf} \nu\tilde{a}.(\Pi_{i\in I} P_i)$. Combining the arguments yields $\nu\tilde{b}.Q^{\neq\nu} \equiv_{sf} \nu\tilde{a}.Q^{\neq\nu} \equiv_{sf} \nu\tilde{a}.P^{\neq\nu}$ and we conclude $P^{sf} \equiv_{sf} Q^{sf}$. The induction step is again trivial as $\equiv_{sf}$ is an equivalence relation. ■

Proof (of Lemma 7.1.11)
We do an induction on the structure of processes. The base cases of empty choices and elementary fragments are trivial. We turn to the induction step.

Assume $\mathcal{G}[\![P]\!] = \mathcal{G}[\![sf(P)]\!]$ holds for process P. Consider $\nu a.P$ with $a \in fn(P)$:

$$\begin{aligned} & & \mathcal{G}[\![\nu a.P]\!] \\ (\text{ Definition of } \mathcal{G}[\![-]\!] \;) & = & \mathcal{G}[\![P]\!] \otimes a \\ (\text{ Hypothesis } \;) & = & \mathcal{G}[\![sf(P)]\!] \otimes a \\ (\text{ Definition of } \mathcal{G}[\![-]\!] \;) & = & \mathcal{G}[\![\nu a.sf(P)]\!] \\ (\text{ Definition of } sf \;) & = & \mathcal{G}[\![sf(\nu a.P)]\!]. \end{aligned}$$

The proofs for $\nu a.P$ with $a \notin fn(P)$ and for $P \mid Q$ are similar. ■

It is important to note that graph equivalence preserves the length of the longest simple path and the graph degree. Combined with the fact that structurally congruent processes yield equivalent graphs, we conclude that the degree and the length of the longest simple paths are invariant under structural congruence of processes.

Lemma 7.1.12 (Invariance of *lsp* and *deg* under $\equiv_{\mathcal{G}}$)
For all $P, Q \in \mathcal{P}$ with $\mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!]$ we have $deg(\mathcal{G}[\![P]\!]) = deg(\mathcal{G}[\![Q]\!])$ and $lsp(\mathcal{G}[\![P]\!]) = lsp(\mathcal{G}[\![Q]\!])$.

Proof
We use induction on the derivations of graph equivalence. For graphs that are equivalent by reflexivity, i.e., $\mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!]$ where $\mathcal{G}[\![P]\!] = \mathcal{G}[\![Q]\!]$, the degree and the length of the simple paths are preserved by the compatibility of the incidence functions. Consider $\mathcal{G}[\![\nu a.P^{sf}]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]$ where $\{b\} \cap (fn(P^{sf}) \cup bn(P^{sf})) = \emptyset$. Let $P^{sf} = \nu\tilde{x}.P^{\neq\nu}$ with $P^{\neq\nu} = \Pi_{i\in I} P_i$. We compute the graph representations:

$$\mathcal{G}[\![\nu a.P^{sf}]\!] = (\{v_i \mid i \in I\}, \tilde{x} \cup \{a\}, l, inc),$$

where $l(v_i) = P_i$ and $v_i \in inc(m)$ iff $m \in fn(P_i)$. Similarly,

$$\mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!] = (\{v_i \mid i \in I\}, \tilde{x} \cup \{b\}, l', inc'),$$

where $l'(v_i) = P_i\{b/a\}$ and $v_i \in inc'(m)$ iff $m \in fn(P_i\{b/a\})$. First, we show equality of the graph degrees. We observe, that inc and inc' coincide on $x \in \tilde{x}$:

$$\begin{aligned} & & v_i \in inc(x) \\ (\text{ Definition of } inc\ ) & \Leftrightarrow & x \in fn(P_i) \\ (\ a \neq x \neq b\ ) & \Leftrightarrow & x \in fn(P_i\{b/a\}) \\ (\text{ Definition of } inc'\ ) & \Leftrightarrow & v_i \in inc'(x). \end{aligned}$$

Hence, $|inc(x)| = |inc'(x)|$ holds. Also a and b are incident with the same vertices, $inc(a) = inc'(b)$, and thus $|inc(a)| = |inc'(b)|$. The degrees of the graphs coincide.

To see that *lsp* is preserved by graph equivalence, let the longest simple path in $\mathcal{G}[\![\nu a.P^{sf}]\!]$ be $p = (v_1, e_1, \ldots, v_n, e_n, v_{n+1})$. If $e_i \in \tilde{x}$, we have $inc(e_i) = inc'(e_i)$ as argued above. Since the definition of paths requires $v_i, v_{i+1} \in inc(e_i)$, we conclude $v_i, v_{i+1} \in inc'(e_i)$. If $e_i = a$, we have $inc(a) = inc'(b)$. Thus, $v_i, v_{i+1} \in inc(a)$ implies $v_i, v_{i+1} \in inc'(b)$. Together, we conclude that the path p' obtained from p by replacing a by b is a path in $\mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]$. Since $length(p') = n$, we conclude that $lsp(\mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]) \geq lsp(\mathcal{G}[\![\nu a.P^{sf}]\!])$. Similarly, if we start with the longest simple path in $\mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]$, we can show that $lsp(\mathcal{G}[\![\nu b.(P^{sf}\{b/a\})]\!]) \leq lsp(\mathcal{G}[\![\nu a.P^{sf}]\!])$. Equality holds.

For replacing non-empty choices, $\mathcal{G}[\![\nu\tilde{a}.(M^{\neq \mathbf{0}} \mid P^{\neq \nu})]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![\nu\tilde{a}.(N^{\neq \mathbf{0}} \mid P^{\neq \nu})]\!]$ with $M^{\neq \mathbf{0}} \equiv N^{\neq \mathbf{0}}$, we again show equality of the incidence functions. This follows from $fn(M^{\neq \mathbf{0}}) = fn(N^{\neq \mathbf{0}})$ due to Lemma 2.1.19. The rest of the proof is analogue. The induction step is trivial as equality (of degree and length) is an equivalence. ■

Combining Lemma 7.1.12 with the fact that structurally congruent processes yield equivalent graphs, we conclude that the degree and the length of the longest simple path are invariant under structural congruence of processes. We shall apply this insight several times in the remainder of the chapter.

Corollary 7.1.13 (Invariance of *lsp* and *deg* under $\equiv$)
Consider processes $P, Q \in \mathcal{P}$ with $P \equiv Q$. Then $lsp(\mathcal{G}[\![P]\!]) = lsp(\mathcal{G}[\![Q]\!])$ and $deg(\mathcal{G}[\![P]\!]) = deg(\mathcal{G}[\![Q]\!])$.

Proof
With Proposition 7.1.9, $P \equiv Q$ implies $\mathcal{G}[\![P]\!] \equiv_{\mathcal{G}} \mathcal{G}[\![Q]\!]$. With Lemma 7.1.12, the corollary holds. ■

We continue with the characterisation of structural stationarity over the functions depth and breadth. In Section 7.4, we return to the graph interpretation of processes to give intuitive explanations for both functions.

7.2 A Second Characterisation of Structural Stationarity

The characterisation of structural stationarity we elaborate in this section refers to the restriction operator. We observe that a bounded number of restricted names does not imply structural stationarity. In fact, a process with only one restricted name may not be structurally stationary. Consider $\nu a.K\lfloor a \rfloor$ with $K(x) := \overline{x}\langle x \rangle \mid K\lfloor x \rfloor$. It generates processes sending on the restricted channel a. The reaction sequence

$$\nu a.K\lfloor a \rfloor \to \nu a.(\overline{a}\langle a \rangle \mid K\lfloor a \rfloor) \to \nu a.(\overline{a}\langle a \rangle \mid \overline{a}\langle a \rangle \mid K\lfloor a \rfloor) \to \ldots$$

forms infinitely many fragments that are pairwise not structurally congruent. Referring to $\nu a.K\lfloor a \rfloor$ as fragment F_1, to $\nu a.(\overline{a}\langle a \rangle \mid K\lfloor a \rfloor)$ as F_2 etc. we obtain the infinite structural semantics $\mathcal{N}[\![F_1]\!]$ depicted in Figure 7.3.

In the graph interpretation, depicted in Figure 7.4, there is no bound on the number of vertices connected with the hyperedge of the restricted name a, i.e., the degree of this edge is not bounded. Thus, the degree of the graphs, which is the maximum of the node degrees, is not bounded.

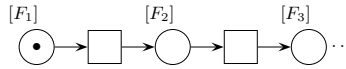


Figure 7.3: Infinite structural semantics in unbounded breadth and depth.

At process level, the degree of a hyperedge labelled by a is the number of sequential processes that share the restricted name. In the restricted form, the hyperedge degree is reflected by the *maximal number of fragments under a restriction*, denoted by $\|F\|_{|}$. For example, the execution above yields $\|\nu a.K\lfloor a\rfloor\|_{|} = 1$ and $\|\nu a.(\overline{a}\langle a\rangle \mid K\lfloor a\rfloor)\|_{|} = 2$. To represent the degree of the graphs, i.e., the maximum of the node degrees, as a function on fragments we search for the widest representation $F_{\mathcal{B}}$ of a fragment F. Widest means that the number of fragments under a restriction in $F_{\mathcal{B}}$ is maximal in the congruence class. The *breadth* of F is then defined by the number of fragments under a restriction in $F_{\mathcal{B}}$. In Section 7.5 we show that this definition of breadth fits to the graph interpretation. The breadth of F equals the degree of $\mathcal{G}[\![F]\!]$.

Figure 7.4: Reaction sequence illustrating unbounded breadth.

Definition 7.2.1 ($\|-\|_{\mathcal{B}} : \mathcal{P}_{\mathcal{F}} \to \mathbb{N}$, boundedness in breadth)
The function $\|-\|_{|} : \mathcal{P}_{\mathcal{F}} \to \mathbb{N}$ gives the *maximal number of fragments under a restriction*:

$$\|F^e\|_{|} := 1 \qquad \|\nu a.(F_1 \mid \ldots \mid F_n)\|_{|} := max\{n, \|F_1\|_{|}, \ldots, \|F_n\|_{|}\}.$$

The *breadth* of fragment F is the maximal number of fragments under a restriction in all fragments structurally congruent with F, i.e.,

$$\|F\|_{\mathcal{B}} := max\{\|G\|_{|} \mid G \equiv F\}.$$

A process $P \in \mathcal{P}$ is *bounded in breadth*, if the breadth of all reachable fragments is bounded by some $k_{\mathcal{B}} \in \mathbb{N}$, i.e.,

$$\exists k_{\mathcal{B}} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf(Q)) : \|F\|_{\mathcal{B}} \leq k_{\mathcal{B}}.$$

The *set of all processes that are bounded in breadth* is $\mathcal{P}_{\mathcal{B}<\infty}$. ♦

By definition, the function $\| - \|_{\mathcal{B}}$ is invariant under structural congruence.

Observation 7.2.2 (Invariance of $\| - \|_{\mathcal{B}}$ under $\equiv$)
For all $F, G \in \mathcal{P}_{\mathcal{F}}$ with $F \equiv G$ it holds $\|F\|_{\mathcal{B}} = \|G\|_{\mathcal{B}}$. ■

As it refers to all fragments in a congruence class, the notion of breadth is hard to grasp. We provide a small example that illustrates the definition.

Example 7.2.3 (Breadth)
Consider $\nu a.L\lfloor a \rfloor$ with $L(x) := \nu b.(\overline{x}\langle b\rangle \mid \overline{x}\langle b\rangle \mid L\lfloor x \rfloor)$. The only reaction sequence is given by

$$\begin{aligned}\nu a.L\lfloor a \rfloor &\to \nu a.(\nu a_1.(\overline{a}\langle a_1\rangle \mid \overline{a}\langle a_1\rangle) \mid L\lfloor a \rfloor)\\ &\to \nu a.(\nu a_1.(\overline{a}\langle a_1\rangle \mid \overline{a}\langle a_1\rangle) \mid \nu a_2.(\overline{a}\langle a_2\rangle \mid \overline{a}\langle a_2\rangle) \mid L\lfloor a \rfloor) \to \dots\end{aligned}$$

After $n \in \mathbb{N}$ reactions we have the following fragment $F_{\mathcal{D}} \equiv F_{\mathcal{B}}$:

$$\begin{aligned}F_{\mathcal{D}} &= \nu a.(\Pi_{i=1}^{n} \nu a_i.(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a \rfloor)\\ F_{\mathcal{B}} &= \nu a_1.(\dots(\nu a_n.(\nu a.(\Pi_{i=1}^{n}(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a \rfloor)))\dots).\end{aligned}$$

We compute $\|F_{\mathcal{D}}\|_{|} = n+1$ and $\|F_{\mathcal{B}}\|_{|} = 2n+1$. In $F_{\mathcal{B}}$ the number of fragments under a restriction is maximal in the congruence class of $F_{\mathcal{D}} \equiv F_{\mathcal{B}}$. So after n reactions we have $\|F_{\mathcal{D}}\|_{\mathcal{B}} = \|F_{\mathcal{B}}\|_{\mathcal{B}} = \|F_{\mathcal{B}}\|_{|} = 2n+1$. There is no bound on the breadth of the reachable fragments, i.e., $\nu a.L\lfloor a \rfloor \notin \mathcal{P}_{\mathcal{B}<\infty}$. ♦

Intuitively, the fragment $F_{\mathcal{B}}$ that maximises $\| - \|_{|}$ minimises the scope of the restricted name, which is shared by most sequential processes. We give the construction of $F_{\mathcal{B}}$ from a given fragment in Section 7.5.

Imposing a bound on the breadth of all reachable fragments does not suffice to show structural stationarity. Consider $\nu a.K\lfloor a \rfloor$ with $K(x) := \nu b.(\overline{b}\langle x\rangle \mid K\lfloor b \rfloor)$. The process generates infinitely many fragments that are pairwise not structurally congruent but have a breadth of two:

$$\nu a.K\lfloor a \rfloor \to \nu a.(\nu b.(\overline{b}\langle a\rangle \mid K\lfloor b \rfloor)) \to \nu a.(\nu b.(\overline{b}\langle a\rangle \mid \nu c.(\overline{c}\langle b\rangle \mid K\lfloor c \rfloor))) \to \dots$$

If we let $F_1 = \nu a.K\lfloor a \rfloor$, $F_2 = \nu a.(\nu b.(\overline{b}\langle a\rangle \mid K\lfloor b \rfloor))$ etc. we again arrive at the infinite structural semantics in Figure 7.3.

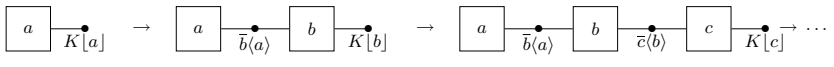


Figure 7.5: Reaction sequence illustrating unbounded depth.

In the graph interpretation in Figure 7.5, the length of the simple paths is not bounded. At process level, this length is mimicked by the nesting of restrictions $\|F\|_\nu$. In the example, $\|\nu a.K\lfloor a\rfloor\|_\nu = 1$ and $\|\nu a.(\nu b.(\overline{b}\langle a\rangle \mid K\lfloor b\rfloor))\|_\nu = 2$. To ensure the restrictions contribute to the length of a simple path, we take a representation $F_\mathcal{D}$ of the given fragment F where this nesting is minimal. Intuitively, $F_\mathcal{D}$ is the flattest representation of F. The *depth* of fragment F is then defined by the nesting of restrictions in this flattest representation $F_\mathcal{D}$. In Section 7.4, we show that this definition of depth corresponds to the intuitive understanding. The depth of all reachable fragments is bounded if and only if the length of all simple paths in the graph interpretation is bounded.

Definition 7.2.4 ($\|-\|_\mathcal{D} : \mathcal{P}_\mathcal{F} \to \mathbb{N}$, boundedness in depth)
The *nesting of restrictions* in a fragment is given by the function $\|-\|_\nu : \mathcal{P}_\mathcal{F} \to \mathbb{N}$ defined inductively as follows:

$$\|F^e\|_\nu := 0 \qquad \|\nu a.(F_1 \mid \ldots \mid F_n)\|_\nu := 1 + max\{\|F_1\|_\nu, \ldots, \|F_n\|_\nu\}.$$

For a fragment F, we define the *depth* to be the minimal nesting of restrictions in all fragments in the congruence class:

$$\|F\|_\mathcal{D} := min\{\|G\|_\nu \;\mid\; G \equiv F\}.$$

A process $P \in \mathcal{P}$ is *bounded in depth*, if there is a bound on the depth of all reachable fragments, i.e.,

$$\exists k_\mathcal{D} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf(Q)) : \|F\|_\mathcal{D} \leq k_\mathcal{D}.$$

The *set of all processes of bounded depth* is $\mathcal{P}_{\mathcal{D}<\infty}$. ♦

Like the breadth, the depth of fragments is invariant under structural congruence.

Observation 7.2.5 (Invariance of $\|-\|_\mathcal{D}$ under $\equiv$)
For all $F, G \in \mathcal{P}_\mathcal{F}$ with $F \equiv G$ it holds $\|F\|_\mathcal{D} = \|G\|_\mathcal{D}$. ■

We continue the investigation of process $\nu a.L\lfloor a\rfloor$ defined in Example 7.2.3.

Example 7.2.6 (Depth)
We observed that all processes reachable via $n \in \mathbb{N}$ reactions are structurally congruent with $F_\mathcal{D} \equiv F_\mathcal{B}$:

$$F_\mathcal{D} = \nu a.(\Pi_{i=1}^n \nu a_i.(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)$$
$$F_\mathcal{B} = \nu a_1.(\ldots(\nu a_n.(\nu a.(\Pi_{i=1}^n(\overline{a}\langle a_i\rangle \mid \overline{a}\langle a_i\rangle) \mid L\lfloor a\rfloor)))\ldots).$$

The nesting function yields $\|F_{\mathcal{D}}\|_\nu = 2$ and $\|F_{\mathcal{B}}\|_\nu = n+1$. Since the nesting of restrictions in $F_{\mathcal{D}}$ is minimal in the congruence class, we have $\|F_{\mathcal{B}}\|_{\mathcal{D}} = \|F_{\mathcal{D}}\|_{\mathcal{D}} = \|F_{\mathcal{D}}\|_\nu = 2$. So the depth all fragments reachable from $\nu a.L\lfloor a \rfloor$ is bounded by two, i.e., $\nu a.L\lfloor a \rfloor \in \mathcal{P}_{\mathcal{D}<\infty}$. ♦

In a fragment F, there are at most $\|F\|_|$ fragments under a restriction. The nesting of restrictions is bounded by $\|F\|_\nu$. Thus, F contains at most $\|F\|_|^{\|F\|_\nu}$ sequential processes.

Lemma 7.2.7 (Elementary Inequality)
For all $F \in \mathcal{P}_{\mathcal{F}} : \|F\|_{\mathcal{S}} \leq \|F\|_|^{\|F\|_\nu}$.

Proof
We proceed by an induction on the structure of fragments.

Base Case For F^e, the inequality holds with $\|F^e\|_{\mathcal{S}} = 1 = 1^0 = \|F^e\|_|^{\|F^e\|_\nu}$.

Induction Step Let $\|F_i\|_{\mathcal{S}} \leq \|F_i\|_|^{\|F_i\|_\nu}$ for all F_i with $1 \leq i \leq n$. We then have for $F = \nu a.(F_1 \mid \ldots \mid F_n)$:

$$
\begin{aligned}
& \|F\|_{\mathcal{S}} \\
(\text{ Def. } \|F\|_{\mathcal{S}} \text{ }) &= \Sigma_{i=1}^n \|F_i\|_{\mathcal{S}} \\
(\text{ Hypothesis }) &\leq \Sigma_{i=1}^n \|F_i\|_|^{\|F_i\|_\nu} \\
(\text{ Def. } max \text{ }) &\leq \Sigma_{i=1}^n max\{\|F_i\|_| \mid 1 \leq i \leq n\}^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}}.
\end{aligned}
$$

With $max_| := max\{\|F_i\|_| \mid 1 \leq i \leq n\}$ we continue:

$$
\begin{aligned}
&= \Sigma_{i=1}^n max_|^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}} \\
&= n \cdot max_|^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}} \\
(\text{ Def. } max \text{ }) &\leq max\{n, max_|\} \cdot max\{n, max_|\}^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}} \\
&= max\{n, max_|\}^{1+max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}} \\
(\text{ Def. } \|F\|_\nu \text{ }) &= max\{n, max_|\}^{\|F\|_\nu} \\
(\text{ Def. } max_| \text{ }) &= max\{n, \|F_1\|_|, \ldots, \|F_n\|_|\}^{\|F\|_\nu} \\
(\text{ Def. } \|F\|_| \text{ }) &= \|F\|_|^{\|F\|_\nu}.
\end{aligned}
$$

■

In fact, boundedness in breadth and in depth entails structural stationarity—the main result in this section. Since the reverse direction is trivial, Theorem 7.2.8

provides the following complete characterisation of structural stationarity. A process is structurally stationary if and only if it is bounded in breadth and bounded in depth. While the proof of structural stationarity from boundedness in the sequential processes in Theorem 4.3.2 is direct and cumbersome, the application of the theorem yields an elegant proof of Theorem 7.2.8. We briefly sketch it.

If the process under consideration is bounded in depth by $k_{\mathcal{D}}$, the nesting of restrictions in the flattest representation $F_{\mathcal{D}}$ of a given fragment F is bounded by $k_{\mathcal{D}}$, $\|F_{\mathcal{D}}\|_\nu \leq k_{\mathcal{D}}$. The number of fragments under a restriction in $F_{\mathcal{D}}$ is bounded by the breadth of F, which in turn is assumed to be bounded by $k_{\mathcal{B}}$. Hence, $\|F_{\mathcal{D}}\|_| \leq k_{\mathcal{B}}$. With the elementary inequality in Lemma 7.2.7 we have a bounded number of sequential processes in $F_{\mathcal{D}}$ and so in F. With Theorem 4.3.2 we conclude structural stationarity.

Theorem 7.2.8 (Characterisation of Structural Stationarity via ν)
$\mathcal{P}_{FG<\infty} = \mathcal{P}_{\mathcal{B}<\infty} \cap \mathcal{P}_{\mathcal{D}<\infty}$.

Proof
$\Rightarrow$ If the process is structurally stationary, there is a finite set of fragments $\{F_1, \ldots, F_n\}$ so that every reachable fragment is structurally congruent with some F_i. Then the maxima $max\{\|F_i\|_{\mathcal{D}} \mid 1 \leq i \leq n\}$ and $max\{\|F_i\|_{\mathcal{B}} \mid 1 \leq i \leq n\}$ exist and bind the depth and the breadth of all reachable fragments.

$\Leftarrow$ If we assume boundedness in breadth and depth there are $k_{\mathcal{B}}$ and $k_{\mathcal{D}}$ so that for all $Q \in Reach(P)$ and all $F \in fg\,(rf(Q))$ we have $\|F\|_{\mathcal{B}} \leq k_{\mathcal{B}}$ and $\|F\|_{\mathcal{D}} \leq k_{\mathcal{D}}$. We establish boundedness in the number of sequential processes:

$$\exists k_{\mathcal{S}} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf(Q)) : \|F\|_{\mathcal{S}} \leq k_{\mathcal{S}}.$$

We claim that $k_{\mathcal{B}}^{k_{\mathcal{D}}}$ is a bound. Consider $Q \in Reach(P)$ and $F \in fg\,(rf(Q))$. For every fragment F, there is $F_{\mathcal{D}} \equiv F$ so that $\|F_{\mathcal{D}}\|_\nu = min\{\|G\|_\nu \mid G \equiv F\} = \|F\|_{\mathcal{D}}$. For this $F_{\mathcal{D}}$ the inequality $\|F_{\mathcal{D}}\|_| \leq max\{\|G\|_| \mid G \equiv F\} = \|F\|_{\mathcal{B}}$ holds. We compute

$$\begin{aligned}
& & \|F\|_{\mathcal{S}} \\
(\ \|-\|_{\mathcal{S}} \text{ invariant under } \equiv\ ) & = & \|F_{\mathcal{D}}\|_{\mathcal{S}} \\
(\ \text{Elementary inequality, Lemma 7.2.7}\ ) & \leq & \|F_{\mathcal{D}}\|_|^{\|F_{\mathcal{D}}\|_\nu} \\
(\ \text{Observation } \|F_{\mathcal{D}}\|_| \leq \|F\|_{\mathcal{B}}\ ) & \leq & \|F\|_{\mathcal{B}}^{\|F_{\mathcal{D}}\|_\nu} \\
(\ \text{Observation } \|F_{\mathcal{D}}\|_\nu = \|F\|_{\mathcal{D}}\ ) & = & \|F\|_{\mathcal{B}}^{\|F\|_{\mathcal{D}}} \\
(\ k_{\mathcal{B}} \text{ and } k_{\mathcal{D}} \text{ bounds on breadth and depth}\ ) & \leq & k_{\mathcal{B}}^{k_{\mathcal{D}}}.
\end{aligned}$$

This proves P is bounded in the number of sequential processes. With Theorem 4.3.2, P is structurally stationary. ■

The reformulation of Theorem 7.2.8 is useful in disproving structural stationarity. A process is not structurally stationary if and only if it is not bounded in breadth or not bounded in depth. Thus, there are two sources of infinity for the structural semantics. We discuss that they are of different quality.

Consider $\nu a.L\lfloor a \rfloor$ with $L(x) := \nu b.(\overline{x}\langle b\rangle \mid \overline{x}\langle b\rangle \mid L\lfloor x \rfloor)$ in Example 7.2.3 and 7.2.6. Removing the restriction νa gives the process $L\lfloor a \rfloor$. The semantics $\mathcal{N}[\![L\lfloor a \rfloor]\!]$ is depicted in Figure 7.6. In any execution, the number of tokens on $[\nu b.(\overline{a}\langle b\rangle \mid \overline{a}\langle b\rangle)]$ is not bounded. We conclude that $\nu a.L\lfloor a \rfloor$ is not bounded in breadth. With Theorem 7.2.8 the process is not structurally stationary.

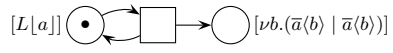


Figure 7.6: Identifying and handling unboundedness in breadth.

The example suggests that processes of bounded in depth but unbounded breadth are not Turing complete. In fact, we show that termination is decidable for these processes in Section 8.2. Conversely, processes of bounded breadth but unbounded depth are Turing complete. This follows from a well-known encoding of counter machines that we recall in Section 8.3

Before we proceed to these decidability results, we prove our graph-theoretic intuition to $\mathcal{P}_{\mathcal{D}<\infty}$ and $\mathcal{P}_{\mathcal{B}<\infty}$ correct (cf. Figure 7.4 and 7.5). The main technical contribution is the definition of a syntactic subclass of fragments, called *anchored fragments*. In this chapter, we use them to derive boundedness in depth from boundedness in the simple paths (Lemma 7.4.1). In Chapter 8 they help us find a well-quasi-ordering on processes of bounded depth (Lemma 8.2.14).

7.3 Anchored Fragments

Anchored fragments are a restricted class of fragments, which enjoys the following technical property. The nesting of restrictions in an anchored fragment corresponds to the length of a simple path in the graph interpretation (Lemma 7.3.2). This relation allows us to prove that anchored fragments are particularly flat: the nesting of restrictions in $F^{\mathcal{A}}$ is bounded by the depth of $F^{\mathcal{A}}$ (Corollary 7.4.5).[2]

The importance of anchored fragments stems from Proposition 7.3.4: *anchored fragments are a normal form under structural congruence*. This means, for a given fragment F we can construct a structurally congruent anchored fragment $F^{\mathcal{A}}$, where $\|F^{\mathcal{A}}\|_{\nu}$ is bounded by $\|F\|_{\mathcal{D}}$. Lemma 7.4.1 and Lemma 8.2.14 are two important applications of the corresponding inequality.

[2]Fragment $F_{\mathcal{B}}$ in Example 7.2.6 illustrates that this does not hold for arbitrary fragments.

In a fragment $F = \nu a.(F_1 \mid \ldots \mid F_n)$, all F_i share the name a. In an *anchored fragment* $F^{\mathcal{A}} = \nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}})$, distinguished processes inside the fragments $F_i^{\mathcal{A}}$ share the name a. These processes are called *anchors* and denoted by $anc(F_i^{\mathcal{A}})$. The definition guarantees that the vertices labelled by the anchors $anc(F_i^{\mathcal{A}})$ are connected via a in the graph interpretation of $F^{\mathcal{A}}$. Figure 7.7 illustrates this idea.

Definition 7.3.1 (Anchored Fragments)
The *set of anchored fragments* $\mathcal{P}_{\mathcal{A}}$ with elements $F^{\mathcal{A}}$ and $G^{\mathcal{A}}$ is defined by

$$F^{\mathcal{A}} \quad ::= \quad F^e \;\mid\; \nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}}),$$

where $a \in fn(anc(F_i^{\mathcal{A}}))$ for all i, $anc(F^e) := F^e$ and $anc(\nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}})) := anc(F_1^{\mathcal{A}})$. ♦

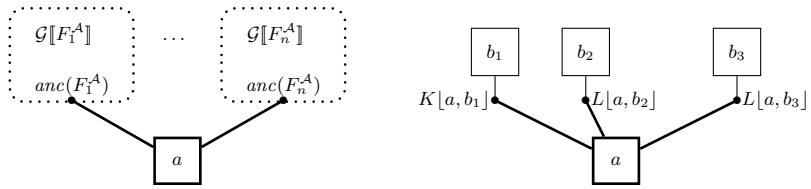


Figure 7.7:
The idea of anchored fragments $F^{\mathcal{A}} = \nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}})$ is to connect the anchors $anc(F_i^{\mathcal{A}})$ in the graph. This is illustrated schematically to the left and for the concrete anchored fragment in Example 7.3.3 to the right.

Since anchored fragments are defined inductively, connectedness not only holds in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$ with $F^{\mathcal{A}} = \nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}})$ but also in each of the $F_i^{\mathcal{A}}$. Hence, when descending the anchored fragment $F^{\mathcal{A}}$ with the function $\| - \|_\nu$, we follow a simple path p in the graph $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. Consequently, for an anchored fragment $F^{\mathcal{A}}$ the nesting of restrictions corresponds to the length of a simple path p in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$, $\|F^{\mathcal{A}}\|_\nu = length(p)$. This is stated in the following Lemma 7.3.2. In the proof, we need that the first element of p is labelled by the anchor of $F^{\mathcal{A}}$, $l(fe(p)) = anc(F^{\mathcal{A}})$.

Lemma 7.3.2
Let $F^{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$. There is a simple path $p \in Paths(\mathcal{G}[\![F^{\mathcal{A}}]\!])$ with $length(p) = \|F^{\mathcal{A}}\|_\nu$ and $l(fe(p)) = anc(F^{\mathcal{A}})$.

We illustrate the construction of a suitable path p in the induction step. The idea is to extend a path p' that exists by the hypothesis by an edge and a vertex.

Example 7.3.3 (Illustration of Lemma 7.3.2)
Consider $F^{\mathcal{A}} = \nu a.(F_1^{\mathcal{A}} \mid F_2^{\mathcal{A}} \mid F_3^{\mathcal{A}})$ with $F_1^{\mathcal{A}} = \nu b_1.K\lfloor a, b_1\rfloor$, $F_2^{\mathcal{A}} = \nu b_2.L\lfloor a, b_2\rfloor$, and $F_3^{\mathcal{A}} = \nu b_3.L\lfloor a, b_3\rfloor$. The vertex labels are unique in this example. Therefore, we identify v with $l(v)$. The nesting of restrictions is

$$\|F^{\mathcal{A}}\|_\nu = 1 + max\{\|F_1^{\mathcal{A}}\|_\nu, \|F_2^{\mathcal{A}}\|_\nu, \|F_3^{\mathcal{A}}\|_\nu\} = 1 + \|\nu b_3.L\lfloor a, b_3\rfloor\|_\nu = 2.$$

Since the nesting of restrictions is equal in $F_1^{\mathcal{A}}$, $F_2^{\mathcal{A}}$, and $F_3^{\mathcal{A}}$ we can choose any $F_i^{\mathcal{A}}$ to compute the nesting of restrictions of $F^{\mathcal{A}}$, in particular $F_3^{\mathcal{A}}$. Figure 7.8 shows a simple path $p = (K\lfloor a, b_1\rfloor, a, L\lfloor a, b_3\rfloor, b_3, L\lfloor a, b_3\rfloor)$ in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$, which satisfies $length(p) = 2 = \|F^{\mathcal{A}}\|_\nu$ and $l(fe(p)) = K\lfloor a, b_1\rfloor = anc(F^{\mathcal{A}})$. We explain its construction.

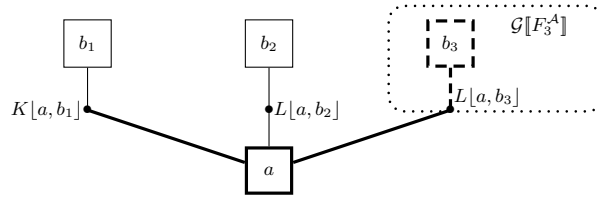


Figure 7.8:
Construction of a path p in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$ with $length(p) = \|F^{\mathcal{A}}\|_\nu$ and $l(fe(p)) = anc(F^{\mathcal{A}})$.

According to the hypothesis, there is a simple path p' in $\mathcal{G}[\![F_3^{\mathcal{A}}]\!]$ that satisfies $length(p') = 1 = \|F_3^{\mathcal{A}}\|_\nu$ and $l(fe(p')) = L\lfloor a, b_3\rfloor = anc(F_3^{\mathcal{A}})$. This path is $p' = (L\lfloor a, b_3\rfloor, b_3, L\lfloor a, b_3\rfloor)$, depicted by dashed lines. As $\mathcal{G}[\![F_3^{\mathcal{A}}]\!]$ is embedded in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$, indicated by the dotted frame, p' is a path in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. The anchor $L\lfloor a, b_3\rfloor$ and $K\lfloor a, b_1\rfloor$, the anchor of $F^{\mathcal{A}}$, are connected with a by definition of anchored fragments. We define $p = (anc(F^{\mathcal{A}}), a, p')$. The path extends p' by the bold lines. ♦

Proof (of Lemma 7.3.2)
We conduct an induction on the structure of anchored fragments.

Base Case Consider F^e with $\mathcal{G}[\![F^e]\!] = (\{v\}, \emptyset, \{(v, F^e)\}, \emptyset)$. For the only simple path $p = (v)$, we get $length(p) = 0 = \|F^e\|_\nu$ and $l(fe(p)) = F^e = anc(F^e)$.

Induction Step Let the statement hold for $F_1^{\mathcal{A}}, \ldots, F_n^{\mathcal{A}}$ with $a \in fn(anc(F_i^{\mathcal{A}}))$ and consider $F^{\mathcal{A}} = \nu a.(F_1^{\mathcal{A}} \mid \ldots \mid F_n^{\mathcal{A}})$. The nesting of restrictions is

$$\|F^{\mathcal{A}}\|_\nu = 1 + max\{\|F_i^{\mathcal{A}}\|_\nu \mid 1 \leq i \leq n\} = 1 + \|F_k^{\mathcal{A}}\|_\nu, \text{ for some } k.$$

By the hypothesis, there are simple paths $p_i \in Paths(\mathcal{G}[\![F_i^{\mathcal{A}}]\!])$ with $length(p_i) = \|F_i^{\mathcal{A}}\|_\nu$ and $l_i(fe(p_i)) = anc(F_i^{\mathcal{A}})$ for all i. We show that $p = (fe(p_1), a, p_k)$ is a simple path in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$ that satisfies the requirements.

By definition, $\mathcal{G}[\![F^{\mathcal{A}}]\!] = (\mathcal{G}[\![F_1^{\mathcal{A}}]\!] \uplus \ldots \uplus \mathcal{G}[\![F_n^{\mathcal{A}}]\!]) \otimes a$. Thus, $\mathcal{G}[\![F_1^{\mathcal{A}}]\!]$ and $\mathcal{G}[\![F_k^{\mathcal{A}}]\!]$ are subgraphs of $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. Therefore, $fe(p_1)$ is a vertex and p_k is a path in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. We observe that $a \in fn(anc(F_1^{\mathcal{A}})) = fn(l_1(fe(p_1)))$. By definition of the connect operator, the first element of p_1 is connected with a in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$, $fe(p_1) \in inc(a)$. Similarly, we derive $fe(p_k) \in inc(a)$. We conclude that $fe(p_1)$ and $fe(p_k)$ are connected via a in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. Thus, p is a path in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. Since p_k is simple and a is a hyperedge that is not in $\mathcal{G}[\![F_k^{\mathcal{A}}]\!]$ (cf. definition of $\otimes$), p is simple as well. By construction, the length and the first element of p are correct. ■

Any fragment can be rewritten into an anchored fragment using structural congruence. In the proof, it is important that every sequential process inside a fragment can be chosen as the anchor.

Proposition 7.3.4 (Normal Form Result)
Consider $F \in \mathcal{P}_{\mathcal{F}}$ and a process $P \in \mathcal{S}(F)$. Then there is an anchored fragment $F^{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$ so that $F^{\mathcal{A}} \equiv F$, $\mathcal{S}(F^{\mathcal{A}}) = \mathcal{S}(F)$, and $anc(F^{\mathcal{A}}) = P$.

We explain the construction in two steps. First, we give an explanation with help of the graph interpretation, then we rephrase the idea on processes.

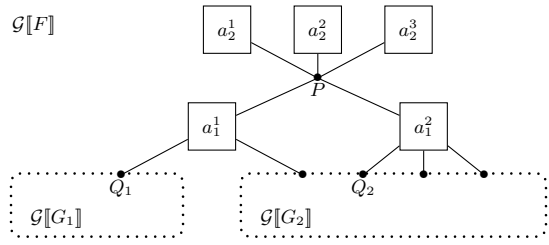


Figure 7.9:
Construction of the anchored fragment $F^{\mathcal{A}} \equiv F$ with $\mathcal{S}(F^{\mathcal{A}}) = \mathcal{S}(F)$ and $anc(F^{\mathcal{A}}) = P$. The figure itself is explained in the text.

When we understand a fragment F as hypergraph $\mathcal{G}[\![F]\!]$, a process $P \in \mathcal{S}(F)$ is a vertex, cf. Figure 7.9. This vertex is connected with several hyperedges, which

can be devided into two sets $\tilde{a}_1$ and $\tilde{a}_2$. Edges in $\tilde{a}_1$ connect P with the remainder of the graph, edges in $\tilde{a}_2$ are only connected with P. In the figure, $\tilde{a}_1 = \{a_1^1, a_1^2\}$ and $\tilde{a}_2 = \{a_2^1, a_2^2, a_2^3\}$. The remaining graph consists of several unconnected graphs $\mathcal{G}[\![G_1]\!], \ldots, \mathcal{G}[\![G_m]\!]$. In Figure 7.9, we have $\mathcal{G}[\![G_1]\!]$ and $\mathcal{G}[\![G_2]\!]$. Since $\mathcal{G}[\![F]\!]$ is connected, each of the graphs $\mathcal{G}[\![G_i]\!]$ contains a vertex that is connected with a name in $\tilde{a}_1$. In the figure, the vertices are Q_1 and Q_2.

To construct the anchored fragment $F^{\mathcal{A}}$, we first recursively apply the construction to each of the fragments G_i. This means, we build anchored fragments $G_i^{\mathcal{A}}$ with the processes Q_i as anchors. The fragment $F^{\mathcal{A}}$ is then (the restricted form of) $\nu\tilde{a}_1.(\nu\tilde{a}_2.P \mid G_1^{\mathcal{A}} \mid \ldots \mid G_m^{\mathcal{A}})$.

On processes, the construction works as follows. We first compute the standard form of the given fragment, $sf(F) = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$. Since computing the standard form does not change the sequential processes, one process P_i is the process P of interest, say P_1. We split the set of names $\tilde{a}$ into three subsets $\tilde{a}_1, \tilde{a}_2, \tilde{a}_3$ as follows. A name a that is shared by P and $P_2 \mid \ldots \mid P_n$ is in the set $\tilde{a}_1$. A name that is free only in P is in $\tilde{a}_2$. The remaining names are in $\tilde{a}_3$. While the sets $\tilde{a}_1$ and $\tilde{a}_2$ are like in the graph construction, the names $\tilde{a}_3$ were hidden in the dotted boxes in Figure 7.9. Shrinking the scopes yields $\nu\tilde{a}_1.(\nu\tilde{a}_2.P \mid \nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n))$. To transform $\nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n)$ into a parallel composition of anchored fragments, we compute the restricted form. It consists of several fragments, $rf(\nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n)) = G_1 \mid \ldots \mid G_m$. By construction, every G_i contains a process Q_i sharing a name with P. Since each G_i has less processes than F we can apply the hypothesis. This yields anchored fragments $G_i^{\mathcal{A}}$, where $anc(G_i^{\mathcal{A}}) = Q_i$ shares a name with P. We now have $\nu\tilde{a}_1.(\nu\tilde{a}_2.P \mid G_1^{\mathcal{A}} \mid \ldots \mid G_m^{\mathcal{A}})$. As the names in $\tilde{a}_1$ are shared by different $G_i^{\mathcal{A}}$, we minimise their scopes to get the required anchored fragment. Before we turn this argumentation into a formal proof, we illustrate it on an example.

