ACTA UNIVERSITATIS UPSALIENSIS
Uppsala Dissertations from the Faculty of Science and Technology
1214

Jesper Bengtson

Formalising process calculi

UPPSALA
UNIVERSITET

Abstract page

As the complexity of programs increase, so does the complexity of the models required to reason about them. Process calculi were introduced in the early 1980s and have since then been used to model communication protocols of varying size and scope. Whereas modeling sophisticated protocols in simple process algebras like CCS or the pi-calculus is doable, expressing the models required is often gruesome and error prone. To combat this, more advanced process calculi were introduced, which significantly reduce the complexity of the models. However, this simplicity comes at a price – the theories of the calculi themselves instead become gruesome and error prone, and establishing their mathematical and logical properties has turned out to be difficult. Many of the proposed calculi have later turned out to be inconsistent.

The contribution of this thesis is twofold. Firstly we provide methodologies to formalise the meta-theory of process calculi in an interactive theorem prover. These are used to formalise significant parts of the meta-theory of CCS and the pi-calculus in the theorem prover Isabelle, using Nominal Logic to allow for a smooth treatment of the binders. Secondly we introduce and formalise psi-calculi, a framework for process calculi incorporating several existing ones, including those we already formalised, and which is significantly simpler and substantially more expressive. Our methods scale well as complexity of the calculi increases.

The formalised results include congruence results for both strong and weak bisimilarities, in the case of the pi-calculus for both the early and the late operational semantics. We also formalise the proof that the axiomatisation of strong late bisimilarity is sound and complete in the finite pi-calculus. We believe psi-calculi to be one of the most expressive frameworks for mobile process calculi, and our Isabelle formalisation to be the most extensive formalisation of process calculi ever done inside a theorem prover.

To my beloved Eva – for her love, her understanding,
and her infinite support.

Contents

1. Introduction

> Adrian carefully replaced the small fluffy teddy bear above Hex's keyboard. Things immediately began to whirr. The ants started to trot again. The mouse squeaked.
>
> They'd tried this three times.
>
> Ponder looked again at the single sentence Hex had written.
>
> +++ Mine! Waaaah! +++
>
> 'I don't actually think,' he said, gloomily, 'that I want to tell the Archchancellor that this machine stops working if we take its fluffy teddy bear away. I just don't think I want to live in that kind of world.'
>
> 'Er,' said Mad Drongo, 'you could always, you know, sort of say it needs to work with the FTB enabled . . . ?'
>
> 'You think that's better?' said Ponder, reluctantly. It wasn't as if it was even a very realistic interpretation of a bear.
>
> 'You mean, better than "fluffy teddy bear"?'
>
> Ponder nodded. 'It's better,' he said.
>
> Terry Pratchett, Hogfather (1996)

How do we ensure that a computer program is correct? This question is as old as computer science itself. To obtain an answer it must first be established what it means for a program to be correct. We can agree that the program should not crash – we want to avoid any blue screens of death, or images of bombs with an accompanying restart button. But that is only part of the story. Most computers are not the types found on desktops, but small embedded devices that control the functions of cars, airplanes, trains, medical equipment, or MP3 players. A valid requirement of the software in a car is that in the case of a collision, the airbag is inflated within five hundredths of a second, and not within five seconds; if a piece of medical equipment is distributing medicine, the correct amount of the drug must be administered, possibly over a period of time; as for the MP3 player, it should not play those favourite songs at dangerously loud levels. Moreover, modern computers require that several programs run simultaneously on the same machine, and interact with each other in desired ways only, but when hundreds or even thousands of programs are running at the same time, the sheer number of possible interactions quickly becomes overwhelming. The Internet also imposes requirements on software. For instance, any transactions with an Internet bank is required to be secure – no one should be able

to eavesdrop, learn any authentication codes, or empty the accounts. The requirements that programs must be able to share resources with others and withstand attacks from malicious users add a level of complexity not present for programs running in isolation. This thesis focuses on how such parallel systems can be modeled in simple intuitive ways, and how to prove with absolute certainty that a program behaves the way it should. Consider the following analogy:

We have been constructing bridges for thousands of years. In the beginning they were small, just big enough to allow people to cross. As expertise increased we learned how to build sturdier bridges that would support more weight, such as that of carriages and horses, and today we are building huge technological marvels that transport thousands of cars and hundreds of trains every day. We have been writing computer programs for a bit over sixty years, and the lack of several thousands of years of experience is apparent. When a bridge is built, there are extensive planning phases, blueprints, and mathematical calculations to ensure that all parts of the bridge will support the weight of whatever we are putting on it. When the bridge is completed we are confident that it will not topple into the ocean when the first train drives across. When a computer program is created, in the worst case scenario, the programmer gets a sloppily written description of what it is to do, the program is written in a rush since the deadline was yesterday, and then fingers are crossed.

Often circumstances are better than this, but the fact is that we do not know how to make blueprints for software of the complexity being written today. A modern programmer is more of a craftsman and an artist than an engineer – the correctness of a program is inferred from experience and careful attention to detail, rather than from mathematical rigour. There is a distinct gap between the programs being developed and the theories that are designed to prove their correctness.

The purpose of this thesis is to reduce this gap. Process calculi is an area of computer science designed to provide blueprints for concurrently running programs. The contribution is twofold. Firstly, we provide computer verified proofs of theorems for existing process calculi; the proof strategies are general enough to be used for calculi of varying complexity. Secondly, we extend the state of the art by introducing a framework of calculi that encompasses several existing ones, but which is substantially simpler and more expressive.

1.1 Formal methods

Formal methods use mathematical models of programs and programming languages and are created in such a way that many desired properties of programs can be proven with absolute certainty. They are extensively used

in industrial applications. Airbus uses the SCADE suite from Esterel technologies to generate software for their aircraft [24]; the Paris Metro line 14 shuttles Parisians every day without a driver, and it had its software verified using the B method [9]; NASA has a Laboratory for Reliable Software (LaRS), which was created in 2003, and are actively researching means to make the software used in the space program more reliable [4]. The focus lies on proving that a program will avoid certain undesired behaviours, such as using too much memory or consuming resources required by other programs.

Software in embedded systems is typically smaller and more tailored to do one specific thing, and analysing it is therefore not as daunting as for bigger computer systems. Moreover, there are often economic incentives to ensure that the software in cars, medical equipment, or rockets actually works. In 1999 NASA lost a $125 million Mars orbiter because the software confused English imperial units of measurements with those of the metric system [1], and in 1996 an Ariane 5 rocket and its cargo, worth a total of $375 million, exploded because of a software error [37]. More recently, in 2010 Toyota announced that they would recall approximately 400 000 of their Prius hybrid cars due to a software glitch that causes poor performance of the anti-lock breaks [2].

Clearly there is a lot of money to be made by ensuring that programs function the way they should from the start. For several years, a research group at NICTA laboured to prove a micro-kernel for an operating system correct [50]; a mathematical model was written which detailed the exact desired behaviour of the kernel, and the code was then proven to correspond precisely to this model. The program is around 7500 lines of C code, and the effort was roughly 40 man years. Techniques were developed along the way to make these types of tasks simpler in the future, but the amount of work required to prove full functional correctness of a system, i.e. ensuring that the system conforms completely to its specification, remains gargantuous. Still, this project proves a point – it is becoming increasingly realistic to verify complete software systems.

One difficulty with software verification is that the programming languages that the computer understands are not the same as the mathematical languages suited for proofs. A common approach when proving a program correct is to formulate a model of its algorithms, using some high level language, and prove that model correct. One problem then is translating the model to a programming language, as there is always the risk that this translation is incorrect. Moreover, simplifications are often made, for example by ignoring the possibility of running out of memory. This is not necessarily a bad thing. If a model is simple and easy to understand then it is easier to prove that the program does what it is supposed to. It is important to ensure that the simplifications are safe – just because an algorithm is correct if it is assumed to have infinite amounts of memory at its disposal, there is

no a priori guarantee that it will work with the finite memory of a computer, or in conjunction with other programs which might be running at the same time.