Example 7.3.5
Let $F = \nu b_1, b_2, b_3, a.(K\lfloor a, b_1\rfloor \mid L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor)$. We construct an anchored fragment with $K\lfloor a, b_1\rfloor$ as anchor. Fragment F is in standard form. We split the set of names $\{a, b_1, b_2, b_3\}$ into $\tilde{a}_1 = \{a\}$, $\tilde{a}_2 = \{b_1\}$, and $\tilde{a}_3 = \{b_2, b_3\}$. Shrinking the scopes yields $\nu a.(\nu b_1.K\lfloor a, b_1\rfloor \mid \nu b_2, b_3.(L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor))$. The restricted form of $\nu b_2, b_3.(L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor)$ is $\nu b_2.L\lfloor a, b_2\rfloor \mid \nu b_3.L\lfloor a, b_3\rfloor$. Both fragments, $\nu b_2.L\lfloor a, b_2\rfloor$ and $\nu b_3.L\lfloor a, b_3\rfloor$, are anchored fragments where the anchors share the name a with $K\lfloor a, b_1\rfloor$. The scope of a is minimal and the computation returns the anchored fragment $\nu a.(\nu b_1.K\lfloor a, b_1\rfloor \mid \nu b_2.L\lfloor a, b_2\rfloor \mid \nu b_3.L\lfloor a, b_3\rfloor)$. ♦

Proof (of Proposition 7.3.4)
We do a well-founded induction on the number of sequential processes in a fragment.

Base Case Consider $F \in \mathcal{P}_\mathcal{F}$ with $\|F\|_\mathcal{S} = 1$ and $P \in \mathcal{S}(F)$. As the function $\|F\|_\mathcal{S}$ does not count restrictions, we conclude $F = \nu\tilde{a}.F^e$ for some set of names $\tilde{a} \subseteq fn(F^e)$. Obviously, F is an anchored fragment. The anchor is correct since $P \in \mathcal{S}(\nu\tilde{a}.F^e) = \{F^e\}$ implies $P = F^e = anc(\nu\tilde{a}.F^e)$.

Induction Step Assume the statement holds for all G with $1 \leq \|G\|_\mathcal{S} < n$. We prove that it also holds for F with $\|F\|_\mathcal{S} = n$. Let $P \in \mathcal{S}(F)$. To begin with, we compute the standard form of F with Lemma 2.1.28, i.e., $F \equiv sf(F) = \nu\tilde{a}.(P_1 \mid \ldots \mid P_n)$:

$$\begin{array}{rcl} & & F \\ (\text{ Standard form, } \|-\|_\mathcal{S} \text{ invariant under } \equiv) & \equiv & \nu\tilde{a}.(P_1 \mid \ldots \mid P_n) \\ (\text{ Lemma 2.1.28: } \mathcal{S}(P) = \mathcal{S}(sf(P)), \text{ wlog. } P = P_1) & \equiv & \nu\tilde{a}.(P \mid P_2 \mid \ldots \mid P_n). \end{array}$$

We decompose $\tilde{a}$ into three disjoint subsets

$$\begin{aligned} \tilde{a}_1 &:= \tilde{a} \cap fn(P) \cap fn(P_2 \mid \ldots \mid P_n) \\ \tilde{a}_2 &:= (\tilde{a} \cap fn(P)) \setminus fn(P_2 \mid \ldots \mid P_n) \\ \tilde{a}_3 &:= (\tilde{a} \cap fn(P_2 \mid \ldots \mid P_n)) \setminus fn(P) \end{aligned}$$

and apply the rule for scope extrusion to shrink the scopes. This explains the first of the following congruences. In the next, we compute the restricted form of $\nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n)$ with Lemma 3.2.7, $rf(\nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n)) = G_1 \mid \ldots \mid G_m$:

$$\begin{array}{rcl} (\text{ Scope extrusion }) & \equiv & \nu\tilde{a}_1.(\nu\tilde{a}_2.P \mid \nu\tilde{a}_3.(P_2 \mid \ldots \mid P_n)) \\ (\text{ Restricted form }) & \equiv & \nu\tilde{a}_1.(\nu\tilde{a}_2.P \mid G_1 \mid \ldots \mid G_m) \\ (\text{ Let } \tilde{a}_1 = a_k, \ldots, a_1) & = & \nu a_k, \ldots, a_1.(\nu\tilde{a}_2.P \mid G_1 \mid \ldots \mid G_m). \end{array}$$

For every fragment G_i there is a name $a_l \in fn(G_i)$. Otherwise, F would be decomposed into at least two fragments when the restricted form is computed, $F = rf(F) \equiv_{rf} rf(\nu\tilde{a}_1.(\ldots)) = rf(\nu\tilde{a}_1.(\ldots)) \mid G_i$. This contradicts $\equiv_{rf}$. We define $I_l \subseteq \{1, \ldots, m\}$ by $i \in I_l$ iff $a_l \in fn(G_i)$. The argumentation shows that for every G_i the index i is in some set I_l. With scope extrusion, we continue the congruence:

$$\equiv \nu a_k.(\ldots \nu a_1.(\nu\tilde{a}_2.P \mid \Pi_{i \in I_1} G_i) \ldots \Pi_{i \in I_k \setminus (I_{k-1} \cup \ldots \cup I_1)} G_i).$$

Since $a_l \in fn(G_i)$ with $i \in I_l \setminus (I_{l-1} \cup \ldots \cup I_1)$, there is $Q_i \in \mathcal{S}(G_i)$ with $a_l \in fn(Q_i)$. As $\|G_i\|_\mathcal{S} \leq n - 1$, the induction hypothesis is applicable to G_i. This yields $G_i^\mathcal{A} \equiv G_i$ with $\mathcal{S}(G_i^\mathcal{A}) = \mathcal{S}(G_i)$ and $anc(G_i^\mathcal{A}) = Q_i$:

$$\begin{array}{rcl} & \equiv & \nu a_k.(\ldots \nu a_1.(\nu\tilde{a}_2.P \mid \Pi_{i \in I_1} G_i^\mathcal{A}) \ldots \Pi_{i \in I_k \setminus (I_{k-1} \cup \ldots \cup I_1)} G_i^\mathcal{A}) \\ & =: & F^\mathcal{A}. \end{array}$$

Of course, $F^\mathcal{A} \in \mathcal{P}_\mathcal{A}$, $\mathcal{S}(F^\mathcal{A}) = \mathcal{S}(F)$, and $anc(F^\mathcal{A}) = P$. ■

7.4 Characterisation of Boundedness in Depth

With the interpretation of processes as graphs, we call process $P \in \mathcal{P}$ *bounded in the simple paths*, if there is $k_{lsp} \in \mathbb{N}$ so that the length of the longest simple path in the hypergraphs of all reachable fragments is less or equal to k_{lsp}, i.e.,

$$\exists k_{lsp} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf\,(Q)) : lsp(\mathcal{G}[\![F]\!]) \leq k_{lsp}.$$

The *set of all processes that are bounded in the simple paths* is $\mathcal{P}_{lsp<\infty}$. We prove that a process is bounded in depth if and only if it is bounded in the simple paths. Thus, processes in $\mathcal{P}_{\mathcal{D}<\infty}$ can be intuitively understood as hypergraphs where the length of the simple paths is bounded.

With the inequality $\|F\|_{\mathcal{D}} \leq lsp(\mathcal{G}[\![F]\!])$, boundedness in depth follows from boundedness in the simple paths.

Lemma 7.4.1
For all $F \in \mathcal{P}_{\mathcal{F}}$ the inequality $\|F\|_{\mathcal{D}} \leq lsp(\mathcal{G}[\![F]\!])$ holds.

Proof
With Proposition 7.3.4, there is an anchored fragment $F^{\mathcal{A}} \equiv F$. By definition of depth, we have $\|F\|_{\mathcal{D}} = min\{\|G\|_{\nu} \mid G \equiv F\} \leq \|F^{\mathcal{A}}\|_{\nu}$. Lemma 7.3.2 gives a simple path p in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$ with $\|F^{\mathcal{A}}\|_{\nu} = length(p)$. Combining the arguments yields:

$$\|F\|_{\mathcal{D}} \leq \|F^{\mathcal{A}}\|_{\nu} = length(p) \leq lsp(\mathcal{G}[\![F^{\mathcal{A}}]\!]) = lsp(\mathcal{G}[\![F]\!]).$$

The second inequality holds by definition of the longest simple path, the last equality is Corollary 7.1.13. ■

To prove the reverse direction, we need that the length of the longest simple path in $\mathcal{G}[\![F]\!]$ is bounded by the nesting of restrictions in F.

Lemma 7.4.2
For $F \in \mathcal{P}_{\mathcal{F}}$ the inequality $lsp(\mathcal{G}[\![F]\!]) \leq 2^{\|F\|_{\nu}} - 1$ holds.

Proof
We use induction on the structure of fragments.

Base Case Elementary fragments F^e form the base case:

$$lsp(\mathcal{G}[\![F^e]\!]) = 0 = 2^0 - 1 = 2^{\|F^e\|_{\nu}} - 1.$$

Induction Step Assume the inequality holds for $F_1, \ldots, F_n$ with $a \in fn(F_i)$ for all i. We consider $F = \nu a.(F_1 \mid \ldots \mid F_n)$:

$$\begin{array}{rcl} & & lsp(\mathcal{G}[\![F]\!]) \\ (\text{ Def. } \mathcal{G}[\![-]\!]) & = & lsp((\mathcal{G}[\![F_1]\!] \uplus \ldots \uplus \mathcal{G}[\![F_n]\!]) \otimes a). \end{array}$$

The graphs $\mathcal{G}[\![F_i]\!]$ are not connected in $\mathcal{G}[\![F_1]\!] \uplus \ldots \uplus \mathcal{G}[\![F_n]\!]$. Thus, a is the only connection between $\mathcal{G}[\![F_i]\!]$ and $\mathcal{G}[\![F_j]\!]$ in $(\mathcal{G}[\![F_1]\!] \uplus \ldots \uplus \mathcal{G}[\![F_n]\!]) \otimes a$. In the worst case, a connects two simple paths of length $max\{lsp(\mathcal{G}[\![F_i]\!]) \mid 1 \leq i \leq n\}$:

$$\begin{array}{rcl} & \leq & 2 \cdot max\{lsp(\mathcal{G}[\![F_i]\!]) \mid 1 \leq i \leq n\} + 1 \\ (\text{ Def. } max) & = & 2 \cdot lsp(\mathcal{G}[\![F_k]\!]) + 1, \text{ for some } k \\ (\text{ Hypothesis }) & \leq & 2 \cdot (2^{\|F_k\|_\nu} - 1) + 1, \text{ for some } k \\ & = & 2^{\|F_k\|_\nu + 1} - 1, \text{ for some } k \\ (\text{ Def. } max) & \leq & 2^{max\{\|F_i\|_\nu \mid 1 \leq i \leq n\}+1} - 1 \\ (\text{ Def. } \|-\|_\nu) & = & 2^{\|F\|_\nu} - 1. \end{array}$$

■

An application of Lemma 7.4.2 shows that the longest simple path in $\mathcal{G}[\![F]\!]$ is bounded by the depth of F. Hence, boundedness in depth implies boundedness in the simple paths.

Lemma 7.4.3
For all $F \in \mathcal{P}_\mathcal{F}$ the inequality $lsp(\mathcal{G}[\![F]\!]) \leq 2^{\|F\|_\mathcal{D}} - 1$ holds.

Proof
There is a fragment $F_\mathcal{D}$ in the congruence class of F, where the nesting of restrictions is minimal, i.e., $\|F_\mathcal{D}\|_\nu = min\{\|G\|_\nu \mid G \equiv F\} = \|F\|_\mathcal{D}$. With Corollary 7.1.13, Lemma 7.4.2, and the choice of $F_\mathcal{D}$ we derive

$$lsp(\mathcal{G}[\![F]\!]) = lsp(\mathcal{G}[\![F_\mathcal{D}]\!]) \leq 2^{\|F_\mathcal{D}\|_\nu} - 1 = 2^{\|F\|_\mathcal{D}} - 1.$$

■

Lemma 7.4.1 and Lemma 7.4.3 prove Theorem 7.4.4.

Theorem 7.4.4 (Characterisation of Boundedness in Depth)
$\mathcal{P}_{\mathcal{D}<\infty} = \mathcal{P}_{lsp<\infty}$.

In Section 7.3, we claimed that anchored fragments are particularly flat. The property follows as a corollary of Lemma 7.3.2 and Lemma 7.4.3. We shall need it in Section 8.2 to understand anchored fragments as trees of bounded height.

Corollary 7.4.5
For $F^{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$ the inequality $\|F^{\mathcal{A}}\|_{\nu} \leq 2^{\|F^{\mathcal{A}}\|_{\mathcal{D}}} - 1$ holds.

Proof
Consider an anchored fragment $F^{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$. With Lemma 7.3.2, the definition of the longest simple path, and Lemma 7.4.3, the (in)equalities

$$\|F^{\mathcal{A}}\|_{\nu} = \mathit{length}(p) \leq \mathit{lsp}(\mathcal{G}[\![F^{\mathcal{A}}]\!]) \leq 2^{\|F^{\mathcal{A}}\|_{\mathcal{D}}} - 1$$

hold for some simple path p in $\mathcal{G}[\![F^{\mathcal{A}}]\!]$. ■

7.5 Characterisation of Breadth

We establish the following result. For non-elementary fragments F the equation $\|F\|_{\mathcal{B}} = \mathit{deg}(\mathcal{G}[\![F]\!])$ holds. In the proof, we construct a fragment $F_{\mathcal{B}}$ where the number of fragments under a restriction is maximal in the congruence class, i.e., $\|F\|_{\mathcal{B}} = \mathit{max}\{\|G\|_{|} \mid G \equiv F\} = \|F_{\mathcal{B}}\|_{|}$. As corollary we conclude that a process is bounded in breadth if and only if it is bounded in the degree of the hypergraphs.

To show $\|F\|_{\mathcal{B}} = \mathit{deg}(\mathcal{G}[\![F]\!])$, we establish two inequalities. To begin with, we consider $\|F\|_{\mathcal{B}} \leq \mathit{deg}(\mathcal{G}[\![F]\!])$ in Lemma 7.5.2. In the proof, we exploit the following lemma that relates the functions $\|F\|_{|}$ and $\mathit{deg}(\mathcal{G}[\![F]\!])$.

Lemma 7.5.1
For a non-elementary fragment $F \in \mathcal{P}_{\mathcal{F}}$ the inequality $\|F\|_{|} \leq \mathit{deg}(\mathcal{G}[\![F]\!])$ holds.

Proof
We conduct an induction on the structure of fragments.

Base Case Consider $F = \nu a.(F_1^e \mid \ldots \mid F_n^e)$. The graph is

$$\mathcal{G}[\![F]\!] = (\{v_1, \ldots, v_n\}, \{a\}, \{(v_i, F_i^e) \mid 1 \leq i \leq n\}, \{(a, \{v_1, \ldots, v_n\})\}).$$

We have $\mathit{deg}(\mathcal{G}[\![F]\!]) = |\mathit{inc}(a)| = n = \|F\|_{|}$.

Induction Step Assume the inequality $\|F_i\|_{|} \leq \mathit{deg}(\mathcal{G}[\![F_i]\!])$ holds for the non-elementary fragments $F_1, \ldots, F_n$ and consider $F = \nu a.(F_1 \mid \ldots \mid F_n)$. To compute the graph of F, let $\mathcal{G}[\![F_i]\!] = (V_i, E_i, l_i, \mathit{inc}_i)$:

$$\mathcal{G}[\![F]\!] = (V := \textstyle\bigcup_{i=1}^{n} V_i, \bigcup_{i=1}^{n} E_i \cup \{a\}, l := \bigcup_{i=1}^{n} l_i, \bigcup_{i=1}^{n} \mathit{inc}_i \cup \{(a, V_a)\}),$$

where $V_a \subseteq V$ with $v \in V_a$ iff $a \in \mathit{fn}(l(v))$. By definition of fragments, the name a is free in every F_i. This implies every fragment has a process $P_i \in \mathcal{S}(F_i)$ that

knows the name, i.e., $a \in fn(P_i)$. Since there is a vertex v_i in the graph of F_i that is labelled by P_i, we conclude $|inc(a)| = |V_a| \geq n$. We derive the inequality $deg(\mathcal{G}[\![F]\!]) \geq \|F\|_{|}$ as follows:

$$\begin{aligned}
& & & deg(\mathcal{G}[\![F]\!]) \\
& (\text{ Definition of } deg \text{ }) & = & max\{deg(e) \mid e \in \textstyle\bigcup_{i=1}^{n} E_i \cup \{a\}\}. \\
& (\text{ Definition of } max \text{ }) & = & max\{deg(a), max\{deg(e) \mid e \in E_1\}, \\
& & & \qquad \ldots, max\{deg(e) \mid e \in E_n\}\} \\
& (\text{ Definition of } deg(\mathcal{G}[\![F_i]\!]) \text{ }) & = & max\{deg(a), deg(\mathcal{G}[\![F_1]\!]), \ldots, deg(\mathcal{G}[\![F_n]\!])\} \\
& (\text{ } deg(a) = |inc(a)| \geq n \text{ }) & \geq & max\{n, deg(\mathcal{G}[\![F_1]\!]), \ldots, deg(\mathcal{G}[\![F_n]\!])\} \\
& (\text{ Hypothesis }) & \geq & max\{n, \|F_1\|_{|}, \ldots, \|F_n\|_{|}\} \\
& (\text{ Definition of } \|-\|_{|} \text{ }) & = & \|F\|_{|}.
\end{aligned}$$

■

The inequality $\|F\|_{\mathcal{B}} \leq deg(\mathcal{G}[\![F]\!])$ follows directly from Lemma 7.5.1 and the invariance of deg under structural congruence.

Lemma 7.5.2
For non-elementary fragments $F \in \mathcal{P}_{\mathcal{F}}$ the inequality $\|F\|_{\mathcal{B}} \leq deg(\mathcal{G}[\![F]\!])$ holds.

Proof
Let $F \in \mathcal{P}_{\mathcal{F}}$ and let H be the fragment in the congruence class of F where $\|-\|_{|}$ is maximal, i.e., $\|H\|_{|} = max\{\|G\|_{|} \mid G \equiv F\} = \|F\|_{\mathcal{B}}$. With Lemma 7.5.1 and Corollary 7.1.13, we get

$$\|F\|_{\mathcal{B}} = \|H\|_{|} \leq deg(\mathcal{G}[\![H]\!]) = deg(\mathcal{G}[\![F]\!]).$$

■

The inequality $\|F\|_{\mathcal{B}} \geq deg(\mathcal{G}[\![F]\!])$ is established in two steps. Let $sf(F) = \nu\tilde{a}.P^{\neq\nu}$. (1) We observe that the degree of $\mathcal{G}[\![F]\!]$ is the maximal number of processes in $P^{\neq\nu}$ that share a name in $\tilde{a}$. (2) We then construct a fragment $F_{\mathcal{B}} \equiv F$ where the number of fragments under a restriction, $\|F_{\mathcal{B}}\|_{|}$, exceeds this number. Definition 7.5.3 makes the maximal number of processes sharing a name precise.

Definition 7.5.3 ($\|-\|_{I_a} : \mathcal{P}_{sf} \to \mathbb{N}$)
Consider $P^{sf} \in \mathcal{P}_{sf}$ with $P^{sf} = \nu\tilde{a}.(\Pi_{i\in I} P_i)$. For every $a \in \tilde{a}$, the index set $I_a \subseteq I$ contains those processes P_i that have a as a free name, i.e., $i \in I_a$ iff $a \in fn(P_i)$. Since $\tilde{a}$ is finite, the maximum $max\{|I_a| \mid a \in \tilde{a}\} =: \|P^{sf}\|_{I_a}$ exists—the *maximal number of processes that share a restricted name.* ♦

With this definition, we rephrase the statements above. (1) In Lemma 7.5.4, we prove the equality $deg(\mathcal{G}[\![sf(F)]\!]) = \|sf(F)\|_{I_a}$. (2) In Lemma 7.5.5, we construct a fragment $F_{\mathcal{B}}$ from F, where $\|F_{\mathcal{B}}\|_{\mid} \geq \|sf(F)\|_{I_a}$ holds. It turns out to be the fragment in the congruence class of F where $\|-\|_{\mid}$ is maximal, i.e., $\|F\|_{\mathcal{B}} = \|F_{\mathcal{B}}\|_{\mid}$. We illustrate Lemma 7.5.4 and Lemma 7.5.5 in Example 7.5.6.

Lemma 7.5.4
For a process $P^{sf} \in \mathcal{P}_{sf}$ the equality $deg(\mathcal{G}[\![P^{sf}]\!]) = \|P^{sf}\|_{I_a}$ holds.

Proof
Let $P^{sf} = \nu\tilde{a}.(\Pi_{i\in I} P_i)$. We compute the graph interpretation:

$$\mathcal{G}[\![P^{sf}]\!] = (\{v_i \mid i \in I\}, \tilde{a}, \{(v_i, P_i) \mid i \in I\}, \{(a, V_a) \mid a \in \tilde{a}\}),$$

where $v_i \in V_a$ iff $a \in fn(P_i)$. We observe that $|inc(a)| = |V_a| = |I_a|$ for all $a \in \tilde{a}$. The desired equality holds by definition of degree and $\| - \|_{I_a}$. ■

We turn to the construction of fragment $F_{\mathcal{B}}$ from fragment F. The idea is to find the active restriction x in F that is shared by most sequential processes. Then the scope of x is minimised so that it contains all these processes.

Lemma 7.5.5
For every $F \in \mathcal{P}_{\mathcal{F}}$ there is a fragment $F_{\mathcal{B}} \equiv F$ with $\|sf(F)\|_{I_a} \leq \|F_{\mathcal{B}}\|_{\mid}$.

Proof
Let $sf(F) = \nu\tilde{a}.(\Pi_{i\in I} P_i)$ be the standard form of F. Take the name $x \in \tilde{a}$ which is shared by most processes, i.e., $|I_x| = max\{|I_a| \mid a \in \tilde{a}\} = \|sf(F)\|_{I_a}$. The remaining names are $\tilde{a}' := \tilde{a} \setminus \{x\}$. We minimise the scope of x and compute the restricted form to get the fragment $F_{\mathcal{B}}$:

$$\begin{aligned} & F \\ (\text{ Scope extrusion }) \quad \equiv \quad & \nu\tilde{a}'.(\nu x.(\Pi_{i\in I_x} P_i) \mid \Pi_{i\in I\setminus I_x} P_i) \\ (\text{ Lemma 3.2.7 }) \quad \equiv \quad & rf(\nu\tilde{a}'.(\nu x.(\Pi_{i\in I_x} P_i) \mid \Pi_{i\in I\setminus I_x} P_i)) \\ =: \quad & F_{\mathcal{B}}. \end{aligned}$$

As $F \equiv \nu\tilde{a}'.(\nu x.(\Pi_{i\in I_x} P_i) \mid \Pi_{i\in I\setminus I_x} P_i)$ we have $F \equiv_{rf} F_{\mathcal{B}}$ with Proposition 3.2.10. The definition of $\equiv_{rf}$ ensures $F_{\mathcal{B}}$ is a fragment. We compute

$$\|F_{\mathcal{B}}\|_{\mid} = max\{\ldots, \|\nu x.(\Pi_{i\in I_x} P_i)\|_{\mid}, \ldots\} \geq \|\nu x.(\Pi_{i\in I_x} P_i)\|_{\mid} = |I_x| = \|sf(F)\|_{I_a}.$$

■

.

Example 7.5.6 (Construction of $F_{\mathcal{B}}$ and Equality in Lemma 7.5.4)
Consider $\nu a.(\nu b_1.K\lfloor a, b_1\rfloor \mid \nu b_2.L\lfloor a, b_2\rfloor \mid \nu b_3.L\lfloor a, b_3\rfloor)$. The standard form of this fragment is $\nu a, b_1, b_2, b_3.(K\lfloor a, b_1\rfloor \mid L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor)$. Hence, in this example $\tilde{a} = \{a, b_1, b_2, b_3\}$. To compute the index sets, let $K\lfloor a, b_1\rfloor$ have index 1 and $L\lfloor a, b_i\rfloor$ index i with $i = 2, 3$. Since a is in the free names of all processes, $I_a = \{1, 2, 3\}$ holds. The remaining names give $I_{b_i} = \{i\}$. Thus, a is shared by most processes, $|I_a| = 3 = max\{|I_a|, |I_{b_1}|, |I_{b_2}|, |I_{b_3}|\}$. It is the name x in the construction. We shrink the scope of a and get $\nu b_1, b_2, b_3.\nu a(K\lfloor a, b_1\rfloor \mid L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor)$. Computing the restricted form does not change the process and we return the fragment $F_{\mathcal{B}} = \nu b_1, b_2, b_3.\nu a(K\lfloor a, b_1\rfloor \mid L\lfloor a, b_2\rfloor \mid L\lfloor a, b_3\rfloor)$. We check the inequality:

$$\|F_{\mathcal{B}}\|_{\mid} = 3 \geq max\{|I_a|, |I_{b_1}|, |I_{b_2}|, |I_{b_3}|\} = \|sf(F)\|_{I_a}.$$

To illustrate Lemma 7.5.4, we observe that $F_{\mathcal{B}}$ is in standard form. Its graph is depicted in Figure 7.7 and 7.8. The equality $\|F_{\mathcal{B}}\|_{I_a} = max\{|I_a|, |I_{b_1}|, |I_{b_2}|, |I_{b_3}|\} = 3 = deg(\mathcal{G}[\![F_{\mathcal{B}}]\!])$ holds. ♦

With Lemma 7.5.4 and Lemma 7.5.5, we prove the inequality $\|F\|_{\mathcal{B}} \geq deg(\mathcal{G}[\![F]\!])$, which is yet missing.

Lemma 7.5.7
Consider the non-elementary fragment $F \in \mathcal{P}_{\mathcal{F}}$. Then $\|F\|_{\mathcal{B}} \geq deg(\mathcal{G}[\![F]\!])$.

Proof
With Lemma 7.5.5, we compute $F_{\mathcal{B}} \equiv F$. By definition of breadth, we have $\|F\|_{\mathcal{B}} = max\{\|G\|_{\mid} \mid G \equiv F\} \geq \|F_{\mathcal{B}}\|_{\mid}$. With Lemma 7.5.5 and Lemma 7.5.4 we get

$$\|F\|_{\mathcal{B}} \geq \|F_{\mathcal{B}}\|_{\mid} \geq \|sf(F)\|_{I_a} = deg(\mathcal{G}[\![sf(F)]\!]) = deg(\mathcal{G}[\![F]\!]).$$

The last equation holds with $\mathcal{G}[\![F]\!] = \mathcal{G}[\![sf(F)]\!]$ according to Lemma 7.1.11. ■

Lemma 7.5.2 and Lemma 7.5.7 prove that the breadth of a fragment equals the degree of its graph.

Theorem 7.5.8 (Characterisation of Breadth)
Let $F \in \mathcal{P}_{\mathcal{F}}$ be non-elementary and $F_{\mathcal{B}} \equiv F$ the fragment constructed in Lemma 7.5.5. Then $\|F\|_{\mathcal{B}} = \|F_{\mathcal{B}}\|_{\mid} = deg(\mathcal{G}[\![F]\!])$.

Proof
The proof of Lemma 7.5.7 yields $\|F\|_{\mathcal{B}} \geq \|F_{\mathcal{B}}\|_{\mid} \geq deg(\mathcal{G}[\![F]\!])$. With Lemma 7.5.2, $\|F\|_{\mathcal{B}} \leq deg(\mathcal{G}[\![F]\!])$. We conclude equality. ■

For an elementary fragment $F^e = K\lfloor\tilde{a}\rfloor$ or $F^e = M^{\neq \mathbf{0}}$, the breadth is one while the graph degree is zero. Although different, both values are bounded. We say that a process is *bounded in the degree*, if there is $k_{deg} \in \mathbb{N}$ that bounds the degree of the graphs of all reachable fragments, i.e.,

$$\exists k_{deg} \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf(Q)) : deg(\mathcal{G}[\![F]\!]) \leq k_{deg}.$$

If we define the *set of all processes that are bounded in the degree* to be $\mathcal{P}_{deg<\infty}$, the following corollary of Theorem 7.5.8 holds.

Corollary 7.5.9 (Characterisation of Boundedness in Breadth)
$\mathcal{P}_{\mathcal{B}<\infty} = \mathcal{P}_{deg<\infty}$.

In the following section, we check well-known examples for boundedness in depth and boundedness in breadth with the help of Theorem 7.4.4 and Theorem 7.5.8.

7.6 Applications

We take up the two basic counterexamles for structural stationarity in the beginning of the chapter (cf. Figure 7.4 and 7.5) and extend them to full models of data structures that are known from the literature. Both processes fail to be finitely representable using the structural semantics in Chapter 3. The results in this chapter explain why.

Example 7.6.1 (Lists)
We consider lists [Mil99, SW01] where only an append operation is available. If we have a list item, LI, the append operation a receives a value y and forwards it to the neighbouring list element, $\overline{an}\langle y\rangle$. If we have a list end, LE, a call to the append operation creates a new list end with y as parameter, $LE\lfloor an, y\rfloor$. The former list end becomes a list item, $LI\lfloor a, an, x\rfloor$:

$$\begin{aligned} LI(a, an, x) &:= a(y).\overline{an}\langle y\rangle.LI\lfloor a, an, x\rfloor \\ LE(a, x) &:= a(y).\nu an.(LI\lfloor a, an, x\rfloor \mid LE\lfloor an, y\rfloor). \end{aligned}$$

Consider system $S_1 := \nu a.(FILL\lfloor a\rfloor \mid \nu c.LE\lfloor a, c\rfloor)$, where process $FILL(a) := \nu c.\overline{a}\langle c\rangle.FILL\lfloor a\rfloor$ generates fresh values c which it appends to the list by sending $\overline{a}\langle c\rangle$. After $FILL$ sent $n-1$ fresh names, we have the anchored fragment

$$F^{\mathcal{A}} = \nu a.(FILL\lfloor a\rfloor \mid \nu a_2.(\nu c.LI\lfloor a, a_2, c\rfloor \mid \nu a_3.(\ldots(\nu c_n.LE\lfloor a_n, c_n\rfloor)\ldots))).$$

We argue that S_1 is not bounded in depth but bounded in breadth. With Theorem 7.2.8 it is not structurally stationary.

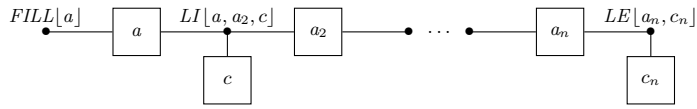


Figure 7.10:
The graph of fragment $F^{\mathcal{A}}$ in Example 7.6.1. It represents a list.

The graph representation of fragment $F^{\mathcal{A}}$ is depicted in Figure 7.10. The length of the simple paths grows in the number of sent names. By Theorem 7.4.4, the process is not bounded in depth, $S_1 \notin \mathcal{P}_{\mathcal{D}<\infty}$. The degree of every graph is two. By Theorem 7.5.8, the breadth of the reachable fragments is two. Hence, $S_1 \in \mathcal{P}_{\mathcal{B}<\infty}$ holds. In the graph interpretation, unboundedness in the length of the simple paths is evident. The actual depth of the reachable fragments is not obvious. It is logarithmic in the number of restrictions. ♦

Example 7.6.2 (Bags)
A bag is a data structure that stores arbitrarily many values without ordering them [Fok07]. We model a bag by the process $BAG\lfloor in, out\rfloor$ which receives a value y on channel in. Then the bag is ready to accept new values, $BAG\lfloor in, out\rfloor$, and has the value y available on channel out:

$$BAG(in, out) := in(y).(\overline{out}\langle y\rangle \mid BAG\lfloor in, out\rfloor).$$

Consider $S_2 := \nu in, out.(FILL\lfloor in\rfloor \mid BAG\lfloor in, out\rfloor)$, where $FILL$ is taken from Example 7.6.1. After n communications between $FILL$ and BAG, we have

$$F_{\mathcal{B}} = \nu in.(\nu c_1, \ldots, c_n.\nu out.(\overline{out}\langle c_1\rangle \mid \ldots \mid \overline{out}\langle c_n\rangle \mid BAG\lfloor in, out\rfloor) \mid FILL\lfloor in\rfloor).$$

As the name indicates, fragment $F_{\mathcal{B}}$ is constructed according to Lemma 7.5.5.

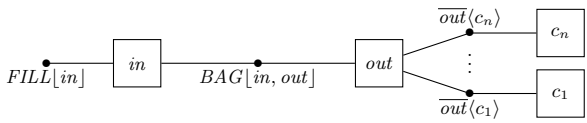


Figure 7.11:
The graph of fragment $F_{\mathcal{B}}$ in Example 7.6.2. It represents a bag.

The graph interpretation is depicted in Figure 7.11. By Theorem 7.5.8, $\|F_{\mathcal{B}}\|_{\mathcal{B}} = deg(\mathcal{G}[\![F_{\mathcal{B}}]\!]) = n+1$. As the degree of the graphs grows unboundedly, the process is not bounded in breadth, $S_2 \notin \mathcal{P}_{\mathcal{B}<\infty}$. With Theorem 4.3.2, the system is not

structurally stationary. The length of the longest simple path, $lsp(\mathcal{G}[\![F_{\mathcal{B}}]\!])$, is three. It is bounded by three in all reachable fragments. By Theorem 7.4.4, S_2 is bounded in depth, i.e., in $\mathcal{P}_{\mathcal{D}<\infty}$. If we remove the restriction νout from S_2, we get $S_2' := \nu in.(FILL\lfloor in \rfloor \mid BAG\lfloor in, out \rfloor)$. This process is bounded in breadth and bounded in depth and hence structurally stationary with Theorem 7.2.8. Note that the number of active restrictions and the number of sequential processes are not bounded during system execution. The graph interpretation coincides with Figure 7.11 but lacks the hyperedge *out*. ⧫

7.7 Related Work and Conclusion

We established a second complete characterisation of structural stationarity. A process is structurally stationary if and only if it is bounded in depth and bounded in depth. The novel characteristic functions *depth* and *breadth* indicate the quality of the connections that are induced by restricted names. As the functions are hard to grasp, we formally defined the graph interpretation of π-Calculus processes and worked out graph-theoretic characterisations. Boundedness in depth is equivalent to boundedness in the simple paths, the function breadth equals the graph degree. To establish the characterisation of depth, we defined a new normal form called *anchored fragments*. In anchored fragments, the nesting of restrictions corresponds to the length of a simple path in the graphs. This relationship allowed us to prove that anchored fragments are particularly flat in a well-defined sense. It would be interesting to see whether this flatness can be established without the graph interpretation. We do not know a different proof.

For a user of the tool Petruchio [SM08], it would be beneficial to know in advance whether the computation of the structural semantics terminates for the process of interest. Unfortunately, the problem is undecidable according to Lemma 4.1.3. We plan to include approximative algorithms to answer the question in our tool. In Section 7.2, we showed how the semantics of subprocesses may reveal unboundedness in breadth. Ideally, the approximation would return the source of infinity: unbounded breadth $\mathcal{B}$ or unbounded depth $\mathcal{D}$. Depending on the source, finite Petri nets $\mathcal{N}_{\mathcal{B}}[\![P]\!]$ and $\mathcal{N}_{\mathcal{D}}[\![P]\!]$ could be computed that approximate the state space of the infinite Petri net $\mathcal{N}[\![P]\!]$. The additional information will help designing precise approximations, which preserve intricate properties of processes.

The idea of understanding the term structure of a process, i.e., the syntax, as a graph was proposed by Milner in his work on flow graphs [MM79, Mil79] (cf. Section 2.1). For the π-Calculus it has been recalled in [MPW92, Mil99, SW01]. We related the functions depth and breadth on processes P to functions on their graphs $\mathcal{G}[\![P]\!]$. We are not aware of similar results in the literature.

As discussed in Section 3.6, Engelfriet and Gelsema proposed normal forms that are related to the restricted form [EG99, EG04b, EG07]. The anchored fragments we presented in this chapter are more stringent than any of the known normal forms, including Engelfriet's and Gelsema's normal forms, Milner's standard form, and the restricted form. Thus, they reveal more information about the connection structure of process terms.

8 Decidability in Bounded Depth and Undecidability in Bounded Breadth

Contents

In the previous chapter, we showed that boundedness in depth and breadth is equivalent to structural stationarity. Hence, important classes of processes—where interesting verification problems are decidable—are bounded in depth and breadth (cf. to the hierarchy of processes in Figure 4.4). Thus, boundedness in depth and breadth seems fundamental for decidability. From a theoretical point of view, it is interesting to investigate whether boundedness in only one of the functions, depth or breadth, is sufficient to achieve decidability results. We argue that this research also has a practical motivation.

Two decidable classes of processes known from the literature fail to be structurally stationary but allow for modelling interesting features of reconfigurable systems. Busi and Gorrieri investigated systems where the number of restricted names during runtime is bounded [BG95, BG09]. They allow, e.g. for modelling the elementary bag data structure in Figure 7.4. Amadio and Meyssonnier proposed bounded input and unique receiver systems [AM02], which turned out

useful for modelling Internet applications [Ama00]. Although not structurally stationary, both classes of processes are *bounded in depth*.

The main contribution in this chapter is a decision procedure for termination and infinity of states for processes of bounded depth. Taking the expressiveness arguments above into account, we claim that $\mathcal{P}_{\mathcal{D}<\infty}$ is the up-to-now most expressive and yet decidable subclass of π-Calculus. It subsumes all other decidable subclasses in the literature. As argued in the introduction to this thesis, termination is a fundamental problem in computation. In computer-aided verification, it is the basis for the automata-theoretic verification of liveness properties [Var91]—and thus of particular importance. The decision procedure for infinity of states can be used as an approximation for structural stationarity in our tool Petruchio. If it reports a finite state space, the process of interest is structurally stationary and the structural semantics a finite and even bounded net. If the procedure decides infinity of states, the structural semantics is either an unbounded or an infinite net. In the latter case, the compilation does not terminate.

For processes of bounded breadth, we recall a well-known encoding of counter machines. It shows that the class $\mathcal{P}_{\mathcal{B}<\infty}$ is *Turing complete*. With the characterisation of structural stationarity in Chapter 7, this Turing completeness has interesting consequences. We prove undecidability of structural stationarity for processes of bounded breadth and derive undecidability of boundedness in depth as an immediate corollary. We also settle undecidability of boundedness in breadth.

The decidability results for $\mathcal{P}_{\mathcal{D}<\infty}$ are obtained by viewing this class as an instance of *well-structured transition systems (WSTS)* [Fin90, AČJT00, FS01]. WSTS are a framework for infinite state systems that generalises decidability results for particular models. Technically, a WSTS is a transition system with an ordering relation on the states, which is compatible with the transition relation. Depending on the ordering, the compatibility, and decidability properties the framework yields decision procedures e.g. for termination [Fin90, FS01] or simulation [AČJT00].

The technical contribution in the instantiation of the WSTS framework is a new ordering $\preceq_{\mathcal{P}}$ on processes, which we show to be a *well-quasi-ordering (wqo)* (i.e., in every infinite sequence of processes two comparable ones can be found) for processes of bounded depth. In the proof, the anchored fragments in Section 7.3 again play a vital role. Since the ordering $\preceq_{\mathcal{P}}$ is a simulation relation it is compatible with the reaction relation of the π-Calculus in a strong sense.

To sum up our contributions:

- We define a new ordering on processes, which we prove to be a wqo on processes of bounded depth.
- With the new ordering, processes of bounded depth have well-structured transition systems. As a consequence, termination and infinity of states are decidable for $\mathcal{P}_{\mathcal{D}<\infty}$.

- We recall a well-known process model of counter machines. It proves Turing completeness of $\mathcal{P}_{\mathcal{B}<\infty}$. Moreover, the encoding reveals undecidability of structural stationarity, boundedness in depth, and boundedness in breadth.

The chapter is organised as follows. In Section 8.1, we review the basics on wqos. Section 8.2 is devoted to the instantiation of the WSTS framework for $\mathcal{P}_{\mathcal{D}<\infty}$, Section 8.3 proves Turing completeness in bounded breadth and gives several undecidability results. We conclude with Section 8.4.

8.1 Well-Quasi-Orderings and the Rooted Tree Embedding

We recall *well-quasi-orderings (wqo)* and present the rooted tree embedding, a particular wqo that we will exploit to define a wqo on fragments in the next section. A brief introduction to wqos and the main results on wqos on graphs can be found in Diestel's book on graph theory [Die06].

A *quasi-ordering (qo)* on a set of elements A is a reflexive and transitive relation $\preceq \subseteq A \times A$. We also call $(A, \preceq)$ a qo. If we have a qo $(A, \preceq)$ and a subset $B \subseteq A$, then we also refer to $(B, \preceq)$ as a qo, without explicitly restricting $\preceq \subseteq A \times A$ to $B \times B$. A qo $(A, \preceq)$ is a *well-quasi-ordering (wqo)*, if in every infinite sequence $(a_i)_{i \in \mathbb{N}}$ there are two comparable elements, i.e., there are indices $i < j$ with $a_i \preceq a_j$. To give an example, the natural numbers $(\mathbb{N}, \leq)$ are a wqo.

A result by Higman [Hig52] lifts a wqo $\preceq$ on a set of elements A to a wqo $\preceq^*$ on the set of *finite sequences or words* A^*.

Lemma 8.1.1 (and Definition [Hig52])
If $(A, \preceq)$ is a wqo, then $(A^*, \preceq^*)$ is a wqo. The ordering $u \preceq^* v$ demands u to be a subsequence of v, which is dominated elementwise. To define $\preceq^* \subseteq A^* \times A^*$ formally, let $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_n)$. We have $u \preceq^* v$ if there are indices $1 \leq i_1 < \ldots < i_m \leq n$ so that $u_k \preceq v_{i_k}$ for all $1 \leq k \leq m$.

Example 8.1.2 (Higman's Result)
Consider the finite set $A = \{a, b\}$, which is a wqo with the identity, i.e., we have $(\{a, b\}, id)$ as wqo. In a sequence starting with the word (a, a), we may choose for example

$$(a, a), (a, b), (b, b), (b, a), a, b, \epsilon, \ldots$$

as first elements (with ϵ denoting the *empty word*). In the following step, we have to take an element that is larger by id^*. For example (a, b, b), which covers (a, b), (b, b), a, b, and ϵ, but not (b, a). ♦

In Section 8.2, we define a qo on fragments. To prove it is a wqo, we relate it with a wqo on trees. Since fragments are process terms, it is convenient to use a term-based representation of trees.

Definition 8.1.3 (Trees over A)
Given a set A, the *trees over* A are defined inductively by

$$T \quad ::= \quad a \mid (a, (T_1, \ldots, T_n)),$$

where $a \in A$. The *set of all trees over* A is $\mathcal{T}(A)$. ⧫

We draw tree a as a single vertex labelled by a that has an incoming arc to indicate it is the root of the tree. For $T = (a, (T_1, \ldots, T_n))$ we draw as before a labelled vertex with an incoming arc. For every tree T_i, we add an arc labelled by i to the root of T_i. The arc labelling reflects the fact that the trees are ordered, i.e., $(a, (T_1, T_2)) \neq (a, (T_2, T_1))$. Figure 8.1 illustrates the graphical representation.

The *height* of a tree is measured like the nesting of restrictions in fragments. For the tree T in Figure 8.1, we have

$$\mathit{height}(T) = 1 + \mathit{max}\{\mathit{height}(T_1), \mathit{height}(T_2)\} = 1 + \mathit{max}\{1, 0\} = 2.$$

Definition 8.1.4 ($\mathit{height} : \mathcal{T}(A) \to \mathbb{N}$)
Consider the trees $\mathcal{T}(A)$ over the set A. The *height* of $T \in \mathcal{T}(A)$ is defined by $\mathit{height}(a) := 0$ and $\mathit{height}((a, (T_1, \ldots, T_n))) := 1 + \mathit{max}\{\mathit{height}(T_i) \mid 1 \leq i \leq n\}$. For $n \in \mathbb{N}$, we denote by $\mathcal{T}(A)_n$ the trees of height less or equal to n. ⧫

A qo $\preceq$ on the set of elements A gives rise to the *rooted tree embedding* $\preceq_{\mathcal{T}}$ as qo on the trees over A, $\mathcal{T}(A)$. Intuitively, $T_1 \preceq_{\mathcal{T}} T_2$ if T_1 is a subtree of T_2 so that the levels of T_1 are preserved in T_2. In particular, the root of T_1 is mapped to the root of T_2 and the leaves in T_1 are leaves in T_2. Figure 8.1 gives an example.

Definition 8.1.5 (Rooted Tree Embedding)
Consider a qo $(A, \preceq)$. The *rooted tree embedding* $\preceq_{\mathcal{T}} \subseteq \mathcal{T}(A) \times \mathcal{T}(A)$ contains all pairs that can be derived with the following rules:

$$\text{(Elem)} \quad \frac{a \preceq a'}{a \preceq_{\mathcal{T}} a'} \qquad \text{(Comp)} \quad \frac{a \preceq a' \quad \text{and} \quad (T_1, \ldots, T_m) \preceq^*_{\mathcal{T}} (T'_1, \ldots, T'_n)}{(a, (T_1, \ldots, T_m)) \preceq_{\mathcal{T}} (a', (T'_1, \ldots, T'_n))}.$$

⧫

To give an intuition to the definition, imagine the leaves in the tree are sequential processes composed in parallel and the remaining nodes are restricted names.

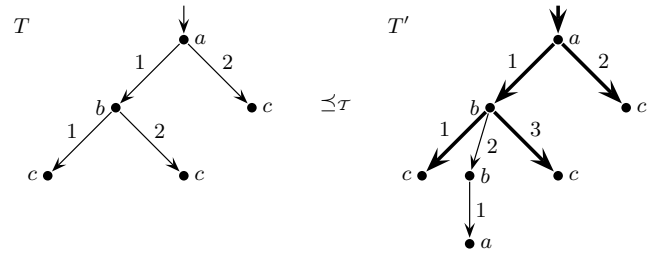


Figure 8.1:
Let $A = \{a, b, c\}$ be ordered by the identity $\preceq = id$. Consider $T = (a, (T_1, T_2))$ with $T_1 = (b, (c, c))$ and $T_2 = c$ as well as $T' = (a, (T_1', T_2'))$ with $T_1' = (b, (c, T_3', c))$, $T_2' = c$, and $T_3' = (b, (a))$. Clearly, $T_1 \preceq_{\mathcal{T}} T_1'$ and $T_2 \preceq_{\mathcal{T}} T_2'$ and so we derive the rooted tree embedding $T \preceq_{\mathcal{T}} T'$, which is depicted by bold edges.

To ensure the ordering is a simulation relation, we may have further processes below a restriction, e.g. $T_1 = (b, (c, c)) \preceq_{\mathcal{T}} (b, (c, T_3', c)) = T_1'$ in Figure 8.1. But a restricted name will not simulate a process, hence $a \not\preceq_{\mathcal{T}} (a, (b))$.

The rooted tree embedding is in fact a qo. While we establish this for all trees over A, wqo only holds for trees of bounded height.

Lemma 8.1.6
If $(A, \preceq)$ is a qo, then $(\mathcal{T}(A), \preceq_{\mathcal{T}})$ is a qo.

Proof
The proof of reflexivity is trivial, we only consider transitivity here. We have to prove that for all T, T', T'' the following holds. If $T \preceq_{\mathcal{T}} T'$ and $T' \preceq_{\mathcal{T}} T''$ then $T \preceq_{\mathcal{T}} T''$. We use induction on the structure of T and show that for all T', T'' we have: $T \preceq_{\mathcal{T}} T'$ and $T' \preceq_{\mathcal{T}} T''$ implies $T \preceq_{\mathcal{T}} T''$.

Base Case Consider $T = a$. Since we assume $T \preceq_{\mathcal{T}} T'$, we conclude $T' = b$ with $a \preceq b$ because only Rule (Elem) gives an ordering for leaves. Similarly, as we assume $T' \preceq_{\mathcal{T}} T''$ and as $T' = b$, we conclude $T'' = c$ with $b \preceq c$. Transitivity of $\preceq$ yields $a \preceq c$. Thus, $a \preceq_{\mathcal{T}} c$ which means $T \preceq_{\mathcal{T}} T''$.

Induction Step Assume the proposition holds for $T_1, \ldots, T_m$ and consider the tree $T = (a, (T_1, \ldots, T_m))$ with $T \preceq_{\mathcal{T}} T' \preceq_{\mathcal{T}} T''$. As only Rule (Comp) gives an ordering on composed trees, we get $T' = (b, (T_1', \ldots, T_n'))$ with $a \preceq b$ and $(T_1, \ldots, T_m) \preceq_{\mathcal{T}}^* (T_1', \ldots, T_n')$. Similarly, from the form of T' it follows that $T'' = (c, (T_1'', \ldots, T_o''))$ with $b \preceq c$ and $(T_1', \ldots, T_n') \preceq_{\mathcal{T}}^* (T_1'', \ldots, T_o'')$.

Transitivity of $\preceq$ yields $a \preceq c$. The ordering $(T_1, \ldots, T_m) \preceq^*_{\mathcal{T}} (T'_1, \ldots, T'_n)$ means there are $1 \leq i_1 < \ldots < i_m \leq n$ with $T_k \preceq_{\mathcal{T}} T'_{i_k}$. Similarly, there are indices $1 \leq j_1 < \ldots < j_n \leq o$ with $T'_l \preceq_{\mathcal{T}} T''_{j_l}$. Thus $T_k \preceq_{\mathcal{T}} T'_{i_k}$ and $T'_{i_k} \preceq_{\mathcal{T}} T''_{j_{i_k}}$ holds. The hypothesis gives $T_k \preceq_{\mathcal{T}} T''_{j_{i_k}}$. We conclude the ordering $(T_1, \ldots, T_m) \preceq^*_{\mathcal{T}} (T''_1, \ldots, T''_o)$ and get $T \preceq_{\mathcal{T}} T''$ with Rule (Comp). ■

We are not aware that the rooted tree embedding has been used elsewhere. Hence, we provide a proof that it is a wqo on trees of bounded height.

Proposition 8.1.7 (The Rooted Tree Embedding is a WQO)
If $(A, \preceq)$ is a wqo then $(\mathcal{T}(A)_n, \preceq_{\mathcal{T}})$ is a wqo for all $n \in \mathbb{N}$.

The proposition is established by an induction on n, where Higman's result is applied in the induction step. For a clean proof, we require some basics on wqos. To keep the presentation self-contained, we provide the proofs of all lemmas.

Consider an infinite sequence $(a_i)_{i \in \mathbb{N}}$ in the set A. An infinite *subsequence* $(a_{f(i)})_{i \in \mathbb{N}}$ is defined by a strictly monotonic function $f : \mathbb{N} \to \mathbb{N}$. A function is strictly monotonic if $i < j$ implies $f(i) < f(j)$.

Lemma 8.1.8
If $(A, \preceq)$ is a wqo then every infinite sequence $(a_i)_{i \in \mathbb{N}}$ contains an infinite increasing subsequence $(a_{f(i)})_{i \in \mathbb{N}}$, i.e., $a_{f(i)} \preceq a_{f(i+1)}$ for all i.

Proof
Consider an infinite sequence $(a_i)_{i \in \mathbb{N}}$. Take the subsequence of elements $a_{nd(i)}$ that are not dominated by subsequent elements, i.e., there is no j with $nd(i) < j$ so that $a_{nd(i)} \preceq a_j$. This sequence is finite due to the wqo assumption (an infinite sequence $(a_{nd(i)})_{i \in \mathbb{N}}$ would contain comparable elements). The lemma holds. ■

We apply the lemma to show that wqos are closed under Cartesian products.

Lemma 8.1.9
Let $(A, \preceq)$ and $(B, \sqsubseteq)$ be two wqos. Then $(A \times B, \preceq\!\!\sqsubseteq)$ is a wqo, where $(a, b) \preceq\!\!\sqsubseteq (a', b')$ iff $a \preceq a'$ and $b \sqsubseteq b'$.

Proof
Let $((a_i, b_i))_{i \in \mathbb{N}}$ be an infinite sequence in $A \times B$. Since $(a_i)_{i \in \mathbb{N}}$ is an infinite sequence in A and $(A, \preceq)$ is a wqo, there is an infinite increasing subsequence $(a_{f(i)})_{i \in \mathbb{N}}$ of $(a_i)_{i \in \mathbb{N}}$ by Lemma 8.1.8. Consider the subsequence $((a_{f(i)}, b_{f(i)}))_{i \in \mathbb{N}}$ of $((a_i, b_i))_{i \in \mathbb{N}}$. As $(b_{f(i)})_{i \in \mathbb{N}}$ is infinite and $(B, \sqsubseteq)$ is a wqo, there are $i < j$ so that $b_{f(i)} \sqsubseteq b_{f(j)}$. With strict monotonicity, we found indices $f(i) < f(j)$ with $a_{f(i)} \preceq a_{f(j)}$ and $b_{f(i)} \sqsubseteq b_{f(j)}$. Thus, $(a_{f(i)}, b_{f(i)}) \preceq\!\!\sqsubseteq (a_{f(j)}, b_{f(j)})$ holds. ■

It is interesting to note that an induction on the previous lemma proves Dickson's result, which is well-known in Petri net theory. It states that for every $k \in \mathbb{N}$ the set $(\mathbb{N}^k, \leq^k)$ is wqo, where $v \leq^k w$ holds for two vectors $v = (v_1, \ldots, v_k)$ and $w = (w_1, \ldots, w_k)$, if $v_i \leq w_i$ for all i. By definition, wqos are closed under taking subsets.

Lemma 8.1.10
If $(A, \preceq)$ be a wqo and $B \subseteq A$, then $(B, \preceq)$ is a wqo.

If we have a qo $(X, \preceq)$ whose subsets $(A, \preceq)$ and $(B, \preceq)$ are wqos, then the union $(A \cup B, \preceq)$ is a wqo. Wqos are not closed under arbitrary union. More precisely, if $(A, \preceq)$ and $(B, \sqsubseteq)$ are wqos then $(A \cup B, \preceq \cup \sqsubseteq)$ is a wqo only if A and B are disjoint. Otherwise, transitivity (and hence even qo) may fail.

Lemma 8.1.11
Let $(X, \preceq)$ be a qo with $A, B \subseteq X$. If $(A, \preceq)$ and $(B, \preceq)$ are wqos then $(A \cup B, \preceq)$ is a wqo.

Proof
Let $(c_i)_{i \in \mathbb{N}} \downarrow A$ be the projection of the infinite sequence $(c_i)_{i \in \mathbb{N}}$ in $A \cup B$ onto the elements in A and similar for $(c_i)_{i \in \mathbb{N}} \downarrow B$. If both sequences were finite, $(c_i)_{i \in \mathbb{N}}$ would be finite as it consists of elements in A and B only. A contradiction. Thus, at least one of the sequences, say $(c_i)_{i \in \mathbb{N}} \downarrow A$, has to be infinite. Since $(c_i)_{i \in \mathbb{N}} \downarrow A = (c_{f(i)})_{i \in \mathbb{N}}$ is a sequence in A and $(A, \preceq)$ is a wqo, there are $i < j$ with $c_{f(i)} \preceq c_{f(j)}$. As f is strictly monotonic, we have indices $f(i) < f(j)$ with $c_{f(i)} \preceq c_{f(j)}$. ■

If we already established a set $(A, \preceq)$ to be wqo, then of course any larger ordering $\sqsubseteq$ that includes $\preceq$ will also yield a wqo.

Lemma 8.1.12
If $(A, \preceq)$ is a wqo and $\preceq \subseteq \sqsubseteq$ (set inclusion) then $(A, \sqsubseteq)$ is a wqo.

Before we turn to the proof of Proposition 8.1.7, we state a set-theoretic observation. The trees of height at most $n+1$ in $\mathcal{T}(A)$ can be viewed as the Cartesian product of A with the trees of height at most n.

Lemma 8.1.13 (and Definition)
Let $\mathcal{T}(A)_n^+ := \mathcal{T}(A)_n^* \setminus \{\varepsilon\}$ denote the non-empty sequences of trees of height at most n. With this definition, the following equality holds:

$$A \times \mathcal{T}(A)_n^+ = \mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0.$$

Proof
The inclusion from left to right is clear. To show the reverse direction, consider $T \in \mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0$. This means $T = (a, (T_1, \ldots, T_m))$ with $\mathit{height}(T_i) \leq n$ for all i. We delay the proof of this fact for a moment. Thus, all T_i are in $\mathcal{T}(A)_n$ and so $(T_1, \ldots, T_m)$ is in $\mathcal{T}(A)_n^+$. We conclude $T = (a, (T_1, \ldots, T_m)) \in A \times \mathcal{T}(A)_n^+$.

To see that $T = (a, (T_1, \ldots, T_m))$ with $\mathit{height}(T_i) \leq n$ has to hold, assume $\mathit{height}(T_i) > n$ for some T_i. Then

$$\mathit{height}(T) = 1 + \mathit{max}\{\mathit{height}(T_i) \mid 1 \leq i \leq m\} > 1 + n$$

and $T \notin \mathcal{T}(A)_{n+1}$, a contradiction. If $T = a$, the height would be zero, which means $T \in \mathcal{T}(A)_0$ and thus $T \notin \mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0$. Again a contradiction. ■

Proof (of Proposition 8.1.7)
We do an induction on n.

Base Case Consider $\mathcal{T}(A)_0$. Since $\mathcal{T}(A)_0 = A$, a sequence $(T_i)_{i \in \mathbb{N}}$ in $\mathcal{T}(A)_0$ is a sequence in A. Since $(A, \preceq)$ is a wqo, there are indices $i < j$ with $T_i = a_i \preceq a_j = T_j$. With Rule (Elem) we conclude $T_i \preceq_{\mathcal{T}} T_j$.

Induction Step Assume $(\mathcal{T}(A)_n, \preceq_{\mathcal{T}})$ is a wqo for some $n \in \mathbb{N}$. We prove that $(\mathcal{T}(A)_{n+1}, \preceq_{\mathcal{T}})$ is a wqo. With Higman's result, $(\mathcal{T}(A)_n^*, \preceq_{\mathcal{T}}^*)$ is a wqo. Since $\mathcal{T}(A)_n^+ \subseteq \mathcal{T}(A)_n^*$, we conclude that $(\mathcal{T}(A)_n^+, \preceq_{\mathcal{T}}^*)$ is a wqo with Lemma 8.1.10. With Lemma 8.1.9, the product $(A \times \mathcal{T}(A)_n^+, \preceq \preceq_{\mathcal{T}}^*)$ is a wqo. Lemma 8.1.13 yields the equality $A \times \mathcal{T}(A)_n^+ = \mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0$. Thus, $(\mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0, \preceq \preceq_{\mathcal{T}}^*)$ is a wqo. Below we prove that $\preceq \preceq_{\mathcal{T}}^* \subseteq \preceq_{\mathcal{T}}$. Lemma 8.1.12 allows us to take the larger ordering, so $(\mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0, \preceq_{\mathcal{T}})$ is a wqo. Since $(\mathcal{T}(A)_0, \preceq_{\mathcal{T}})$ is a wqo, the union $((\mathcal{T}(A)_{n+1} \setminus \mathcal{T}(A)_0) \cup \mathcal{T}(A)_0, \preceq_{\mathcal{T}})$ is a wqo by Lemma 8.1.11. It is the set $(\mathcal{T}(A)_{n+1}, \preceq_{\mathcal{T}})$ and so the statement holds.

To see that $\preceq \preceq_{\mathcal{T}}^* \subseteq \preceq_{\mathcal{T}}$, consider $(a, (T_1, \ldots, T_i)) \preceq \preceq_{\mathcal{T}}^* (a', (T_1', \ldots, T_j'))$. This means, $a \preceq a'$ and $(T_1, \ldots, T_i) \preceq_{\mathcal{T}}^* (T_1', \ldots, T_j')$. With Rule (Comp), we conclude $(a, (T_1, \ldots, T_i)) \preceq_{\mathcal{T}} (a', (T_1', \ldots, T_j'))$. ■

The rooted tree embedding is no wqo on the set of all trees over A. Consider the sequence of trees where the height grows in every step, but no new branches are created.

Example 8.1.14 (Counterexample for Wqo on $\mathcal{T}(A)$)
Let $T_1 = a$, $T_2 = (a, (a))$, $T_3 = (a, (a, (a)))$ etc. All these trees are incomparable, i.e., for all $i < j$ we have $T_i \not\preceq_{\mathcal{T}} T_j$. Hence, $(\mathcal{T}(A), \preceq_{\mathcal{T}})$ is not a wqo. ♦

8.2 Well-Structure and Decidability in Bounded Depth

Our main result states that processes of bounded depth have *well-structured transition systems (WSTS)* [Fin90, AČJT00, FS01]. A WSTS is an image-finite *transition system* $(S, \rightsquigarrow)$ with a *wqo* $\preceq$ on the states, which is required to be a *simulation*. By definition, the relation $s \preceq t$ is a simulation if state t imitates the transition behaviour of s.

Definition 8.2.1 (Well-Structured Transition System)
A *well-structured transition system* is a triple $(S, \rightsquigarrow, \preceq)$ where S is a set of states, $\rightsquigarrow \subseteq S \times S$ is a transition relation, and $\preceq \subseteq S \times S$ is a wqo and a simulation. This means the following implication holds for all $s \preceq t$: if $s \rightsquigarrow s'$ then there is $t' \in S$ with $t \rightsquigarrow t'$ and $s' \preceq t'$. The WSTS *has a non-terminating computation from* $s_0 \in S$ if an infinite sequence $s_0 \rightsquigarrow s_1 \rightsquigarrow \ldots$ exists. The *set of states reachable from* s_0 is $Reach(s_0) := \{s \in S \mid s_0 \rightsquigarrow^* s\}$. ♦

It is well-known that Petri nets have well-structured transition systems with Dickson's ordering (cf. Section 8.1). A comprehensive overview of models with WSTS can be found in [FS01].

Consider the WSTS $(S, \rightsquigarrow, \preceq)$. If $\rightsquigarrow$ is effectively computable and $\preceq$ is decidable then the following algorithm decides termination and infinity of states [Fin90, FS01]. For $s_0 \in S$, we construct the *finite reachability tree* $FRT(s_0)$. The root is labelled by s_0. For every vertex labelled by s in the tree, we create a new vertex for every successor t of s. We connect the vertex labelled by s and the new vertex. If there is a vertex labelled by s' on the path from the root to the new vertex with $s' \preceq t$, we label the new vertex by t_+. Otherwise we label it by t. We do not create successors for vertices t_+. The idea is that t with $s' \preceq t$ can simulate the behaviour of s' and thus repeat $s' \rightsquigarrow \ldots \rightsquigarrow t$. Figure 8.2 shows the finite reachability tree of a Petri net transition system. It is interesting to compare it with the coverability tree in Figure 2.4. The latter also determines the limits $(1, 0, \omega)$ of computation sequences [Fin90].

Figure 8.2:
A Petri net and the finite reachability tree $FRT((1,0,0))$ of its transition system. Since $(1,0,0) \leq^3 (1,0,1)$ but $(1,0,1) \not\leq^3 (1,0,0)$, we conclude that the Petri net does not terminate and that it has an infinite state space.

Proposition 8.2.2 ([Fin90, FS01])
A WSTS $(S, \rightsquigarrow, \preceq)$ has a non-terminating computation from $s_0 \in S$ if and only if $FRT(s_0)$ contains a vertex t_+. If $\preceq$ is a partial ordering,[1] then $Reach(s_0)$ is infinite if and only if $FRT(s_0)$ contains t_+ with some predecessor s so that $s \preceq t_+$ and $t_+ \not\preceq s$. As $\preceq$ is a wqo, the tree $FRT(s_0)$ is finite and both problems are decidable.

To instantiate the framework, we define a qo $\preceq_{\mathcal{P}}$ on processes and prove it to be a wqo on $Reach(P)$ where P is bounded in depth (Section 8.2.1) and to be a simulation (Section 8.2.2). In fact, our wqo is also a partial ordering but we omit the proof here. We apply the decision procedures in Section 8.2.3.

8.2.1 An Adequate Well-Quasi-Ordering

Our wqo $\preceq_{\mathcal{P}}$ on processes is derived from a wqo on fragments. The idea of the *fragment ordering* $\preceq_{\mathcal{F}}$ is to use the rooted tree embedding and close it under structural congruence. The leafs in the trees are sequential processes or elementary fragments. Therefore, Rule (Elem) is mimicked by Rule (1): $F^e \preceq_{\mathcal{F}} F^e$. Fragment $\nu a.(\Pi_{i\in I} F_i)$ is dominated by $\nu a.(\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j)$ if the G_i dominate the F_i. This imitates Rule (Comp). If F' is smaller than G' then every $F \equiv F'$ is smaller than $G \equiv G'$, Rule (3).

Definition 8.2.3 (Fragment Ordering)
The *fragment ordering* $\preceq_{\mathcal{F}} \subseteq \mathcal{P}_{\mathcal{F}} \times \mathcal{P}_{\mathcal{F}}$ is defined by:

$$(1)\quad \frac{}{F^e \preceq_{\mathcal{F}} F^e} \qquad (2)\quad \frac{F_i \preceq_{\mathcal{F}} G_i \text{ for all } i \in I}{\nu a.(\Pi_{i\in I} F_i) \preceq_{\mathcal{F}} \nu a.(\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j)}$$

$$(3)\quad \frac{F \equiv F' \preceq_{\mathcal{F}} G' \equiv G}{F \preceq_{\mathcal{F}} G.}$$

♦

While reflexivity of $\preceq_{\mathcal{F}}$ is immediate, the proof of transitivity is more involved. It follows from Lemma 8.2.19 and we defer it until Section 8.2.2.

Lemma 8.2.4
$(\mathcal{P}_{\mathcal{F}}, \preceq_{\mathcal{F}})$ is a qo.

The crucial point is to prove that $\preceq_{\mathcal{F}}$ is a wqo on fragments of bounded depth. We sketch the proof before we plunge into the details.

[1] A partial ordering is a quasi-ordering that is antisymmetric, i.e., if $s \leq t$ and $t \leq s$ then $s = t$.

Proof Sketch
The idea is to conclude from wqo of the rooted tree embedding to wqo of the fragment ordering. To this end, we interpret fragments F as trees $\mathcal{T}[\![F]\!]$ (Definition 8.2.5). More precisely, we take the syntax tree of a fragment but do not decompose sequential processes. This means, the vertices in $\mathcal{T}[\![F]\!]$ are the sequential processes and the active restrictions in F (Lemma 8.2.6). The height of the resulting trees is the nesting of restrictions in the fragments, i.e., $\|F\|_\nu = \mathit{height}(\mathcal{T}[\![F]\!])$ holds (Lemma 8.2.7). Hence, we can now understand a sequence of fragments $(F_i)_{i\in\mathbb{N}}$ as a sequence of trees $(\mathcal{T}[\![F_i]\!])_{i\in\mathbb{N}}$ over a set A, which contains the sequential processes and the active restrictions in all F_i.

If the height of the trees is bounded, wqo of the rooted tree embedding ensures that there are two comparable trees, i.e., $\mathcal{T}[\![F_i]\!] \preceq_{\mathcal{T}} \mathcal{T}[\![F_j]\!]$ for some $i < j$. The aim is now to conclude the fragment ordering $F_i \preceq_{\mathcal{F}} F_j$ from this. Lemma 8.2.8 shows that this conclusion is valid as long as A is ordered by the identity. Combined with the requirement that (A, id) has to be a wqo in order for $id_{\mathcal{T}}$ to be a wqo (Proposition 8.1.7), we conclude that A has to be finite (exactly the finite sets are wqos with the identity). To sum up, the sequence of fragments $(F_i)_{i\in\mathbb{N}}$ has to satisfy the following requirements. (1) The nesting of restrictions $\|F_i\|_\nu$ needs to be bounded to ensure the height of the trees is bounded. (2) The sequential processes $\mathcal{S}(F_i)$ and the active restricted names $arn(F_i)$ in the fragments have to belong to a set A, to ensure we get trees over A. (3) The set A has to be finite to conclude from rooted tree embedding to fragment ordering.

Proposition 8.2.13 shows that we can in fact assume our sequence of fragments to satisfy these requirements. With the theory of derivatives in Section 4.2, we construct fragments that consist of a finite set of sequential processes. With the theory of anchored fragments in Section 7.3, we rewrite these fragments to anchored fragments so that the nesting of restrictions is bounded by the depth. There may still be arbitrarily many active restricted names composed in parallel:

$$\nu a.(\nu b.K\lfloor a, b\rfloor) \to \nu a.(\nu b.K\lfloor a, b\rfloor \mid \nu c.K\lfloor a, c\rfloor) \to \dots$$

The idea is to reuse restricted names or, stated differently, to use one restricted name for every nesting level. So, the above sequence is replaced by

$$\nu u_0.(\nu u_1.K\lfloor u_0, u_1\rfloor) \to \nu u_0.(\nu u_1.K\lfloor u_0, u_1\rfloor \mid \nu u_1.K\lfloor u_0, u_1\rfloor) \to \dots$$

Technically, the function con_0 maps a fragment F with $\|F\|_\nu = n$ to a fragment $con_0(F)$ where the active restrictions are included in $\{u_0, \dots, u_n\}$ (Lemma 8.2.10). Requirements (1), (2), and (3) above are met. We prove that the fragment ordering is a wqo in Lemma 8.2.14. ■

We start with the interpretation of fragments as trees. An elementary fragment F^e is a tree consisting of a single leaf, a composed fragment $\nu a.(F_1 \mid \dots \mid F_n)$ yields a composed tree where a is the root.

Definition 8.2.5 (Interpretation of Fragments as Trees)
The function $\mathcal{T}[\![-]\!]$ interprets a fragment $F \in \mathcal{P}_{\mathcal{F}}$ as a tree $\mathcal{T}[\![F]\!]$ in $\mathcal{T}(A)$, where the set A contains $\mathcal{S}(F) \cup arn(F)$:

$$\mathcal{T}[\![F^e]\!] := F^e \qquad \mathcal{T}[\![\nu a.(F_1 \mid \ldots \mid F_n)]\!] := (a, (\mathcal{T}[\![F_1]\!], \ldots, \mathcal{T}[\![F_n]\!])).$$

♦

Lemma 8.2.6 shows that the codomain is correct, i.e., if the set A contains the sequential processes and the active restrictions in F, then $\mathcal{T}[\![F]\!]$ is a tree over A.

Lemma 8.2.6 (The Codomain of $\mathcal{T}[\![-]\!]$ is Correct)
For every fragment $F \in \mathcal{P}_{\mathcal{F}}$ with $\mathcal{S}(F) \cup arn(F) \subseteq A$ it holds $\mathcal{T}[\![F]\!] \in \mathcal{T}(A)$.

Since the rooted tree embedding is a wqo only on trees of bounded height, it is crucial that the tree interpretation $\mathcal{T}[\![F]\!]$ yields trees where the height is related to the depth of F. The following lemma relates it to the nesting of restrictions. We create the relationship to the depth of F by translating an anchored fragment $F^{\mathcal{A}} \equiv F$ into a tree $\mathcal{T}[\![F^{\mathcal{A}}]\!]$.

Lemma 8.2.7
For all fragments $F \in \mathcal{P}_{\mathcal{F}}$ we have $\|F\|_\nu = height(\mathcal{T}[\![F]\!])$.

The following lemma allows us to conclude fragment ordering $F \preceq_{\mathcal{F}} G$ from rooted tree embedding $\mathcal{T}[\![F]\!] \preceq_{\mathcal{T}} \mathcal{T}[\![G]\!]$, as long as the trees are ordered by the identity.

Lemma 8.2.8 (From Rooted Tree Embedding to Fragment Ordering)
Consider a qo (A, id). If $\mathcal{T}[\![F]\!] id_{\mathcal{T}} \mathcal{T}[\![G]\!]$ then $F \preceq_{\mathcal{F}} G$ holds for all fragments $F, G \in \mathcal{P}_{\mathcal{F}}$.

Proof
We do an induction on the structure of F and show that for all G the ordering $\mathcal{T}[\![F]\!] id_{\mathcal{T}} \mathcal{T}[\![G]\!]$ implies $F \preceq_{\mathcal{F}} G$.

Base Case An elementary fragment F^e is interpreted as a a single leaf, $\mathcal{T}[\![F^e]\!] = F^e$. Since we assume $\mathcal{T}[\![F^e]\!] id_{\mathcal{T}} \mathcal{T}[\![G]\!]$ and only Rule (Elem) allows us to derive an ordering on leaves, we conclude that $\mathcal{T}[\![G]\!] = G^e$ is a single leaf as well. As $F^e id_{\mathcal{T}} G^e$ means $F^e = G^e$, we conclude $F^e \preceq_{\mathcal{F}} G^e$ with Rule (1).

Induction Step Assume the proposition holds for the fragments $F_1, \ldots, F_m$ and consider $F = \nu a.(F_1 \mid \ldots \mid F_m)$. We derive the ordering

$$\mathcal{T}[\![F]\!] = (a, (\mathcal{T}[\![F_1]\!], \ldots, \mathcal{T}[\![F_m]\!])) id_{\mathcal{T}} \mathcal{T}[\![G]\!]$$

only with Rule (Comp). Thus, $\mathcal{T}[\![G]\!] = (b, (\mathcal{T}[\![G_1]\!], \ldots, \mathcal{T}[\![G_n]\!]))$ with $a = b$ and $(\mathcal{T}[\![F_1]\!], \ldots, \mathcal{T}[\![F_m]\!]) id^*_{\mathcal{T}} (\mathcal{T}[\![G_1]\!], \ldots, \mathcal{T}[\![G_n]\!])$. Higman's ordering $id^*_{\mathcal{T}}$ implies there are indices $1 \leq i_1 < \ldots < i_m \leq n$ with $\mathcal{T}[\![F_k]\!] id_{\mathcal{T}} \mathcal{T}[\![G_{i_k}]\!]$. The hypothesis gives $F_k \preceq_{\mathcal{F}} G_{i_k}$. With Rule (2), we conclude

$$\nu a.(F_1 \mid \ldots \mid F_m) \preceq_{\mathcal{F}} \nu a.(G_{i_1} \mid \ldots \mid G_{i_m} \mid \Pi_{i \in I_{rem}} G_i),$$

where the index set I_{rem} contains the remaining indices different from the i_k. With structural congruence we reorder the fragments G_i:

$$\nu a.(F_1 \mid \ldots \mid F_m) \preceq_{\mathcal{F}} \nu a.(G_{i_1} \mid \ldots \mid G_{i_m} \mid \Pi_{i \in I_{rem}} G_i) \equiv \nu a.(G_1 \mid \ldots \mid G_n).$$

Rule (3) yields $F \preceq_{\mathcal{F}} G$. ■

To conclude $\preceq_{\mathcal{F}}$ is a wqo from the fact that $id_{\mathcal{T}}$ is a wqo with Lemma 8.2.8, (A, id) needs to be a wqo (cf. Proposition 8.1.7). This is the case if (and only if) A is finite. Thus, we need fragments that consist of a finite set of sequential processes and a finite set of restricted names. As discussed above, the idea is to reuse restricted names. Technically, we apply the function con_i to a give a unique name u_i to every nesting level i of restrictions. This means, for fragments $con_i(F)$ we relax the requirement that a name is bound at most once, Convention 2.1.11.

Definition 8.2.9 ($con_i : \mathcal{P}_{\mathcal{F}} \to \mathcal{P}_{\mathcal{F}}$)
For every $i \in \mathbb{N}$ the function $con_i : \mathcal{P}_{\mathcal{F}} \to \mathcal{P}_{\mathcal{F}}$ renames the active restrictions in a fragment into fresh names:

$$\begin{aligned} con_i(F^e) &:= F^e \\ con_i(\nu a.(F_1 \mid \ldots \mid F_n)) &:= \nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\}), \end{aligned}$$

where without loss of generality $\{u_i\}$ is fresh for all $con_{i+1}(F_k)$ with $1 \leq k \leq n$, i.e., $\{u_i\} \cap (fn(con_{i+1}(F_k)) \cup bn(con_{i+1}(F_k))) = \emptyset$. ♦

Of course, renaming yields structurally congruent fragments, i.e., $F \equiv con_i(F)$, and the nesting of restrictions $\|F\|_\nu$ does not change in $con_i(F)$. Most important is the fact that the nesting of restrictions $\|F\|_\nu$ determines the number of restricted names in $con_i(F)$ in the following way: $arn(con_i(F)) \subseteq \{u_i, \ldots, u_{i+\|F\|_\nu}\}$. Finally, the function con_i only changes the sequential processes by a substitution.

Lemma 8.2.10 (Properties of con_i)
For every $F \in \mathcal{P}_{\mathcal{F}}$ and every $i \in \mathbb{N}$ we have $con_i(F) \in \mathcal{P}_{\mathcal{F}}$, $con_i(F) \equiv F$, $\|con_i(F)\|_\nu = \|F\|_\nu$, $arn(con_i(F)) \subseteq \{u_i, \ldots, u_{i+\|F\|_\nu}\}$, and $\mathcal{S}(con_i(F)) = \mathcal{S}(F)\sigma$, where $\sigma : arn(F) \to arn(con_i(F))$.

Proof
We do an induction on the structure of fragments. The base case of elementary fragments is trivial. We directly turn to the induction step where we assume the proposition holds for the fragments $F_1, \ldots, F_n$ and consider $F = \nu a.(F_1 \mid \ldots \mid F_n)$. We have

$$con_i(F) \quad = \quad \nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\}).$$

By the hypothesis, each $con_{i+1}(F_k)$ is a fragment. As substitutions only change free names, the set of fragments is closed under the application of substitutions and $con_{i+1}(F_k)\{u_i/a\}$ is a fragment. Since F is a fragment, we have $a \in fn(F_k)$. Free names are preserved under structural congruence, so $a \in fn(con_{i+1}(F_k))$ holds. Hence, $u_i \in fn(con_{i+1}(F_k))\{u_i/a\} = fn(con_{i+1}(F_k)\{u_i/a\})$ by application of Lemma 2.1.16. We conclude that $con_i(F)$ is a fragment in $\mathcal{P}_{\mathcal{F}}$.