Another problem with software verification is that even if a program has been completely verified, and contains no mistakes, the language it is implemented in can be incorrect. Usually there are extensive reference manuals that describe in detail what each command of the language does. These are often written in English, which as any human language is subject to interpretation. It is not uncommon that the same programming language is interpreted differently by different computers.

An alternative to the reference manuals is to use a formal semantics for the programming language. The semantics provides a mathematical description of each command of the language, and makes it possible to prove general properties, such that a particular command always has a desired effect. Without a semantics, the correctness of programs cannot be proven – it is not possible to mathematically prove correctness of something that cannot be mathematically interpreted. Still, most modern programming languages, like C, Java, Erlang, or Scala, do not have a formal semantics, and language designers do not have program verification in mind when designing programming languages.

By reverse engineering a formal semantics, software written in these languages can still be verified. These semantics generally do not encompass the full expressive power of the programming language but they are expressive enough to prove correctness of simpler programs. The micro-kernel mentioned above, which is written in C, is just one example.

In this thesis we will focus on the design of high level languages targeted at parallel systems. We will provide semantics for these languages, discuss what properties need to be proven and why. Moreover, we will ensure that these proofs are correct with absolute certainty by having them checked by a computer.

1.2 Parallel systems

Parallel systems are notoriously difficult to formalise. A sequential program running in isolation has unique access to the resources of the machine it is running on, and keeping track of the state of the system with each command is relatively straightforward; a parallel program must share its resources with other programs running at the same time, making it more difficult to determine the state of the system at any given time, and hence also the effect of each command.

The difficulty to check whether a program has the desired behaviour is only one side of the coin; it is often difficult to write specifications for parallel systems for much the same reasons – the state space of a system with

many parallel components is too large to account for in a program, and it is very difficult, if not impossible, to get a good view of how a parallel system will react at any point in time. A famous example is the Needham-Schröder public key protocol [63] from 1978. This protocol is designed such that two parties can communicate with each other using encrypted messages. A trusted server is used to set up the communication, and manages the encryption keys of the parties. This protocol was proven to be insecure by Denning and Sacco in 1981 [35] – a malicious third party could crack the protocol and take the place of one of the original trusted parties, compromising the system. The Needham-Schröder protocol is not particularly large, but it still took three years to find the bug and fix it.

One reason that the bug was not found sooner was that there were few formalisms to reason about parallel programs. Dijkstra had created a variant of ALGOL60 with a parallel construct in the language [36], and Hoare extended on these ideas with his theory of Communicating Sequential Processes (CSP) [27].

1.3 Process calculi

In 1980, Milner introduced a new field of research which today is commonly referred to as process calculi, or process algebras, with his Calculus of Communicating Systems (CCS) [55]. Process calculi are a family of related formalisms that provide high level descriptive languages to reason about concurrent systems. They also introduce a concept of equality between processes, and provide algebraic laws to reason about these equalities. One such equality is bisimilarity, and its intuitive definition is that two processes P and Q are bisimilar, written $P \sim Q$, if for every action that any of the processes can do, the other can do the same action, and the states they end up in are still bisimilar. An example of an algebraic law is the compositionality law which states that if two processes are bisimilar, $P \sim Q$, then the processes resulting from putting another process in parallel with these processes are also bisimilar, $P \mid R \sim Q \mid R$.

CCS was groundbreaking in that it introduced a formalism for comparing programs based on how they communicate – which data is sent, which data is received, and where do the programs go from there. It is a minimalistic formalism with only a few basic operators – most notably processes may run in parallel, and they can contain local information not available to any other process. CCS will be described in detail in Part II of this thesis.

The pi-calculus was introduced by Milner, Parrow, and Walker in the late 1980s [58]. A pi-calculus process has the capability to create a local communication channel, which only that process knows about, and which can be sent to another process allowing for secure communication between the two. The pi-calculus will be covered in detail in Part III.

Process calculi to date have been used extensively to model communication protocols. Many protocols make use of cryptography to be able to send information over an insecure medium where anyone can intercept messages, and be confident that only the intended recipient can decipher and read the message. In 1999, Abadi and Gordon introduced the spi-calculus [8], which included cryptographic primitives such as encryption and decryption as primitive operators of the calculus.

The spi-calculus has been used to verify a number of security protocols. Its algebraic properties are more complicated than previous calculi. For the pi-calculus and CCS, equality on processes is inferred just by looking at how the processes interact with the environment; a spi-calculus process must also keep track of information available to each process, as the knowledge of cryptographic keys admits decryption of messages. There is a multitude of different equivalences for spi-calculus processes, each suited for slightly different tasks [26].

Many process calculi are tailored to solve a specific problem. This is problematic as it invariably leads to duplication in proof effort – whenever a new calculus is created, all of its proofs must be redone, and these are often very similar to corresponding proofs in previous calculi. Moreover, as the complexity of the calculi increases, so does the complexity of their proofs. There is therefore a need for frameworks that encompass a wide range of applications and calculi.

The applied pi-calculus was introduced by Abadi and Fournet in 2001 [7]. It was novel in the sense that the user supplies what data processes can use; some examples are linked list, binary trees, or encrypted or decrypted messages. The user also supplies an equation system to reason about the data. A typical equation could state that

$$\mathtt{dec}(\mathtt{enc}(M,\ k),\ k) = M$$

which means that a message M encrypted with a key k can be decrypted with the same key. This generality allows the applied pi-calculus to model the same cryptographic protocols as the spi-calculus, and it does this with a leaner algebraic theory.

The applied pi-calculus is extensively used, with hundreds of papers citing it, but one of its semantics was discovered to be non-compositional in 2009 [17]. The fact that such a widely used calculus can still have a bug in it after eight years hints at the difficulty of the proofs involved.

1.4 Theorem proving

Pen and paper proofs are often plagued by sweeping statements such as: *from Definition A we can clearly see that ...*, or *the proof follows trivially by*

induction on x. These styles of proofs make use of the human intuition to deduce what is *clear* or *trivial*, but care has to be taken to ensure that these simplifications do not introduce any inconsistencies or flaws in the proof.

The main point of any mathematical proof is to form a convincing argument so that with a reasonable degree of certainty, the proof is correct, no cases have been missed, and all appeals to intuition are safe. As the complexity of the proofs increases, this becomes more and more time consuming and increasingly error prone. Therefore, in order to ensure that a proof actually is correct, it is desirable to have the proofs checked by a theorem prover.

A theorem prover is a computer program, that given a proof in a language the prover understands can check if the proof is correct. There are many advantages of using theorem provers. Primarily they are used to ensure that all proofs actually are correct and no cases have been overlooked, but that is only half the story. Once a theory has been proven correct inside a theorem prover, the user can make changes, and the ramifications of these changes become instantly apparent. Consider doing the same to a big pen-and-paper formalisation – it would be nearly impossible to foresee all possible effects that a change has on different parts of the formalisation, except by redoing all of the proofs. This process would be time consuming, boring, and the risk of doing a mistake is far from negligible.

There exist several theorem provers: Coq [25], Isabelle [64], Agda [3], PVS [66], Nuprl [31] and HOL [42], just to name a few. These theorem provers are interactive. They have many automated tactics, and the user can provide additional proof strategies. Many are also getting better and easier to use, and so the concept of having fully machine checked proofs has recently become far more realistic. As an indication of this, several major results have been proven over the last few years, including the four and five colour theorems [14, 41], Kepler's conjecture [65] and Gödel's incompleteness theorem [72]. Significant advances in applications related to software are summarized in the POPLmark Challenge [11], a set of benchmarks intended both for measuring progress and for stimulating discussion and collaboration in mechanizing the meta-theory of programming languages. There are, for example, results on analysis of typing in System F and light versions of Java. The theorem prover Isabelle was recently used to verify software in the Verisoft project [5]. Moreover, the verification of the operating system micro-kernel discussed previously was verified using Isabelle.

A common criticism of theorem provers is that they are hard to use and the amount of work required to formalise proofs significantly exceeds doing them on paper. The reason for this is mainly that it is difficult to model human intuition in a straightforward way – for a theorem prover, nothing is *clear* or *trivial*, and a lot of time has to be spent proving the seemingly obvious. However, the reason that intuitive truths are difficult to model can be that they are actually not true. One famous such example is the Baren-

dregt variable convention, which intuitively states that the names chosen for arguments of functions are unimportant. It will be discussed in detail in Section 3.1.1.

So whereas the argument that theorem provers are difficult to use and require a considerable amount of work has merit, the fact is that they provide a robust way of ensuring that a formalisation is correct, and they provide a flexible working environment where theories can be modified without running the risk of introducing inconsistencies. Moreover, modern theorem provers are becoming increasingly powerful and easy to use.

1.5 Contributions

The main contribution of this thesis is to formalise the meta-theory of different dialects of process calculi in a theorem prover. I have created extensive formalisations of three major process calculi: CCS [57] by Milner, the pi-calculus [58] by Milner, Parrow, and Walker, and the psi-calculi [17] by myself, Johansson, Parrow, and Victor. These calculi vary greatly in complexity, but the proof strategy used to formalise their meta-theories is the same, and have scaled remarkably well as complexity increases.

Another main contribution is the psi-calculi framework, which was developed in our research group. I participated in the development at the same time as I formalised all theories in Isabelle. In this way the framework was formalised in parallel with its development. Psi-calculi represents the current state of the art of process calculi. We believe it to be one of the most expressive frameworks for concurrent systems currently available, and its formalisation in Isabelle to be the most extensive formalisation of process calculi ever done in a theorem prover.

Every proof in this thesis has been machine checked using the interactive theorem prover Isabelle – all definitions have been encoded, and all lemmas and theorems have been proven. The advantage of this is clear – we know that our proofs are correct and that nothing has been overlooked. Isabelle also provides support for typesetting the theories which have been proven. All lemmas in this thesis are generated directly from the Isabelle sources, significantly reducing the risk that the formulas presented contain errors.

1.6 Thesis outline

This thesis is composed of five parts. Part I serves as an introduction, and provides the technical background required for the rest of the thesis. A reader familiar with the subjects may want to skip some or all of the chapters presented. Part II describes how Milner's Calculus of Concurrent Systems (CCS) [55] is formalised in Isabelle, and Part III does the same

for the pi-calculus [58]. Part IV introduces and formalises psi-calculi – a general framework which captures both CCS, the pi-calculus and many others. Part V concludes the thesis.

1.6.1 Part I: Background

Chapter 2 introduces process calculi, their background, structure, and applications. A reader familiar with process calculi may still want to read Sections 2.4 and onwards, as they cover the proof strategies that are used for the rest of the thesis.

Chapter 3 introduces the concept of alpha-equivalence, how it is used in process calculi, and different attempts to provide a smooth treatment in theorem provers.

Chapter 4 describes Nominal Logic [69], which provides the logical infrastructure upon which the rest of the thesis builds.

Chapter 5 describes the interactive theorem prover Isabelle, and covers the required material for understanding the Isabelle proofs presented in this thesis.

1.6.2 Part II: The Calculus of Communicating Systems

Chapter 6 introduces the semantics of CCS, some example derivations, and how the semantics is modeled in Isabelle.

Chapter 7 defines strong bisimilarity – an equivalence relation that equates processes having the same behaviour.

Chapter 8 defines structural congruence – an equivalence relation that equates processes that are intuitively considered equal. One such example is that processes differing only by the order of their parallel components are equal. Moreover, we prove that all structurally congruent terms are bisimilar.

In Chapter 9 we define weak bisimilarity, which is an equivalence relation similar to strong bisimilarity, but it abstracts away from the internal actions of the processes. We also prove that all strongly bisimilar processes are weakly bisimilar.

1.6.3 Part III: The pi-calculus

Chapter 12 introduces the pi-calculus, its history and impact.

There are two types of operational semantics for the pi-calculus – the early semantics, and the late one. In Chapter 13 we describe the early operational semantics, and how it is implemented in Isabelle. All the following chapters up to Chapter 17 use the early semantics.

In Chapter 14 we define strong bisimilarity for the early semantics of the pi-calculus.

In Chapter 15 we define weak bisimilarity.

In Chapter 16 we define weak congruence.

In Chapter 17 we define the late semantics of the pi-calculus, we also define all the equivalences from the early semantics and prove the corresponding results.

In Chapter 18 we define structural congruence for the pi-calculus, and prove that all structurally congruent processes are also late bisimilar.

In Chapter 19 we prove that the axiomatisation of strong late bisimilarity for the finite pi-calculus is sound and complete.

In Chapter 20 we prove that all late bisimilar processes are also early bisimilar.

1.6.4 Part IV: Psi-calculi

In Chapter 22 we provide an in depth exposition of parametric process calculi. We also introduce the psi-calculi framework including its strong bisimulation equivalences.

In Chapter 23 we introduce the notion of binding sequences – a mechanism for treating sequences of binders atomically, rather than working with one binder at a time. The concept of binders in process calculi is defined in Chapter 2.

In Chapter 24 we provide the Isabelle definitions for psi-calculi processes, and cover how the parametricity of the framework is encoded in Isabelle.

Chapter 25 covers the operational semantics of psi-calculi, as well as the rules used to do induction over the transition system.

In Chapter 26 we describe a technique to derive general inversion rules for calculi using binding sequences. Inversion rules are used for case analysis on transitions of the calculi.

In Chapter 27 we model strong bisimilarity in Isabelle.

Chapter 28 covers the structural congruence rules for psi-calculi, proves that all bisimilar processes are also structurally congruent, and that bisimilarity is a congruence.

Chapter 29 describes weak bisimilarity for psi-calculi. Weak bisimilarity is considerably more complex than for other process calculi, and motivating examples are provided as to why this is the case. We also define a subset of psi-calculi, where the logical environment satisfies weakening, i.e. that nothing known by the environment can be made untrue by adding extra information.

In Chapter 30 we formalise weak bisimilarity for arbitrary psi-calculi in Isabelle.

In Chapter 31 we define weak congruence for psi-calculi and prove that it is a congruence.

In Chapter 32 we add logical weakening to the psi-calculi framework, define the simpler version of weak bisimilarity and prove that the two versions coincide.

In Chapter 33 we discuss extensions to the psi-calculi framework, and encode new operators by adding extra constraints to the framework.

In Chapter 34 we compare psi-calculi to other calculi, and provide the counter-examples to why the semantics for the applied pi-calculus and CC-pi are not compositional. We also discuss in detail our experiences from formalising a framework parallel to the development of its theories.

1.6.5 Part V: Conclusions

The thesis is concluded with a discussion on what has been achieved and learned through the formalisation efforts. We cover possible extensions to Isabelle to make these types of formalisations easier. We come back to related work, what other process calculi have been formalised in theorem provers, and which techniques were used. We also discuss possible future work.

1.7 My publications

I have published eleven articles with different constellations of people, but mostly with my supervisor and the rest of my research group. This thesis builds on eight of these articles, where two are journal versions of conference articles.

1.7.1 Articles contributing to this thesis

1. Jesper Bengtson. Generic implementations of process calculi in Isabelle. In *The 16th Nordic Workshop on Programming Theory (NWPT'04)*, pages 74–78, 2004.
2. Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using Nominal Logic. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 4423 of *LNCS*, pages 63–77, 2007.
3. Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2008.
4. Jesper Bengtson and Joachim Parrow. A completeness proof for bisimulation in the pi-calculus using Isabelle. *Electronic Notes in Theoretical Computer Science*, 192(1):61–75, 2007.
5. Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of LICS 2009*, pages 39–48. IEEE, 2009.

6. Jesper Bengtson and Joachim Parrow. Psi-calculi in Isabelle. In *Proceedings of TPHOLs 2009*, volume 5674 of *LNCS*, pages 99–114. Springer, 2009.
7. Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: A framework for mobile processes with nominal data and logic Submitted to *LICS 2009 special issue of LMCS*.
8. Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Proceedings of LICS 2010 (to appear)*. IEEE, 2010.