To show structural congruence, we observe:

$$\begin{aligned}
& & & \nu a.(F_1 \mid \ldots \mid F_n) \\
& (\text{ Hypothesis }) & \equiv & \nu a.(con_{i+1}(F_1) \mid \ldots \mid con_{i+1}(F_n)) \\
& (\ \alpha\text{-conversion }) & \equiv & \nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\}) \\
& (\text{ Def. } con_i\ ) & = & con_i(\nu a.(F_1 \mid \ldots \mid F_n)).
\end{aligned}$$

To see that the nesting of restrictions $\|F\|_\nu$ does not change under con_i, we exploit the invariance of $\| - \|_\nu$ under the application of substitutions:

$$\begin{aligned}
& & & \|con_i(F)\|_\nu \\
& (\text{ Def. } con_i\ ) & = & \|\nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\})\|_\nu \\
& (\text{ Def. } \|-\|_\nu\ ) & = & 1 + max\{\|con_{i+1}(F_k)\{u_i/a\}\|_\nu \mid 1 \leq k \leq n\} \\
& (\ \|F\sigma\|_\nu = \|F\|_\nu\ ) & = & 1 + max\{\|con_{i+1}(F_k)\|_\nu \mid 1 \leq k \leq n\} \\
& (\text{ Hypothesis }) & = & 1 + max\{\|F_k\|_\nu \mid 1 \leq k \leq n\} \\
& (\text{ Def. } \|-\|_\nu\ ) & = & \|F\|_\nu.
\end{aligned}$$

We check the restricted names:

$$\begin{aligned}
& & arn(con_i(F)) \\
& (\text{ Def. } con_i\ ) & = & arn(\nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\})) \\
& (\text{ Def. } arn\ ) & = & \{u_i\} \cup arn(con_{i+1}(F_1)\{u_i/a\}) \cup \ldots \cup arn(con_{i+1}(F_n)\{u_i/a\}).
\end{aligned}$$

Substitutions do not change the active restrictions, i.e., $arn(F\sigma) = arn(F)$. With this observation, we continue the equation:

$$\begin{aligned}
& & = & \{u_i\} \cup arn(con_{i+1}(F_1)) \cup \ldots \cup arn(con_{i+1}(F_n)) \\
& (\text{ Hypothesis }) & \subseteq & \{u_i\} \cup \{u_{i+1}, \ldots, u_{i+1+\|F_1\|_\nu}\} \cup \ldots \cup \{u_{i+1}, \ldots, u_{i+1+\|F_n\|_\nu}\}
\end{aligned}$$

$$\begin{array}{rcl} (\text{ Def. } max\ ) & = & \{u_i, \ldots, u_{i+1+max\{\|F_k\|_\nu \mid 1 \leq k \leq n\}}\} \\ (\text{ Def. } \|-\|_\nu\ ) & = & \{u_i, \ldots, u_{i+\|F\|_\nu}\}. \end{array}$$

Finally, we consider the sequential processes:

$$\begin{array}{rcl} & & \mathcal{S}(con_i(F)) \\ (\text{ Def. } con_i\ ) & = & \mathcal{S}(\nu u_i.(con_{i+1}(F_1)\{u_i/a\} \mid \ldots \mid con_{i+1}(F_n)\{u_i/a\})) \\ (\text{ Def. } \mathcal{S}\ ) & = & \mathcal{S}(con_{i+1}(F_1)\{u_i/a\}) \cup \ldots \cup \mathcal{S}(con_{i+1}(F_n)\{u_i/a\}) \\ (\text{ Lemma 2.1.21 }) & = & \mathcal{S}(con_{i+1}(F_1))\{u_i/a\} \cup \ldots \cup \mathcal{S}(con_{i+1}(F_n))\{u_i/a\} \\ (\text{ Hypothesis }) & = & \mathcal{S}(F_1)\sigma_1\{u_i/a\} \cup \ldots \cup \mathcal{S}(F_n)\sigma_n\{u_i/a\}, \end{array}$$

where $\sigma_k : arn(F_k) \to arn(con_{i+1}(F_k))$. By Convention 2.1.11, a name is bound at most once in F. Hence, the sets of restricted names $arn(F_k)$ are all disjoint and disjoint with $\{a\}$. Consequently, the substitutions σ_k as well as $\{u_i/a\}$, which are sets of pairs, are disjoint. So their union is well defined:

$$\sigma := \sigma_1 \cup \ldots \cup \sigma_n \cup \{u_i/a\}.$$

In the following step, we exploit the equality $\mathcal{S}(F_k)\sigma_k\{u_i/a\} = \mathcal{S}(F_k)\sigma$. To see that it holds, assume we have a name $b \in fn(\mathcal{S}(F_k))$, which is mapped by σ while it is kept identical by $\sigma_k\{u_i/a\}$. Then there is some σ_l with b in its domain. This means, $b \in arn(F_l)$ and at the same time free in F_k. Since the bound and free names are disjoint in F, this cannot be the case. The equality holds:

$$\begin{array}{rcl} & = & \mathcal{S}(F_1)\sigma \cup \ldots \cup \mathcal{S}(F_n)\sigma \\ (\text{ Applic. } \sigma \text{ to sets }) & = & (\mathcal{S}(F_1) \cup \ldots \cup \mathcal{S}(F_n))\sigma \\ (\text{ Def. } \mathcal{S}\ ) & = & (\mathcal{S}(\nu a.(F_1 \mid \ldots \mid F_n))\sigma. \end{array}$$

We check that the domain of σ is $arn(F)$ and the codomain is $arn(con_i(F))$. For the domain, we have:

$$\begin{array}{rcl} & & arn(F) \\ & = & \{a\} \cup arn(F_1) \cup \ldots \cup arn(F_n), \end{array}$$

which is the domain of $\sigma = \sigma_1 \cup \ldots \cup \sigma_n \cup \{u_i/a\}$. For the codomain, we get

$$\begin{array}{rcl} & & arn(con_i(F)) \\ (\text{ Def. } arn\ ) & = & \{u_i\} \cup arn(con_{i+1}(F_1)\{u_i/a\}) \cup \ldots \cup arn(con_{i+1}(F_n)\{u_i/a\}) \\ & = & \{u_i\} \cup arn(con_{i+1}(F_1)) \cup \ldots \cup arn(con_{i+1}(F_n)), \end{array}$$

where we again exploit the invariance of the active restrictions under substitutions. The last term is the codomain of σ, which concludes the proof. ■

Example 8.2.11
Consider $F^{\mathcal{A}} = \nu a.(\nu b_1.K\lfloor a, b_1\rfloor \mid \nu b_2.L\lfloor a, b_2\rfloor \mid \nu b_3.L\lfloor a, b_3\rfloor)$. An application of con_0 yields $con_0(F^{\mathcal{A}}) = \nu u_0.(\nu u_1.K\lfloor u_0, u_1\rfloor \mid \nu u_1.L\lfloor u_0, u_1\rfloor \mid \nu u_1.L\lfloor u_0, u_1\rfloor)$. ♦

In the next Proposition 8.2.13, we construct the particular fragments that satisfy the Requirements (1), (2), and (3) above. In Section 8.2.3, we consider a larger example that illustrates the concept.

Remark 8.2.12
The proof of Proposition 8.2.13 exploits all deeper results in this thesis, from the characterisation of structural congruence with the restricted form in Section 3.2 over the construction of derivatives in Section 4.2 to the anchored fragments in Section 7.3. In the following Lemma 8.2.14, we combine the proposition with the rooted tree embedding in Section 8.1 to show that the fragment ordering is a wqo. With respect to the required foundations, we consider Proposition 8.2.13 and Lemma 8.2.14 to be the deepest results in this thesis. At the same time, they are the most beautiful results as they shed new light on the order-theoretic structure of the transition systems of π-Calculus processes.

We furthermore remark that the proofs of Proposition 8.2.13 and Lemma 8.2.14 are both remarkably simple: straightforward derivations. All previous results fit together smoothly and are just plugged in at the right position. This justifies our claim that the elaborated theories—*restricted form*, *derivatives*, *depth*, *anchored fragments*, and *rooted tree embedding*—are natural notions. They should be applicable also outside the theory of structural stationarity. ♦

Proposition 8.2.13 (Particular Fragments)
Consider process $P \in \mathcal{P}$ and the reachable fragment $F \in fg\,(rf(Reach(P)))$. There is a fragment $G \equiv F$ so that $\|G\|_\nu \leq 2^{\|F\|_\mathcal{D}} - 1$, $arn(G) \subseteq \{u_0, \ldots, u_{\|G\|_\nu}\}$, and $\mathcal{S}(G) \subseteq \{Q\sigma \mid Q \in derivatives(P) \text{ and } \sigma : fn(Q) \to fn(P) \cup arn(G)\}$.

Proof
Let $F \in fg\,(rf(Reach(P)))$ be reachable. With Proposition 4.2.2, we have

$$F \equiv \nu\tilde{a}.Q^{\neq\nu}, \quad \text{where} \quad Q^{\neq\nu} = \Pi_{i\in I} Q_i\sigma_i$$

with $Q_i \in derivatives(P)$ and $\sigma_i : fn(Q_i) \to \tilde{a} \cup fn(P)$. With Lemma 3.2.7, we compute the restricted form $\nu\tilde{a}.Q^{\neq\nu} \equiv rf(\nu\tilde{a}.Q^{\neq\nu})$. Proposition 3.2.10 yields restricted equivalence:

$$F = rf(F) \equiv_{rf} rf(\nu\tilde{a}.Q^{\neq\nu}).$$

Since F is a fragment, restricted equivalence ensures $rf(\nu\tilde{a}.Q^{\neq\nu})$ is a fragment. For $rf(\nu\tilde{a}.Q^{\neq\nu})$, we compute the structurally congruent anchored fragment with

Proposition 7.3.4. We denote it by $F^{\mathcal{A}}$. Applying con_0 to $F^{\mathcal{A}}$ yields the desired fragment G, i.e., we now have

$$G = con_0(F^{\mathcal{A}}) \equiv F^{\mathcal{A}} \equiv rf(\nu\tilde{a}.Q^{\neq\nu}) \equiv \nu\tilde{a}.Q^{\neq\nu} \equiv F.$$

We first check the nesting of restrictions:

$$\begin{aligned}
& & \|G\|_\nu \\
(\text{ Def. } G) & = & \|con_0(F^{\mathcal{A}})\|_\nu \\
(\text{ Lemma 8.2.10 }) & = & \|F^{\mathcal{A}}\|_\nu \\
(\text{ Corollary 7.4.5 }) & \leq & 2^{\|F^{\mathcal{A}}\|_{\mathcal{D}}} - 1 \\
(\text{ Invariance of } \|-\|_{\mathcal{D}} \text{ under } \equiv) & = & 2^{\|F\|_{\mathcal{D}}} - 1.
\end{aligned}$$

The active restrictions are included in the set $\{u_0, \ldots, u_{\|G\|_\nu}\}$ with the properties of con_0. More precisely:

$$\begin{aligned}
& & arn(G) \\
(\text{ Def. } G) & = & arn(con_0(F^{\mathcal{A}})) \\
(\text{ Lemma 8.2.10 }) & \subseteq & \{u_0, \ldots, u_{\|F^{\mathcal{A}}\|_\nu}\} \\
(\ \|G\|_\nu = \|F^{\mathcal{A}}\|_\nu, \text{ Lemma 8.2.10 }) & \subseteq & \{u_0, \ldots, u_{\|G\|_\nu}\}.
\end{aligned}$$

It remains to be shown that the sequential processes satisfy the conditions above. In the following chain of equations, we omit the domains and codomains of σ and σ'. We discuss that they are correct afterwards:

$$\begin{aligned}
& & \mathcal{S}(G) \\
(\text{ Def. } G) & = & \mathcal{S}(con_0(F^{\mathcal{A}})) \\
(\text{ Lemma 8.2.10 }) & = & \mathcal{S}(F^{\mathcal{A}})\sigma \\
(\text{ Proposition 7.3.4 }) & = & \mathcal{S}(rf(\nu\tilde{a}.Q^{\neq\nu}))\sigma \\
(\text{ Lemma 3.2.7 }) & = & \mathcal{S}(\nu\tilde{a}.Q^{\neq\nu})\sigma \\
(\text{ Def. } \mathcal{S}, \text{ form of } Q^{\neq\nu}) & = & \{Q\sigma' \mid Q \in derivatives(P)\}\sigma \\
& = & \{Q\sigma'\sigma \mid Q \in derivatives(P)\}.
\end{aligned}$$

We now have to show that $\sigma'\sigma$ maps $fn(Q)$ into $fn(P) \cup arn(G)$. With Lemma 2.1.15, it is sufficient to show $fn(Q\sigma'\sigma) \subseteq fn(P) \cup arn(G)$. We prove this as follows:

$$\begin{aligned}
& & fn(Q\sigma'\sigma) \\
(\ Q\sigma'\sigma \in \mathcal{S}(G), \text{ Lemma 2.1.22 }) & \subseteq & fn(G) \cup arn(G) \\
(\ fn(G) = fn(F), \text{ Lemma 2.1.19 }) & = & fn(F) \cup arn(G).
\end{aligned}$$

Let F be a fragment in $fg\,(rf(R))$, where R is a reachable process of P. This means, $rf(R) = R_1^{rf} \mid F \mid R_2^{rf}$. Hence, the free names of F are included in the free names of $rf(R)$. This continues the inclusion:

$$\begin{aligned} & & \subseteq & \quad fn(rf(R)) \cup arn(G) \\ & (\ rf(R) \equiv R, \text{ Lemma 2.1.19 }) & = & \quad fn(R) \cup arn(G) \\ & (\ R \in Reach(P), \text{ Lemma 2.1.37 }) & \subseteq & \quad fn(P) \cup arn(G). \end{aligned}$$

This concludes the proof. ■

The fragment ordering is a wqo on fragments of bounded depth.

Lemma 8.2.14
For every process $P \in \mathcal{P}_{\mathcal{D}<\infty}$ the set $(fg\,(rf(Reach(P))), \preceq_{\mathcal{F}})$ is a wqo.

Proof
Consider $P \in \mathcal{P}_{\mathcal{D}<\infty}$ where $k_{\mathcal{D}} \in \mathbb{N}$ is a bound on the depth of the reachable fragments. Our aim is to understand the reachable fragments as trees over a suitable set A. This set is defined by

$$\begin{aligned} A := \quad & \{u_0, \ldots, u_{2^{k_{\mathcal{D}}}-1}\} \\ & \cup \{Q\sigma \mid Q \in derivatives(P) \text{ and } \sigma : fn(Q) \to fn(P) \cup \{u_0, \ldots, u_{2^{k_{\mathcal{D}}}-1}\}\}. \end{aligned}$$

Since the set of derivatives is finite by Lemma 4.2.5, the set A is finite as well. Any finite set is a wqo with the identity as ordering, i.e., (A, id) is a wqo.

Consider a sequence $(F_i)_{i\in\mathbb{N}}$ in $fg\,(rf(Reach(P)))$. We show that it contains two comparable elements. Every F_i is structurally congruent with a fragment G_i as defined in Proposition 8.2.13: $\|G_i\|_\nu \leq 2^{\|F_i\|_{\mathcal{D}}} - 1$, $arn(G_i) \subseteq \{u_0, \ldots, u_{\|G_i\|_\nu}\}$, and $\mathcal{S}(G_i) \subseteq \{Q\sigma \mid Q \in derivatives(P) \text{ and } \sigma : fn(Q) \to fn(P) \cup arn(G_i)\}$. To see that the tree $\mathcal{T}[\![G_i]\!]$ is in $\mathcal{T}(A)$, the set A needs to contain all active restrictions and sequential processes in G_i. This is the case, because

$$\|G_i\|_\nu \leq 2^{\|F_i\|_{\mathcal{D}}} - 1 \leq 2^{k_{\mathcal{D}}} - 1$$

with the boundedness assumption on $\|F_i\|_{\mathcal{D}}$. Hence, $\mathcal{T}[\![G_i]\!] \in \mathcal{T}(A)$ for all i.

According to Lemma 8.2.7, the height of $\mathcal{T}[\![G_i]\!]$ is equal to the nesting of restrictions in G_i. Thus, we have a sequence $(\mathcal{T}[\![G_i]\!])_{i\in\mathbb{N}}$ of trees in $\mathcal{T}(A)_{2^{k_{\mathcal{D}}}-1}$. With Proposition 8.1.7, $(\mathcal{T}(A)_{2^{k_{\mathcal{D}}}-1}, id_{\mathcal{T}})$ is a wqo. Hence there are $i < j$ with $\mathcal{T}[\![G_i]\!] id_{\mathcal{T}} \mathcal{T}[\![G_j]\!]$. Since A is ordered by the identity, $G_i \preceq_{\mathcal{F}} G_j$ with Lemma 8.2.8. With Rule (3), we conclude $F_i \preceq_{\mathcal{F}} F_j$. The lemma holds. ■

Recall that the states in the transition system $\mathcal{T}\,(P)$ are the classes of the reachable processes of P under structural congruence. Hence, to prove the transition system of a process $P \in \mathcal{P}_{\mathcal{D}<\infty}$ to be well-structured requires a qo $\preceq_{\mathcal{P}}$ on

the congruence classes $Reach(P)/_{\equiv}$. The idea is to exploit the restricted form of a process and define $\preceq_{\mathcal{P}}$ in terms of the fragment ordering. We have $[Q] \preceq_{\mathcal{P}} [R]$ if every fragment in the restricted form of Q is dominated by a fragment in the restricted form of R. Since parallel composition is associative and commutative, we can assume that the fragments F_i in $rf(Q)$ and G_i in $rf(R)$ are ordered so that F_i is dominated by G_i. For example, $[Q] = [F \mid F] \preceq_{\mathcal{P}} [G \mid G' \mid H] = [R]$ if $F \preceq_{\mathcal{F}} G$ and $F \preceq_{\mathcal{F}} G'$.

Definition 8.2.15 ($\preceq_{\mathcal{P}} \subseteq \mathcal{P}/_{\equiv} \times \mathcal{P}/_{\equiv}$)
With Lemma 3.2.7, every process $P \in \mathcal{P}$ is structurally congruent to a parallel composition of fragments. Therefore, we define the relation $\preceq_{\mathcal{P}} \subseteq \mathcal{P}/_{\equiv} \times \mathcal{P}/_{\equiv}$ by

$$[\Pi_{i\in I} F_i] \preceq_{\mathcal{P}} [\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j],$$

where $F_i \preceq_{\mathcal{F}} G_i$ for all $i \in I$. ♦

Note that in particular the stop process $\mathbf{0}$ is dominated by any process as it is represented by $\Pi_{i\in\emptyset} F_i$, where the index set is empty.

Lemma 8.2.16
$(\mathcal{P}/_{\equiv}, \preceq_{\mathcal{P}})$ is a qo.

Proof
Reflexivity of $\preceq_{\mathcal{P}}$ follows immediately from reflexivity of $\preceq_{\mathcal{F}}$.

Transitivity Assume that $[P] \preceq_{\mathcal{P}} [Q] \preceq_{\mathcal{P}} [R]$. With $[P] \preceq_{\mathcal{P}} [Q]$, we have $[P] = [\Pi_{i\in I} F_i] \preceq_{\mathcal{P}} [\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j] = [Q]$ so that $F_i \preceq_{\mathcal{F}} G_i$ for all $i \in I$. Similarly, $[Q] \preceq_{\mathcal{P}} [R]$ means $[Q] = [\Pi_{k\in K} G'_k]$ and $[R] = [\Pi_{k\in K} H_k \mid \Pi_{l\in L} H_l]$ so that $G'_k \preceq_{\mathcal{F}} H_k$ for all $k \in K$. Since structural congruence and restricted equivalence coincide on processes in restricted form due to Corollary 3.2.11, we have $\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j \equiv_{rf} \Pi_{k\in K} G'_k$. This means, the fragments G'_k can be reordered so that $\Pi_{k\in K} G'_k \equiv_{rf} \Pi_{k_i\in K_I} G'_{k_i} \mid \Pi_{k_j\in K_J} G'_{k_j}$ with $G_i \equiv G'_{k_i}$ and $G_j \equiv G'_{k_j}$. We also reorder the fragments H_k: $\Pi_{k\in K} H_k \equiv_{rf} \Pi_{k_i\in K_I} H_{k_i} \mid \Pi_{k_j\in K_J} H_{k_j}$ so that $G'_{k_i} \preceq_{\mathcal{F}} H_{k_i}$. We now have $G_i \equiv G'_{k_i} \preceq_{\mathcal{F}} H_{k_i}$. Rule (3) in the definition of $\preceq_{\mathcal{F}}$ yields $G_i \preceq_{\mathcal{F}} H_{k_i}$. With transitivity of $\preceq_{\mathcal{F}}$, we conclude $F_i \preceq_{\mathcal{F}} H_{k_i}$. Transitivity of $\preceq_{\mathcal{P}}$ holds:

$$\begin{aligned}
[P] &= [\Pi_{i\in I} F_i] \\
&\preceq_{\mathcal{P}} [\Pi_{k_i\in K_I} H_{k_i} \mid \Pi_{k_j\in K_J} H_{k_j} \mid \Pi_{l\in L} H_l] \\
&= [\Pi_{k\in K} H_k \mid \Pi_{l\in L} H_l] \\
&= [R].
\end{aligned}$$

■

The main result in this section states that $\preceq_{\mathcal{P}}$ is a wqo on the reachable processes of $P \in \mathcal{P}_{\mathcal{D}<\infty}$. This follows from Lemma 8.2.14 and Higman's result.

Proposition 8.2.17 ($\preceq_{\mathcal{P}}$ is a WQO for $\mathcal{P}_{\mathcal{D}<\infty}$)
If $P \in \mathcal{P}_{\mathcal{D}<\infty}$, then $(\mathit{Reach}(P)/_{\equiv}, \preceq_{\mathcal{P}})$ is a wqo.

Proof
Consider $P \in \mathcal{P}_{\mathcal{D}<\infty}$. With Lemma 8.2.14, $(\mathit{fg}(\mathit{rf}(\mathit{Reach}(P))), \preceq_{\mathcal{F}})$ is a wqo. Thus, the words over $\mathit{fg}(\mathit{rf}(\mathit{Reach}(P)))$ are a wqo with the $\preceq^*_{\mathcal{F}}$ ordering according to Higman's result. We interpret the parallel composition of fragments $\Pi_{i\in I}F_i = F_{i_1} \mid \ldots \mid F_{i_n}$ as such a word $(F_{i_1}, \ldots, F_{i_n})$. Since every process Q is structurally congruent with a process in restricted form, in every infinite sequence $([P_i])_{i\in\mathbb{N}}$ in $\mathit{Reach}(P)/_{\equiv}$ there are $i < j$ with

$$P_i \equiv \Pi_{i\in I}F_i \preceq^*_{\mathcal{F}} \Pi_{j\in J}G_j \equiv P_j.$$

The ordering $\preceq^*_{\mathcal{F}}$ demands that each F_i is dominated by some G_{j_i}. We reorder the fragments G_j so that J_I contains these indices j_i. This gives

$$[P_i] = [\Pi_{i\in I}F_i] \preceq_{\mathcal{P}} [\Pi_{j\in J_I}G_j \mid \Pi_{j\in J\setminus J_I}G_j] = [P_j].$$

Thus, $(\mathit{Reach}(P)/_{\equiv}, \preceq_{\mathcal{P}})$ is a wqo. ■

To conclude, we remark that $\preceq_{\mathcal{P}}$ is in fact a partial ordering. We omit the proof as it requires a number of additional insights about the fragment ordering $\preceq_{\mathcal{F}}$, the number of sequential processes in fragments $\|-\|_{\mathcal{S}}$, and structural congruence.

Remark 8.2.18
The ordering $\preceq_{\mathcal{P}} \subseteq \mathcal{P}/_{\equiv} \times \mathcal{P}/_{\equiv}$ is antisymmetric and so a partial ordering. ♦

8.2.2 Proof of Simulation

In the proof that $\preceq_{\mathcal{P}}$ is a simulation, the following Lemma 8.2.19 is crucial. It relates the fragment ordering $F \preceq_{\mathcal{F}} G$ with the standard form of F. This standard form is covered by G in a way that reveals $\preceq_{\mathcal{F}}$ is a simulation. We do not use the function *sf* as it is more convenient in the induction step to have the freedom of structural congruence.

Lemma 8.2.19
For all $F, G \in \mathcal{P}_{\mathcal{F}}$ we have $F \preceq_{\mathcal{F}} G$ if and only if $F \equiv \nu\tilde{a}.P^{\neq\nu}$ in standard form and $G \equiv \nu\tilde{a}.(P^{\neq\nu} \mid R)$ for some $R \in \mathcal{P}$.

Proof
$\Rightarrow$ We proceed by induction on the derivations of $\preceq_{\mathcal{F}}$. In the base case, we have elementary fragments $F^e \preceq_{\mathcal{F}} F^e$, which are are non-empty choices or calls to identifiers and hence in standard form. The proposition holds with $R = \mathbf{0}$.

Induction Step Let the statement hold for $F \preceq_{\mathcal{F}} G$ and $F_i \preceq_{\mathcal{F}} G_i$ with $i \in I$.

Rule (2) Consider $\nu a.(\Pi_{i\in I} F_i) \preceq_{\mathcal{F}} \nu a.(\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j)$. By the hypothesis, we have $F_i \equiv \nu\tilde{a}_i.P_i^{\neq\nu}$ and $G \equiv \nu\tilde{a}_i.(P_i^{\neq\nu} \mid R_i)$. Since structural congruence is preserved by α-conversion, we can assume $\tilde{a}_i$ disjoint from the free names in $P_j^{\neq\nu} \mid R_j$, in particular $\tilde{a}_j$, and from the free names in G_j for all $j \neq i$. Therefore, the scope extrusions in the following two systems of congruences are correct. We start with the form of $\nu a.(\Pi_{i\in I} F_i)$ and let $I = \{i_1, \ldots, i_n\}$:

$$\begin{aligned} & & \nu a.(\Pi_{i\in I} F_i) \\ (\text{ Hypothesis }) & \equiv & \nu a.(\Pi_{i\in I} \nu\tilde{a}_i.P_i^{\neq\nu}) \\ (\text{ Scope extrusion }) & \equiv & \nu a, \tilde{a}_{i_1}, \ldots, \tilde{a}_{i_n}.(\Pi_{i\in I} P_i^{\neq\nu}). \end{aligned}$$

To see that the latter process is in standard form, we check that a is in the free names of some $P_i^{\neq\nu}$. This holds even for all $i \in I$, since $a \in fn(F_i) = fn(\nu\tilde{a}_i.P_i^{\neq\nu})$ by the definition of fragments and the invariance of free names under structural congruence. For the process $\nu a.(\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j)$ we proceed similarly:

$$\begin{aligned} & & \nu a.(\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j) \\ (\text{ Hypothesis }) & \equiv & \nu a.(\Pi_{i\in I} \nu\tilde{a}_i.(P_i^{\neq\nu} \mid R_i) \mid \Pi_{j\in J} G_j) \\ (\text{ Scope extrusion }) & \equiv & \nu a, \tilde{a}_{i_1}, \ldots, \tilde{a}_{i_n}.(\Pi_{i\in I} (P_i^{\neq\nu} \mid R_i) \mid \Pi_{j\in J} G_j) \\ (\text{ Assoc. and commut. } \mid) & \equiv & \nu a, \tilde{a}_{i_1}, \ldots, \tilde{a}_{i_n}.(\Pi_{i\in I} P_i^{\neq\nu} \mid \Pi_{i\in I} R_i \mid \Pi_{j\in J} G_j) \\ (\ R := \Pi_{i\in I} R_i \mid \Pi_{j\in J} G_j\ ) & \equiv & \nu a, \tilde{a}_{i_1}, \ldots, \tilde{a}_{i_n}.(\Pi_{i\in I} P_i^{\neq\nu} \mid R). \end{aligned}$$

Rule (3) Consider $F' \preceq_{\mathcal{F}} G'$ where $F' \equiv F$ and $G' \equiv G$. By the hypothesis, we have structurally congruent processes in the required form for F and G. By transitivity of structural congruence, they also work for F' and G'.

$\Leftarrow$ Let $F \equiv \nu\tilde{a}.P^{\neq\nu}$ and $G \equiv \nu\tilde{a}.(P^{\neq\nu} \mid R)$ with $P^{\neq\nu} = \Pi_{i\in I} P_i$. We restrict the scopes of all $a \in \tilde{a}$ to get fragments:

$$\begin{aligned} & & \nu\tilde{a}.P^{\neq\nu} \\ (\text{ Let } \tilde{a} = a_k, \ldots, a_1\ ) & = & \nu a_k, \ldots, a_1.(\Pi_{i\in I} P_i) \\ (\ i \in I_l \text{ iff } a_l \in fn(P_i)\ ) & \equiv & \nu a_k.(\ldots \nu a_1.(\Pi_{i\in I_1} P_i) \ldots \Pi_{i\in I_k \setminus (I_{k-1}\cup\ldots\cup I_1)} P_i). \end{aligned}$$

Similarly, we get

$$\nu\tilde{a}.(P^{\neq\nu} \mid R)$$

$$
\begin{aligned}
&(\text{ Let } rf(R) = \Pi_{j\in J} H_j \ ) \quad\equiv\quad \nu a_k, \ldots, a_1.(\Pi_{i\in I} P_i \mid \Pi_{j\in J} H_j)\\
&(\ j \in J_l \text{ iff } a_l \in fn(H_j)\ ) \quad\equiv\quad \nu a_k.(\ldots \nu a_1.(\Pi_{i\in I_1} P_i \mid \Pi_{j\in J_1} H_j)\ldots\\
&\qquad\qquad \Pi_{i\in I_k\setminus(I_{k-1}\cup\ldots\cup I_1)} P_i \mid \Pi_{j\in J_k\setminus(J_{k-1}\cup\ldots\cup J_1)} H_j).
\end{aligned}
$$

Since fragments cannot be decomposed (cf. definition of $\equiv_{rf}$) and since G is a fragment, for every H_j there is a name $a_l \in fn(H_j)$. With Rule (1) and (2), we establish the fragment ordering

$$
\begin{aligned}
&\nu a_k.(\ldots \nu a_1.(\Pi_{i\in I_1} P_i)\ldots \Pi_{i\in I_k\setminus(I_{k-1}\cup\ldots\cup I_1)} P_i)\\
\preceq_{\mathcal{F}}\quad &\nu a_k.(\ldots \nu a_1.(\Pi_{i\in I_1} P_i \mid \Pi_{j\in J_1} H_j)\ldots\\
&\qquad \Pi_{i\in I_k\setminus(I_{k-1}\cup\ldots\cup I_1)} P_i \mid \Pi_{j\in J_k\setminus(J_{k-1}\cup\ldots\cup J_1)} H_j).
\end{aligned}
$$

Since $F \equiv \nu\tilde{a}.P^{\neq\nu}$ and $G \equiv \nu\tilde{a}.(P^{\neq\nu} \mid R)$, we conclude $F \preceq_{\mathcal{F}} G$ with Rule (3) and the congruences for $\nu\tilde{a}.P^{\neq\nu}$ and $\nu\tilde{a}.(P^{\neq\nu} \mid R)$ above. ■

The lemma immediately shows that the fragment ordering is transitive.

Proof (of Lemma 8.2.4)
Consider $F, G, H \in \mathcal{P}_{\mathcal{F}}$ with $F \preceq_{\mathcal{F}} G \preceq_{\mathcal{F}} H$. We establish $F \preceq_{\mathcal{F}} H$. As $F \preceq_{\mathcal{F}} G$, Lemma 8.2.19 gives $F \equiv \nu\tilde{a}.P^{\neq\nu}$ in standard form and $G \equiv \nu\tilde{a}.(P^{\neq\nu} \mid R)$ for some $R \in \mathcal{P}$. We inspect G:

$$
\begin{aligned}
& G\\
\equiv\quad & \nu\tilde{a}.(P^{\neq\nu} \mid R)\\
\equiv\quad & \nu\tilde{a}.(P^{\neq\nu} \mid \nu\tilde{a}_R.R^{\neq\nu})\\
\equiv\quad & \nu\tilde{a}, \tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu}).
\end{aligned}
\qquad
\begin{aligned}
&(\text{ Lemma 2.1.28: } sf(R) = \nu\tilde{a}_R.R^{\neq\nu}\ )\\
&(\text{ Scope extrusion })
\end{aligned}
$$

We justify the scope extrusion in the last step. By Convention 2.1.11, the free names in $P^{\neq\nu}$ are disjoint from the bound names in R. With Lemma 2.1.28, we get $\tilde{a}_R = arn(sf(R)) \subseteq arn(R)$. Hence, $\tilde{a}_R \cap fn(P^{\neq\nu}) = \emptyset$ and the scope extrusion is correct.

As $G \preceq_{\mathcal{F}} H$, Lemma 8.2.19 gives $G \equiv \nu\tilde{c}.Q^{\neq\nu}$ and $H \equiv \nu\tilde{c}.(Q^{\neq\nu} \mid S)$ for some process $S \in \mathcal{P}$. We now have $\nu\tilde{c}.Q^{\neq\nu} \equiv G \equiv \nu\tilde{a}, \tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu})$, where the first and the last process are in standard form. Structural congruence and standard equivalence coincide on processes in standard form according to Corollary 2.1.32, i.e., we have $\nu\tilde{a}, \tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu}) \equiv_{sf} \nu\tilde{c}.Q^{\neq\nu}$. With Lemma 2.1.33, there is a bijective substitution $\sigma : \tilde{c} \to \tilde{a} \cup \tilde{a}_R$ so that $Q^{\neq\nu}\sigma \equiv_{sf} P^{\neq\nu} \mid R^{\neq\nu}$. We apply this substitution to H and get

$$
\begin{aligned}
& H\\
\equiv\quad & \nu\tilde{c}.(Q^{\neq\nu} \mid S)\\
\equiv\quad & \nu\tilde{a}, \tilde{a}_R.(Q^{\neq\nu}\sigma \mid S\sigma)
\end{aligned}
\qquad
\begin{aligned}
&(\text{ Form of } H\ )\\
&(\text{ Applic. } \sigma \text{ above })
\end{aligned}
$$

$$\begin{array}{rcl}(\ Q^{\neq\nu}\sigma \equiv_{sf} P^{\neq\nu} \mid R^{\neq\nu} \text{ above }) & \equiv & \nu\tilde{a},\tilde{a}_R.(P^{\neq\nu} \mid R^{\neq\nu} \mid S\sigma) \\ (\ \text{Scope extrusion, } \tilde{a}_R \cap fn(P^{\neq\nu}) = \emptyset \text{ above }) & \equiv & \nu\tilde{a}.(P^{\neq\nu} \mid \nu\tilde{a}_R.(R^{\neq\nu} \mid S\sigma)).\end{array}$$

Lemma 8.2.19 gives $F \preceq_{\mathcal{F}} H$. ■

We now show that the ordering $\preceq_{\mathcal{P}}$ is a simulation relation. This concludes the proof that processes of bounded depth have WSTS. Before we turn to the technicalities, we briefly outline our arguments. The ordering

$$[P] = [\Pi_{i\in I} F_i] \preceq_{\mathcal{P}} [\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j] = [Q],$$

means each fragment F_i is dominated by G_i in the fragment ordering, i.e., $F_i \preceq_{\mathcal{F}} G_i$. For the moment, let us assume we have a single fragment $F \preceq_{\mathcal{F}} G$. Lemma 8.2.19 shows that F is structurally congruent with a process $\nu\tilde{a}.P^{\neq\nu}$ in standard form, while G is structurally congruent with $\nu\tilde{a}.(P^{\neq\nu} \mid R)$. By a case distinction, we check that $\nu\tilde{a}.P^{\neq\nu} \to \nu\tilde{a}.Q^{\neq\nu}$ can be mimicked by $\nu\tilde{a}.(P^{\neq\nu} \mid R) \to \nu\tilde{a}.(Q^{\neq\nu} \mid R)$. To show that the resulting processes are related by the fragment ordering, i.e., to establish $[\nu\tilde{a}.Q^{\neq\nu}] \preceq_{\mathcal{P}} [\nu\tilde{a}.(Q^{\neq\nu} \mid R)]$, we apply the direction from right to left in Lemma 8.2.19. Of course, in the proof of Proposition 8.2.20, we consider several fragments F_i. We first extrude the scopes of the names $\tilde{a}_i$ and then proceed in the explained way.

Proposition 8.2.20 ($\preceq_{\mathcal{P}}$ is a Simulation)
The relation $\preceq_{\mathcal{P}} \subseteq \mathcal{P}/_{\equiv} \times \mathcal{P}/_{\equiv}$ is a simulation.

Proof
Let $[P] = [\Pi_{i\in I} F_i] \preceq_{\mathcal{P}} [\Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j] = [Q]$. We show that for all $[P'] \in \mathcal{P}/_{\equiv}$ with $[P] \to_{\mathcal{T}} [P']$ there is $[Q'] \in \mathcal{P}/_{\equiv}$ with $[Q] \to_{\mathcal{T}} [Q']$ and $[P'] \preceq_{\mathcal{P}} [Q']$.

The definition of $\preceq_{\mathcal{P}}$ gives $F_i \preceq_{\mathcal{F}} G_i$ for all $i \in I$. With Lemma 8.2.19, every F_i is structurally congruent with a standard form $\nu\tilde{a}_i.P_i^{\neq\nu}$. The G_i are structurally congruent with $\nu\tilde{a}_i.(P_i^{\neq\nu} \mid R_i)$. Structural congruence allows us to assume that the names $\tilde{a}_i$ are disjoint from the free names of the other $P_k^{\neq\nu}$, R_k, and the free names in the processes G_j. Hence, we can extrude the scopes of the name $\tilde{a}_i$:

$$\begin{array}{rcl}P & \equiv & \Pi_{i\in I} F_i \\ (\ \text{Lemma 8.2.19}\ ) & \equiv & \Pi_{i\in I} \nu\tilde{a}_i.P_i^{\neq\nu} \\ (\ \text{Scope extrusion, } \tilde{a} := \bigcup_{i\in I} \tilde{a}_i\ ) & \equiv & \nu\tilde{a}.(\Pi_{i\in I} P_i^{\neq\nu}) \\ (\ P^{\neq\nu} := \Pi_{i\in I} P_i^{\neq\nu}\ ) & = & \nu\tilde{a}.P^{\neq\nu}.\end{array}$$

Similarly we get for Q:

$$\begin{array}{rcl}Q & \equiv & \Pi_{i\in I} G_i \mid \Pi_{j\in J} G_j \\ (\ \text{Lemma 8.2.19}\ ) & \equiv & \Pi_{i\in I} \nu\tilde{a}_i.(P_i^{\neq\nu} \mid R_i) \mid \Pi_{j\in J} G_j\end{array}$$

$$\begin{aligned}
(\text{ Scope extrusion, } \tilde{a} \text{ defined above }) &\equiv \nu\tilde{a}.(\Pi_{i\in I}(P_i^{\neq\nu} \mid R_i)) \mid \Pi_{j\in J} G_j \\
(\text{ Assoc. and commut. }) &\equiv \nu\tilde{a}.(P^{\neq\nu} \mid R) \mid \Pi_{j\in J} G_j,
\end{aligned}$$

where $P^{\neq\nu}$ is defined above as $P^{\neq\nu} := \Pi_{i\in I} P_i^{\neq\nu}$ and $R := \Pi_{i\in I} R_i$.

Consider the reaction $P \to P'$. With Rule (Struct) we get $\nu\tilde{a}.P^{\neq\nu} \to P'$. According to Proposition 2.1.38, there are three possibilities for this reaction. Either a process identifier calls its defining equation, a τ-action is consumed, or two processes communicate. We consider the latter case, where we assume without loss of generality that the first two processes communicate, i.e., $\nu\tilde{a}.P^{\neq\nu} = \nu\tilde{a}.(P_1 \mid P_2 \mid P_{rem}^{\neq\nu})$ with $P_1 = M_1 + a(x).P_1' + N_1$ and $P_2 = M_2 + \overline{a}\langle b\rangle.P_2' + N_2$. Proposition 2.1.38 gives

$$P' \equiv \nu\tilde{a}.(P_1'\{b/x\} \mid P_2' \mid P_{rem}^{\neq\nu}).$$

We compute the standard form $sf(P_1') = \nu\tilde{b}_1.P_1^{\neq\nu}$ and extrude the scope of the names $\tilde{b}_1$. We observe that $\tilde{b}_1 \subseteq bn(P_1') \subseteq bn(P_1)$. Hence, with the disjointness of $bn(P_1)$ and $fn(P_2 \mid P_{rem}^{\neq\nu})$, we can extrude the scope without α-conversion:

$$\begin{aligned}
& & & \nu\tilde{a}.(P_1'\{b/x\} \mid P_2' \mid P_{rem}^{\neq\nu}) \\
(\text{ Standard form }) & & \equiv & \nu\tilde{a}.((\nu\tilde{b}_1.P_1^{\neq\nu})\{b/x\} \mid P_2' \mid P_{rem}^{\neq\nu}) \\
(\text{ Applic. } \sigma) & & = & \nu\tilde{a}.(\nu\tilde{b}_1.(P_1^{\neq\nu}\{b/x\}) \mid P_2' \mid P_{rem}^{\neq\nu}) \\
(\text{ Scope extrusion }) & & \equiv & \nu\tilde{a},\tilde{b}_1.(P_1^{\neq\nu}\{b/x\} \mid P_2' \mid P_{rem}^{\neq\nu}) \\
(\text{ Treat } P_2' \text{ similarly }) & & \equiv & \nu\tilde{a},\tilde{b}_1,\tilde{b}_2.(P_1^{\neq\nu}\{b/x\} \mid P_2^{\neq\nu} \mid P_{rem}^{\neq\nu}) \\
& & =: & P''.
\end{aligned}$$

We prove that Q can mimic the reaction. The argumentation above yields

$$\nu\tilde{a}.(P^{\neq\nu} \mid R) \to \nu\tilde{a},\tilde{b}_1,\tilde{b}_2.(P_1^{\neq\nu}\{b/x\} \mid P_2^{\neq\nu} \mid P_{rem}^{\neq\nu} \mid R) =: Q'.$$

As $Q \equiv \nu\tilde{a}.(P^{\neq\nu} \mid R) \mid \Pi_{j\in J} G_j$ we get $Q \to Q' \mid \Pi_{j\in J} G_j$ with Rule (Par) and Rule (Struct).

We now have to show that $[P'] \preceq_{\mathcal{P}} [Q' \mid \Pi_{j\in J} G_j]$. Process P'' need not be structurally congruent with a single fragment. To apply Lemma 8.2.19 we compute its restricted form, which consists of several fragments: $rf(P'') = \Pi_{i\in I_H} H_i$. Consider such a fragment H. We compute the standard form, $H \equiv \nu\tilde{c}.Q^{\neq\nu}$. Computing the restricted form and the standard form does not change the sequential processes, so $\mathcal{S}(H) = \mathcal{S}(Q^{\neq\nu}) \subseteq \mathcal{S}(P'')$. Furthermore, α-conversion is not required and thus $\tilde{c} \subseteq \tilde{a} \cup \tilde{b}_1 \cup \tilde{b}_2$.

We compute the restricted form $rf(Q') = \Pi_{i\in I_H} H_i' \mid \Pi_{j\in J_H} H_j$. Again this does not change the sequential processes. So there are fragments H_j that consist of sequential processes in R only. For every fragment H_i there is a fragment H_i' which consists of at least the sequential processes $Q^{\neq\nu}$ and the names $\tilde{c}$ but

may additionally contain processes $R^{\neq\nu}$ and names $\tilde{c}_R$ from R. We compute the standard form of such a H' and shrink the scopes of the names $\tilde{c}_R$:

$$H' \equiv \nu\tilde{c}, \tilde{c}_R.(Q^{\neq\nu} \mid R^{\neq\nu}) \equiv \nu\tilde{c}.(Q^{\neq\nu} \mid \nu\tilde{c}_R.R^{\neq\nu}).$$

Lemma 8.2.19 now gives $H_i \preceq_{\mathcal{F}} H_i'$ for all $i \in I_H$. We thus have $P' \equiv P'' \equiv \Pi_{i\in I_H} H_i$ and $Q' \equiv \Pi_{i\in I_H} H_i' \mid \Pi_{j\in J_H} H_j$, which means $[P'] \preceq_{\mathcal{P}} [Q' \mid \Pi_{j\in J} G_j]$. ■

With Proposition 8.2.17, Proposition 8.2.20, and the fact that the reaction relation is image-finite up to structural congruence (Lemma 2.1.39) we conclude that processes of bounded depth have WSTS.

Theorem 8.2.21 (Well-Structure in Bounded Depth)
If $P \in \mathcal{P}_{\mathcal{D}<\infty}$, then $(\mathit{Reach}(P)/_{\equiv}, \rightarrow_{\mathcal{T}}, \preceq_{\mathcal{P}})$ is a WSTS.

8.2.3 Decidability Results

The reaction relation is effectively computable and $\preceq_{\mathcal{P}}$ is decidable. Hence, we can instantiate the decidability result for termination and infinity of states in Proposition 8.2.2 for processes of bounded depth.

Corollary 8.2.22 (Decidability Results)
For a process $P \in \mathcal{P}_{\mathcal{D}<\infty}$ it is decidable whether there is a non-terminating computation starting from $[P]$ and whether $\mathit{Reach}(P)/_{\equiv}$ is infinite.

Figure 8.2 shows the finite reachability tree of a Petri net. To illustrate the decidability result in Corollary 8.2.22, we complement the picture by the finite reachability tree of a process of bounded depth. Recall that the bag data structure in the previous chapter was defined by the equation $BAG(in, out) := in(y).(\overline{out}\langle y\rangle \mid BAG\lfloor in, out\rfloor)$. We observed that the system

$$F_0 = \nu in.(FILL\lfloor in\rfloor \mid \nu out.BAG\lfloor in, out\rfloor)$$

with $FILL(in) := \nu c.\overline{in}\langle c\rangle.FILL\lfloor in\rfloor$ is bounded in depth by two. Hence, its transition system is well-structured by Theorem 8.2.21. We explain the computation of the finite reachability tree $FRT([F_0])$ depicted in Figure 8.3. We give for every reachable fragment the particular fragment in Proposition 8.2.13.

The root of the tree is $[F_0]$. Note that F_0 is structurally congruent with

$$\nu u_0.(FILL\lfloor u_0\rfloor \mid \nu u_1.BAG\lfloor u_0, u_1\rfloor).$$

A call to the defining equation of $FILL$ yields $[F_0] \rightarrow_{\mathcal{T}} [F_1]$ with

$$F_1 \quad = \quad \nu in.(\nu c.\overline{in}\langle c\rangle.FILL\lfloor in\rfloor \mid \nu out.BAG\lfloor in, out\rfloor)$$

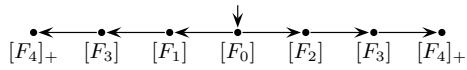


Figure 8.3:
The finite reachability tree of a process of bounded depth. The processes are explained in the text.

$$\equiv \quad \nu u_0.(\nu u_1.\overline{u_0}\langle u_1\rangle.FILL\lfloor u_0\rfloor \mid \nu u_1.BAG\lfloor u_0, u_1\rfloor).$$

We insert a new vertex into the tree and check that $[F_0] \not\preceq_{\mathcal{P}} [F_1]$. Hence, the vertex is labelled by $[F_1]$ and not marked by a $+$. If BAG calls its definition first, we get the reaction $[F_0] \rightarrow_{\mathcal{T}} [F_2]$ with

$$\begin{aligned} F_2 &= \nu in.(FILL\lfloor in\rfloor \mid \nu out.in(y).(\overline{out}\langle y\rangle \mid BAG\lfloor in, out\rfloor)) \\ &\equiv \nu u_0.(FILL\lfloor u_0\rfloor \mid \nu u_1.u_0(y).(\overline{u_1}\langle y\rangle \mid BAG\lfloor u_0, u_1\rfloor)). \end{aligned}$$

Since also $[F_0] \not\preceq_{\mathcal{P}} [F_2]$, a new vertex is inserted that is labelled by $[F_2]$. Consider $[F_1]$, where in the next step the BAG identifier unfolds its definition. This results in $[F_1] \rightarrow_{\mathcal{T}} [F_3]$ with

$$\begin{aligned} F_3 &= \nu in.(\nu c.\overline{in}\langle c\rangle.FILL\lfloor in\rfloor \mid \nu out.in(y).(\overline{out}\langle y\rangle \mid BAG\lfloor in, out\rfloor)) \\ &\equiv \nu u_0.(\nu u_1.\overline{u_0}\langle u_1\rangle.FILL\lfloor u_0\rfloor \mid \nu u_1.u_0(y).(\overline{u_1}\langle y\rangle \mid BAG\lfloor u_0, u_1\rfloor)). \end{aligned}$$

We create a new vertex labelled by $[F_3]$ as $[F_0] \not\preceq_{\mathcal{P}} [F_3]$ and $[F_1] \not\preceq_{\mathcal{P}} [F_3]$. In F_3, the $FILL$ process passes content νc to the bag, which gives $[F_3] \rightarrow_{\mathcal{T}} [F_4]$ with

$$\begin{aligned} F_4 &= \nu in.(FILL\lfloor in\rfloor \mid \nu out.\nu c.(\overline{out}\langle c\rangle \mid BAG\lfloor in, out\rfloor)) \\ &\equiv \nu u_0.(FILL\lfloor u_0\rfloor \mid \nu u_1.(\nu u_2.\overline{u_1}\langle u_2\rangle \mid BAG\lfloor u_0, u_1\rfloor)). \end{aligned}$$

We create a new vertex and check the labels on the path from the root to the new vertex. In fact, $\nu u_1.BAG\lfloor u_0, u_1\rfloor \preceq_{\mathcal{F}} \nu u_1.(\nu u_2.\overline{u_1}\langle u_2\rangle \mid BAG\lfloor u_0, u_1\rfloor)$ and thus $F_0 \preceq_{\mathcal{F}} F_4$. By definition of $\preceq_{\mathcal{P}}$, we conclude $[F_0] \preceq_{\mathcal{P}} [F_4]$ and label the new vertex by $[F_4]_+$. The successors of $[F_2]$ are computed similarly. With Corollary 8.2.22, we have the following result.

Result 8.2.23
The bag process $\nu in.(FILL\lfloor in\rfloor \mid \nu out.BAG\lfloor in, out\rfloor)$ does not terminate since the finite reachability tree contains $[F_4]_+$. Moreover, as $\preceq_{\mathcal{P}}$ is a partial ordering and $[F_0] \preceq_{\mathcal{P}} [F_4]$ but $[F_4] \not\preceq_{\mathcal{P}} [F_0]$ the state space is not finite.

8.3 Undecidability in Bounded Breadth

There are several machine models with the ability to perform arithmetic operations on data variables, which are known to be Turing complete. For the undecidability proofs in this thesis, we use a model introduced by Minsky in [Min67]. Although Minsky called his formalism a *program machine* that operates on *registers*, the model is nowadays well-known under the name of *(2-)counter machines* acting on *counter variables*. In this section, we exploit Turing completeness of counter machines to show Turing completeness for processes of bounded depth and to establish undecidability of structural stationarity, boundedness in depth, and boundedness in breadth. In Section 9.3, counter machines help us prove undecidability of reachability for processes of depth one.

8.3.1 Counter Machines

A 2-counter machine has two *counters* c_1 and c_2 that store arbitrarily large natural numbers and a *finite sequence of labelled instructions* $l : op$. There are two kinds of operations op. The first increments a counter, say c_1, by one and then jumps to the instruction labelled by l':

$$c_1 := c_1 + 1 \text{ goto } l' \tag{8.1}$$

The second operation has the form

$$\text{if } c_1 = 0 \text{ then goto } l'; \text{ else } c_1 := c_1 - 1; \text{ goto } l''; \tag{8.2}$$

It checks counter c_1 for being zero and—if this is the case—jumps to the instruction labelled by l'. If the value of c_1 is positive, the counter is decremented by one and the machine jumps to l''.

More formally, a *(2-)counter machine* is a triple $CM = (c_1, c_2, instr)$, where c_1, c_2 are counters and

$$instr = l_0 : op_0; \ldots, l_n : op_n; l_{n+1} : halt$$

is a finite sequence of the labelled instructions defined above. The sequence ends with operation *halt*, which terminates the execution. The *set of all counter machines* is $\mathcal{CM}$.

To define the operational semantics of a counter machine CM, we require the notion of a state. A *state* of CM is a triple $s = (v_1, v_2, l)$, where $v_i \in \mathbb{N}$ is the current value of counter c_i with $i = 1, 2$ and $l \in \{l_0, \ldots, l_{n+1}\}$ is the label of the operation to be executed next. A finite or infinite sequence of states

$$s_0, s_1, s_2, \ldots$$

is called a *run* of CM. Runs are subject to the following constraints. Initially, the counter values are zero and instruction l_0 is executed, i.e., $s_0 = (0, 0, l_0)$. For every state change s_i, s_{i+1} with $s_i = (v_1, v_2, l)$ the values of the counters and the instruction are changed according to the current operation op with $l : op$. In case op is an increment operation for the first counter as defined in (8.1), we have $s_{i+1} = (v_1 + 1, v_2, l')$, i.e., value v_1 is incremented, v_2 is not changed, and the current label is changed to l'. The decrement operation on c_1 in (8.2) depends on whether $v_1 = 0$ holds. In this case, we jump to instruction l' without modifying the counter values, i.e., $s_{i+1} = (v_1, v_2, l')$. If the content of c_1 is positive, we decrement it and jump to l'', which yields $s_{i+1} = (v_1 - 1, v_2, l'')$. Action *halt* does not change a state.

We say that counter machine CM *terminates* if all its runs are finite. A state $s = (v_1, v_2, l)$ is *reachable* in CM, if there is a run $s_0, s_1, s_2, \ldots$ with $s_i = s$ for some $i \in \mathbb{N}$. Since counter machines are Turing complete, termination and reachability are undecidable.

Theorem 8.3.1 (Theorem 14.1-1 in [Min67])
Counter machines are Turing complete. Hence, for a counter machine CM and a state $s = (v_1, v_2, l)$ it is undecidable whether (1) CM terminates and (2) whether s is reachable in CM.

We shall need undecidability of termination in this section, undecidability of reachability is exploited in Section 9.3 to separate processes of depth one from finite place/transition Petri nets.

8.3.2 From Counter Machines to Bounded Breadth

The construction we present in this section is folklore in concurrency theory and can be found, e.g. in [Mil89]. For the π-Calculus, Amadio and Meyssonnier presented several variants in [AM02].

The idea is to encode counters as list processes as introduced in Section 7.6. The number of list items represents the value of the counter. To model tests for zero, we extend the definition in Example 7.6.1. Every list item and list end has three channels it communicates on—reflecting the three operations on counters. Channel i is used for *increment* operations. It corresponds to the append operation in the previous list model. Communications on channel d *decrement* the counter value. A message on t is a *test for zero*. We first explain the behaviour of a list item. To keep the definition short, we abbreviate the parameters i, d, t by $\tilde{c}$. Similarly, the channels i', d', t' of the following list element are abbreviated by $\tilde{c}'$:

$$LI(\tilde{c}, \tilde{c}') \quad := \quad i.\overline{i}'.LI\lfloor \tilde{c}, \tilde{c}' \rfloor + d.\left(\overline{d}'.LI\lfloor \tilde{c}, \tilde{c}' \rfloor + \overline{t}'.LE\lfloor \tilde{c} \rfloor\right).$$

An increment operation received on channel i is passed to the following list element with the send action $\overline{i'}$. As a list item stands for a positive counter value, the test for zero fails, i.e., a list item does not communicate on channel t. If a list item receives a decrement operation, it contacts the following list element. Since it is unknown whether this is a list item LI or a list end LE, the current list item tries to communicate on both channels $\overline{d'}$ and $\overline{t'}$. If the next element is a list item, it answers the decrement call. A list end receives the $\overline{t'}$ message and as reaction to it terminates. Now the current list item is the last element and therefore calls the defining equation $LE\lfloor\tilde{c}\rfloor$ with

$$LE(\tilde{c}) \quad := \quad t + i.\nu\tilde{c}'.(LI\lfloor\tilde{c},\tilde{c}'\rfloor \mid LE\lfloor\tilde{c}'\rfloor).$$

As explained, the list end terminates on a test for zero. As it represents counter value zero, it does not listen on the decrement channel. If the list end receives an increment operation, it creates new control channels $\tilde{c}' = i', d', t'$ and a new list end process $LE\lfloor\tilde{c}'\rfloor$. The former list end becomes a list item by calling the defining equation $LI\lfloor\tilde{c},\tilde{c}'\rfloor$.

Every instruction $l : op$ of the counter machine is translated into a process identifier K_l whose defining process is determined by the operation op. For the increment operation (8.1) on counter c_1, we get

$$K_l(\tilde{c}_1,\tilde{c}_2) := i_1.K_{l'}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor.$$

The parameters $\tilde{c}_1 = i_1, d_1, t_1$ and $\tilde{c}_2 = i_2, d_2, t_2$ are the control channels of the lists that represent the counters c_1 and c_2, respectively.

The encoding of the decrement operation in (8.2) contains a subtlety. If the test for zero is successful, we delete the list end of counter c_1 and have to create a new one. This yields

$$K_l(\tilde{c}_1,\tilde{c}_2) := t_1.\nu\tilde{c}_1'.\left(K_{l'}\lfloor\tilde{c}_1',\tilde{c}_2\rfloor \mid LE\lfloor\tilde{c}_1'\rfloor\right) + d_1.K_{l''}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor.$$

The instruction $l : \mathit{halt}$ is translated into $K_l(\tilde{c}_1,\tilde{c}_2) := \overline{\mathit{halt}}$. The send action will be helpful later to prove undecidability of boundedness in breadth.

To sum up, the counter machine CM is translated into the process

$$\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!] := \nu\tilde{c}_1.\nu\tilde{c}_2.(LE\lfloor\tilde{c}_1\rfloor \mid LE\lfloor\tilde{c}_2\rfloor \mid K_{l_0}\lfloor\tilde{c}_1,\tilde{c}_2\rfloor)$$

It uses the defining equations we just discussed. To provide an intuition to the encoding of counter values in lists, we give a brief example.

Example 8.3.2 ($\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}} : \mathcal{CM} \to \mathcal{P}_{\mathcal{B}<\infty}$)
Consider the state $(2, 0, l)$ of a counter machine. It is represented by

$$\nu\tilde{c}_1.(\nu\tilde{c}_1'.(LI\lfloor\tilde{c}_1,\tilde{c}_1'\rfloor \mid \nu\tilde{c}_1''.(LI\lfloor\tilde{c}_1',\tilde{c}_1''\rfloor \mid LE\lfloor\tilde{c}_1''\rfloor)) \mid \nu\tilde{c}_2.(LE\lfloor\tilde{c}_2\rfloor \mid K_l\lfloor\tilde{c}_1,\tilde{c}_2\rfloor)).$$

There are two list items in the list for c_1 to represent counter value two. Similarly, the list of counter c_2 consists of a single list end. The label of the current instruction can be deduced from the process identifier K_l. ♦

Example 8.3.2 suggests a tight relationship between the states reachable in a counter machine CM and the processes reachable in its encoding $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!]$. We do not bother with the technicalities here and just remark that the encoding preserves termination.

Proposition 8.3.3
The counter machine CM terminates if and only if $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!]$ terminates.

Like lists in Example 8.3.2, the process representation of a counter machine is bounded in breadth by two. We exploit this observation in the following section to establish undecidability of boundedness in depth and breadth.

Lemma 8.3.4
For every counter machine CM we have $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!] \in \mathcal{P}_{\mathcal{B}<\infty}$.

With proper synchronisation mechanisms (cf. action *act* in Section 4.5.1) the construction can be modified so that the steps of the counter machine coincide with step sequences of the corresponding process of bounded breadth.

Remark 8.3.5
Processes of bounded breadth $\mathcal{P}_{\mathcal{B}<\infty}$ are Turing complete. ♦

8.3.3 Undecidability Results

To show undecidability of structural stationarity for processes of bounded breadth, we reduce the termination problem of counter machines. This works since non-structurally stationary processes do not terminate and for structurally stationary processes we can use the structural semantics to decide termination.

Proposition 8.3.6 (Undecidability of Structural Stationarity)
For a process $P \in \mathcal{P}_{\mathcal{B}<\infty}$ it is undecidable whether P is structurally stationary, i.e., whether $P \in \mathcal{P}_{FG<\infty}$ holds.

Proof
Assume structural stationarity is decidable for processes of bounded breadth using the procedure *isStructurallyStationary*. The algorithm in Table 8.1 then decides termination of a given counter machine CM as follows. We compute the process $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!] \in \mathcal{P}_{\mathcal{B}<\infty}$. If the process is not structurally stationary it

Let CM be the given counter machine;
Compute $\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!]$;
If $\neg$*isStructurallyStationary*$(\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!])$
  return CM does not terminate;
else
  return *terminates*$(\mathcal{N}[\![\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!]]\!])$;

Table 8.1:
Proof of undecidability of structural stationarity. The procedure checks whether a counter machine terminates, under the assumption the procedure *isStructurallyStationary* decides structural stationarity for processes in $\mathcal{P}_{\mathcal{B}<\infty}$. Procedure *terminates* decides termination for finite place/transition Petri nets.

does not terminate according to Lemma 4.1.4. By Proposition 8.3.3 CM does not terminate.

If $\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!]$ is a structurally stationary process, the structural semantics $\mathcal{N}[\![\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!]]\!]$ is a finite place/transition Petri net by Lemma 4.1.2. For finite place/transition Petri nets, termination is decidable, e.g. by inspecting the finite reachability tree (cf. Proposition 8.2.2). Moreover, the net terminates if and only if the counter machine does due to the following equivalence:

$$\begin{array}{rcl} & & CM \text{ terminates} \\ (\text{ Proposition 8.3.3 }) & \Leftrightarrow & \mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!] \text{ terminates} \\ (\text{ Theorem 3.4.3 }) & \Leftrightarrow & \mathcal{N}[\![\mathcal{P}^{\mathcal{B}<\infty}_{\mathcal{CM}}[\![CM]\!]]\!] \text{ terminates.} \end{array}$$

Hence we would be able to decide termination of a counter machine. So, the assumption that structural stationarity is decidable for $\mathcal{P}_{\mathcal{B}<\infty}$ has to be false. ■

For a process of bounded breadth the condition of structural stationarity is equivalent to boundedness in depth according to Theorem 7.2.8. Since structural stationarity is undecidable, boundedness in depth is.

Corollary 8.3.7 (Undecidability of Boundedness in Depth)
Consider a process $P \in \mathcal{P}_{\mathcal{B}<\infty}$. It is undecidable whether $P \in \mathcal{P}_{\mathcal{D}<\infty}$ holds.

To conclude the section, we reduce termination of a counter machines CM

to deciding boundedness in breadth. Again, we exploit the fact that our process representation of counter machines is bounded in breadth. The idea of the reduction is to compose $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!]$ in parallel with

$$halt.\nu a.K_{\mathcal{B}=\infty}\lfloor a \rfloor.$$

The process consumes the $\overline{halt}$ message and starts an execution where fragments of unbounded breadth are generated. The counter machine terminates if and only if the parallel composition is not bounded in breadth.

Lemma 8.3.8 (Undecidability of Boundedness in Breadth)
For a process $P \in \mathcal{P}$ it is undecidable whether P is bounded in breadth, i.e., whether $P \in \mathcal{P}_{\mathcal{B}<\infty}$ holds.

Proof
Consider the counter machine CM and the process

$$\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!] \mid halt.\nu a.K_{\mathcal{B}=\infty}\lfloor a \rfloor$$

with $K_{\mathcal{B}=\infty}(a) = \overline{a}\langle a \rangle \mid K_{\mathcal{B}=\infty}\lfloor a \rfloor$. The counter machine terminates if and only if it reaches its *halt* operation. This is the case if and only if process $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!]$ reaches a process that contains the send action $\overline{halt}$. Since $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!]$ is bounded in breadth, reachability of $\overline{halt}$ is equivalent to unboundedness in breadth for $\mathcal{P}_{\mathcal{CM}}^{\mathcal{B}<\infty}[\![CM]\!] \mid halt.\nu a.K_{\mathcal{B}=\infty}\lfloor a \rfloor$. ■

8.4 Related Work and Conclusion

We investigated the expressiveness of processes of bounded depth and processes of bounded breadth. The main result is that processes of bounded depth have WSTS, and with our instantiation of the framework termination and infinity of states are decidable. This thesis is the first to instantiate the WSTS framework for the π-Calculus. Compatibility with the reaction relation required a non-trivial ordering $\preceq_{\mathcal{P}}$ on the reachable processes. For processes of bounded breadth, we recalled a simulation of counter machines that is folklore in concurrency theory and proves this class Turing complete. Moreover, the encoding shows undecidability of structural stationarity, boundedness in depth, and boundedness in breadth—which is not surprising as all properties are semantical.

Finkel generalised the coverability graph procedure for Petri nets to what he called WSTS [Fin90]. He presented algorithms to decide termination and boundedness problems in the general setting. Abdulla et. al. generalised decidability results of temporal properties and simulation relations for lossy channel systems to their notion of WSTS [AČJT00]. Both definitions were unified by Finkel and Schnoebelen in [FS01].

In [BGZ03, BGZ04, BGZ08], the expressiveness of CCS with recursion, replication, and iteration is investigated. The authors employ the WSTS framework to prove termination decidable for processes with replication—while it is undecidable for those with with recursion. We remark that their ordering is related to our fragment ordering, but it is considerably simpler to establish well-quasi-orderedness in their setting due to the limited expressiveness of CCS (without mobility of names). Recall that we rely on the theory of anchored fragments from Section 7.3 to find flat representations for fragments.

In [BGZ04, BGZ08], it is shown to be decidable whether a process is reachable that communicates on some public channel (the process has a so-called *barb*). It is likely that this property is decidable also for processes of bounded depth. Practically more relevant is the question whether boundedness in breadth is decidable for processes of bounded depth. If this is the case, an implementation of the algorithm inside Petruchio [SM08] can check a system of interest for structural stationarity and hence decide whether it is amenable to verification with the structural semantics. Recall that for processes of bounded depth, structural stationarity is equivalent to boundedness in breadth with Theorem 7.2.8.

Based on a translation of π-Calculus into multisets, orderings on processes defined by multiset containment relations are studied in [EG01]. The main result is that the orderings of Engelfriet and Gelsema characterise structural congruence: $P \preceq Q$ and $Q \preceq P$ holds if and only if $P \equiv Q$. The authors call this equivalence a Cantor Bernstein property. We considered the more intricate wqos, i.e., $\preceq_{\mathcal{P}}$ needed to be well-behaved under reaction. Moreover, we remark that also our fragment ordering satisfies the Cantor Bernstein property and we rely on it when proving $\preceq_{\mathcal{P}}$ to be antisymmetric, i.e., $[P] \preceq_{\mathcal{P}} [Q]$ and $[Q] \preceq_{\mathcal{P}} [P]$ implies $[P] = [Q]$.

In [YBH04, DS06], type systems for the π-Calculus were presented that ensure termination of well-typed processes. We argue that our result is more general in the sense that we do not define a dedicated analysis, but instantiate the WSTS framework for processes of bounded depth and then derive decidability of termination and infinity of states as a corollary of the finite reachability construction. While our approach is restricted to processes of bounded depth, the type systems above apply to any process but—as termination is undecidable—may not succeed in typing it although it terminates. Recently, the authors proposed a so-called dynamic typing system [DHS08]. It statically inspects a process term and annotates it with assertions, which are then checked at runtime. If they are found violated, an exception is raised and the system terminates.

To conclude the section, we remark that heuristics are needed to avoid computing the full finite reachability tree to detect infinity of states or non-termination. Also approximations on the ordering $\preceq_{\mathcal{P}}$ should be developed to prune the finite reachability tree and turn our decidability result into a practical procedure.

9 Structure and Concurrency

Contents

Processes of bounded depth have well-structured transition systems. This insight from the previous chapter allows us to decide termination and infinity of states on the finite reachability tree of such a process. However, this decidability result is unsatisfactory in two respects. It is limited to the two mentioned properties and it does not give us tool support—we first have to implement the algorithm that decides $\preceq_{\mathcal{P}}$. For structurally stationary processes, the situation is more convenient. Our translation allows us to reuse all existing verification approaches and in particular all tools for Petri nets. Therefore, the aim of this chapter is to recover a Petri net translation for processes of bounded depth.[1] The main finding is that for a strictly larger class than structurally stationary processes and within bounded depth, we can still give a translation into Petri nets. Unfortunately, the full class $\mathcal{P}_{\mathcal{D}<\infty}$ turns out more expressive than Petri nets as we prove reachability to be undecidable (already in depth one). The approach to extend the structurally stationary processes is as follows.

[1] Note that the results in the previous chapter do not forbid the existence of such a translation.

In the introduction, we mentioned that dynamically reconfigurable systems are an extension of concurrent systems. For concurrent systems, classical Petri net semantics highlight the interaction between the sequential processes inside the system. Therefore, these semantics are called *concurrency semantics* as opposed to the structural semantics that reflects the connection structure between sequential processes.

We show that the view to processes taken by the structural semantics is orthogonal to the classical view of concurrency semantics. The main result is that both semantics can be combined by typing the restrictions in a process. Restrictions of type one are handled according to the structural semantics and restrictions of type two according to the concurrency semantics. This yields a Petri net translation that extends both, classical concurrency semantics and the structural semantics in Chapter 3, in the following sense. If the process yields a finite Petri net under one of the semantics, then it does so under the mixed semantics. The result is stronger, we show that the process can be typed so that the mixed semantics yields the same Petri net as the original semantics. Conversely, there are processes that are finitely represented under the mixed semantics but neither under the structural nor the concurrency semantics.

To conclude the section, we show that the mixed semantics forms the borderline to place/transition Petri nets. If we leave the class of processes it finitely represents reachability becomes undecidable. Our contributions are as follows:

- We define a concurrency semantics for the π-Calculus. It is the first that satisfies three indispensable quality criteria. It yields a bisimilar transition system and so retrievability holds (fails for [BG95, BG09]). It is expressive as it allows for translating processes with restricted names (fails for [AM02]). It has an intuitive *finiteness characterisation* (fails for [Eng96]). The technical tool that facilitates the definition is the so-called *name-aware transition system* of a process, which manages the use of restricted names.

- We combine the concurrency semantics with the structural semantics in Chapter 3. The idea is to type the restricted names. The definition of the semantics itself again requires a (mixed) normal form on processes. For names of type one, it resembles the restricted form, for names of type two it imitates the standard form. The combined semantics is finite if and only if names of type one form finitely many fragments and only finitely many names of type two are used.

- We prove that this combined semantics is the borderline to place/transition Petri nets in the following sense. If we relax the finiteness requirement and consider a strictly larger class of processes, reachability becomes undecidable. Hence, there can be no translation into place/transition Petri nets. In this sense, the process class is complete.

The chapter is organised as follows. In Section 9.1, we define the name-aware transition system and the concurrency semantics. We combine the latter with the structural semantics in Section 9.2. In Section 9.3, we show that the resulting class of processes cannot be extended since reachability becomes undecidable outside. We conclude with a discussion of related concurrency semantics in Section 9.4.

9.1 A Concurrency Semantics for the π-Calculus

Concurrency semantics reflect the communications between sequential processes. As opposed to the structural semantics, the scopes of restricted names are not important. Therefore, the idea to define concurrency semantics is to treat restricted names as if they were global. If a restricted name that was hidden by a prefix is discovered, a fresh global name is invented. In Section 9.1.1, we define the *name-aware transition system* of the π-Calculus in order to invent fresh names in a systematic way and to trace the names that have been invented so far. These name-aware transition systems are the basis for the definition of our concurrency semantics.

9.1.1 Name-aware Transition System

The *name-aware transition system* of the π-Calculus uses so-called *name-aware processes* of the form $(P^{\neq\nu}, \tilde{a})$, i.e., pairs of processes in standard form $P^{\neq\nu}$ and sets of names $\tilde{a}$. The idea is that in an execution sequence leading to process $(P^{\neq\nu}, \tilde{a})$ the restricted names $\tilde{a}$ have been invented. A second characteristic of name-aware transition systems is that restricted names, which are discovered and have to be added to the set $\tilde{a}$, may not be chosen arbitrarily but are computed. To allow for this computation, we assume that every restricted name carries an index. More precisely, we stick to the following convention.

Convention 9.1.1
Consider a process $P \in \mathcal{P}$ that relies on the defining equations $K_i(\tilde{x}_i) := P_i$ for $1 \leq i \leq m$. We make the following assumptions:

- Every restricted name has the form a_n, i.e., it carries an *index* $n \in \mathbb{N}$. We assume that the indices in process P as well as P_i are 0.
- α-conversion only changes the index n but not the name a of a restricted name a_n.

♦

Definition 9.1.2
For a set $\tilde{a}$ of names that carry indices, we define $\tilde{a}+1 := \{a_{n+1} \mid a_n \in \tilde{a}\}$. ♦

To illustrate the computation of the restricted names, consider the name-aware process $(\tau.\nu b_0.K\lfloor b_0\rfloor, \{b_0, b_1, b_2\})$ for some process identifier K. It consumes a silent action and generates a restricted name b_k. The idea is to take the restricted name b_3 since $k=3$ is the smallest index so that $b_k \notin \{b_0, b_1, b_2\}$.

Definition 9.1.3 (Name-aware Reaction Relation)
The *name-aware reaction relation*, denoted by $\rightarrow^{na}$, is defined as follows:

$$(P^{\neq\nu}, \tilde{a}) \rightarrow^{na} (Q^{\neq\nu}, \tilde{a} \uplus \tilde{b}) \quad :\Leftrightarrow \quad \begin{array}{l}(1)\ P^{\neq\nu} \rightarrow \nu\tilde{b}.Q^{\neq\nu} \text{ in standard form and} \\ (2)\ \forall b_k \in \tilde{b} : k-1 = max\{i \mid b_i \in \tilde{a}\}.\end{array}$$

For the empty set, the maximum is defined by $max\emptyset = 0$, i.e., we choose 0 as index if there is no name $b_i \in \tilde{a}$. Note that disjointness of $\tilde{a}$ and $\tilde{b}$ is always satisfied by constraint (2). For a process $(P^{\neq\nu}, \tilde{a})$, we define the *set of processes reachable by the name-aware reaction relation*:

$$Reach_{na}((P^{\neq\nu}, \tilde{a})) := \{(Q^{\neq\nu}, \tilde{b}) \mid (P^{\neq\nu}, \tilde{a}) \rightarrow^{na*} (Q^{\neq\nu}, \tilde{b})\},$$

where $\rightarrow^{na*}$ is the reflexive and transitive closure of $\rightarrow^{na}$. ♦

The name-aware reaction relation in fact determines the restricted name b_3 for the example process $(\tau.\nu b_0.K\lfloor b_0\rfloor, \{b_0, b_1, b_2\})$ defined above:

$$(\tau.\nu b_0.K\lfloor b_0\rfloor, \{b_0, b_1, b_2\}) \rightarrow^{na} (K\lfloor b_3\rfloor, \{b_0, b_1, b_2\} \uplus \{b_3\}).$$

Like for the reaction relation, we define the *name-aware transition system* by factorising the reachable processes $Reach_{na}((P^{\neq\nu}, \tilde{a}))$ along structural congruence, denoted by $Reach_{na}((P^{\neq\nu}, \tilde{a}))/_{\equiv}$. This means, we take process $([P^{\neq\nu}], \tilde{a})$ instead of $(P^{\neq\nu}, \tilde{a})$ as a state in the transition system. The transition relation is lifted like for the reaction relation.

Definition 9.1.4 (Name-aware Transition System)
The *name-aware transition system* of process $(P^{\neq\nu}, \tilde{a})$ is

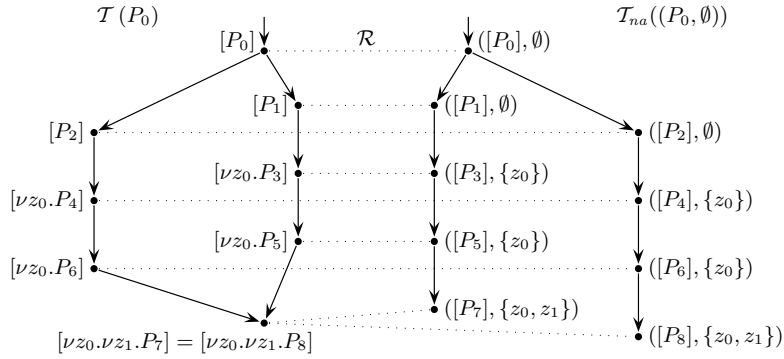
$$\mathcal{T}_{na}((P^{\neq\nu}, \tilde{a})) := (Reach_{na}((P^{\neq\nu}, \tilde{a}))/_{\equiv}, \rightarrow^{na}_{\mathcal{T}}, ([P^{\neq\nu}], \tilde{a})),$$

where $([Q^{\neq\nu}], \tilde{b}) \rightarrow^{na}_{\mathcal{T}} ([R^{\neq\nu}], \tilde{c}) :\Leftrightarrow (Q^{\neq\nu}, \tilde{b}) \rightarrow^{na} (R^{\neq\nu}, \tilde{c})$. ♦

Before we turn to the precise relationship between the name-aware transition system and the original transition system of a π-Calculus process, we give a more elaborate example illustrating name-aware behaviour.

Example 9.1.5 (Name-aware Transition System)
Consider $P_0 = \overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle \mid a(y).K\lfloor a, y\rfloor$ with $K(a,x) := \nu z_0.\overline{x}\langle z_0\rangle \mid a(y).K\lfloor a, y\rfloor$. Two processes $\overline{a}\langle b\rangle$ and $\overline{a}\langle c\rangle$ send on the public channel a. Their messages b and c are received by the process $a(y).K\lfloor a, y\rfloor$, which generates in response a message $\overline{b}\langle z_0\rangle$ and $\overline{c}\langle z_0\rangle$, respectively, where z_0 is a restricted name. After $\overline{a}\langle b\rangle$ and $\overline{a}\langle c\rangle$ have sent their messages, the system deadlocks.

Figure 9.1 gives the original and the name-aware transition system of P_0. Note that in the name-aware transition system the ordering of messages $\overline{a}\langle b\rangle$ and $\overline{a}\langle c\rangle$ leads to different states. Process P_7 contains $\overline{c}\langle z_0\rangle$ and $\overline{b}\langle z_1\rangle$ while process P_8 contains $\overline{b}\langle z_0\rangle$ and $\overline{c}\langle z_1\rangle$. In the original transition system, both processes are represented by $[\nu z_0.\nu z_1.P_7] = [\nu z_0.\nu z_1.P_8]$. ♦



$$P_0 = \overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle \mid a(y).K\lfloor a, y\rfloor$$

$$P_1 = \overline{a}\langle b\rangle \mid K\lfloor a, c\rfloor \qquad P_2 = \overline{a}\langle c\rangle \mid K\lfloor a, b\rfloor$$
$$P_3 = \overline{a}\langle b\rangle \mid \overline{c}\langle z_0\rangle \mid a(y).K\lfloor a, y\rfloor \qquad P_4 = \overline{a}\langle c\rangle \mid \overline{b}\langle z_0\rangle \mid a(y).K\lfloor a, y\rfloor$$
$$P_5 = \overline{c}\langle z_0\rangle \mid K\lfloor a, b\rfloor \qquad P_6 = \overline{b}\langle z_0\rangle \mid K\lfloor a, c\rfloor$$
$$P_7 = \overline{c}\langle z_0\rangle \mid \overline{b}\langle z_1\rangle \mid a(y).K\lfloor a, y\rfloor \qquad P_8 = \overline{b}\langle z_0\rangle \mid \overline{c}\langle z_1\rangle \mid a(y).K\lfloor a, y\rfloor$$

Figure 9.1:
The original transition system $\mathcal{T}(P_0)$ and the name-aware transition system $\mathcal{T}_{na}((P_0,\emptyset))$ of process P_0 in Example 9.1.5. The dotted lines illustrate the bisimilarity between the two, which is established in Lemma 9.1.6.

Lemma 9.1.6 states bisimilarity of the name-aware and the original transition system of a process. We defer the proof until Section 9.1.3

Lemma 9.1.6 (Bisimilarity)
Let $P \in \mathcal{P}$ with $sf(P) = \nu\tilde{a}.P^{\neq\nu}$. The bisimilarity $\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ holds.

9.1.2 Concurrency Semantics

Before we turn to the technicalities, we explain the definition of the concurrency semantics. Starting with the name-aware transition system of a process, we compute two disjoint sets of places. The first set is given by the names $\tilde{a}$ in the reachable name-aware processes $([P^{\neq\nu}], \tilde{a})$. The second set is given by the sequential processes in $P^{\neq\nu}$, more precisely, by the structural congruence classes of sequential processes. We also refer to the first set of places as *name places* and to the second set as *process places*.

Like the set of places, the initial marking is composed of two disjoint markings, $M_0 = M_0^{\mathcal{P}} + M_0^{\mathcal{N}}$. Function $M_0^{\mathcal{P}}$ marks the process places while $M_0^{\mathcal{N}}$ marks the name places. Let $([P_0^{\neq\nu}], \tilde{a}_0)$ be the initial process in the name-aware transition system. The initial marking $M_0^{\mathcal{P}}$ is given by the sequential processes in $P_0^{\neq\nu}$. Marking $M_0^{\mathcal{N}}$ puts a single token on all name places that have index zero—except the names $\tilde{a}_0$. If $a_0 \in \tilde{a}_0$, the name a_1 is marked by one token. In fact, all name places will be safe in the concurrency semantics. We briefly explain the intuition to the construction and marking of name places.

If a name place is marked, the name is the next to be invented in the name-aware transition system. For example, if we have $([\tau.\nu b_0.K\lfloor b_0\rfloor], \{b_0, b_1, b_2\})$ inventing the name b_3, we expect the name place b_3 to be marked. The transition that corresponds to the reaction

$$([\tau.\nu b_0.K\lfloor b_0\rfloor], \{b_0, b_1, b_2\}) \rightarrow_{\mathcal{T}}^{na} ([K\lfloor b_3\rfloor], \{b_0, b_1, b_2, b_3\})$$

moves the token from b_3 to the name place b_4. This means b_4 is the next name to be invented and b_3, b_2, b_1, b_0 have already been invented. Technically, the transitions imitating the name-aware behaviour are defined as follows.

Like for the structural semantics we have two disjoint sets of transitions. The first set contains transitions of the form $([F^e], \tilde{a}, [Q^{\neq\nu}])$ with the requirement that the sequential process F^e reacts to $\nu\tilde{a}.Q^{\neq\nu}$. In the example above, we have a transition $([\tau.\nu b_0.K\lfloor b_0\rfloor], \{b_3\}, [K\lfloor b_3\rfloor])$.

The preset of a transition $([F^e], \tilde{a}, [Q^{\neq\nu}])$ are the process place $[F^e]$ and the name places $\tilde{a}$. Having $\tilde{a}$ in the preset reflects the idea that names can only be invented if their places are marked. The postset is given by the sequential

processes in $Q^{\neq\nu}$ and the names $\tilde{a}+1$. Thus, the transition moves a token from $a_k \in \tilde{a}$ and to a_{k+1} as required in the explanation above.

The second set of transitions models communications between sequential processes. Here we have transitions $([F_1^e \mid F_2^e], \tilde{a}, [Q^{\neq\nu}])$ with the condition that $F_1^e \mid F_2^e$ reacts to $\nu\tilde{a}.Q^{\neq\nu}$. The preset are the sequential processes in $F_1^e \mid F_2^e$. More precisely, we have an arc weighted two from place $[F^e]$ to the transition if $[F^e]$ coincides with $[F_1^e]$ and $[F_2^e]$. In this case, two structurally congruent sequential processes communicate.[2] If place $[F^e]$ is one of the sequential processes, i.e., $[F^e] = [F_1^e]$ or $[F^e] = [F_2^e]$, we draw an arc weighted one from place $[F^e]$ to the transition. In any other case there is no arc, which means the transition represents a reaction the process is not involved in. Like for the first set of transitions, the places $\tilde{a}$ in the preset ensure restricted names are invented in the correct order. The postset is similar as well.

Note that a process $P^{\neq\nu}$ in *restricted form* is a parallel composition of sequential processes, which are elementary fragments. Hence, we can use the fragment function *fg* and the decomposition function *dec* to access the sequential processes and their numbers in $P^{\neq\nu}$. More precisely, $fg\left((P^{\neq\nu}, \tilde{a})\right) := fg\left(P^{\neq\nu}\right)$ and $dec((P^{\neq\nu}, \tilde{a})) := dec(P^{\neq\nu})$. By $nms((P^{\neq\nu}, \tilde{a})) := \tilde{a}$ we refer to the names in a name-aware process.