In Article 1 I formalised an extensive part of the meta-theory of Milner's CCS. All of Part II stems from this article.

Article 2 presents the formalisation of the pi-calculus by Milner Parrow and Walker. We later published Article 3, which is a journal version of Article 2. All of Part III except Chapter 19 is based on these two articles.

In Article 4 we extended the pi-calculus formalisation to include the axiomatisation of strong bisimilarity for the finite subset of the pi-calculus. Chapter 19 builds on results from this article. I alone wrote all of the Isabelle formalisations in Articles 1-4. The articles are written with my supervisor.

In Article 5 we introduce psi-calculi. This is a group effort where the theories were developed in parallel with my formalisation efforts. The only part I did not have hand in is Section 3, which explains how to encode other process calculi using psi-calculi. Chapter 22, and Sections 34.1.1 and 34.1.2 borrow heavily from this article.

In Article 6 we formalised all results of Article 5. I wrote the complete formalisation. Chapters 23-28 are based on this article.

Article 7 is a journal version of Article 5. I participated significantly in all parts except Sections 3 and 4.

In Article 8 we present weak equivalences for psi-calculi. I am responsible for all of the Isabelle formalisation and participated significantly in all parts except Section 6 on barbed congruence. Chapter 29 borrows heavily from this article. All of the results of Article 8, except the barbed equivalence, have been formalised by me in Isabelle, and the results are presented in Chapters 30-33. This work is still unpublished.

1.7.2 Other publications

The following are articles which I have coauthored during my Ph.D. but which do not appear, or are only briefly touched upon in the dissertation.

1. Michael Baldamus, Jesper Bengtson, Gianluigi Ferrari, and Roberto Raggi. Web services as a new approach to distributing and coordinating semantics-based verification toolkits. *Electronic Notes of Theoretical Computer Science*, 105:11–20, 2004.
2. Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security*

Foundations Symposium, pages 17–32, Washington, DC, USA, 2008. IEEE Computer Society.

3. Magnus Johansson, Joachim Parrow, Björn Victor, and Jesper Bengtson. Extended pi-calculi. In *Proceedings of ICALP 2008*, volume 5126 of *LNCS*, pages 87–98. Springer, July 2008.

In article 1 we presented a framework of tools for formal methods on the web – the idea was to have them collaborate with each other and use each other's results. I wrote a few pages for this article, but the scientific work was done by my coauthors.

Article 2 was written as a part of a three month internship at Microsoft Research in Cambridge. We created a security type system for F#, which is the .NET version of ML. The type system uses refinement types with logical predicates, and a type safe program is secure in the sense that e.g. only trusted parties can decrypt messages being sent. The type system is expressive enough to verify other safety properties as well. Programs are annotated with types and the type checker sends proof obligations to an automatic theorem prover. At the time we used SPASS [78], but later upgrades uses Z3 [62]. I implemented the type checker, and coauthored the theories.

Article 3 describes extended pi-calculi, a precursor to the psi-calculi framework. The theoretical development was a group effort, most of my work was to build the infrastructure in Isabelle required to formalise calculi of this caliber. Psi-calculi does everything that extended pi-calculi does and more, and in a more elegant way. Work on extended pi-calculi has been abandoned.

Part I:
Background

2. Process calculi

Process calculi, introduced in the early 1980s, were pioneered by Milner with the Calculus of Communicating Systems (CCS). The main contribution of CCS is that it provides a clear and intuitive way to reason about parallel systems in terms of their interactions with the environment.

This chapter introduces a simple process calculus which is used to cover the basic concepts of process calculi, their terminology, and the proof strategies that will be used throughout this thesis. This calculus is intended only for explanatory purposes, and is not practically useful as a modeling language – calculi which are suited for this purpose will be covered in Parts II, III, and IV.

2.1 Syntax

Process calculi use names, which are an infinite number of atomic building blocks, to build the data structures required by the calculus. There is also a notion of actions that can be performed by the agents, which will henceforth be denoted as agents. This thesis will use the following notation.

- Names are denoted by a, b, c, ...
- Agents are denoted by P, Q, R, ...
- Actions are denoted by α, β, γ, ... and represent the visible capabilities of an agent.

In our simple process calculus, actions are defined as follows:

Definition 2.1 (Actions)**.**

$$\alpha \stackrel{\text{def}}{=} \tau \mid a$$

A τ-action represents an internal action of an agent, whereas an action consisting of a name is visible to the environment.

Agents can now be defined in the following way:

Definition 2.2 (Agents)**.**

$$\begin{array}{lll} P \stackrel{\text{def}}{=} & \alpha . P & \textit{Prefix} \\ & P \mid Q & \textit{Parallel} \\ & (\nu x) P & \textit{Restriction} \\ & \mathbf{0} & \textit{Nil} \end{array}$$

The structural congruence $\equiv$ is defined as the smallest congruence satisfying the following laws:

1. The abelian monoid laws for Parallel: commutativity $P \mid Q \equiv Q \mid P$, associativity $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and $\mathbf{0}$ as unit $P \mid \mathbf{0} \equiv P$; and the same laws for Sum.
2. The scope extension laws

$$\begin{array}{lcll} (\nu x)\mathbf{0} & \equiv & \mathbf{0} & \\ (\nu x)(P \mid Q) & \equiv & P \mid (\nu x)Q & \text{if } x \sharp P \\ (\nu x)\alpha.P & \equiv & \alpha.(\nu x)P & \text{if } x \sharp \alpha \\ (\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P & \end{array}$$

Figure 2.1: The definition of structural congruence.

The empty agent, denoted $\mathbf{0}$, represents a deadlocked agent i.e. an agent with no actions. An agent P running in parallel with an agent Q is denoted $P \mid Q$. An agent $\alpha.P$ can do the action α and then become P. An agent can generate names local to that agent through a ν-operator, where the agent $(\nu x)P$ denotes an agent P with the name x local to it – intuitively, x may not occur in any other agent.

The *free names* are the names in an agent except those restricted by Restriction. The term $x \sharp P$, pronounced *x fresh for P*, means that x is not in the free names of P. An exact definition of this operator, and a discussion of its origins, will be given in Chapter 4.

2.2 Structural congruence

Structural congruence is an equivalence relation that relates agents which are syntactically different, but intuitively considered equal. For instance, it is reasonable to assume that the parallel operator is associative and commutative and that restricting a name in an agent where that name does not exist has no effect. The structural congruence rules can be found in figure 2.1.

2.3 Operational semantics

The notation $P \xrightarrow{\alpha} P'$ is used to represent an agent P doing an action α and ending up in the state P'. The agent P' is often referred to as an α-derivative of P or just a derivative of P.

$$\frac{P \equiv Q \qquad Q \xrightarrow{\alpha} Q' \qquad Q' \equiv P'}{P \xrightarrow{\alpha} P'}\ \textsc{Struct} \qquad \frac{}{\alpha.P \xrightarrow{\alpha} P}\ \textsc{Action}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par} \qquad \frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Sync}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ \textsc{Scope}$$

Figure 2.2: An operational semantics for a simple process calculus.

The operational semantics is a collection of rules through which transitions can be inferred, and can be found in Fig. 2.2. The STRUCT rule can be used to rewrite an agent or its derivatives to structurally congruent counterparts. The ACTION rule allows an agent $\alpha.P$ to do an α-action and end up in the state P. The PAR rule allows the agent P in $P \mid Q$ to do an action while Q does nothing. If Q does an action, a symmetric version of this rule can be inferred through the use of STRUCT. The SYNC rule allows two agents P and Q to synchronise provided they have the same action. The SCOPE rule is designed to block actions containing names which are local to the agents. An agent $(\nu x)P$ can only do an action α if x does not occur free in α. Since alpha is just a name or τ, this means that $x \neq \alpha$.