Definition 9.1.7 (Concurrency Semantics $\mathcal{N}_\mathcal{C} : \mathcal{P} \to \mathcal{PN}$)
Consider a process $P \in \mathcal{P}$ with $sf(P) = \nu\tilde{a}.P^{\neq\nu}$. The *concurrency semantics* is the function $\mathcal{N}_\mathcal{C} : \mathcal{P} \to \mathcal{PN}$ defined in Table 9.1. It assigns to P a Petri net $\mathcal{N}_\mathcal{C}[\![P]\!]$, which we also call *the concurrency semantics of process* P. ♦

We comment on the definition. The inclusion $nms(Reach_{na}((P^{\neq\nu}, \tilde{a}))) + 1 \subseteq S$ ensures that for every name a_k reachable in the name-aware transition system, we also have a_{k+1} as place. Hence, there is always a place a_{n+1} to move the token to even if only the names $a_0, \ldots, a_n$ are invented. This eases the definition of the bisimulation relation in the proof of Lemma 9.1.9. Without these places, bisimilarity still holds but the proof is less elegant.

Like the structural semantics, the concurrency semantics has additional transitions of the form $([F_1^e \mid F_2^e], \tilde{a}, [Q^{\neq\nu} \mid F_2^e])$, where F_1^e reacts to $\nu\tilde{a}.Q^{\neq\nu}$ and thus $F_1^e \mid F_2^e$ reacts to $\nu\tilde{a}.(Q^{\neq\nu} \mid F_2^e)$.[3] We again safely omit them when computing the Petri net. We do not exclude them by definition as this induces additional case distinctions in the proof of bisimilarity in Lemma 9.1.9. To become familiar with the concurrency semantics, we illustrate it on the process in Example 9.1.5. Afterwards, we establish the mentioned bisimilarity.

[2] See the explanation of transitions $([F_1 \mid F_2], [Q])$ in the structural semantics in Section 3.3, where a similar phenomenon occurs when structurally congruent fragments communicate.
[3] For the structural semantics, Figure 3.3 illustrates the additional transitions.

$$S := fg\left(Reach_{na}((P_0^{\neq\nu}, \tilde{a}_0))\right)/_{\equiv} \cup nms(Reach_{na}((P_0^{\neq\nu}, \tilde{a}_0))) \cup nms(Reach_{na}((P_0^{\neq\nu}, \tilde{a}_0))) + 1.$$

Let the fragments $[F^e], [F_1^e]$, and $[F_2^e]$ as well as the names $\tilde{a}$ be places in S:

$$T := \{([F^e], \tilde{a}, [Q^{\neq\nu}]) \mid F^e \to \nu\tilde{a}.Q^{\neq\nu} \in \mathcal{P}_{sf}\} \cup \{([F_1^e \mid F_2^e], \tilde{a}, [Q^{\neq\nu}]) \mid F_1^e \mid F_2^e \to \nu\tilde{a}.Q^{\neq\nu} \in \mathcal{P}_{sf}\}.$$

Consider transitions $t = ([F^e], \tilde{a}, [Q^{\neq\nu}])$, $t' = ([F_1^e \mid F_2^e], \tilde{a}, [Q^{\neq\nu}])$, and places a and $[G^e]$. To avoid case distinctions, we let a condition $a \in \tilde{a}$ or $a \in (\tilde{a} + 1)$ yield 1 if it is satisfied and 0 otherwise:

$$W([G^e], t) := (dec(F^e))([G^e]) \qquad W([G^e], t') := (dec(F_1^e \mid F_2^e))([G^e])$$
$$W(a, t) := a \in \tilde{a} \qquad W(a, t') := a \in \tilde{a}$$
$$W(t, [G^e]) := (dec(Q^{\neq\nu}))([G^e]) \qquad W(t, [G^e]) := (dec(Q^{\neq\nu}))([G^e])$$
$$W(t, a) := a \in (\tilde{a} + 1) \qquad W(t', a) := a \in (\tilde{a} + 1).$$

The initial marking is $M_0 := M_0^{\mathcal{P}} + M_0^{\mathcal{N}}$. Since name places receive a single token, we define $M_0^{\mathcal{N}}$ by the set of marked places:

$$M_0^{\mathcal{P}} := dec(P^{\neq\nu})$$
$$M_0^{\mathcal{N}} := (\{a_0 \in S\} \setminus \tilde{a}_0) \cup (\tilde{a}_0 + 1).$$

Table 9.1:
Definition of $\mathcal{N}_{\mathcal{C}}[\![P_0]\!] = (S, T, W, M_0)$ for process P_0 with $sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$.

Example 9.1.8 (Concurrency Semantics)
Consider $P_0 = \overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle \mid a(y).K\lfloor a, y\rfloor$ with $K(a, x) := \nu z_0.\overline{x}\langle z_0\rangle \mid a(y).K\lfloor a, y\rfloor$. The concurrency semantics is depicted in Figure 9.2. To begin with, we compute the set of process places. It is given by the fragments reachable in the name-aware transition system. With respect to Figure 9.1, we get

$$fg\,(Reach_{na}((P_0, \emptyset)))/_{\equiv} = (fg\,(P_0) \cup \ldots \cup fg\,(P_8))/_{\equiv}.$$

Computing the fragment function gives the set of process places in Figure 9.2. Only the names z_0 and z_1 are reachable in the name-aware transition system. With $nms(Reach_{na}((P^{\neq\nu}, \tilde{a}))) + 1 \subseteq S$ we also add the name place z_2.

Initially, the processes $\overline{a}\langle b\rangle$, $\overline{a}\langle c\rangle$, and $a(x).K\lfloor a, x\rfloor$ are present. Hence the corresponding places are marked by $M_0^{\mathcal{P}}$. The marking $M_0^{\mathcal{N}}$ is given by

$$(\{z_0 \in S\} \setminus \emptyset) \cup (\emptyset + 1) = \{z_0\}.$$

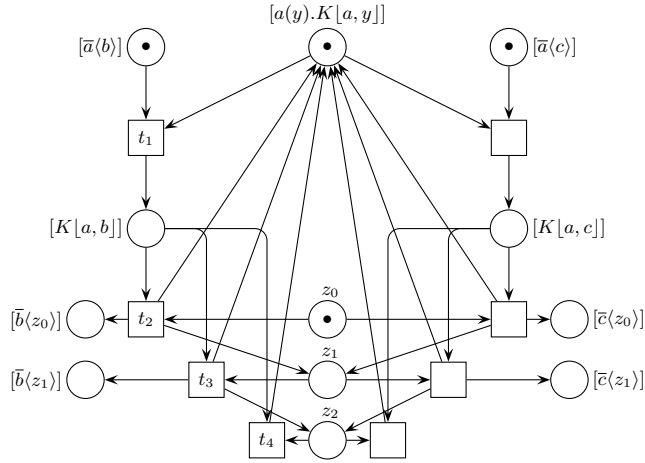


Figure 9.2:
The concurrency semantics of process P_0 in Example 9.1.5. The construction as well as transition names are explained in Example 9.1.8.

We now explain the transition set. The processes $\overline{a}\langle b\rangle$ and $a(x).K\lfloor a, x\rfloor$ react to $K\lfloor a, b\rfloor$ without inventing restricted names. This behaviour is reflected by transition $t_1 = ([\overline{a}\langle b\rangle \mid a(y).K\lfloor a, y\rfloor], \emptyset, [K\lfloor a, b\rfloor])$, which takes a token from $[\overline{a}\langle b\rangle]$ and from $[a(y).K\lfloor a, y\rfloor]$ and puts a token on $K\lfloor a, b\rfloor$.

The process $K\lfloor a, b\rfloor$ reacts to $\nu z_k.(\overline{b}\langle z_k\rangle \mid a(y).K\lfloor a, y\rfloor)$ for any k. Since we only have z_0, z_1, and z_2 as name places, we get three different transitions:

$$\begin{aligned} t_2 &= ([K\lfloor a, b\rfloor], \{z_0\}, [\overline{b}\langle z_0\rangle \mid a(y).K\lfloor a, y\rfloor]) \\ t_3 &= ([K\lfloor a, b\rfloor], \{z_1\}, [\overline{b}\langle z_1\rangle \mid a(y).K\lfloor a, y\rfloor]) \\ t_4 &= ([K\lfloor a, b\rfloor], \{z_2\}, [\overline{b}\langle z_2\rangle \mid a(y).K\lfloor a, y\rfloor]). \end{aligned}$$

Transition t_2 invents the restricted name z_0. It removes the token from z_0 and adds a token to z_1. If $\overline{a}\langle c\rangle$ and $a(x).K\lfloor a, x\rfloor$ communicate first, z_0 has already been invented. In this case, no token on z_0 but a token on z_1 is present. Therefore, transition t_2 is disabled, which forbids reinventing z_0. Instead, transition t_3 allows for generating z_1. The transition moves the token from z_1 to z_2. Like t_2 it consumes a token from $[K\lfloor a, b\rfloor]$ and creates a token on $[a(y).K\lfloor a, y\rfloor]$, but while t_2 puts a token on $[\overline{b}\langle z_0\rangle]$, transition t_3 marks $[\overline{b}\langle z_1\rangle]$.

We expect transition t_4 to behave similar to t_2 and t_3, i.e., to put a token on z_3 and $[\overline{b}\langle z_3\rangle]$. But since the weight function W is determined by the reachable

processes and names and as z_3 and $\overline{b}\langle z_3\rangle$ are not reachable, they are not considered. The name-aware transition system in Figure 9.1 shows that no name z_2 is generated, therefore transition t_4 is never enabled.

To conclude the example, we observe that the set of places determines the transition set, as shown by the computation of the dead transition t_4. A similar phenomenon occurred in Example 3.3.8, where a dead transition (also named t_4) was created in the structural semantics. ♦

The transition system of the concurrency semantics $\mathcal{N}_C[\![P_0]\!]$ in Figure 9.2 is isomorphic to the name-aware transition system of P_0 in Figure 9.1. We conjecture that this isomorphism holds in general. In the following Lemma 9.1.9, we only establish bisimilarity between the transition systems. The reason is that we combine the result with Lemma 9.1.6 to prove bisimilarity between a process and its concurrency semantics in Theorem 9.1.10. Example 9.1.5 now shows that the bisimilarity in Lemma 9.1.6 cannot be strengthened to isomorphism. Hence, combining the lemmas only yields bisimilarity for the composed relation.

Lemma 9.1.9
Consider $P \in \mathcal{P}$ with $sf(P) = \nu\tilde{a}.P^{\neq\nu}$. We have $\mathcal{T}(\mathcal{N}_C[\![P]\!]) \approx \mathcal{T}_{na}((P^{\neq\nu}, \tilde{a}))$.

We defer the proof until Section 9.1.3 and continue with the first main result. The transition system of a process and that of its concurrency semantics are bisimilar. Moreover, the bisimulation allows us to compute the reachable processes from the markings. Note that this is the first concurrency semantics for the π-Calculus that deals with restricted names and yields a bisimilar and finite place/transition Petri net. We discuss the problems with related approaches in Section 9.4.

Theorem 9.1.10 (Full Retrievability)
For every process $P \in \mathcal{P}$ we have $\mathcal{T}(\mathcal{N}_C[\![P]\!]) \approx \mathcal{T}(P)$.

Proof
Consider process $P_0 \in \mathcal{P}$ with $sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$. By Lemma 9.1.6 we have

$$\mathcal{T}_{na}((P_0^{\neq\nu}, \tilde{a}_0)) \approx \mathcal{T}(P_0).$$

With Lemma 9.1.9

$$\mathcal{T}(\mathcal{N}_C[\![P_0]\!]) \approx \mathcal{T}_{na}((P_0^{\neq\nu}, \tilde{a}_0)).$$

By transitivity of bisimilarity we derive

$$\mathcal{T}(\mathcal{N}_C[\![P_0]\!]) \approx \mathcal{T}(P_0).$$

We give the bisimulation relation that connects $\mathcal{T}(\mathcal{N}_C[\![P_0]\!])$ and $\mathcal{T}(P_0)$. It is the composition of the two bisimulations in the Lemmas 9.1.9 and 9.1.6:

$$\begin{aligned}\mathcal{R} := \big\{(M^{\mathcal{P}} + M^{\mathcal{N}}, [\nu\tilde{a}.P^{\neq\nu}]) \mid P^{\neq\nu} \;&\equiv\; \Pi_{[F^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([F^e])}F^e \\ \tilde{a} \;&=\; \{a_i \in S \mid M^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\}\big\}.\end{aligned}$$

Hence, we can compute from a given marking the corresponding reachable process and full retrievability holds. ■

Our second main result is a finiteness characterisation for the concurrency semantics. The semantics is finite if and only if the process generates finitely many restricted names. We call those processes *restriction bounded*. Since in the original transition system unused restrictions can be removed by $\nu a.P \equiv P$ if $a \notin fn(P)$, we again rely on the name-aware transition system to define restriction boundedness.

Definition 9.1.11 (Restriction Boundedness)
Consider a process $P \in \mathcal{P}$ with $sf(P) = \nu\tilde{a}.P^{\neq\nu}$. We call P *restriction bounded* if there is a finite set of names $\tilde{m}$ so that for every process $(Q, \tilde{b})$ reachable in the name-aware transition system $\mathcal{T}\left(P^{\neq\nu}, \tilde{a}\right)$ the inclusion $\tilde{b} \subseteq \tilde{m}$ holds, i.e.,

$$\exists\tilde{m} \subseteq \mathcal{N} : \forall(Q, \tilde{b}) \in Reach_{na}(P^{\neq\nu}, \tilde{a}) : \tilde{b} \subseteq \tilde{m} \text{ and } |\tilde{m}| < \infty.$$

♦

If the process is not restriction bounded, clearly the concurrency semantics is infinite as every restricted name leads to a place. The main task is to show the reverse. If we assume that a bounded number of restricted names is used in all reachable processes, then the concurrency semantics is a finite Petri net. Theorem 9.1.12 shows that this is the case. The proof again uses the theory of derivatives and in particular applies Proposition 4.2.2.

Theorem 9.1.12 (Finiteness Characterisation)
For any process $P \in \mathcal{P}$, the concurrency semantics $\mathcal{N}_C[\![P]\!]$ is finite if and only if P is restriction bounded.

Proof
$\Leftarrow$ Consider a restriction bounded process $P_0 \in \mathcal{P}$ with $sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$. Let $\tilde{m}$ be the finite set of names that contains the names $\tilde{b}$ of all processes $(Q^{\neq\nu}, \tilde{b})$ reachable in the name-aware transition system.

Like for the structural semantics, we observe that the concurrency semantics is finite if and only if its set of places is. For the set of name places finiteness holds by definition of restriction boundedness. To establish finiteness of the set

of process places, consider $(Q^{\neq\nu}, \tilde{b}) \in \mathit{Reach}_{na}((P_0, \tilde{a}_0))$. We show that for any fragment $F^e \in \mathit{fg}\left(Q^{\neq\nu}\right)$ we have

$$F^e \equiv R\sigma \text{ with } R \in \mathit{derivatives}(P_0) \text{ and } \sigma : \mathit{fn}(R) \to \tilde{m} \cup \mathit{fn}(P_0).$$

This shows that the process places are included in

$$\{R\sigma \mid R \in \mathit{derivatives}(P_0) \text{ and } \sigma : \mathit{fn}(R) \to \tilde{m} \cup \mathit{fn}(P_0)\}/_{\equiv},$$

which is a finite set. By Lemma 4.2.5, the set of derivatives is finite. Moreover, $\mathit{fn}(R)$, $\mathit{fn}(P_0)$, and $\tilde{m}$ are finite. Hence, there are finitely many mappings from $\mathit{fn}(R)$ into $\tilde{m} \cup \mathit{fn}(P_0)$.

To prove that any fragment $F^e \in \mathit{fg}\left(Q^{\neq\nu}\right)$ is structurally congruent with a process $R\sigma$, we observe that $\nu\tilde{b}.Q^{\neq\nu}$ is reachable from P_0 by the bisimilarity in Lemma 9.1.6. Without loss of generality, we assume the process to be in standard form.[4] Proposition 4.2.2 shows that

$$\nu\tilde{b}.Q^{\neq\nu} \equiv \nu\tilde{c}.R^{\neq\nu},$$

where the latter process is in standard form and moreover $R^{\neq\nu} = \Pi_{i\in I} R_i\sigma_i$ with $R_i \in \mathit{derivatives}(P_0)$ and $\sigma_i : \mathit{fn}(R_i) \to \mathit{fn}(P_0) \cup \tilde{c}$ holds. With Corollary 2.1.32 we strengthen the relationship to standard equivalence, $\nu\tilde{b}.Q^{\neq\nu} \equiv_{sf} \nu\tilde{c}.R^{\neq\nu}$. By Lemma 2.1.33 there is a substitution $\sigma : \nu\tilde{c} \to \nu\tilde{b}$ so that

$$R^{\neq\nu}\sigma = \Pi_{i\in I} R_i\sigma_i\sigma \equiv_{sf} Q^{\neq\nu}.$$

By definition of standard equivalence, for every sequential process $F^e \in \mathit{fg}\left(Q^{\neq\nu}\right)$ there is a structurally congruent process $R_i\sigma_i\sigma$. Since $\nu\tilde{b}.(R^{\neq\nu}\sigma) \equiv \nu\tilde{b}.Q^{\neq\nu}$ the process is reachable. Hence, $\mathit{fn}(\nu\tilde{b}.(R^{\neq\nu}\sigma)) \subseteq \mathit{fn}(P_0)$ by Lemma 2.1.37. We conclude $\mathit{fn}(R_i\sigma_i\sigma) \subseteq \mathit{fn}(P_0) \cup \tilde{b} \subseteq \mathit{fn}(P_0) \cup \tilde{m}$, which shows that the codomain of $\sigma_i\sigma$ is correct.

$\Rightarrow$ If process $P \in \mathcal{P}$ is not restriction bounded, then for every finite set of names $\tilde{m}$ a reachable process $(Q, \tilde{b})$ exists in the name-aware transition system where $\tilde{b}$ contains a name outside $\tilde{m}$. Since the set of name places in $\mathcal{N}_{\mathcal{C}}[\![P]\!]$ is the union of all sets $\tilde{b}$, it is not finite and so the concurrency semantics is not. ■

Corollary 9.1.13 (Restriction Boundedness and Bounded Depth)
If $P \in \mathcal{P}$ is restriction bounded, then P is bounded in depth, i.e., $P \in \mathcal{P}_{\mathcal{D}<\infty}$.

Proof
Assume $\tilde{m} \subseteq \mathcal{N}$ is the finite set of names that includes all names reachable in the name-aware transition system. Then $|\tilde{m}|$ bounds the depth of all reachable fragments with the bisimilarity in Theorem 9.1.10. ■

[4] If this is not the case, we remove the names from $\nu\tilde{b}$ that are not in $\mathit{fn}(Q^{\neq\nu})$.

To illustrate that the classes of structurally stationary and restriction bounded processes are incomparable, we consider two examples.

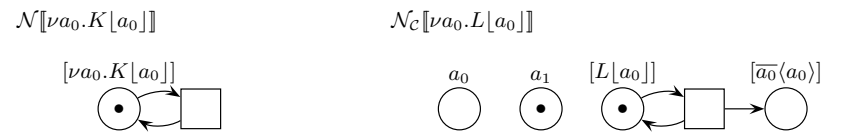


Figure 9.3:
Structurally stationary and restriction bounded processes are incomparable, $\nu a_0.K\lfloor a_0\rfloor$ defined in Example 9.1.14 is not restriction bounded but structurally stationary. Its structural semantics is depicted to the left. Process $\nu a_0.L\lfloor a_0\rfloor$ in the same example is restriction bounded but not structurally stationary. Its concurrency semantics is depicted to the right.

Example 9.1.14 (Structurally Stationary and Restriction-bounded Processes)
The following two processes separate the classes. The corresponding semantics that finitely represent them are depicted in Figure 9.3.

1. In Example 4.3.8, we considered $\nu a_0.K\lfloor a_0\rfloor$ with $K(x) := \nu b_0.K\lfloor b_0\rfloor$. It has the reaction sequence

   $$\nu a_0.K\lfloor a_0\rfloor \to \nu b_0.K\lfloor b_0\rfloor \to \nu b_1.K\lfloor b_1\rfloor \to \dots$$

   We observed that the process is structurally stationary, but as it generates a new restricted name in every reaction it is not restriction bounded.

2. In Section 7.2, we defined $\nu a_0.L\lfloor a_0\rfloor$ with $L(x) := \overline{x}\langle x\rangle \mid L\lfloor x\rfloor$, which generates processes that send on the restricted channel a_0. The sequence

   $$\nu a_0.L\lfloor a_0\rfloor \to \nu a_0.(\overline{a_0}\langle a_0\rangle \mid L\lfloor a_0\rfloor) \to \nu a_0.(\overline{a_0}\langle a_0\rangle \mid \overline{a_0}\langle a_0\rangle \mid L\lfloor a_0\rfloor) \to \dots$$

   forms fragments of arbitrary breadth and therefore the process is not structurally stationary with Theorem 7.2.8. It is restriction bounded as only one restricted name is generated in all execution sequences.

Note that both processes are bounded in depth. ⧫

Before we turn to the combination of the presented concurrency semantics with our structural semantics, we give the proofs missing in this section.

9.1.3 Proofs of Lemma 9.1.6 and Lemma 9.1.9

We first show bisimilarity between the original and the name-aware transition system of a process.

Proof (of Lemma 9.1.6)
Consider process $P_0 \in \mathcal{P}$ with $sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$. The index of $\tilde{a}_0$ indicates that by Convention 9.1.1 all names in the initial process carry zero as index. As bisimulation relation we choose

$$\mathcal{R} := \{ \Big(([P^{\neq\nu}], \tilde{a}), [\nu\tilde{a}.P^{\neq\nu}]\Big) \quad | \quad ([P^{\neq\nu}], \tilde{a}) \in Reach_{na}(([P_0^{\neq\nu}], \tilde{a}_0))/_{\equiv} \text{ and } [\nu\tilde{a}.P^{\neq\nu}] \in Reach(P_0)/_{\equiv} \}.$$

To show that $\mathcal{R}$ is indeed a bisimulation, we establish the following:

(1) The relation $\mathcal{R}$ connects the initial states, i.e., $(([P_0^{\neq\nu}], \tilde{a}_0), [\nu\tilde{a}_0.P^{\neq\nu}]) \in \mathcal{R}$.

(2) For all $(([P^{\neq\nu}], \tilde{a}), [P]) \in \mathcal{R}$ two implications hold.
   (2.1) If $([P^{\neq\nu}], \tilde{a}) \rightarrow_{\mathcal{T}}^{na} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ then there is $[Q] \in Reach(P_0)/_{\equiv}$ so that $[P] \rightarrow_{\mathcal{T}} [Q]$ and $Q \equiv \nu\tilde{a}.\nu\tilde{b}.Q^{\neq\nu}$.
   (2.2) If $[P] \rightarrow_{\mathcal{T}} [Q]$ then there is $([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b}) \in Reach_{na}(([P_0^{\neq\nu}], \tilde{a}_0))/_{\equiv}$ with $([P^{\neq\nu}], \tilde{a}) \rightarrow_{\mathcal{T}}^{na} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ and $Q \equiv \nu\tilde{a}.\nu\tilde{b}.Q^{\neq\nu}$.

Case (1) With Lemma 2.1.28, we have $P_0 \equiv sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$. Hence,

$$\Big(([P_0^{\neq\nu}, \tilde{a}_0]), [P_0]\Big) \in \mathcal{R}.$$

Case (2.1) Consider the name-aware reaction $([P^{\neq\nu}], \tilde{a}) \rightarrow_{\mathcal{T}}^{na} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$. By definition of $\rightarrow^{na}$ this means

$$P^{\neq\nu} \rightarrow \nu\tilde{b}.Q^{\neq\nu}.$$

We prove that $[P] = [\nu\tilde{a}.P^{\neq\nu}]$ can imitate the reaction. Since $P^{\neq\nu} \rightarrow \nu\tilde{b}.Q^{\neq\nu}$ we get $\nu\tilde{a}.P^{\neq\nu} \rightarrow \nu\tilde{a}.\nu\tilde{b}.Q^{\neq\nu}$ with Rule (Res). We choose $[Q] = [\nu\tilde{a}.\nu\tilde{b}.Q^{\neq\nu}]$ to get

$$\Big(([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b}), [Q]\Big) \in \mathcal{R}.$$

Case (2.2) Let $[P] = [\nu\tilde{a}.P^{\neq\nu}] \rightarrow_{\mathcal{T}} [Q]$, which means $\nu\tilde{a}.P^{\neq\nu} \rightarrow Q$. Without loss of generality, let $\nu\tilde{a}.P^{\neq\nu}$ be in standard form.[5] By Proposition 2.1.38 there are three possibilities for the reaction $\nu\tilde{a}.P^{\neq\nu} \rightarrow Q$. We consider the consumption of τ-actions, which we assume to take place in the first sequential process. Proposition 2.1.38 yields

$$\nu\tilde{a}.P^{\neq\nu} \quad = \quad \nu\tilde{a}.(M + \tau.R + N \mid P_{rem}^{\neq\nu})$$

[5] Otherwise we remove the names in $\tilde{a}$ that are not free in $P^{\neq\nu}$.

$$Q \quad \equiv \quad \nu\tilde{a}.(R \mid P_{rem}^{\neq\nu}).$$

To find a corresponding name-aware reaction, we observe that

$$P^{\neq\nu} = M + \tau.R + N \mid P_{rem}^{\neq\nu} \quad \rightarrow \quad R \mid P_{rem}^{\neq\nu}.$$

Let the standard form of R be $sf(R) = \nu\tilde{x}.R^{\neq\nu}$. We change the indices of $\tilde{x}$ as required for a name-aware reaction, i.e., we rename $\tilde{x}$ to $\tilde{b}$ so that for all $b_k \in \tilde{b}$ we have $k = max\{i \mid b_i \in \tilde{a}\} + 1$ or $k = 0$ if there is no $b_i \in \tilde{a}$. To extrude the scope of $\tilde{b}$, we argue that $\tilde{b} \cap fn(P_{rem}^{\neq\nu}) = \emptyset$. Since $\nu\tilde{a}.(R \mid P_{rem}^{\neq\nu})$ is reachable from P_0, we have $fn(P_{rem}^{\neq\nu}) \subseteq \tilde{a} \cup fn(P_0)$ with Lemma 2.1.37. Since the names $\tilde{x}$ are bound in P_0, they are disjoint with $fn(P_0)$ and so the names $\tilde{b}$ where the indices are changed are disjoint with $fn(P_0)$. By construction $\tilde{b} \cap \tilde{a} = \emptyset$. Scope extrusion of $\tilde{b}$ is valid:

$$\begin{aligned}
& & & R \mid P_{rem}^{\neq\nu} \\
&(\ R \equiv sf(R) = \nu\tilde{x}.R^{\neq\nu}, \text{ Lemma 2.1.28 }) & \equiv \quad & \nu\tilde{x}.R^{\neq\nu} \mid P_{rem}^{\neq\nu} \\
&(\ \alpha\text{-conversion }) & \equiv \quad & \nu\tilde{b}.(R^{\neq\nu}\{\tilde{b}/\tilde{x}\}) \mid P_{rem}^{\neq\nu} \\
&(\text{ Scope extrusion }) & \equiv \quad & \nu\tilde{b}.(R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu}).
\end{aligned}$$

With Rule (Struct) we derive

$$P^{\neq\nu} = M + \tau.R + N \mid P_{rem}^{\neq\nu} \rightarrow \nu\tilde{b}.(R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu}).$$

By construction of $\tilde{b}$, the requirements of $\rightarrow^{na}$ are satisfied and we have the name-aware reaction

$$(P^{\neq\nu}, \tilde{a}) \rightarrow^{na} (R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu}, \tilde{a} \uplus \tilde{b}).$$

Furthermore, with $R \mid P_{rem}^{\neq\nu} \equiv \nu\tilde{b}.(R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu})$ it holds

$$Q \equiv \nu\tilde{a}.(R \mid P_{rem}^{\neq\nu}) \equiv \nu\tilde{a}.\nu\tilde{b}.(R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu}).$$

We thus have

$$\Big(([R^{\neq\nu}\{\tilde{b}/\tilde{x}\} \mid P_{rem}^{\neq\nu}], \tilde{a} \uplus \tilde{b}), [Q]\Big) \in \mathcal{R}.$$

This shows that $\mathcal{R}$ is a bisimulation relation and concludes the proof. ■

We now turn to the bisimilarity between the name-aware transition system and the concurrency semantics.

Proof (of Lemma 9.1.9)
Consider $P_0 \in \mathcal{P}$ with $sf(P_0) = \nu\tilde{a}_0.P_0^{\neq\nu}$. As bisimulation relation we choose

$$\begin{aligned}
\mathcal{R} := \Big\{ \Big(M^{\mathcal{P}} + M^{\mathcal{N}}, ([P^{\neq\nu}], \tilde{a})\Big) \mid P^{\neq\nu} \ &\equiv \ \Pi_{[F^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([F^e])}F^e \text{ and} \\
\tilde{a} \ &= \ \{a_i \in S \mid M^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\}\Big\}.
\end{aligned}$$

To prove that $\mathcal{R}$ is a bisimulation, we establish the following:

(1) The initial states are related by $\mathcal{R}$, i.e., $(M_0, ([P_0^{\neq\nu}], \tilde{a}_0)) \in \mathcal{R}$.

(2) For all $\left(M^{\mathcal{P}} + M^{\mathcal{N}}, ([P^{\neq\nu}], \tilde{a})\right) \in \mathcal{R}$ we have two implications.

(2.1) If $M^{\mathcal{P}} + M^{\mathcal{N}} \to N^{\mathcal{P}} + N^{\mathcal{N}}$ then there is a process $([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ with $([P^{\neq\nu}], \tilde{a}) \to_T^{na} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ and $(N^{\mathcal{P}} + N^{\mathcal{N}}, ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})) \in \mathcal{R}$.

(2.2) If $([P^{\neq\nu}], \tilde{a}) \to_T^{na} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ then there is a marking $N^{\mathcal{P}} + N^{\mathcal{N}}$ so that $M^{\mathcal{P}} + M^{\mathcal{N}} \to N^{\mathcal{P}} + N^{\mathcal{N}}$ and $(N^{\mathcal{P}} + N^{\mathcal{N}}, ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})) \in \mathcal{R}$.

Case (1) Since $P_0^{\neq\nu}$ is in restricted form, we can apply the elementary equivalence in Lemma 3.3.4:

$$\begin{array}{rcl} & & P_0^{\neq\nu} \\ (\text{ Lemma 3.3.4 }) & \equiv & \Pi_{[F^e] \in supp(dec(P_0^{\neq\nu}))} \Pi^{(dec(P_0^{\neq\nu}))([F^e])} F^e \\ (\text{ Def. } M_0^{\mathcal{P}} := dec(P^{\neq\nu}) \text{ }) & \equiv & \Pi_{[F^e] \in supp(M_0^{\mathcal{P}})} \Pi^{M_0^{\mathcal{P}}([F^e])} F^e. \end{array}$$

To show that the sets of names coincide, we observe

$$\begin{array}{rcl} & & \{a_i \in S \mid M_0^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\} \\ (\text{ Def. } M_0^{\mathcal{N}} \text{ }) & = & \{a_0 \mid a_1 \in (\tilde{a}_0 + 1)\} \\ (\text{ Def. } \tilde{a}_0 + 1 \text{ }) & = & \tilde{a}_0. \end{array}$$

This shows $(M_0^{\mathcal{P}} + M_0^{\mathcal{N}}, ([P_0^{\neq\nu}], \tilde{a}_0)) \in \mathcal{R}$.

Case (2.1) Consider the transition $M \to N$ with $M = M^{\mathcal{P}} + M^{\mathcal{N}}$ and $N = N^{\mathcal{P}} + N^{\mathcal{N}}$. The following equivalences hold by definition:

$$\begin{array}{rl} & M \to N \\ \Leftrightarrow & \exists t \in T : M(s) \geq W(s,t) \text{ for all } s \in {}^\bullet t \text{ and} \\ & \quad N(s) = M(s) - W(s,t) + W(t,s) \text{ for all } s \in S. \end{array}$$

We consider the case $t = ([F^e], \tilde{b}, [Q^{\neq\nu}])$, the case of two communicating sequential processes is similar but technically more involved. The choice of t gives

$$\begin{array}{rcl} M^{\mathcal{P}}([F^e]) & \geq & W([F^e], ([F^e], \tilde{b}, [Q^{\neq\nu}])) = 1 \text{ and} \\ M^{\mathcal{N}}(b) & \geq & W(b, ([F^e], \tilde{b}, [Q^{\neq\nu}])) = 1 \text{ for all } b \in \tilde{b}. \end{array}$$

The process $P^{\neq\nu}$ contains a fragment F^e, since

$$\begin{array}{rcl} & & P^{\neq\nu} \\ (\text{ Def. } \mathcal{R} \text{ }) & \equiv & \Pi_{[G^e] \in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])} G^e \\ (\text{ Assoc. and comm. } \mid \text{ }) & \equiv & F^e \mid \Pi_{[G^e] \in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e]) - W([G^e], t)} G^e. \end{array}$$

The last equivalence is correct for two reasons. We showed $M^{\mathcal{P}}([F^e]) \geq 1$ above, hence we can extract a parallel composition of F^e. Furthermore, by definition of W we get $W([G^e], t) = (dec([F^e]))([G^e]) = 1$, if $G^e \equiv F^e$ and 0 otherwise. Hence, the term $W([G^e], t)$ compensates for the extraction of F^e.

By definition of the transition set we have the reaction $F^e \to \nu\tilde{b}.Q^{\neq\nu}$. We derive the following reaction of $P^{\neq\nu}$:

$$\begin{aligned}
& P^{\neq\nu} \\
(\text{ Argumentation above }) \quad &\equiv \quad F^e \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e \\
(\text{ Rule (Par) }) \quad &\to \quad \nu\tilde{b}.Q^{\neq\nu} \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e \\
(\text{ Scope extrusion }) \quad &\equiv \quad \nu\tilde{b}.(Q^{\neq\nu} \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e).
\end{aligned}$$

Scope extrusion in the last step requires some consideration. We have

$$fn(\Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e) \subseteq fn(P^{\neq\nu}) \subseteq \tilde{a} \cup fn(\nu\tilde{a}_0.P_0^{\neq\nu}),$$

where the latter inclusion relies on $\nu\tilde{a}.P^{\neq\nu} \in Reach(\nu\tilde{a}_0.P_0^{\neq\nu})$ and Lemma 2.1.37. That $\nu\tilde{a}.P^{\neq\nu}$ is reachable from $\nu\tilde{a}_0.P_0^{\neq\nu}$ holds with the previous Lemma 9.1.6 and the fact that $([P^{\neq\nu}], \tilde{a}) \in Reach_{na}(([P_0^{\neq\nu}], \tilde{a}_0))/_{\equiv}$.

We now show that $\tilde{b}$ is disjoint with $\tilde{a}$ as well as $fn(\nu\tilde{a}_0.P_0^{\neq\nu})$. For the latter, this holds by the fact that $\tilde{b}$ is a set of bound names and bound names are disjoint with free names. To see disjointness with $\tilde{a}$, we observe that names in $\tilde{b}$ are marked while by definition of $\mathcal{R}$ the names in $\tilde{a}$ are dominated by marked names, i.e., if $a_i \in \tilde{a}$ then there is a name a_k with $M(a_k) = 1$ and $k > i$. Hence, $\tilde{a} \cap \tilde{b} = \emptyset$ and the scope extrusion is correct.

To establish the name-aware reaction

$$(P^{\neq\nu}, \tilde{a}) \to^{na} (Q^{\neq\nu} \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e, \tilde{a} \uplus \tilde{b}),$$

we have to show that the index of $b_k \in \tilde{b}$ is correct, i.e., $k-1 = max\{i \mid b_i \in \tilde{a}\}$:

$$\begin{aligned}
& max\{i \mid b_i \in \tilde{a}\} \\
(\text{ Def. } \mathcal{R} \text{ }) \quad &= \quad max\{i \mid M^{\mathcal{N}}(b_l) = 1 \text{ for some } b_l \text{ with } l > i\} \\
(\text{ } l = k \text{ since } t \text{ fires }) \quad &= \quad max\{i \mid M^{\mathcal{N}}(b_k) = 1 \text{ with } k > i\} \\
&= \quad k - 1.
\end{aligned}$$

This proves the reaction. We now establish

$$(N^{\mathcal{P}} + N^{\mathcal{N}}, ([Q^{\neq\nu} \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})} \Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)} G^e], \tilde{a} \uplus \tilde{b})) \in \mathcal{R}.$$

To begin with, we show that the process is given by the marking $N^{\mathcal{P}}$. This holds (a) by the elementary equivalence in Lemma 3.3.4, (b) by definition of W, (c) by

associativity and commutativity of parallel composition, and (d) by the equation defining the marking N with $M[t\rangle N$:

$$
\begin{array}{lcl}
 & & Q^{\neq\nu} \mid \Pi_{[G^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)}G^e \\
(\text{ a }) & \equiv & \Pi_{[G^e]\in supp(dec(Q^{\neq\nu}))}\Pi^{(dec(Q^{\neq\nu}))([G^e])}G^e \mid \\
 & & \Pi_{[G^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)}G^e \\
(\text{ b }) & \equiv & \Pi_{[G^e]\in supp(dec(Q^{\neq\nu}))}\Pi^{W(t,[G^e]}G^e \mid \\
 & & \Pi_{[G^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)}G^e \\
(\text{ c }) & \equiv & \Pi_{[G^e]\in(supp(dec(Q^{\neq\nu}))\cup supp(M^{\mathcal{P}}))}\Pi^{M^{\mathcal{P}}([G^e])-W([G^e],t)+W(t,[G^e])}G^e \\
(\text{ d }) & \equiv & \Pi_{[G^e]\in supp(N^{\mathcal{P}})}\Pi^{N^{\mathcal{P}}([G^e])}G^e.
\end{array}
$$

The following equations show that the set of names is correct. They hold by definition of $\mathcal{R}$ and W:

$$
\begin{array}{lcl}
 & & \tilde{a} \uplus \tilde{b} \\
= & & \{a_i \mid M^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\} \uplus \{b_k \mid W(b_k, t) = 1)\} \\
= & & \{a_i \mid M^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\} \uplus \{b_k \mid N^{\mathcal{N}}(b_{k+1}) = 1 = M^{\mathcal{N}}(b_k)\} \\
= & & \{a_i \mid N^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\}.
\end{array}
$$

The inclusion $\subseteq$ in the last equation holds since only the indices of names $b \in \tilde{b}$ have been increased. To see that $\supseteq$ holds, we distinguish two cases. If $N^{\mathcal{N}}(a_k) = M^{\mathcal{N}}(a_k)$ then all a_i with $i < k$ are in $\{a_i \mid M^{\mathcal{N}}(a_k) = 1 \text{ with } k > i\}$. If $N^{\mathcal{N}}(b_{k+1}) = 1$ but $M^{\mathcal{N}}(b_k) = 1$ then the name is included in the second set. Hence, the two sets are equal and $\mathcal{R}$ holds.

Case (2.2) Consider the name-aware reaction

$$([P^{\neq\nu}], \tilde{a}) \rightarrow^{na}_{\mathcal{T}} ([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b}).$$

We show that there is a transition enabled in M that can imitate the behaviour.