2.4 Bisimilarity

Intuitively, two agents are said to be bisimilar if they can mimic each other step by step. Traditionally, a bisimulation is a symmetric binary relation $\mathscr{R}$ such that for all agents P and Q in $\mathscr{R}$, if P can do an action, then Q can mimic that action and their corresponding derivatives are in $\mathscr{R}$. The largest such bisimulation is denoted $\sim$, i.e. a P being bisimilar to an agent Q is written $P \sim Q$.

There is a multitude of different bisimulation relations for the different kinds of process calculi in existence, ranging from the very simple to the very complex. This section introduces the proof strategies that will be used for the rest of this thesis. When designing process calculi it is important to use a congruence – i.e. an equivalence relation preserved by all operators. For an operator to preserve a bisimilarity, it must be the case that applying the operator to two bisimilar agents will not produce two agents which

are not bisimilar. For instance, if the fact that P and Q are bisimilar implies that also $(\nu x)P$ and $(\nu x)Q$ are bisimilar, then bisimilarity is preserved by Restriction. The property that a bisimilarity is preserved by an operator is called a preservation property.

Congruences have the advantage that they are preserved by all operators, which ensures that any part of an agent can be replaced by a congruent one without changing its behaviour. This allows specifications and implementations to be designed modularly – a specification for the entire system can be created, but bisimilarity must only be proven for each subcomponent, they can then be freely interchanged and the result is still guaranteed to be bisimilar.

An important application area for process calculi is security protocols. A specification will generally require that no private information is leaked to the environment. If bisimilarity is preserved by the parallel operator, the bisimilar agents will behave the same even in the presence of an arbitrary attacker running in parallel.

Formally, an agent P can simulate an agent Q in a relation $\mathscr{R}$, if for every transition Q can do, P can mimic that transition and the derivatives are in $\mathscr{R}$. We use the terminology that a simulation preserves $\mathscr{R}$ if the derivatives of all possible simulations are in R.

Definition 2.3 (Simulation)**.** *An agent P simulating an agent Q preserving $\mathscr{R}$ is written $P \hookrightarrow_{\mathscr{R}} Q$*

$$P \hookrightarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R})$$

Bisimilarity can then very conveniently be defined coinductively, i.e. the greatest fixed point derived from a monotonic function.

Definition 2.4 (Bisimilarity)**.** *Bisimilarity, denoted $\sim$, is defined as the greatest fixed point satisfying:*

$$\begin{aligned} P \sim Q \Longrightarrow P &\hookrightarrow_{\sim} Q && \textsc{Simulation} \\ \wedge\ Q &\sim P && \textsc{Symmetry} \end{aligned}$$

Proving that two agents are bisimilar boils down to choosing a symmetric candidate bisimulation relation $\mathscr{X}$ containing the two agents, and proving that for all $(P, Q) \in \mathscr{X}$, $P \hookrightarrow_{\mathscr{X}} Q$.

2.5 Weak bisimilarity

Weak bisimilarity abstracts from the τ-actions. The idea is that two agents are bisimilar if they can mimic each other's visible actions, ignoring all internal computations.

$$\frac{P \equiv Q \qquad Q \overset{\alpha}{\Longrightarrow} Q' \qquad Q' \equiv P'}{P \overset{\alpha}{\Longrightarrow} P'}\ \textsc{Struct} \qquad \frac{}{\alpha.P \overset{\alpha}{\Longrightarrow} P}\ \textsc{Action}$$

$$\frac{P \overset{\alpha}{\Longrightarrow} P'}{P \mid Q \overset{\alpha}{\Longrightarrow} P' \mid Q}\ \textsc{Par} \qquad \frac{P \overset{\alpha}{\Longrightarrow} P' \qquad Q \overset{\alpha}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'}\ \textsc{Sync}$$

$$\frac{P \overset{\alpha}{\Longrightarrow} P' \qquad x \sharp \alpha}{(\nu x)P \overset{\alpha}{\Longrightarrow} (\nu x)P'}\ \textsc{Scope}$$

Figure 2.3: A lifted weak operational semantics. All rules are derived from the strong semantics found in Figure 2.2.

An agent P can do a τ-chain to P', written $P \Longrightarrow P'$ if P and P' are in the reflexive transitive closure of τ-actions from P.

Definition 2.5 (τ-chain)**.**

$$P \Longrightarrow P' \overset{\text{def}}{=} (P, P') \in \{(P, P') : P \xrightarrow{\tau} P'\}^*$$

A weak transition, written $P \overset{\alpha}{\Longrightarrow} P'$ is defined as a strong transition with a τ-chain appended before and after the action.

Definition 2.6 (Weak transition)**.**

$$P \overset{\alpha}{\Longrightarrow} P' \overset{\text{def}}{=} \exists P'' P'''.\ P \Longrightarrow P'' \wedge P'' \xrightarrow{\alpha} P''' \wedge P''' \Longrightarrow P'$$

Definition 2.7 (Weak simulation)**.** *An agent P weakly simulating an agent Q preserving $\mathscr{R}$ is written $P \rightsquigarrow_{\mathscr{R}} Q$*

$$P \rightsquigarrow_{\mathscr{R}} Q \overset{\text{def}}{=} \forall \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \overset{\alpha}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R})$$

It is important to note that in weak simulations, a weak action mimics a strong one.

Definition 2.8 (Weak bisimilarity)**.** *Weak bisimilarity, denoted $\approx$, is defined as the greatest fixed point satisfying:*

$$\begin{aligned} P \approx Q \Longrightarrow P &\rightsquigarrow_{\approx} Q && \textsc{Simulation} \\ \wedge\ Q &\approx P && \textsc{Symmetry} \end{aligned}$$

Proving properties of weak bisimilarity is more involved than proofs for strong bisimilarity as the τ-chains must be taken into consideration. In or-

der to abstract from this added complexity, we introduce the concept of lifting. A strong semantic rule can be lifted, if all of its strong transitions can be replaced by weak ones. The semantics in Figure 2.3 illustrate this.

If a semantic rule can be lifted, it can be used in the same way as its strong counterpart, and the proof strategies which use strong semantic rules can also use the weak ones. This significantly cuts down on the amount of work required to formalise properties of weak bisimilarity, as the proofs for strong bisimilarity can be reused, modulo changing which semantic rules are used.

2.6 Structural congruence revisited

In this chapter we have introduced process calculi through a simple example with a structural congruence rule in the semantics. In reality, this is not always a good design decision. The arguments in favour are that the semantics becomes leaner and easier to understand.

The main disadvantage is that whenever a proof involving the semantics is done, it is not enough to consider the agents at hand, but all structurally congruent agents must also be considered. This makes the proofs more difficult and mare cumbersome to work with. Consider as an example the following lemma.

Lemma 2.9. *If* $P \xrightarrow{\alpha} P'$ *and* $x \sharp P$ *then* $x \sharp P'$.

Proof. By induction on the transition $P \xrightarrow{\alpha} P'$ □

In the STRUCT case, an auxiliary lemma is needed to show that the structural congruence laws introduce no new fresh names.

Lemma 2.10. *If* $P \equiv Q$ *and* $x \sharp P$ *then* $x \sharp Q$.

Proof. By induction on the construction of $P \equiv Q$.

The problem arises in the case for symmetry of structural congruence ($P \equiv Q \longrightarrow Q \equiv P$). The induction hypothesis provides $x \sharp Q$, but the proof requires that $x \sharp P$. The solution is to strengthen the induction hypothesis to $x \sharp P \longrightarrow x \sharp Q \wedge x \sharp Q \longrightarrow x \sharp P$. □

This proof is moderately easy but it is inconvenient to prove structural congruence properties for every proof on the transition system. Moreover, case analysis on a semantics with structural congruence is complicated. For every transition, every structurally congruent agent which could trigger the transition must be considered. For instance, the transition $P \mid Q \xrightarrow{\alpha} P'$, can be derived from eight cases – one each from the PAR and COMM rule, and six from structural congruence – reflexivity, symmetry and transitivity, and

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \textsc{Action} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Sync} \qquad \frac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ \textsc{Scope}$$

Figure 2.4: A STRUCT-free operational semantics for a simple process calculus.

the three abelian monoid laws. For more advanced calculi, this number is even greater.

This problem becomes worse when using a theorem prover which will require you to prove all steps, even if they are similar, when it cannot prove them automatically. Figure 2.4 shows a STRUCT-free version of the operational semantics.

Even though the semantics does not contain structural congruence, it must be possible to derive the structural congruence rules. More precisely, any terms which are structurally congruent must also be bisimilar using semantics without a STRUCT rule.

3. Alpha-equivalence

When defining process algebras or programming languages, the notion of binders must be made precise. Depending on the calculus being formalised, binders serve different functions. The most common notion is for a binder to be a name which acts as a placeholder for terms, and during execution, this placeholder can be instantiated and replaced by an arbitrary term. For process algebras, it is also common to have binders represent local names for an agent. Two agents which are syntactically equal except for the bound names are called alpha-equivalent and changing the bound names of an agent to other valid bound names is called alpha-conversion.

In the process algebra described in Chapter 2, the only binder is the ν-operator, which conforms to the second use of binders mentioned above. The operator creates a unique name which can only appear under the scope of the binder. Which name is chosen is less important, although some restrictions do apply.

Consider the following three agents.

$$P = (\nu x)(x.z.\mathbf{0} \mid x.z.\mathbf{0}) \qquad Q = (\nu y)(y.z.\mathbf{0} \mid y.z.\mathbf{0})$$

$$R = (\nu z)(z.z.\mathbf{0} \mid z.z.\mathbf{0})$$

Here P and Q are alpha-equivalent as they only differ in that the bound name x has been replaced by y. However, neither P nor Q are alpha-equivalent to R, as the binder z will bind all occurrence of z in R, whereas z occurs free in both P and Q. Restriction binds a name in an agent, and this name may not occur anywhere else in the proof context; if it does, it must be alpha-converted to a name which meets this constraint. To be accurate, it is necessary to manually alpha-convert agents such that these freshness constraints are guaranteed; in practice, proofs often abstract away from the notions of alpha-equivalence altogether.

3.1 Manual proofs with pen and paper

When doing paper proofs, the idiosyncrasies of alpha-equivalence are usually glossed over. Generally, agents are assumed to be equal up to alpha-

equivalence, are implicitly assumed to not contain any bound names which clash in an unwanted way.

In *The pi-calculus* [71] Sangiorgi and Walker write:

> In any discussion, we assume that the bound names of any processes or actions under consideration are chosen to be different from the names free in any other entities under consideration, such as processes, actions, substitutions and sets of names.

Similar reasonings can be found in Parrow's *An introduction to the pi-calculus* [67].

> ... we will use the phrase "$bn(\alpha)$ is fresh" in a definition to mean that the name in $bn(\alpha)$, if any, is different from any free name occurring in any of the agents in the definition.

These and other papers make an implicit assumption that alpha-equivalent agents can be freely exchanged in order to ensure any desired freshness properties of the bound names. Intuitively it is difficult to object to this style of reasoning. If two processes are considered equal, then surely it must be possible to replace one for another without breaking the proofs? As it turns out, this can lead to inconsistencies unless proper care is taken.

3.1.1 The Barendregt variable convention

In his book *The lambda calculus: its syntax and semantics* [13], Barendregt introduces what is now commonly known as the Barendregt variable convention.

> **Variable Convention:** If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

It is this convention that has been used in papers like the ones cited above, but it turns out to be unsound in the general case. In [73] Urban provides a variant of the λ-calculus to demonstrate that this can lead to inconsistencies. By using the ideas from [73], and changing the SCOPE-rule of the semantics in Figure 2.2, a demonstration in process algebras is:

$$\frac{P \xrightarrow{\alpha} P' \qquad y \sharp \alpha}{(\nu x)P \xrightarrow{\alpha} P'} \text{SCOPE}$$

This rule differs only from the original SCOPE rule in that the bound name x is dropped from the derivative. Intuitively, the rule reveals the name which is bound by the binder.

The proof of lemma 2.9 still holds, with the following modification to the SCOPE-case.

Faulty lemma: *If* $P \xrightarrow{\alpha} P'$ *and* $y \sharp P$ *then* $y \sharp P'$.

Proof. By induction on the transition $P \xrightarrow{\alpha} P'$.

case SCOPE – $y \sharp (\nu x)P$ and $P \xrightarrow{\alpha} P'$:
Here the variable convention allows us to chose x such that x is not equal to any other name in the proof context, more specifically $x \neq y$. Moreover, since $y \sharp (\nu x)P$ we get that term $y \sharp P$, and hence with the induction hypothesis, $y \sharp P'$.

□

This lemma does not hold. Since $x \sharp (\nu x)y.x.\mathbf{0}$ and $(\nu x)y.x.\mathbf{0} \xrightarrow{y} x.\mathbf{0}$, the lemma will state that $x \sharp x.\mathbf{0}$, which is clearly not true.

The problem arises since the lemma is dependent on the bound names of the agents for its result, and by allowing the user to freely alpha-convert the agents to fit the constraints of the proof, an inconsistency is introduced.

In [73], Urban et. al. propose a fix to this problem. By requiring that any bound name in an inductive rule does not occur free in its conclusions, and all bound names are mutually distinct, the Barendregt variable convention can be used freely when doing proofs with binders. The exact method for this is made precise in [73], and formalised in the interactive theorem prover Isabelle. Briefly put, in the inconsistent semantics defined above, there exists a transition $(\nu x)y.x.\mathbf{0} \xrightarrow{y} x.\mathbf{0}$, containing the bound name x which occurs free after the reduction, making the transition system invalid for use with the Barendregt variable convention.

3.2 Machine checked proofs

When formalising mathematics, all aspects of the proofs must be made precise. Computers are excellent at checking whether or not a proof is correct or not, but having them create the proofs themselves is difficult. Hand waving techniques such as the Barendregt variable convention are hard to formalise, since the user must make precise exactly what it means for a name to be *sufficiently fresh*. Several ways to have computers treat binders have been proposed.

3.2.1 de Bruijn indices

One of the oldest representations of terms with binders was introduced by de Bruijn in [33]. The key idea is that all names are represented by natural numbers, where a name has a number corresponding to the nesting level of its binder. The agent $(\nu x)(\nu y)x.y.z.\mathbf{0}$ would have the de Bruijn representation $\nu\ \nu$ 2.1.3.$\mathbf{0}$ where x is mapped to the number 2 and y to the number 1, as that is their nesting depth from their respective binder. The name z is mapped to the number 3, but is free since it is greater than the binding depth.

Things become more complicated if a name occurs at different binding depths in different parts of an agent. For instance, the agent $(\nu x)x.((\nu y)x.\mathbf{0})$ has the de Bruijn representation ν 1.ν 2.$\mathbf{0}$, where the number x is mapped to increases as another binder is traversed.

If the semantics of a calculus modifies the structure of the agents, the values for binders must be recalculated, and doing this by hand is tedious. Moreover, the representation is not easy to read for humans.

Still, de Bruijn indices have been extensively used in automatic tools which reason about process calculi such as the Concurrency and Mobility Workbenches [61, 77]. As an internal representation for a computer program, de Bruijn indices work well, as creating the infrastructure which calculates the name mappings is quite straight forward; all calculations regarding binders is done in the background and the user does not have to worry about recalculating the values of the binders. The programs can also easily translate their internal representation of the agents to a more human readable form.

There are also formalisations of process calculi where de Bruijn indices are used. In [45], Hirschkoff proves a substantial part of the meta-theory of the pi-calculus in the interactive theorem prover Coq. The work is extensive, but technical parts regarding agent representation make up for a substantial part of the formalisation. Hirschkoff writes:

> Technical work, however, still represents the biggest part of our implementation, mainly due to the managing of De Bruijn indexes: indeed, as stressed above, the De Bruijn notation, while drastically simplifying work for bound names, requires accuracy in dealing with free names. Of our 800 proved lemmas, about 600 are concerned with operators on free names; ...