By definition of name-aware reactions, we have $P^{\neq\nu} \rightarrow \nu\tilde{b}.Q^{\neq\nu}$. By Proposition 2.1.38, there are three possible sources for this reaction. We consider consumption of τ-actions in the first component, i.e., $P^{\neq\nu} = M + \tau.R + N \mid P^{\neq\nu}_{rem}$. Communications between sequential processes and calls to identifiers are similar but require handling of substitutions. The main point in all proofs is to show that exactly the $\tilde{b}$ places are marked. An application of Proposition 2.1.38 yields

$$
\begin{array}{rcl}
P^{\neq\nu} & = & M + \tau.R + N \mid P^{\neq\nu}_{rem} \\
\nu\tilde{b}.Q^{\neq\nu} & \equiv & R \mid P^{\neq\nu}_{rem}.
\end{array}
$$

We compute the standard form $sf(R) = \nu\tilde{x}.R^{\neq\nu}$ and rename $\tilde{x}$ to a set of names $\tilde{y}$ that is disjoint with the free names in $P^{\neq\nu}_{rem}$. Then we extrude the scope of $\tilde{y}$:

$$\begin{aligned} & & R \mid P^{\neq\nu}_{rem} \\ (\ R \equiv sf(R) = \nu\tilde{x}.R^{\neq\nu}, \text{ Lemma 2.1.28 }) & \equiv & \nu\tilde{x}.R^{\neq\nu} \mid P^{\neq\nu}_{rem} \\ (\ \alpha\text{-conversion }) & \equiv & \nu\tilde{y}.(R^{\neq\nu}\{\tilde{y}/\tilde{x}\}) \mid P^{\neq\nu}_{rem} \\ (\ \text{Scope extrusion }) & \equiv & \nu\tilde{y}.(R^{\neq\nu}\{\tilde{y}/\tilde{x}\} \mid P^{\neq\nu}_{rem}). \end{aligned}$$

Since $\nu\tilde{b}.Q^{\neq\nu}$ as well as $\nu\tilde{y}.(R^{\neq\nu}\{\tilde{y}/\tilde{x}\} \mid P^{\neq\nu}_{rem})$ are both in standard form and structurally congruent, Corollary 3.3.6 yields

$$\nu\tilde{b}.Q^{\neq\nu} \equiv_{sf} \nu\tilde{y}.(R^{\neq\nu}\{\tilde{y}/\tilde{x}\} \mid P^{\neq\nu}_{rem}).$$

By Lemma 2.1.33 there is a substitution $\sigma : \tilde{y} \to \tilde{b}$ so that

$$Q^{\neq\nu} \equiv_{sf} R^{\neq\nu}\{\tilde{y}/\tilde{x}\}\sigma \mid P^{\neq\nu}_{rem}\sigma.$$

Abbreviating $\{\tilde{y}/\tilde{x}\}\sigma$ by σ' and referring to $M + \tau.R + N$ by F^e, we get

$$F^e \to R \equiv \nu\tilde{x}.R^{\neq\nu} \equiv \nu\tilde{b}.(R^{\neq\nu}\sigma'),$$

where the second congruence α-converts the names $\tilde{x}$. Rule (Struct) gives the reaction $F^e \to \nu\tilde{b}.(R^{\neq\nu}\sigma')$.

Since $([P^{\neq\nu}], \tilde{a})$ is a reachable process, $([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ is. Hence, the names $\tilde{b}$ as well as $[F^e]$ are places. By construction of the transition set, we have a transition $t = ([F^e], \tilde{b}, [R^{\neq\nu}\sigma'])$. To see that t is enabled, note that by definition of $\mathcal{R}$

$$F^e \mid P^{\neq\nu}_{rem} = P^{\neq\nu} \equiv \Pi_{[G^e]\in supp(M^{\mathcal{P}})}\Pi^{M^{\mathcal{P}}([G^e])}G^e.$$

Thus, place $[F^e]$ is marked by at least one token. To see that the name places $\tilde{b}$ are marked, we observe that $b_k \in \tilde{b}$ implies

$$k - 1 = max\{i \mid b_i \in \tilde{a}\} = max\{i \mid M(b_l) = 1 \text{ with } l > i\}.$$

The first equation holds by definition of name-aware reactions, the second by definition of $\mathcal{R}$. We conclude $l = k$, which means $M(b_k) = 1$ and transition t is enabled. Its firing yields marking N with

$$N(s) = M(s) - W(s,t) + W(t,s) \text{ for all } s \in S.$$

The proof that $\mathcal{R}$ relates N and $([Q^{\neq\nu}], \tilde{a} \uplus \tilde{b})$ is similar to the one in (2.1). The difference is that we need to show that $\tilde{b}+1$ as well as $supp(dec(R^{\neq\nu}\sigma'))$ are included in the set of places. For $\tilde{b}+1$ this holds with the fact that $\tilde{b}$ is reachable by a name-aware reaction and thus by $nms(Reach_{na}((P^{\neq\nu}_0, \tilde{a}_0))) + 1 \subseteq S$ we also have $\tilde{b} + 1 \subseteq S$. For the support, we argue as follows:

$$supp(dec(R^{\neq\nu}\sigma'))$$

$$\begin{aligned}
(\text{ Lemma 3.3.2 }) &= fg\left(R^{\neq\nu}\sigma'\right)/_{\equiv} \\
(\text{ Def. } fg\text{ }) &\subseteq fg\left(R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma\right)/_{\equiv} \\
(\text{ See below }) &= fg\left(Q^{\neq\nu}\right)/_{\equiv} \\
(\text{ Def. } S\text{ }) &\subseteq S.
\end{aligned}$$

For the second equality, note that with Corollary 3.3.6 $R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma \equiv Q^{\neq\nu}$ implies $dec(R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma) = dec(Q^{\neq\nu})$. Hence, we get

$$\begin{aligned}
& fg\left(R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma\right)/_{\equiv} \\
(\text{ Lemma 3.3.2 }) &= supp(dec(R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma)) \\
(\ dec(R^{\neq\nu}\sigma' \mid P^{\neq\nu}_{rem}\sigma) = dec(Q^{\neq\nu})\ ) &= supp(dec(Q^{\neq\nu})) \\
(\text{ Lemma 3.3.2 }) &= fg\left(Q^{\neq\nu}\right)/_{\equiv}.
\end{aligned}$$

This concludes the proof of bisimilarity. ■

Remark 9.1.15
Both bisimilarity proofs only rely on the fact that $P^{\neq\nu}$ is in restricted form. Hence, they hold with an arbitrary process P^{rf} in restricted form instead of $P^{\neq\nu}$. As a consequence, the correctness of the mixed semantics defined in the following section reduces to the correctness proof for the concurrency semantics we just gave. ♦

9.2 Combining Structural and Concurrency Semantics

The idea to combine the structural and the concurrency semantics is to have two types of restricted names. Depending on the type, a name is handled according to the concurrency semantics or according to the structural semantics. Technically, restricted names νa *may* carry a tag $\mathcal{C}$.[6] Tagged names $\nu a^{\mathcal{C}}$ are translated by the concurrency semantic while names νa without tag are handled by the structural semantics. This means tagged names $\nu a^{\mathcal{C}}$ yield name places in the Petri net while untagged names form fragments, which are used like process places in the concurrency semantics. This combined semantics raises two problems. (1) It requires the definition of a name-aware transition system to make the invention of tagged names explicit. (2) For every reachable process, the fragments induced by untagged names need to be computed. The solution to both problems is a

[6] We still assume that restricted names carry indices.

normal form for processes, which combines the standard and the restricted form. It is introduced in the following Section 9.2.1, the combined semantics itself is presented in Section 9.2.2.

We aim at a clear but informal introduction of the combined semantics. Similarly, the statements of bisimilarity and finiteness are given informally. As pointed out in the previous section, the proofs for the concurrency semantics should still hold in the new setting. We refuse to formalise the statements as this causes overhead which distracts from the idea and does not provide new insights. Important definitions are illustrated on the example of a bag data structure introduced in Section 7.6 and also considered in Section 8.2.3. We briefly restate it using the tagged names defined above.

Example 9.2.1 (The Bag Data Structure)
The bag data structure together with a process that fills it forms the system $\nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.(FILL\lfloor in_0^{\mathcal{C}}\rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor)$, where

$$\begin{aligned} BAG(in_0^{\mathcal{C}}, out_0^{\mathcal{C}}) &:= in_0^{\mathcal{C}}(y).(\overline{out_0^{\mathcal{C}}}\langle y\rangle \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor) \\ FILL(in_0^{\mathcal{C}}) &:= \nu val.\overline{in_0^{\mathcal{C}}}\langle val\rangle.FILL\lfloor in_0^{\mathcal{C}}\rfloor. \end{aligned}$$

In Section 7.6, we observed that the $out_0^{\mathcal{C}}$ channel is shared by arbitrarily many processes. Therefore, we type it by $\mathcal{C}$ to treat it by the concurrency semantics. Conversely, we observed that arbitrarily many instances of the value *val* are created. By omitting the type, the name is handled like in the structural semantics. Since the structural semantics does not need indices, we omit them. For the name $in_0^{\mathcal{C}}$ we are free to tag it. Also the untagged name *in* would lead to a finite Petri net representation in the mixed semantics. ♦

9.2.1 Mixed Normal Form

The idea of the mixed normal form is to *maximise* the scopes of tagged names $a^{\mathcal{C}}$ and to *minimise* the scopes of untagged names a. The result is a process where the tagged names surround a process in restricted form, i.e., a process $P^{mf} = \nu\tilde{a}^{\mathcal{C}}.P^{rf}$ so that P^{rf} only contains untagged names. We call P^{mf} a process *in mixed normal form*. More formally, the class is defined by

$$P^{mf} ::= P^{rf} \mid \nu a^{\mathcal{C}}.P^{mf},$$

where $a^{\mathcal{C}} \in fn(P^{mf})$ and P^{rf} is a process in restricted form with $a^{\mathcal{C}} \cap arn(P^{rf}) = \emptyset$, i.e., there is no tagged name in the active restrictions of P^{rf}. The *set of all processes in mixed normal form* is $\mathcal{P}_{mf}$.

Example 9.2.2 (Mixed Normal Form)
Consider the following processes P and P^{mf}:

$$\begin{aligned} P &= \nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.(FILL\lfloor in_0^C \rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}} \rfloor \mid \overline{out_0^{\mathcal{C}}}\langle val\rangle) \\ P^{mf} &= \nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.(FILL\lfloor in_0^C \rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}} \rfloor \mid \nu val.\overline{out_0^{\mathcal{C}}}\langle val\rangle). \end{aligned}$$

Both, P and P^{mf} are in mixed normal form. The process $\nu val.P$, where an untagged outermost restriction is added, is not in mixed normal form. ♦

The mixed normal form $P^{mf} = \nu\tilde{a}^{\mathcal{C}}.P^{rf}$ combines standard form $P^{sf} = \nu\tilde{a}.P^{\neq\nu}$ and restricted form P^{rf}. Like in the standard form, tagged names surround a parallel composition of processes, with the difference that $P^{\neq\nu}$ uses elementary fragments $P^{\neq\nu} = \Pi_{i\in I}F_i^e$ while $P^{rf} = \Pi_{i\in I}F_i$ composes proper fragments. Compared with the restricted form, the mixed normal form additionally restricts free names in P^{rf}.

Before we introduce the name-aware transition system of a tagged process, we make sure that every process is structurally congruent to a process in mixed normal form. To this end, we use a function $mf : \mathcal{P} \to \mathcal{P}_{mf}$ that combines the definitions of $sf : \mathcal{P} \to \mathcal{P}_{sf}$ and $rf : \mathcal{P} \to \mathcal{P}_{rf}$. We explain the recursive definition in detail. Empty sums $M^{=\mathbf{0}} = \mathbf{0} + \ldots + \mathbf{0}$ are represented by $\mathbf{0}$, which are empty parallel compositions of fragments and thus in mixed normal form, $\mathbf{0} = \Pi_{i\in\emptyset}F_i \in \mathcal{P}_{rf} \subseteq \mathcal{P}_{mf}$. Since any non-empty sequential process is in restricted form, it is also in mixed normal form. We have

$$mf(M^{=\mathbf{0}}) := M^{=\mathbf{0}} \qquad\qquad mf(F^e) := F^e.$$

For the parallel composition $P \mid Q$, we recursively compute the mixed normal forms of P and Q. Let them be $mf(P) = \nu\tilde{a}_P^{\mathcal{C}}.P^{rf} \neq \mathbf{0} \neq \nu\tilde{a}_Q^{\mathcal{C}}.Q^{rf} = mf(Q)$. Like the standard form, the mixed normal form $mf(P \mid Q)$ extrudes the scopes of both sets of names:

$$mf(P \mid Q) := \nu\tilde{a}_P^{\mathcal{C}}.\nu\tilde{a}_Q^{\mathcal{C}}.(P^{rf} \mid Q^{rf}).$$

Since P^{rf} and Q^{rf} are parallel compositions of fragments, the process $P^{rf} \mid Q^{rf}$ is in restricted form as well.

Restricted names are treated according to whether they are tagged. Tagged names are handled like in the standard form, i.e., for $\nu a^{\mathcal{C}}.P$ with $a^{\mathcal{C}} \in fn(P)$ we define

$$mf(\nu a^{\mathcal{C}}.P) := \nu a^{\mathcal{C}}.mf(P).$$

For a process $\nu a.P$ with an untagged name $a \in fn(P)$, we recursively compute the mixed normal form $mf(P) = \nu\tilde{a}^{\mathcal{C}}.P^{rf}$. This singles out the tagged names $\tilde{a}^{\mathcal{C}}$ in P. We commute νa with $\nu\tilde{a}^{\mathcal{C}}$, which yields $\nu\tilde{a}^{\mathcal{C}}.\nu a.P^{rf}$. Let $P^{rf} = \Pi_{i\in I}F_i$. Similar to the function rf, the indices I_a of those fragments F_i are determined that have

a as a free name. The scope of a is then restricted accordingly, which yields $\nu\tilde{a}^{\mathcal{C}}.(\nu a.(\Pi_{i\in I_a} F_i) \mid \Pi_{i\in I\setminus I_a} F_i)$. By construction $\nu a.(\Pi_{i\in I_a} F_i)$ is a fragment and thus $\nu a.(\Pi_{i\in I_a} F_i) \mid \Pi_{i\in I\setminus I_a} F_i$ is in restricted form. To sum up, we get

$$mf(\nu a.P) := \nu a^{\mathcal{C}}.\left(\nu a.(\Pi_{i\in I_a} F_i) \mid \Pi_{i\in I\setminus I_a} F_i\right).$$

Example 9.2.3 ($mf : \mathcal{P} \to \mathcal{P}_{mf}$)
In Example 9.2.2, we observed that the process $\nu val.P$ with

$$P \quad = \quad \nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.(FILL\lfloor in_0^{C}\rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor \mid \overline{out_0^{\mathcal{C}}}\langle val\rangle)$$

is not in mixed normal form. We compute the process $mf(\nu val.P)$. To begin with, we recursively determine the mixed normal form of P, which is P itself, i.e., $mf(P) = P$. Commuting val over the tagged names yields

$$\nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.\nu val.(FILL\lfloor in_0^{C}\rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor \mid \overline{out_0^{\mathcal{C}}}\langle val\rangle).$$

We determine the fragments in $FILL\lfloor in_0^{C}\rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor \mid \overline{out_0^{\mathcal{C}}}\langle val\rangle$ that use the name val. This is $\overline{out_0^{\mathcal{C}}}\langle val\rangle$. Restricting the scope of val gives

$$mf(\nu val.P) = \nu in_0^{\mathcal{C}}.\nu out_0^{\mathcal{C}}.(\nu val.\overline{out_0^{\mathcal{C}}}\langle val\rangle \mid FILL\lfloor in_0^{C}\rfloor \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}}\rfloor).$$

The process is in mixed normal form. ♦

Although we have not checked the details, it should be easy to show $mf(P) \equiv P$ and $mf(P) \in \mathcal{P}_{mf}$. The proof should be similar to the proofs of Lemma 2.1.28 for the function sf and Lemma 3.2.7 for the function rf. Moreover, it should be straightforward to adapt the standard equivalence $\equiv_{sf}$ to a mixed equivalence $\equiv_{mf}$, which characterises structural congruence over the mixed normal form, i.e., $P \equiv Q$ if and only if $mf(P) \equiv_{mf} mf(Q)$. Again the proof should be similar to the proofs of Proposition 2.1.31 and Proposition 3.2.10. The propositions characterise structural congruence by standard equivalence $\equiv_{sf}$ and by restricted equivalence $\equiv_{rf}$, respectively. The relation $\equiv_{mf}$ differs from $\equiv_{sf}$ in two aspects. The rules containing processes $P^{\neq\nu}$ are replaced by similar rules where processes in restricted form P^{rf} are used. Moreover, the rule

$$\nu\tilde{a}.(M^{\neq\mathbf{0}} \mid P^{\neq\nu}) \equiv_{sf} \nu\tilde{a}.(N^{\neq\mathbf{0}} \mid P^{\neq\nu}), \text{ where } M^{\neq\mathbf{0}} \equiv N^{\neq\mathbf{0}}$$

is replaced by

$$\nu\tilde{a}^{\mathcal{C}}.(F \mid P^{rf}) \equiv_{sf} \nu\tilde{a}^{\mathcal{C}}.(G \mid P^{rf}), \text{ where } F \equiv G.$$

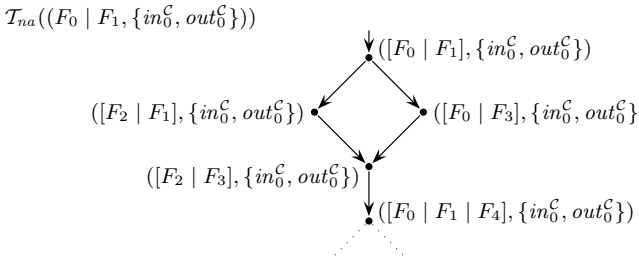
We observe that the latter rule also mimics the replacement of fragments by structurally congruent ones as allowed for in the restricted equivalence $\equiv_{rf}$. We now turn to the definition of the mixed semantics.

9.2.2 Mixed Semantics

Like the concurrency semantics, the combined semantics relies on a name-aware transition system to keep track of the tagged restricted names that are used. With the mixed normal form $\nu\tilde{a}^{\mathcal{C}}.P^{rf}$ in mind, we define a new form of *name-aware processes*, namely $(P^{rf}, \nu\tilde{a}^{\mathcal{C}})$ where the active restrictions in P^{rf} are untagged. For these name-aware processes, the *name-aware reaction relation* $\rightarrow^{na}$ is adapted accordingly. The only difference to Definition 9.1.3 is the use of the mixed normal form instead of the standard form:

$$(P^{rf}, \tilde{a}^{\mathcal{C}}) \rightarrow^{na} (Q^{rf}, \tilde{a}^{\mathcal{C}} \uplus \tilde{b}^{\mathcal{C}}) :\Leftrightarrow (1)\ P^{rf} \rightarrow \nu\tilde{b}^{\mathcal{C}}.Q^{rf} \text{ in mixed normal form and } (2)\ \forall b_k^{\mathcal{C}} \in \tilde{b}^{\mathcal{C}} : k - 1 = max\{i \mid b_i^{\mathcal{C}} \in \tilde{a}^{\mathcal{C}}\}.$$

It is straightforward to modify the definition of the name-aware transition system as well. Without change of notation, let it be $\mathcal{T}_{na}((P^{rf}, \nu\tilde{a}^{\mathcal{C}}))$. Figure 9.4 illustrates it on the bag data structure. As discussed in Section 9.1.3, the bisimilarity



$F_0 = FILL\lfloor in_0^C \rfloor$ $\quad$ $F_1 = BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}} \rfloor$

$F_2 = \nu val.\overline{in_0^{\mathcal{C}}}\langle val\rangle.FILL\lfloor in_0^C \rfloor$ $\quad$ $F_3 = in_0^{\mathcal{C}}(y).(\overline{out_0^{\mathcal{C}}}\langle y\rangle \mid BAG\lfloor in_0^{\mathcal{C}}, out_0^{\mathcal{C}} \rfloor)$

$F_4 = \nu val.\overline{out_0^{\mathcal{C}}}\langle val\rangle$

Figure 9.4:
The name-aware transition system of the bag process in Example 9.2.1.

of the name-aware and the original transition system only makes use of the fact that $P^{\neq\nu}$ is a parallel composition of fragments. Hence, we claim it holds for the adapted definition as well, i.e., if $mf(P_0) = \nu\tilde{a}_0^{\mathcal{C}}.P_0^{rf}$ then

$$\mathcal{T}_{na}((P_0^{rf}, \tilde{a}_0^{\mathcal{C}})) \approx \mathcal{T}(P_0).$$

Also the mixed semantics is an adaptation of the concurrency semantics. We use the new name-aware transition system to define the places. They are now proper fragments like in the structural semantics and no longer elementary fragments. Similarly, the set of transitions is changed. They are of the form

$$([F], \tilde{a}^{\mathcal{C}}, [Q^{rf}]) \quad \text{and} \quad ([F_1 \mid F_2], \tilde{a}^{\mathcal{C}}, [Q^{rf}])$$

with the condition that $F \rightarrow^{na} \nu\tilde{a}^{\mathcal{C}}.Q^{rf}$ so that the latter process is in mixed normal form. The weight function is changed accordingly. Formally, the *mixed semantics* is a function $\mathcal{N}_{\mathcal{M}} : \mathcal{P} \to \mathcal{PN}$ obtained from Table 9.1 by dropping the superscripts of elementary fragments and replacing processes $P^{\neq\nu}$ and $Q^{\neq\nu}$ by P^{rf} and Q^{rf}. Figure 9.5 gives the mixed semantics for the bag data structure. It is worth noting that neither the structural nor the concurrency semantics can finitely represent the process.

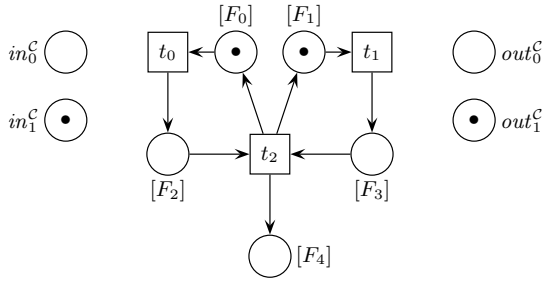


Figure 9.5:
The mixed semantics of the bag data structure. The fragments F_0 to F_4 are defined in Figure 9.4. Obviously, the fragments form the places like in the structural semantics and the tagged names yield name places like in the concurrency semantics. The transitions are $t_0 = ([F_0], \emptyset, [F_2])$, $t_1 = ([F_1], \emptyset, [F_3])$, and $t_3 = ([F_2 \mid F_3], \emptyset, [F_0 \mid F_1 \mid F_4])$.

We also discussed that the proof of bisimilarity in Lemma 9.1.9 does not distinguish between elementary fragments and proper fragments. So it still holds for the combined semantics and with $mf(P_0) = \nu\tilde{a}_0^{\mathcal{C}}.P_0^{rf}$ we have:

$$\mathcal{T}\left(\mathcal{N}_{\mathcal{M}}[\![P_0]\!]\right) \approx \mathcal{T}_{na}((P_0^{rf}, \tilde{a}_0^{\mathcal{C}})).$$

Combining both bisimilarities, we obtain bisimilarity for the mixed semantics,

$$\mathcal{T}_{na}(\mathcal{N}_{\mathcal{M}}[\![P_0]\!]) \approx \mathcal{T}\left(P_0\right),$$

and like for the structural and concurrency semantics, the processes can be reconstructed from the markings.

It is interesting to observe that for processes *without tagged names* the mixed semantics degenerates to the structural semantics. The absence of tagged names in the process leads to absence of name places in the semantics. Hence, transitions do not create names and have the form $([F], \emptyset, [Q^{rf}])$ or $([F_1 \mid F_2], \emptyset, [Q^{rf}])$. They can be identified with the transitions in the structural semantics. In case *all names are tagged* in the process under consideration, the mixed semantics corresponds to the concurrency semantics. This follows from the fact that the mixed normal form coincides with the standard form for these processes. Hence, the places in the mixed semantics are sequential processes.

Remark 9.2.4 (Conservative Extension)
Consider process $P \in \mathcal{P}$. If the process does not use tagged names, the mixed semantics coincides with the structural semantics, i.e., $\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}[\![P]\!]$. If the process only uses tagged names, the mixed semantics coincides with the concurrency semantics, i.e., $\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{C}}[\![P]\!]$. ♦

We conclude the section with a finiteness characterisation for the mixed semantics. According to Lemma 4.1.2, the structural semantics is finite exactly if the process is structurally stationary, i.e., there are finitely many fragments the restricted form of every reachable process consists of. Theorem 9.1.12 states that the concurrency semantics is finite if and only if the process is restriction bounded, i.e., the name-aware transition system generates finitely many restricted names. We prove the mixed semantics to be finite if and only if

- the untagged names form finitely many fragments and
- only finitely many tagged names are generated.

Technically, a process P_0 with $mf(P_0) = \nu\tilde{a}_0^{\mathcal{C}}.P_0^{rf}$ is *mixed-bounded*, if there is a finite set of fragments $\{F_1, \ldots, F_n\}$ and a finite set of names $\tilde{m}^{\mathcal{C}}$ so that for every reachable process $(Q^{rf}, \tilde{a}^{\mathcal{C}})$ we have

$$\forall F \in fg\left(Q^{rf}\right) : \exists i : F \equiv F_i \text{ and } \tilde{a}^{\mathcal{C}} \subseteq \tilde{m}^{\mathcal{C}}.$$

We first show that the mixed semantics $\mathcal{N}_{\mathcal{M}}[\![P]\!]$ is finite if P is mixed-bounded. Note that—like for the structural and for the concurrency semantics—finiteness of the set of places yields finiteness of the mixed semantics. Finiteness of the set of name places immediately follows from the definition of mixed boundedness, which requires a finite set of names $\tilde{m}^{\mathcal{C}}$ including the names $\tilde{a}^{\mathcal{C}}$ of all reachable name-aware processes. Likewise, finiteness of the set of fragment places is ensured by the structural stationarity condition, which asks for a finite set of fragments containing up to structural congruence all reachable fragment.

Conversely, if the mixed semantics is a finite Petri net, then in particular the set of places is finite. Mixed boundedness of the translated process follows from the definition of the set of places.

Theorem 9.2.5 (Finiteness Characterisation)
The mixed semantics $\mathcal{N}_{\mathcal{M}}[\![P]\!]$ is finite if and only if P is mixed-bounded.

Corollary 9.2.6 (Mixed-Bounded Processes are Bounded in Depth)
If $P \in \mathcal{P}$ is mixed-bounded, then P is bounded in depth, $P \in \mathcal{P}_{\mathcal{D}<\infty}$.

Proof
Let P be mixed-bounded, so that all tagged names are included in $\tilde{m}^{\mathcal{C}}$ and all fragments are structurally congruent with $F_1, \ldots, F_n$. Then the depth of all reachable fragments is bounded by $|\tilde{m}^{\mathcal{C}}| + max\{\|F_i\|_{\mathcal{D}} \mid 1 \leq i \leq n\}$. ■

Theorem 9.2.5 yields the following implication. If a process is mixed-bounded, then *there is* a faithful representation of the process as finite place/transition Petri net. Hence, mixed boundedness is a *sound* characterisation of processes with respect to place/transition Petri net semantics. The following section shows that it is also complete. If a process class is not mixed-bounded, then *there is no* faithful finite place/transition Petri net representation. This shows that mixed-bounded processes form the borderline between the π-Calculus and Petri nets.

9.3 Completeness of Mixed Boundedness

We argue that the class of mixed bounded processes is *complete* with respect to finite Petri net semantics. If we have a superclass of mixed bounded processes, there will be no reachability-preserving translation into finite place/transition Petri nets. Since it is always possible to handle particular classes of processes by specialised translations, we make our argument precise. We show that in *slight extensions* of mixed bounded processes reachability becomes undecidable. Since the problem is decidable for finite place/transition Petri nets [May84, Kos82], there can be no reachability-preserving translation for the extended process class.

The processes we consider are *bounded in depth by one*. They are not mixed bounded since (1) they create an arbitrary number of restricted names and since (2) their fragments are not bounded in breadth. Condition (1) states that the restricted names cannot be handled by the concurrency semantics while Condition (2) ensures the processes do not yield a finite structural semantics.

To establish undecidability of reachability, we reduce the reachability problem for 2-counter machines (cf. Section 8.3.1). Since the resulting processes are

bounded in depth by one, the encoding of counter machines presented in this section drastically differs from the one in Section 8.3.2. It is an adaptation of a construction in [DFS98], which shows that reachability is undecidable for Petri nets with transfer.

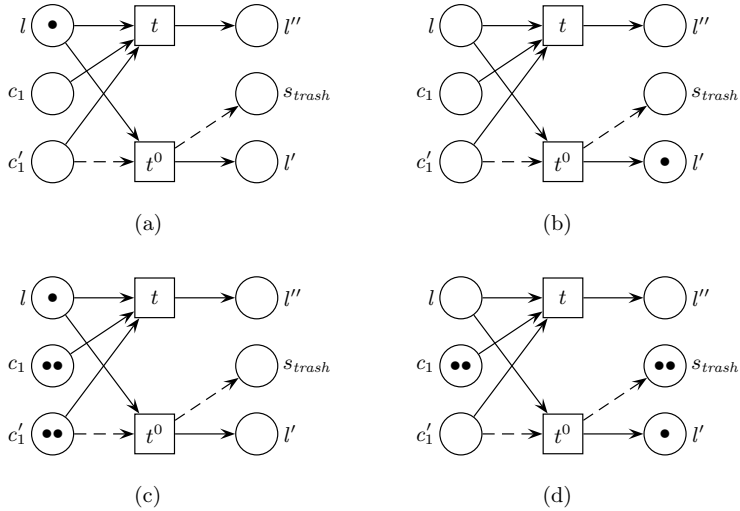


Figure 9.6:
A Petri net with transfer modelling a test for zero in a counter machine. Dashed lines represent transfer arcs of t^0 that move all tokens in c_1' to the trash place s_{trash} when the transition is fired. Note that t^0 is enabled as long as place l carries a token—even if c_1' is zero. For example, firing t^0 in the transfer net in (a) yields the marking in (b). Firing t^0 in (c) yields the marking depicted in (d).

The idea of Dufourd, Finkel, and Schnoebelen is to represent a counter c_1 by two places c_1 and c_1'. The test for zero

$$\text{if } c_1 = 0 \text{ then goto } l'; \text{ else } c_1 := c_1 - 1; \text{ goto } l''; \tag{9.1}$$

is modelled by the transfer net in Figure 9.6. To test counter c_1 for being zero, transition t^0 transfers the content of place c_1' to a trash place s_{trash}. Since the transition is enabled although c_1' is empty (Figure 9.6 (a) and (b)), the content of c_1 and c_1' coincides as long as the net properly simulates the counter machine. If a transfer operation is executed although c_1' is not empty, the amount of tokens in c_1 and c_1' becomes different (Figure 9.6 (c) and (d)). Since increment operations

always add the same amount of tokens to c_1 and c_1', this difference is preserved throughout the computation. Hence, a state (v_1, v_2, l) is reachable in the counter machine if and only if a marking is reachable in the transfer net where place l is marked, counter c_1 as well as its copy c_1' carry v_1 tokens, and similarly c_2 and c_2' carry v_2 tokens.

To adapt the model of Dufourd. et. al. to processes, we represent a counter value by a parallel composition of processes, e.g., $c_1 = 3$ by $\overline{a} \mid \overline{a} \mid \overline{a}$. The transfer operation requires us to change arbitrarily many processes with one communication. We achieve this by attaching the processes $\overline{a}$ to a so-called *process bunch* $PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1}\rfloor$. For the counter value $c_1 = 3$, this results in the process

$$\nu a.(PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1}\rfloor \mid \overline{a} \mid \overline{a} \mid \overline{a}).$$

The processes $\overline{a}$ are attached to the process bunch $PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1}\rfloor$ by sharing the restricted name a. The index c_1 of the free names i_{c_1}, d_{c_1}, and t_{c_1} shows that the process bunch models counter c_1. Since a is a restricted name, the process bunch has exclusive access to its processes. It offers three operations to modify their numbers: i_{c_1}, d_{c_1}, and t_{c_1}. A communication on i_{c_1} stands for *increase* and creates a new process $\overline{a}$. Similarly, a message on d_{c_1} *decreases* the process number by consuming a process $\overline{a}$. A *test for zero* on t_{c_1} creates a new and empty process bunch for counter c_1. The old process bunch terminates. A process $\nu a.(\overline{a} \mid \overline{a} \mid \overline{a})$ without process bunch $PB\lfloor a, i_{c_1}, d_{c_1}, t_{c_1}\rfloor$ is considered to belong to the trash place. To sum up, a process bunch is defined by

$$\begin{aligned} PB(a, d_x, i_x, t_x) \quad := \quad & i_x.(PB\lfloor a, d_x, i_x, t_x\rfloor \mid \overline{a}) \\ & + d_x.a.PB\lfloor a, d_x, i_x, t_x\rfloor \\ & + t_x.\nu b.PB\lfloor b, i_x, d_x, t_x\rfloor. \end{aligned}$$

The translation of the labelled instructions is similar to the one in Section 8.3.2. The increment operation on counter c_1

$$l : c_1 := c_1 + 1 \text{ goto } l'$$

yields a process identifier K_l with defining equation

$$K_l(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') := i_{c_1}.i_{c_1'}K_{l'}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor.$$

The parameter lists are $\tilde{c}_1 = i_{c_1}, d_{c_1}, t_{c_1}$ and similar for the other counters. Note that both, c_1 and c_1', are incremented to keep the numbers of processes in both bunches equal. Like for Petri nets with transfer, the test for zero in (9.1) only changes the value of counter c_1'. The decrement operation acts on both counters:

$$K_l(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') := t_{c_1'}.K_{l'}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor + d_{c_1}.d_{c_1'}.K_{l''}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor.$$

In one respect, our process model is different from the encoding of Dufourd et. al. In the transfer net, a decrement happens only if the places c_1 and c_1' carry a token. A process bunch accepts a decrement operation although it might be empty. In this case, the system deadlocks and reachability is preserved. Finally, a halt instruction $l : \mathit{halt}$ is translated into $K_l(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') := \overline{\mathit{halt}}$. The full translation of a counter machine CM yields the process

$$\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty}[\![CM]\!] = \prod_{x \in \{c_1, \ldots, c_2'\}} \nu a_x . PB\lfloor a_x, \tilde{x}\rfloor \mid K_{l_0}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor. \tag{9.2}$$

Example 9.3.1 ($\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty} : \mathcal{CM} \to \mathcal{P}_{\mathcal{D}<\infty}$)
Consider the counter machine $CM = (c_1, c_2, \mathit{instr})$ with

instr :
$l_0 : c_1 := c_1 + 1;$ goto l_1;
$l_1 :$ if $c_1 = 0$ then goto l_1; else $c_1 := c_1 - 1;$ goto l_2;
$l_2 : \mathit{halt}.$

The machine sets c_1 to one, the following check for zero fails, c_1 is decremented, and the machine stops. The corresponding process has the form in (9.2) with the following defining equations:

$$\begin{aligned}
K_{l_0}(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') &:= i_{c_1} . i_{c_1'} K_{l_1}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor \\
K_{l_1}(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') &:= t_{c_1'} . K_{l_1}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor + d_{c_1} . d_{c_1'} . K_{l_2}\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor \\
K_{l_2}(\tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2') &:= \overline{\mathit{halt}}.
\end{aligned}$$

♦

The reachable states of the counter machine CM can be computed from the reachable processes of $\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty}[\![CM]\!]$. More precisely, the counter machine CM reaches the state (v_1, v_2, l) if and only if its encoding reaches the process

$$\begin{aligned}
&\prod_{x \in \{c_1, c_1'\}} \nu a_x . (PB\lfloor a_x, \tilde{x}\rfloor \mid \Pi^{v_1} \overline{a_x}) \\
\mid\ &\prod_{x \in \{c_2, c_2'\}} \nu a_x . (PB\lfloor a_x, \tilde{x}\rfloor \mid \Pi^{v_2} \overline{a_x}) \\
\mid\ &K_l\lfloor \tilde{c}_1, \tilde{c}_1', \tilde{c}_2, \tilde{c}_2'\rfloor.
\end{aligned}$$

The first parallel composition ensures that the bunches for c_1 and c_1' contain v_1 processes, the construction for the counters c_2 and c_2' is similar. Combined with the observation that the process $\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty}[\![CM]\!]$ is always bounded in depth by one, we arrive at the desired undecidability theorem.

Theorem 9.3.2 (Undecidability of Reachability in Depth One)
Consider two processes $P, Q \in \mathcal{P}_{\mathcal{D}<\infty}$ where the depth is bounded by one. The problem whether $[Q] \in \mathit{Reach}(P)/_{\equiv}$ is undecidable.

Note that Theorem 9.3.2 implies undecidability of reachability for the class $\mathcal{P}_{\mathcal{D}<\infty}$. This means, for processes of bounded depth reachability is undecidable by a reduction from counter machines but termination is decidable according to Corollary 8.2.22. Hence, our mapping $\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty}$ cannot preserve termination since this is undecidable for counter machines. Example 9.3.1 gives a counter machine CM that terminates but whose process representation $\mathcal{P}_{\mathcal{CM}}^{\mathcal{D}<\infty}[\![CM]\!]$ has an infinite run.

Since reachability is decidable for finite place/transition Petri nets [May84, Kos82], we conclude that there does not exist a *reachability-preserving* translation into finite place/transition Petri nets for any class of processes subsuming those of depth one.

Corollary 9.3.3 (Completeness of Mixed Boundedness)
Consider a class of processes $\mathcal{P}' \subseteq \mathcal{P}$ that contains all processes of depth one. For any mapping $f : \mathcal{P}' \to \mathcal{PN}$ one of the following holds. Either $f(P)$ is infinite for some processes or reachability is not preserved, i.e., there is no algorithm to decide on the Petri net $f(P)$ whether a given process $[Q]$ is reachable from P.

In our opinion, mixed bounded processes are close to processes of depth one. Any reasonable extension of this class will subsume the processes of depth one. Hence, the extension will not be translatable into finite place/transition Petri nets without losing reachability. This closes our argumentation for completeness of mixed bounded processes with respect to finite place/transition Petri nets.

9.4 Related Work and Conclusion

We investigated the translation of processes of bounded depth into place transition Petri nets. The main result is that the structural semantics from Chapter 3 and a new concurrency semantics are orthogonal and that they can be combined to a mixed semantics. The class of processes that are finitely represented by the mixed semantics generalises both, structurally stationary and restriction bounded processes (Remark 9.2.4), the latter are finitely represented by the concurrency semantics (Theorem 9.1.12). Mixed processes still lie within the class of processes of bounded depth (Corollary 9.2.6). Figure 9.7 illustrates the relationship between the process classes. We showed it to be impossible to further generalise mixed bounded processes while still achieving reachability-preserving translations to Petri nets. The reason is that reachability becomes undecid-

able immediately outside mixed boundedness, in particular processes of bounded depth are strictly more expressive than Petri nets. In this sense, mixed bounded processes are complete with respect to finite place/transition Petri net semantics.

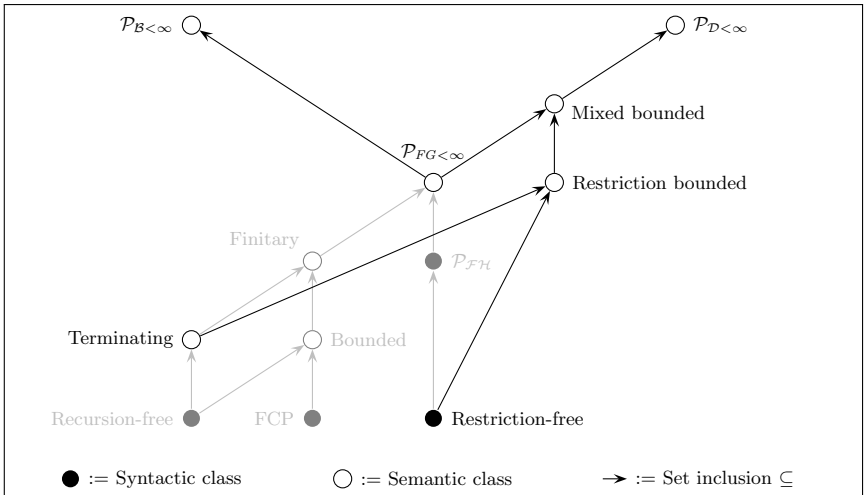


Figure 9.7: A more complete hierarchy of processes.

Although several concurrency Petri net semantics for the π-Calculus have been proposed in the literature [Eng96, BG95, BG09, AM02, KKN06, DKK06a, DKK06b, DKK08], we found them all defective in the sense that they do not satisfy the quality criteria we require for Petri net semantics to be usable for verification purposes. Therefore, we presented a new concurrency semantics. To the best of our knowledge, it is the first that translates processes with restricted names (expressiveness) into bisimilar (retrievability) place/transition Petri nets (analysability), so that finiteness of the nets can be characterised. The proof of bisimilarity and finiteness rely on a new transition system for the π-Calculus that keeps track of the identities of restricted names. It is missing in related approaches.

We discuss the problems with previous results. Engelfriet [Eng96] translates processes with replication into bisimilar place/transition Petri nets. Since the Petri net representation is infinite as soon as the replication operator is used the requirement for finiteness is not satisfied. Amadio and Meyssonnier [AM02] translate recursive but restriction-free processes into bisimilar and finite place/transition Petri nets. Hence, their semantics does not handle an express-

ive class of processes. Similar to our approach, Busi and Gorrieri [BG95, BG09] translate restriction bounded processes into finite place/transition Petri nets. More precisely, they use primitive inhibitor nets and show that the inhibiting places can be removed if reaction semantics is considered. They fail to prove bisimilarity, so retrievability does not hold. Koutny et. al. [KKN06, DKK06a, DKK06b, DKK08] achieve a bisimilar translation into finite but high-level Petri nets, thus violating the requirement for analysability by using a Turing complete formalism.