3.2.2 Higher order abstract syntax

When coding agents using higher order abstract syntax (HOAS), one treats binders as functions from names to agents, i.e. of type `name->agent`. the formalisations need to ensure that those are avoided.

HOAS has been used to model the pi-calculus in both Coq [46], by Honsell et. al., and in Isabelle by Röckl and Hirschkoff [70]. In [46] the late operational semantics is encoded with late strong bisimilarity. The results include that the algebraic laws presented in [58] are sound where the non-trivial proofs include preservation results for bisimulation and the results for structural congruence. In [70], a special well-formedness predicate is used to filter out the exotic agents.

Another problem is that since abstraction is handled by the meta-logic of the theorem prover, reasoning about binders at the object level can become problematic. In [46] we can read:

> The main drawback in HOAS is the difficulty of dealing with meta-theoretic issues concerning names in process contexts, *i.e.* agents of type `name->agent`. As a consequence, some meta-theoretic properties involving substitution and freshness of names inside proofs and processes, cannot be proved inside the framework and instead have to be postulated.

Early attempts to encode the pi-calculus in the HOL theorem prover also include [60, 54].

3.2.3 Nominal logic

Nominal logic, created by Pitts [69], allow for terms with binders to be described and reasoned about in a very intuitive manner. Moreover, it is natively supported by the Isabelle theorem prover which is why it is the formalism of choice for this thesis. Nominal logic is covered in detail in Chapter 4.

Fraenkel Mostowski set theory (FM set theory) was one of the first serious attempts to formalise nominal logic. It is standard ZF set theory but with an extra freshness axiom added. In [38], Gabbay formalises a portion of the pi-calculus in FM set theory. In this approach a N-quantifier (new quantifier) is used to generate names which are fresh for the current context. Gabbay also started work on incorporating a framework for FM set theory inside Isabelle [39] with which formalisations such as ours could be made. Unfortunately, this early version of nominal logic is incompatible with the axiom of choice and has to be used in Isabelle/PURE – a bare boned set of theories, since Isabelle/HOL contains the axiom of choice. The attempt was later abandoned.

4. Nominal logic

Nominal logic is a formalism designed to simplify the treatment of calculi involving binders. It accomplishes this through a mathematical formalism which allows reasoning about terms with binders up to alpha-equivalence, thus removing the hand waving style proofs which are often practiced when dealing with binders.

Nominal logic is a research area in its own right. This chapter will cover the theoretical background needed to understand the concepts of this thesis. Later chapters will cover how these theories are incorporated and used in Isabelle.

At the core of nominal logic is an atom sort which contains a countably infinite set of entities which can be bound, and alpha-converted in the data. A nominal formalisation can utilise several different atom types, denoted by $\mathscr{A}$, $\mathscr{A}'$, ..., and their elements denoted by a, b, c, ... Moreover a notion of atom swapping is introduced, which allows for renaming of atoms.

$$(a\ b)\cdot c = \begin{cases} a & \textit{if } c = b \\ b & \textit{if } c = a \\ c & \textit{otherwise} \end{cases}$$

Sequences of swaps are called permutations, and are denoted by p, q, r ....

4.1 Nominal sets

A nominal set $\mathscr{X}$ consists of a set of elements and a swapping operator on these elements. One of the main contributions of nominal logic is that it is not necessary to know the structure or syntax of the elements of the nominal set, but rather the behaviour of the swapping function. The following axioms must be satisfied:

$$\forall a \in \mathscr{A}.\ \forall x \in \mathscr{X}.\ (a\ a)\cdot x = x$$
$$\forall a,\ a' \in \mathscr{A}.\ \forall x \in \mathscr{X}.\ (a\ a')\cdot(a\ a')\cdot x = x$$
$$\forall a,\ a' \in \mathscr{A}.\ \forall b,\ b' \in \mathscr{A}'.\ \forall x \in \mathscr{X}.$$
$$(a\ a')\cdot(b\ b')\cdot x = ((a\ a')\cdot b,\ (a\ a')\cdot b')\cdot(a\ a')\cdot x$$

Intuitively, the axioms dictate that $(a\ a)$ must be the identity swapping, that applying the same swapping twice does nothing and that applying a swapping to another distributes the swapping over the arguments. This last property is often referred to as *equivariance* and will be covered in detail later.

For the rest of this chapter a member of a nominal set will be called a term.

4.2 Support and freshness

A useful property of atom swapping is that it can be used to derive the free atoms of a term. In nominal logic, this is called the support of a term. More formally, the support of a term T can be defined as follows:

Definition 4.1 (Support)**.** *The support of a term T is denoted supp T.*

$$supp\ T \stackrel{\text{def}}{=} \{a : infinite\,\{b : (a\ b) \cdot T \neq T\}\}$$

Intuitively, support is defined as the atoms of an agent which changes the agent when swapped. No knowledge of the structure of T is needed, other than that it must satisfy the swapping axioms of nominal logic. Consider the agent

$$P \stackrel{\text{def}}{=} (\nu x)a.(c.\mathbf{0} \mid x.\mathbf{0})$$

which has the free names a and c. The following table describes how the support of P is calculated:

	a	b	c	d	
a	$(a\ a) \cdot P = P$	$(a\ b) \cdot P \neq P$	$(a\ c) \cdot P \neq P$	$(a\ d) \cdot P \neq P$	
b	$(b\ a) \cdot P \neq P$	$(b\ b) \cdot P = P$	$(b\ c) \cdot P \neq P$	$(b\ d) \cdot P = P$	
c	$(c\ a) \cdot P \neq P$	$(c\ b) \cdot P \neq P$	$(c\ c) \cdot P = P$	$(c\ d) \cdot P \neq P$	$\ldots$
d	$(d\ a) \cdot P \neq P$	$(d\ b) \cdot P = P$	$(d\ c) \cdot P \neq P$	$(d\ d) \cdot P = P$	
e	$(e\ a) \cdot P \neq P$	$(e\ b) \cdot P = P$	$(e\ c) \cdot P \neq P$	$(e\ d) \cdot P = P$	
f	$(f\ a) \cdot P \neq P$	$(f\ b) \cdot P = P$	$(f\ c) \cdot P \neq P$	$(f\ d) \cdot P = P$	
		$\vdots$			

We obtain that the support of P is $\{a, c\}$. As can be seen in the columns for a and b, there are infinitely many atoms which when swapped for these atoms will change P. The columns for b and d show that the only two atoms which when swapped with either b or d change P are a and c, and hence

neither b nor d is in the support for P. Any swapping of x will not have an effect, as we are working up to alpha-equivalence.

Another key concept of nominal logic is that of freshness, denoted with $\sharp$. An atom is fresh for a term if it does not appear in its support. This operator was informally described in Chapter 2, and now receives a formal definition.

Definition 4.2 (Freshness)**.** *A name x fresh for a term T is denoted $x \sharp T$.*

$$x \sharp T \stackrel{\text{def}}{=} x \notin \mathit{supp}\ T$$

Note that both support and freshness are overloaded to be defined for all atom sorts.

4.3 Binding construct

Nominal sets are equipped with a binding construct, $[a].T$, which binds the atom a in the term T. This construct is often referred to as an atom abstraction. and it must satisfy the following axioms:

$$\forall b,\ b' \in \mathscr{A}.\ \forall a \in \mathscr{A}'.\ \forall T \in \mathscr{X}.\ (b\ b') \cdot [a].T = [(b\ b') \cdot a].(b\ b') \cdot T$$
$$\forall a,\ b \in \mathscr{A}.\ \forall T,\ T' \in \mathscr{X}.\ [a].T = [b].T' \iff (a = b \wedge T = T') \vee (b \sharp T \wedge T' = (a\ b) \cdot T)$$

The first axiom dictates that name swappings must distribute over the atom abstraction. The second axiom is at the core of how nominal logic deals with alpha-equivalences. It provides a very intuitive way to alpha-convert a term. Pick an arbitrary name b which does not occur in a term being alpha-converted and then replace the original abstraction with b and swap all of its occurrences in the term with b.