The definition of the mixed semantics relies on a typing mechanism for restricted names. If a name carries a tag $\mathcal{C}$, it is translated according to the concurrency semantics, otherwise according to the structural semantics. In our tool Petruchio—that besides the structural also implements the concurrency and the mixed semantics—we do not expect a user to type restrictions. Instead, Strazny implemented two algorithms that infer the types of restrictions automatically. The first tries to handle every name with the structural semantics. If it detects unbounded breadth for some name (with an approximate algorithm), it changes the type of the name and repeats the compilation with the new process. The second algorithm tags all names so that they are handled by the concurrency semantics and then iteratively removes the tags.

In Section 9.3, we showed how to imitate an undecidability result for Petri nets with transfer (to a trash place) in processes of bounded depth. In fact, an extension of the presented construction shows that every Petri net with transfer (to a trash place) can be modelled bisimilarly by a process of bounded depth. This has an interesting decidability-theoretic consequence. We discussed in Section 4.5 that action-based linear-time logics are decidable for Petri nets [Esp94] and conjectured that they should be decidable for structurally stationary processes. For Petri nets with transfer, decidability of these logics has been settled negatively [RB04]. Hence, we conjecture that model checking processes of bounded depth against these logics will be undecidable. A comprehensive overview of decidability results for this extended Petri net model can be found in [Gee07]. A second remark about the more general encoding of transfer nets (with trash place) into processes of bounded depth is that it proves input-bounded unique receiver systems to be bisimilarly reflectable in bounded depth. Amadio and Meyssonnier encode their processes into Petri nets with transfer [AM02], which we then in turn model in bounded depth.

10 Conclusion

Contents

We summarise the main results in this thesis and make a remark on the corresponding publications. Afterwards we discuss future work.

10.1 Summary

We presented finite representations of infinite-state DRS classes, which allow for the application of computer-aided verification techniques. The main subject were structurally stationary systems and their representation in the structural semantics. For systems of bounded depth we showed how to compute the finite reachability tree. Restriction bounded systems are finitely represented under the concurrency semantics and mixed bounded systems under the mixed semantics. Systems of bounded breadth turned out Turing complete, so there does not exist a finite and decidable representation. We clarified the relationship among the classes and to existing classes in the literature and obtained a fairly complete picture (Figure 9.7). Moreover, the tight relationship to (extended) Petri net models provides a good intuition to decidability of verification problems for the different classes. To sum up, given a π-Calculus model to be verified, we can (1) judge the class of systems it belongs to, (2) compute a finite representation for it, and (3) depending on the class provide a number of verification techniques.

We organised the thesis along a list of quality criteria finite representations have to satisfy in order to be useful for verification purposes, namely *retrievability*,

finiteness, *expressiveness*, *analysability*, *intuitivity*, and *maximality*. We briefly recapitulate the contributions of the three parts.

Part I was devoted to the theory of structural stationarity. We observed that π-Calculus processes may be interpreted as groups of unconnected graphs and formalised this intuition in terms of a normal form for processes. The main ingredients are fragments, groups of processes connected by restricted names. This restricted form gave rise to the definition of the structural place/transition Petri net semantics. We showed that the transition systems of process and Petri net are isomorphic and that the structural semantics is fully abstract with respect to structural congruence. We then defined the property of structural stationarity for processes, which requires a finite set of fragments every reachable process consists of. The structural semantics is finite precisely for structurally stationary processes. A complete characterisation shows that a process is structurally stationary if and only if there is a bound on the number of sequential processes in all reachable fragments. The corresponding theorem proved structural stationarity for finitary and for restriction-free processes. To the best of our knowledge, the structural semantics is the first automata-theoretic representation that finitely represents both classes. With the aim of modelling client-server architectures, we defined finite handler processes and proved them structurally stationary. We concluded the investigation with a translation of Petri nets back into structurally stationary systems, which showed that the size of the structural semantics is not bounded by a primitive recursive function in the size of the process.

Part II was concerned with automatic verification techniques for structurally stationary processes. We showed how to exploit a particularly efficient verification techniques for safe Petri nets for the verification of finite control processes, a subclass of structurally stationary processes, and applied the resulting tool chain to verify a number of benchmark case studies. Most notably, we verified a realistic model of an automated transportation system. On a highway control system, we demonstrated that also other well-known verification techniques from Petri net theory can be applied successfully to prove structurally stationary systems correct.

Part III aims at an intuitive understanding of structurally stationary systems. The main finding is that the property of structural stationarity can be decomposed into two boundedness requirements. Based on a formalisation of the graph-theoretic interpretation of processes, we showed that boundedness in the novel function *depth* can be characterised by forbidding list structures. Boundedness in the measure *breadth* says that there is a bound on the distribution of restricted names. We proved processes of bounded depth to have well-structured transition systems (WSTS) and inherited the finite reachability tree procedure, which allows

us to decide termination and infinity of states. For systems of bounded breadth, we recalled a folklore construction to prove Turing completeness. Finally, we combined our structural semantics with a classical concurrency semantics to a mixed translation. It forms the borderline to place/transition Petri nets as for processes just beyond this class (but within bounded depth) reachability becomes undecidable.

Publications Chapter 3 and Chapter 4, the verification of the car platoon system, and the second characterisation of structural stationarity can be found in [Mey09]. The unfolding-based verification technique in Chapter 5 appeared as [MKS08] and with full proofs and the transportation case study as journal version [MKS09]. The characterisation of boundedness in depth and the instantiation of the well-structured transition system framework is [Mey08]. Concurrency and mixed semantics have been presented in [MG09].

10.2 Future Work

We gave hints for future work at the end of every chapter, and only summarise the main ideas here.

Compositionality The structural semantics is a non-compositional function. Operators $\oplus$ on Petri nets should be investigated, which validate an equation $\mathcal{N}[\![P \mid Q]\!] \approx \mathcal{N}[\![P]\!] \oplus \mathcal{N}[\![Q]\!]$ [BDK01]. This composition operator is difficult to define as a restricted name sent from outside comes with all processes that use this name. So, communication between the parts is not restricted to synchronisation but exchanges higher-order objects [SW01]. However, such a compositional semantics would offer the possibility for compositional verification. A property of interest for $\mathcal{N}[\![P \mid Q]\!]$ is split up into subproperties for the system parts $\mathcal{N}[\![P]\!]$ and $\mathcal{N}[\![Q]\!]$. These are proven separately and imply the full property taking into account the composition [dRdBH$^+$01].

Logics The theory of structural stationarity should also be extended by a suitable logic to specify properties of processes. Candidate logics are restrictions of the spatial logic of Caires and Cardelli that specify correctness of connections between processes—a crucial property in DRS [CC03]. The idea to check whether a process P satisfies a formula ϕ is to compute the structural semantics $\mathcal{N}[\![P]\!]$ and likewise to compile down the property to an ordinary temporal logic property for Petri nets $\theta_{\mathcal{PN}}(\phi)$. The equivalence

$$P \models \phi \quad \Leftrightarrow \quad \mathcal{N}[\![P]\!] \models \theta_{\mathcal{PN}}(\phi).$$

reduces the problem $P \models \phi$ to checking a Petri net property with standard Petri net verification tools. First steps towards such a translation have been achieved by Sven Linker in his Master's thesis [Lin08].

The logic of Linker only talks about the temporal evolution of structures, hereby neglecting process identities. Properties of DRS often refer to the identities of processes (e.g. if free agent x receives a car ahead message now, it will be a follower later on, cf. [Dam96, FGMP03, Wes08, Tob08] for further examples). Those properties are neither expressible in the logic developed by Linker nor can they be verified with help of the structural semantics. As explained in Section 6.3, the latter loses the identities of restricted names when fragment F evolves to Q with a transition $([F], [Q])$. A possible solution is to equip the transitions in the structural semantics with labels relating the restricted names in pre and postset, similar to History-Dependent automata [MP95a, Pis99, MP01].

Bisimilarity Checking For some processes (e.g. closed ones), the structural semantics yields communication-free Petri nets, which are known to be equivalent to Basic Parallel Processes. For Basic Parallel Processes, bisimulation equivalence is known to be decidable [BCMS01]. It would be interesting to investigate whether the procedure can be adapted to decide a standard bisimilarity on structurally stationary processes—in particular barbed bisimulation equivalence. For the full class of structurally stationary processes, checking bisimilarity is likely to be undecidable due to the negative results for Petri nets [Jan95].

Bounded Depth For systems of bounded depth, we presented the finite reachability tree for analysis purposes. For practical applications, abstractions of the tree are required to efficiently evaluate the state space. A second aspect is decidability of more intricate temporal logic properties. Like in the coverability tree for Petri nets, it should be possible to compute the limits of computation sequences for processes of bounded depth. However, the construction in Section 9.3 combined with the results in [RB04] make a decision procedure for full LTL unlikely.

Bounded Breadth Systems of unbounded depth but bounded breadth deserve more attention. Although we rendered the full and large class Turing complete, there may be reasonable constraints which yield interesting decidable subclasses.

Bibliography

[AČJT00] P. A. Abdulla, K. Čerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2):109–127, 2000.

[AM02] R. M. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous π-calculus. *Nordic Journal of Computing*, 9(1):70–101, 2002.

[Ama00] R. M. Amadio. On modelling mobility. *Theoretical Computer Science*, 240(1):147–176, 2000.

[AVA04] Automatic Verification and Analysis of Complex Systems. URL: `http://www.avacs.org`, last access 2008-11-22, since 2004.

[Bas94] D. A. Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51(2):61–66, 1994.

[Bau06] J. Bauer. *Analysis of Communication Topologies by Partner Abstraction*. PhD thesis, Department of Computer Science, Saarland University, 2006.

[BB90] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. of the 17th Annual ACM Symposium on Principles of Programming Languages, POPL*, pages 81–94. ACM Press, 1990.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In J.A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[BDK01] E. Best, R. Devillers, and M. Koutny. *Petri Net Algebra*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2001.

[BDNN98] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Proc. of the 9th International Conference on Concurrency Theory, CONCUR*, volume 1466 of *LNCS*, pages 84–98. Springer-Verlag, 1998.

[BG95] N. Busi and R. Gorrieri. A Petri net semantics for π-calculus. In *Proc. of the 6th International Conference on Concurrency Theory, CONCUR*, volume 962 of *LNCS*, pages 145–159. Springer-Verlag, 1995.

[BG09] N. Busi and R. Gorrieri. Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs. *Journal of Logic and Algebraic Programming*, 78(1):138–162, 2009.

[BGZ03] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *Proc. of the 30th International Colloquium on Automata, Languages and Programming, ICALP*, volume 2719 of *LNCS*, pages 133–144. Springer-Verlag, 2003.

[BGZ04] N. Busi, M. Gabbrielli, and G. Zavattaro. Comparing recursion, replication, and iteration in process calculi. In *Proc. of the 31th International Colloquium on Automata, Languages and Programming, ICALP*, volume 3142 of *LNCS*, pages 307–319. Springer-Verlag, 2004.

[BGZ08] N. Busi, M. Gabbrielli, and G. Zavattaro. On the expressive power of recursion, replication, and iteration in process calculi. 27 pages, under consideration for publication in Mathematical Structures of Computer Science, 2008.

[BR01] A. Braatz and A. Ritter. Referenzfallstudie Produktionstechnik. Technical report, IFF University Stuttgart and Fraunhofer IPA Stuttgart, 2001. Version 1.3: `http://tfs.cs.tu-berlin.de/projekte/indspec/SPP/RefPAv13.ps`, version 2.0: `http://tfs.cs.tu-berlin.de/projekte/indspec/SPP/RefPAv2.ps`, last access 2008-11-28.

[BRdS86] G. Boudol, G. Roucairol, and R. de Simone. Petri nets and algebraic calculi of processes. In *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory of Petri Nets-selected papers 1986*, volume 222 of *LNCS*, pages 41–58. Springer-Verlag, 1986.

[BTW07] J. Bauer, T. Toben, and B. Westphal. Mind the shapes: Abstraction refinement via topology invariants. In *Proc. of the 5th International Symposium on Automated Technology for Verification and Analysis, ATVA*, volume 4762 of *LNCS*, pages 35–50. Springer-Verlag, 2007.

[Bus02] N. Busi. Analysis issues in Petri nets with inhibitor arcs. *Theoretical Computer Science*, 275(1–2):127–177, 2002.

[Cai04] L. Caires. Behavioural and spatial observations in a logic for the π-Calculus. In *Proc. of the 7th International Conference on Foundations of Software Science and Computation Structures, FOSSACS*, volume 2987 of *LNCS*, pages 72–89. Springer-Verlag, 2004. Spatial Logic Model Checker: `http://ctp.di.fct.unl.pt/SLMC/`, last access 2008-11-28.

[CC03] L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Information and Computation*, 186(2):194–235, 2003.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of the 12th International Conference on Computer Aided Verification, CAV*, volume 1855 of *LNCS*, pages 154–169. Springer-Verlag, 2000.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CL04] L. Caires and É. Lozes. Elimination of quantifiers and undecidability in spatial logics for concurrency. In *Proc. of the 15th International Conference on Concurrency Theory, CONCUR*, volume 3170 of *LNCS*, pages 240–257. Springer-Verlag, 2004.

[Dam96] M. Dam. Model checking mobile processes. *Information and Computation*, 129(1):35–51, 1996.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of the 25th International Colloquium on Automata, Languages and Programming, ICALP*, volume 1443 of *LNCS*, pages 103–115. Springer-Verlag, 1998.

[DHS08] R. Demangeon, D. Hirschkoff, and D. Sangiorgi. Static and dynamic typing for the termination of mobile processes. In *Proc. of the 5th IFIP International Conference on Theoretical Computer Science, IFIP TCS*, volume 273 of *IFIP*, pages 413–427. Springer-Verlag, 2008.

[Die06] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 3rd edition, 2006.

[Dij68] E. W. Dijkstra. The structure of the "THE"-multiprogramming system. *Communications of the ACM*, 11(5):341–346, 1968.

[DKK06a] R. Devillers, H. Klaudel, and M. Koutny. A Petri net semantics of the finite π-Calculus terms. *Fundamenta Informaticae*, 70(3):203–226, 2006.

[DKK06b] R. Devillers, H. Klaudel, and M. Koutny. A Petri net translation of π-Calculus terms. In *Proc. of the 3rd International Colloquium on Theoretical Aspects of Computing, ICTAC*, volume 4281 of *LNCS*, pages 138–152. Springer-Verlag, 2006.

[DKK08] R. Devillers, H. Klaudel, and M. Koutny. A compositional Petri net translation of general π-Calculus terms. *Formal Aspects of Computing*, 20(4–5):429–450, 2008.

[DRB02] G. Delzanno, J.-F. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, volume 2280 of *LNCS*, pages 173–187. Springer-Verlag, 2002.

[dRdBH+01] W.-P. de Roever, F. S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[DS94] C. Dietz and G. Schreiber. A term representation of p/t systems. In *Proc. of the 15th International Conference on Application and Theory of Petri Nets, ATPN*, volume 815 of *LNCS*, pages 239–257. Springer-Verlag, 1994.

[DS06] Y. Deng and D. Sangiorgi. Ensuring termination by typability. *Information and Computation*, 204(7):1045–1082, 2006.

[EDD+04] H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, editors. *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *LNCS*. Springer-Verlag, 2004. Project URL: `http://tfs.cs.tu-berlin.de/projekte/indspec/SPP/index-eng.html`, last access 2008-11-22.

[EEHP06] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Theory of constraints and application conditions: From graphs to high-level structures. *Fundamenta Informaticae*, 74(1):135–166, 2006.

[EG99] J. Engelfriet and T. Gelsema. Multisets and structural congruence of the pi-calculus with replication. *Theoretical Computer Science*, 211(1-2):311–337, 1999.

[EG01] J. Engelfriet and T. Gelsema. Structural inclusion in the pi-calculus with replication. *Theoretical Computer Science*, 258(1–2):131–168, 2001.

[EG04a] J. Engelfriet and T. Gelsema. The decidability of structural congruence for replication restricted pi-calculus processes. Technical report, Leiden Institute of Advanced Computer Science, 2004. Revised 2005.

[EG04b] J. Engelfriet and T. Gelsema. A new natural structural congruence in the pi-calculus with replication. *Acta Informatica*, 40(6):385–430, 2004.

[EG07] J. Engelfriet and T. Gelsema. An exercise in structural congruence. *Information Processing Letters*, 101(1):1–5, 2007.

[EH08] J. Esparza and K. Heljanko. *Unfoldings*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2008.

[Eng96] J. Engelfriet. A multiset semantics for the pi-calculus with replication. *Theoretical Computer Science*, 153(1-2):65–94, 1996.

[ES01] J. Esparza and C. Schröter. Net reductions for LTL model-checking. In *Proc. of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods, CHARME*, volume 2144 of *LNCS*, pages 310–324. Springer-Verlag, 2001.

[Esp94] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. of the 19th International Colloquium on Trees in Algebra and Programming, CAAP*, volume 787 of *LNCS*, pages 115–129. Springer-Verlag, 1994.

[Esp97a] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[Esp97b] J. Esparza. Petri Nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997.

[FGMP03] G.-L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Transactions on Software Engineering and Methodology*, 12(4):440–473, 2003. HAL: `http://fmt.isti.cnr.it:8080/hal/`, last access 2008-11-28.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

[FMPR01] S. Flake, W. Mueller, W. Pape, and J. Ruf. Analyzing timing constraints in flexible manufacturing systems. In *Proc. of the International NAISO Symposium on Information Science Innovations in Intelligent Automated Manufacturing, IAM 2001*, 2001.

[Fok07] W. Fokkink. *Modelling Distributed Systems.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2007.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[Gee07] G. Geeraerts. *Coverability and Expressiveness Properties of Well-Structured Transition Systems.* PhD thesis, Département d'Informatique, Univesité Libre de Bruxelles, 2007.

[Gri07] P. Gringel. Modellierung und Verifikation eines holonischen Transportsystems mit dem Pi-Kalkül. Bachelor's thesis, Department of Computing Science, University of Oldenburg, 2007.

[Hab92] A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *LNCS*. Springer-Verlag, 1992.

[Hab97] P. Habermehl. On the complexity of the linear-time μ-calculus for Petri-nets. In *Proc. of the 18th International Conference on Application and Theory of Petri Nets, ATPN*, volume 1248 of *LNCS*, pages 102–116. Springer-Verlag, 1997.

[Hel02] K. Heljanko. *Combining Symbolic and Partial Order Methods for Model Checking 1-Safe Petri Nets.* PhD thesis, Department of Computer Science and Engineering, Helsinki University of Technology, 2002.

[HESV91] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. Design of platoon maneuver protocols for ivhs. Path research report, Institute of Transportation Studies, University of California, Berkeley, 1991.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.* (3), 2(7):326–336, 1952.

[Hir04] D. Hirschkoff. An extensional spatial logic for mobile processes. In *Proc. of the 15th International Conference on Concurrency Theory, CONCUR*, volume 3170 of *LNCS*, pages 325–339. Springer-Verlag, 2004.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Jan95] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[Kho03] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, School of Computing Science, Newcastle University, 2003.

[Kho08] V. Khomenko. PUNF homepage. URL: `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/tools.html`, last access 2008-11-16, 2008.

[KK06] B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, volume 3920 of *LNCS*, pages 197–211. Springer-Verlag, 2006.

[KKN06] V. Khomenko, M. Koutny, and A. Niaouris. Applying Petri net unfoldings for verification of mobile systems. In *Proc. of the 4th Workshop on Modelling of Objects, Components and Agents, MOCA*, Bericht FBI-HH-B-267/06, pages 161–178. University of Hamburg, 2006.

[KKY04] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state encoding conflicts in stg unfoldings using sat. *Fundamenta Informaticae*, 62(2):221–241, 2004.

[KM09] V. Khomenko and R. Meyer. Checking π-Calculus structural congruence is graph isomorphism complete. In *Proc. of the 9th International Conference on Application of Concurrency to System Design, ACSD*, pages 70–79. IEEE Computer Society Press, 2009.

[Kos82] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. of the 14th Annual ACM Symposium on Theory of Computing, STOC*, pages 267–281. ACM Press, 1982.

[Lin08] S. Linker. Model checking π-Calculus against temporal connectedness properties. Master's thesis, Department of Computing Science, University of Oldenburg, 2008.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[Mey08] R. Meyer. On boundedness in depth in the π-calculus. In *Proc. of the 5th IFIP International Conference on Theoretical Computer Science, IFIP TCS*, volume 273 of *IFIP*, pages 477–489. Springer-Verlag, 2008.

[Mey09] R. Meyer. A theory of structural stationarity in the π-calculus. *Acta Informatica*, 46(2):87–137, 2009.

[MG09] R. Meyer and R. Gorrieri. On the relationship between π-Calculus and finite place/transition Petri nets. In *Proc. of the 20th International Conference on Concurrency Theory 2009, CONCUR*, volume 5710 of *LNCS*, pages 463–480. Springer-Verlag, 2009.

[Mil79] R. Milner. Flowgraphs and flow algebras. *Journal of the Association for Computing Machinery*, 26(4):794–818, 1979.

[Mil89] R. Milner. *Communication and concurrency*. Prentice Hall, 1989.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[MJ84] F. L. Morris and C. B. Jones. An early program proof by alan turing. *IEEE Annals of the History of Computing*, 6(2):139–143, 1984.

[MKS08] R. Meyer, V. Khomenko, and T. Strazny. A practical approach to verification of mobile systems using net unfoldings. In *Proc. of the 29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency, ATPN*, volume 5062 of *LNCS*, pages 327–347. Springer-Verlag, 2008.

[MKS09] R. Meyer, V. Khomenko, and T. Strazny. A practical approach to verification of mobile systems using net unfoldings. *Fundamenta Informaticae*, 94(3–4):439–471, 2009.

[MM79] G. Milne and R. Milner. Concurrent processes and their syntax. *Journal of the Association for Computing Machinery*, 26(2):302–321, 1979.

[MM81] E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for Petri nets. *Journal of the Association for Computing Machinery*, 28(3):561–576, 1981.

[MORW04] M. Möller, E.-R. Olderog, H. Rasch, and H. Wehrheim. Linking csp-oz with uml and java: A case study. In *Proc. of the 4th International Conference on Integrated Formal Methods, IFM*, volume 2999 of *LNCS*, pages 267–286. Springer-Verlag, 2004.

[MP95a] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. of the 6th International Conference on Concurrency Theory, CONCUR*, volume 962 of *LNCS*, pages 42–56. Springer-Verlag, 1995.

[MP95b] U. Montanari and M. Pistore. Concurrent semantics for the π-calculus. *Electronic Notes in Theoretical Computer Science*, 1:411–429, 1995.

[MP01] U. Montanari and M. Pistore. History dependent automata. Technical report, Instituto Trentino di Cultura, 2001.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100(1):1–40, 1992.

[Old91] E.-R. Olderog. *Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1991.

[OP92] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(6):497–543, 1992.

[PAT86] California Partners for Advanced Transit and Highways. URL: `http://www.path.berkeley.edu`, last access 2008-11-22, since 1986.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. Schriften des IMM, Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962.

[Pis99] M. Pistore. *History Dependent Automata*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1999.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, University of Aarhus, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science, FOCS*, pages 46–57. IEEE Computer Society Press, 1977.

[PW03] L. Priese and H. Wimmel. *Petri-Netze*. Springer-Verlag, 2003.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer-Verlag, 1982.

[RB04] J.-F. Raskin and L. Van Begin. Petri nets with non-blocking arcs are difficult to analyze. *Electronic Notes in Theoretical Computer Science*, 98:35–55, 2004.

[Rei85] W. Reisig. *Petri nets: An Introduction*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1985.

[Ren04] A. Rensink. Canonical graph shapes. In *Proc. of the 13th European Symposium on Programming, ESOP*, volume 2986 of *LNCS*, pages 401–415. Springer-Verlag, 2004.

[RWKR04] J. Ruf, R. J. Weiss, T. Kropf, and W. Rosenstiel. Modeling and formal verification of production automation systems. In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *LNCS*, pages 541–566. Springer-Verlag, 2004.

[San01] D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. of the 28th Annual ACM Symposium on Principles of Programming Languages, POPL*, pages 4–13. ACM Press, 2001.

[SM08] T. Strazny and R. Meyer. PETRUCHIO homepage. URL: `http://petruchio.informatik.uni-oldenburg.de`, last access 2008-11-28, 2008.

[Sta03] P. H. Starke. INA homepage. URL: `http://www2.informatik.hu-berlin.de/lehrstuehle/automaten/ina`, last access 2008-11-16, 2003.

[Str07] T. Strazny. Entwurf und Implementierung von Algorithmen zur Berechnung von Petrinetz-Semantiken für Pi-Kalkül-Prozesse. Master's thesis, Department of Computing Science, University of Oldenburg, 2007.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tob08] T. Toben. *Analysis of Dynamic Evolution Systems by Spotlight Abstraction Refinement*. PhD thesis, Department of Computing Science, University of Oldenburg, 2008.

[Var91] M. Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Annals of Pure and Applied Logic*, 51(1–2):79–98, 1991.

[VM94] B. Victor and F. Moller. The mobility workbench: A tool for the π-calculus. In *Proc. of the 6th International Conference on Computer Aided Verification, CAV*, volume 818 of *LNCS*, pages 428–440. Springer-Verlag, 1994. MWB: `http://www.it.uu.se/research/group/mobility/mwb`, last access 2008-11-28.

[Weh00] H. Wehrheim. Specification of an automatic manufacturing system: A case study in using integrated formal methods. In *Proc. of the 3rd International Conference on Fundamental Approaches to Software Engineering, FASE*, volume 1783 of *LNCS*, pages 334–348. Springer-Verlag, 2000.

[Wes08] B. Westphal. *Specification and Verification of Dynamic Topology Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, 2008.

[WW07] B. Wachter and B. Westphal. The spotlight principle. on combining process-summarising state abstractions. In *Proc. of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI*, volume 4349 of *LNCS*, pages 182–198. Springer-Verlag, 2007.

[YBH04] N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the π-Calculus. *Information and Computation*, 191(2):145–202, 2004.

Symbol Index

Subject Index

Curriculum Vitae

20/02/2009	Defense of the dissertation.
10/2005 - 09/2008	Scholarship holder in the graduate school *Trustworthy Software Systems (TrustSoft)* at the University of Oldenburg. Dissertation on structurally stationary reconfigurable systems. Advised by Professor Olderog and Professor Best.
08/2004 - 03/2009	Research assistant in the Transregional Collaborative Research Center on *Automatic Verification and Analysis of Complex Systems (AVACS)*, run by the Universities of Oldenburg, Freiburg, and Saarbrücken. Member of subproject R1 *Beyond Timed Automata* in Oldenburg.
10/2001 - 09/2005	Studies of Computer Science with Mathematics as subsidiary subject at the University of Oldenburg. Master's thesis on model checking phase event automata against duration calculus formulae with the help of test automata.
09/2000 - 06/2001	Military service.
08/1987 - 06/2000	Grundschule Lengenerland, Orientierungsstufe Remels, Gymnasium Westerstede. Abitur 2000.
02/02/1981	Born in Leer, Germany.

Technical Reports

Fakultät II, Department für Informatik, Universität Oldenburg,
Postfach 2503, 26111 Oldenburg, Germany

1/87 A. Viereck: "Klassifikationen, Konzepte und Modelle für den Mensch-Rechner-Dialog" (Dissertation)

2/87 A. Schwill: "Forbidden subgraphs and reduction systems: A comparison"

3/87 J. Kämper: "Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes"

1/88 K. Ambos-Spies, H. Fleischhack, H. Huwig: "Diagonalizing over deterministic polynomial time"

2/88 A. Schwill: " Shortest edge-disjoint paths in geodetically connected graphs"

3/88 V. Claus, U. Lichtblau (Hrsg.): "1. Tagung zur Küsten-Informatik"

1/89 U. van der Valk: "Einige Entscheidbarkeits- und Unentscheidbarkeitsresultate für Klasse von S/T-Netzen unter Maximum Firing Strategie und unter Prioritätenstrategien"

2/89 J. Kämper: "Strukturelle Untersuchungen im Umfeld der Komplexitätsklassen P und NP unter besonderer Berücksichtigung nichtuniformer, probabilistischer und disjunktiv selbstreduzierender Algorithmen" (Dissertation)

3/89 J. Kämper: "Nondeterministic oracle Turing machines with maximal computation paths"

1/90 A. Schwill: "Shortest edge-disjoint paths in graphs" (Dissertation)

2/90 K.R. Apt, E.-R. Olderog: "Using transformations to verify parallel programs"

3/90 U. Lichtblau: "Flußgraphgrammatiken" (Dissertation)

4/90 K.R. Apt, E.-R. Olderog: "Introduction to program verification"

5/90 H. Jasper: "Datenbankunterstützung für Prolog-Programmierumgebungen" (Dissertation)

1/91 F. Korf: "Net-based efficient simulation of AADL specifications"

2/91 S.V. Krishnan, C. Pandu Rangan, A. Schwill, S. Seshadri: "Two disjoint paths in chordal graphs"

3/91 H. Eirund: "Modellierung und Manipulation multimedialer Dokumente" (Dissertation)

4/91 G. Schreiber: "Ein funktionaler Äquivalenzbegriff für den hierarchischen Entwurf von Netzen"

1/92 A. Viereck (Hrsg.): "Ergebnisse der 11. Arbeitstagung, Mensch-Maschine Kommunikation"

2/92 P. Gorny, U. Daldrup, H. Schwab: "Zwischenbilanz: Menschengerechte Gestaltung von Software"

3/92 E.-R. Olderog, St. Rössig, J. Sander, M. Schenke: "ProCoS at Oldenburg: The Interface between Specification Language and occam-like Programming Language"

4/92 F. Korf: "Synthesis of VHDL Test Environments form Temporal Logic Specifications"

5/92 W. Kowalk: "Konstruktorentechnik: Neue Methoden zur Mengenrechnung, Logikrechnung und Intervallrechnung"

1/93 Ch. Dietz, G. Schreiber: "Eine Termdarstellung für S/T-Netze"

2/93 J. Sauer: "Wissensbasiertes Lösen von Ablaufplanungsproblemen durch explizite Heuristiken"

3/93 M. Sonnenschein, U. Lichtblau (Hrsg.): "6. Kolloquium der Arbeitsgruppe Informatik-Systeme"

4/93 H. Fleischhack, U. Lichtblau, M. Sonnenschein, R. Wieting: "Generische Definition hierarchischer zeitbeschrifteter höherer Petrinetze"

5/93 F. Köster, L. Twele, R. Wieting, W. Ziegler: "Fallbeispiele zur Modellierung mit THORNetzen"

1/94 R. Götze: "Dialogmodellierung für multimediale Benutzerschnittstellen"

2/94 B. Müller: "PPO-Eine objektorientierte Prolog-Erweiterung zur Entwicklung wissensbasierter Anwendungssysteme"

3/94 W. Damm/A. Mikschl: "Projekt Entwurf und Implementierung eines Multi-threaded RISC-Prozessors"

4/94 S. Rössig: "A Transformational Approach to the Design of Communicating Systems" (Dissertation)

5/94 G. Schreiber: "Funktionale Äquivalenz von Petri-Netzen" (Dissertation)

1/95 A. Gronewold, H. Fleischhack: "Language Preserving Reductions of Safe Petri-Nets"

2/95 H. Reineke: "Struktur und Verhalten von verteilten endlichen Automaten" (Dissertation)

3/95 H. Behrends: "Beschreibung ereignisgesteuerter Aktivitäten in datenbankgestützten Informationssystemen" (Dissertation)

4/95 U. M. Levens: "Computerunterstütztes Modellieren von Musikstücken mit Petri-Netzen: Das Mailänder Konzept"

1/96 M. Burke: "FDDI und ATM in multimedialen Anwendungsumgebungen" (Dissertation)

2/96 I. Pitschke: "Interaktive Rekonstruktion geometrischer Modelle aus digitalen Bildern" (Dissertation)

1/97 L. Bölke: "Ein akustischer Interaktionsraum für blinde Rechnerbenutzer" (Dissertation)

2/97 S. Schöf: "Verteilte Simulation höherer Petrinetze" (Dissertation)

1/98 S. Kleuker: "Inkrementelle Entwicklung von verifizierten Spezifikationen für verteilte Systeme" (Dissertation)

2/98 J. Bohn: “Mechanical Support and Validation of a Design Calculus for Communicating Systems by a Logic-Based Proof System“ (Dissertation)

3/98 L. Köhler: „Fuzzy Geometrie und Anwendungen in der medizinischen Bildverarbeitung“ (Dissertation)

4/98 J. Helbig: „Linking Visual Formalisms: A ‘Compositional Proof System for Statecharts Based on Symbolic Timing Diagrams“ (Dissertation)

5/98 G. Stiege: „Edge Partitions in Undirected Graphs“

6/98 A. Gerns: „Entwicklung und Bewertung von Objektmigrationsstrategien für verteilte Umgebungen“

7/98 M. Stadler: „Abstrakte Rechnernetzmodelle als Grundlage einer umfassenden Automatisierung des Netzmanagements – Konzepte und Sprachen zu ihrer Umsetzung“ (Dissertation)

8/98 M.-S. Steiner: „Lastverteilung in heterogenen Systemen“

9/98 Clemens Otte: „Fuzzy-Prototyp-Klassifikatoren und deren Anwendung zur automatischen Merkmalsselektion“

1/99 Juliane Vorndamme: „Die Auswirkungen rechtlicher Verpflichtungen auf die Softwareentwicklung“

2/99 E. Best/K.M. Richter: „ Relational Semantics Revisited“

3/99 J. S. Lie: „Einsatz von Objektmigrationssystemen zur Leistungssteigerung in verteilten Systemen“

4/99 ZweiJahresbericht des Fachbereichs Informatik

5/99 Ingo Stierand, Olaf Maibaum, Björn Briel, Günther Stiege: „Cassandra - Generierung, Analyse und Simulation von eingebetteten Multiprozessor-Echtzeitsystemen“

6/99 Gunnar Wittich: „Ein problemorientierter Ansatz zum Nachweis von Realzeiteigenschaften eingebetteter Systeme“

7/99 Annegret Habel, Jürgen Müller, Detlef Plump: „Double-Pushout Graph Transformation Revisited“

8/99 Ingo Stierand: „Eine Konfigurationssprache zur Erstellung von Ambrosia/MP-Systemen“

9/99 Igor V. Tarasyuk: „Equivalences for Concurrent and Distributed Systems“

10/99 Eike Best, Alexander Lavrov: „Generalised Composition Operations for High-Level Petri-Nets“

11/99 Alexander Lavrov: „Enhancing Mixed Nonlinear Optimization: A Hybrid Approach“

12/99 Alexander Lavrov: „Hybrid Techniques in Discrete-Event System Modelling and Control: some Examples“

13/99 Eike Best, Raymond Devillers, Maciej Koutny: „Recursion and Petri Nets“

14/99 Eike Best, Raymond Devillers, Maciej Koutny: „The Box Algebra = Petri Nets + Process Expressions“

15/99 Eike Best, Harro Wimmel: „Reducing *k*-safe Petri Nets to Pomset-equivalent 1-safe Petri Nets“

16/99 Udo Brockmeyer: „Verifikation von STATEMATE Designs“ (Dissertation)

1/00 Henning Dierks: „Specification and Verification of Polling Real-Time Systems“ (Dissertation)

2/00 Clemens Fischer: „Combination and Implementation of Processes and Data: from CSP-OZ to Java“ (Dissertation)

3/00 Cheryl Kleuker: „Constraint Diagrams“ (Dissertation)

4/00 Thomas Thielke: „Linear-algebraische Methoden zur Beschreibung, Verfeinerung und Analyse gefärbter Petrinetze“ (Dissertation)

1/01 Günther Stiege: „Higher Decomposition in Undirected Graphs“ (Bericht)

2/01 Ute Vogel: Zwei-Jahres-Bericht

3/01 Josef Tapken: „Model-Checking of Duration Calculus Specifications“ (Dissertation)

4/01 Björn Briel: „Analyse eingebetteter Systeme mittels verteilter Simulation“ (Dissertation)

5/01 Günther Stiege: „Standard Decomposition and Periodicity of Digraphs“ (Bericht)

6/01 Ingo Stierand: „Ambrosia/MP - Ein Echtzeitbetriebssystem für eingebettete Mehrprozessorsysteme„ (Dissertation)

1/02 Giorgio Busatto, Annegret Habel: „Improving the Quality of Hypertexts Using Graph Transformation“ (Bericht)

2/02 Giorgio Busatto: „Modeling Hyperweb Dynamics through Hierarchical Graph Transformation“ (Bericht)

3/02 Giorgio Busatto: „An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation“ (Dissertation)

4/02 Laila Kabous: „An Object Oriented Design methodology for hard real Time Systems: The OOHARTS approach“ (Dissertation)

1/03 Ute Vogel: „2-Jahres-Bericht“

2/03 Olaf Maibaum: „Bestimmung symbolischer Laufzeiten in eingebetteten Echtzeitsystemen“ (Dissertation)

3/03 Günther Stiege, Ingo Stierand: „Connectedness-Based Hierarchical Decomposition of Undirected Graphs“ (Bericht)

4/03 Willi Hasselbring, Susanne Petersen: „Standards für die medizinische Kommunikation und Dokumentation (Bericht)

5/03 Andreas Möller: „Eine virtuelle Maschine für Graphprogramme“ (Bericht)

6/03 Tom Bienmüller: „Reducing Complexity for the Verification of Statemate Designs“ (Bericht)

7/03 Sandra Steinert: „Graph Programs for Graph Algorithms“ (Bericht)

8/03 Jochen Klose: „Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior“ (Dissertation)

1/04 Jens Oehlerking: „Transformation of Edmonds’ Maximum Matching Algorithm into a Graph Program“ (Bericht)

2/04 Sergej Alekseev: „Dienste Intelligenter Netze Graphentheoretische Methoden in der Kontrollflussanalyse“ (Bericht)

3/04 Giorgio Busatto: „GraJ: A System für Executing Graph Programs in Java“ (Bericht)

1/05 Sergej Alekseev and Johannes Wust: „Graph Theoretical Methods in the Control Flow Analysis of Object Oriented Real Time Software“ (Bericht)

2/05 Ute Vogel: „2-Jahres-Bericht“

3/05 Igor Tarasyuk: „Discrete time stochastic Petri box calculus“ (Bericht)

1/06 Henning Dierks: „Time, Abstraction and Heuristics“ (Habilitation)

2/06 Li Sek Su: „Full-Output Siphons and Deadlock-Freeness for Free Choice Petri Nets“ (Bericht)

3/06 Timo Warns: „Solving Consensus Using Structural Failure Models„ (Bericht)

4/06 Sergej Alekseev: „Graphentheoretische Methoden in der Ablaufanalyse objektorientierter Anwendungen“ (Dissertation)

5/06 Li Sek Su: „Some Considerations on the Foundation of NP-Completeness Theory*“ (Bericht)

6/06 Li Sek Su: „Semitraps and Deadlock-Freeness for Reduced Asymmetric Choice Nets“ (Bericht)

7/06 Li Sek Su: „Algorithms of computing the Deadlock Markings Sets for Petri Nets“ (Bericht)

8/06 Annegret Habel, Karl-Heinz Pennemann, and Arend Rensink: „Weakest Preconditions for High-Level Programs (Long Version)“ (Bericht)

9/06 Jochen Hoenicke: „Combination of Processes, Data, and Time“ (Dissertation)

10/06 Steffen Becker, Marco Boscovic, Abhishek Dhama, Simon Giesecke, Jens Happe, Wilhelm Hasselbring, Heiko Koziolek, Henrik Lipskoch, Roland Meyer, Margarethe Muhle, Alexandra Paul, Jan Ploski, Matthias Rohr, Mani Swaminathan, Timo Warns, Daniel Winteler: „Trustworthy Software Systems: A Discussion of Basic Concepts and Terminology“ (Bericht)

11/06 Christian Zuckschwerdt: „Ein System zur Transformation von Konsistenz- in Anwendungs-bedingungen“ (Bericht)

01/07 Andreas Schäfer: „Specification and Verification of Mobile Real-Time Systems“ (Dissertation)

02/07 Günther Stiege: „General Graphs“ (Bericht)

03/07 Wolfgang Kowalk: „Integralrechnung“ (Bericht)

04/07 Karl Azab, Karl-Heinz Pennemann: „Type Checking C++ Template Instantiation by Graph Programs“ (Bericht)

01/08 Roland Meyer: „On depth and breath in the Pi-Calculus“ (Bericht)

02/08 Ingo Brückner: „Slicing Integrated Formal Specifications for Verification“ (Dissertation)

03/08 Ute Vogel: „2-Jahres-Bericht 2004 - 2006“ (Bericht)

04/08 Günther Stiege: „Summierbare Familien“ (Bericht)

05/08 Igor V. Tarasyuk: „Investigating equivalence relations in dtsPBC“ (Bericht)

01/09 Elke Wilkeit: „2-Jahres-Bericht 2007 – 2008“ (Bericht)

02/09 Roland Meyer: „Structural Stationarity in the pi-Calculus“ (Dissertation)