This method of alpha-conversion requires that all members of the nominal sets have finite support, i.e. only a finite number of atoms. If this is not the case, it would be impossible to pick a fresh name to alpha-convert into. All members of all nominal sets in this thesis will have finite support.

4.4 Equivariance

Equivariance represents the ability for atom swappings to distribute over an operation. For example, a function f is said to be equivariant if for all arguments $\tilde{x}$, $(a,\ b) \cdot f(\tilde{x}) = f((a,\ b) \cdot \tilde{x})$. Similar properties are necessary for set membership, logical predicates and datatype constructors.

The reason that this property is important is that in a proof a term must be replaceable for alpha-equivalent ones. When a term is alpha-converted, a swapping is applied under the scope of the binder. If a term under the scope of this binder exists elsewhere in the proof context, i.e. without the binder, then it must be possible to apply the same swapping to that term as well – equivariance ensures that this is possible.

Equivariance properties are very easy to work with. Name swappings are very well behaved in that they rarely change the meaning of what they are being applied to, as opposed to renaming or substitution where it is not always easy to see what effect their application will have on the proof environment.

5. Isabelle

Isabelle is a generic interactive theorem prover developed at the University of Cambridge, and Technische Universität München. It provides the user with a simple meta-logic and a means to map object logics to the meta-logic. The appropriate object logic depends on what proofs needs to be done, and a wide variety are available, including first order logic, Zermelo-Fraenkel set theory, the logic of computable functions (LCF) and higher order logic. The latter, Isabelle/HOL, is the most developed.

Isabelle/HOL allows the user to define inductive and coinductive predicates and sets, inductively defined datatypes and well-founded recursive functions. It also has a large infrastructure of proof libraries available for use for formalisations. An extension to Isabelle/HOL provides support for nominal logic [76], providing extensive infrastructure for reasoning about formalisms with binders.

All proofs in this thesis have been made using Isabelle/HOL-Nominal. The proofs are written in the human readable proof language Isar [79], and do not require Isabelle expertise. This chapter will cover the Isabelle concepts used in this thesis. For a more in-depth Isabelle tutorial, see [64] and [79].

5.1 The Isabelle meta-logic

Isabelle's meta-logic is a higher order logic with three connectives used to encode standard logical inference rules. There is a universal quantifier ($\bigwedge$), implication ($\Longrightarrow$) and equality ($\equiv$).

As an example, the PAR-rule from the process algebra described in Chapter 2 is written in the Isabelle meta-logic as follows:

Inference rule	meta-logic
$\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	$\bigwedge P\, \alpha\, P'\, Q.\ P \xrightarrow{\alpha} P' \Longrightarrow P \mid Q \xrightarrow{\alpha} P' \mid Q$

Universal quantifiers at the top level can be implicitly assumed, and the meta-logic formula above will be written as:

$$P \xrightarrow{\alpha} P' \Longrightarrow P \mid Q \xrightarrow{\alpha} P' \mid Q$$

Rules with several assumptions are coded using a sequence of implications. The SYNC-rule for instance is coded like this:

$$P \xrightarrow{\alpha} P' \Longrightarrow Q \xrightarrow{\alpha} Q' \Longrightarrow P \mid Q \xrightarrow{\tau} P' \mid Q'$$

For brevity, these types of rules use the following syntactic sugar:

$$[\![P \xrightarrow{\alpha} P'; Q \xrightarrow{\alpha} Q']\!] \Longrightarrow P \mid Q \xrightarrow{\tau} P' \mid Q'$$

Most commonly, lemmas will be written with a horizontal line separating assumptions from conclusions throughout this thesis.

5.2 Writing proofs in Isabelle

Isabelle supports two styles for writing proofs. One lets the user provide a list of proof altering commands, where each command changes the proof state until the proof is resolved. This method has the advantage that it is reasonably fast, and the user has constant feedback from Isabelle and can deduce what the next step of the proof should be. The downside is that the resulting code is not easy to parse, as all it is is a sequence of commands, and the internal proof state between the commands is not visible. Most commonly, this style of proof is called *apply scripting*, as *apply* is the name of the Isabelle command which modifies the proof state.

The alternative is to write proofs using the Isar proof language, which provides proofs which are significantly easier to read and maintain. Both styles will be covered in this chapter.

When displaying proofs, we will use the following headers containing the name of the lemma, the arguments and their types ($T_1::\tau_1, \ldots, T_m::\tau_m$), which assumptions the lemma has ($A_1, \ldots, A_n$) and which conclusion it proves (C).

lemma ⟨*name*⟩ :
fixes $T_1 ::\tau_1$ **and** $T_2 ::\tau_2$ **and** ... **and** $T_m ::\tau_m$
assumes A_1 **and** A_2 **and** ... **and** A_n
shows C

From the information in this header, Isabelle can deduce the following proof goal.

$$\bigwedge T_1::\tau_1 \;\; T_2::\tau_2 \ldots \; T_m::\tau_m.\; [\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow C$$

This level of detail for the headers is not strictly necessary as Isabelle can often infer the types of a proof – type annotations need only be given when there is an ambiguity in the type inference, but for clarity, this style of header will always be provided.

5.2.1 Apply scripts

Apply scripts are often referred to as backwards reasoning. They typically start from the conclusion of a goal, apply an Isabelle command which transforms the conclusion into something closer to the assumptions, and then repeats the process until the goal is proven.

The following lemma states that the agent $a.P$ can communicate with the agent $a.Q$. Every step of the proof is shown, as well as Isabelle's output during the scripting. for the rest of this chapter, the type *act* is the type of actions, defined in Definition 2.1, and the type *toy* is the agent datatype defined in Definition 2.2.

lemma *prefixComm*:
fixes $a :: act$ **and** $P :: toy$ **and** $Q :: toy$

shows $a.P \mid a.Q \xrightarrow{\tau} P \mid Q$

The only applicable rule to prove the first subgoal is the COMM-rule.

apply(*rule semantics.Comm*)

Isabelle provides two subgoals that needs to be proven, one at a time.

1. $a.P \xrightarrow{\alpha} P$
2. $a.Q \xrightarrow{\alpha} Q$

The only applicable rule is the ACTION-rule.

apply(*rule semantics.Action*)

The first subgoal is discharged.

1. $a.Q \xrightarrow{a} Q$

apply(*rule semantics.Action*)

The second subgoal is discharged, and Isabelle informs that there is nothing left to prove.

No subgoals!

done

5.2.2 Inductive proofs

Isabelle supports induction on inductively defined sets and predicates. The operational semantics of the different calculi presented in this thesis will

$$\left[\begin{array}{l}
R \xrightarrow{\alpha} R' \\[2ex]
\bigwedge \alpha\ P.\ \dfrac{}{Prop\ (\alpha\,.\,P)\ \alpha\ P}\ \textsc{Act} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q.\ \dfrac{P \xrightarrow{\alpha} P' \qquad Prop\ P\ \alpha\ P'}{Prop\ (P \mid Q)\ \alpha\ (P' \mid Q)}\ \textsc{Par1} \\[2ex]
\bigwedge Q\ \alpha\ Q'\ P.\ \dfrac{Q \xrightarrow{\alpha} Q' \qquad Prop\ Q\ \alpha\ Q'}{Prop\ (P \mid Q)\ \alpha\ (P \mid Q')}\ \textsc{Par2} \\[2ex]
\dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' & Prop\ P\ \alpha\ P' \\ Q \xrightarrow{\alpha} Q' & Prop\ Q\ \alpha\ q' \end{pmatrix}}{Prop\ (P \mid Q)\ \tau\ (P' \mid Q')}\ \textsc{Sync} \\[2ex]
\bigwedge P\ \alpha\ P'\ x.\ \dfrac{P \xrightarrow{\alpha} P' \qquad Prop\ P\ \alpha\ P' \qquad x \sharp \alpha}{Prop\ ((\nu x)P)\ \alpha\ ((\nu x)P')}\ \textsc{Scope}
\end{array}\right]$$
$$\overline{Prop\ R\ \alpha\ R'}$$

Figure 5.1: An induction rule of the semantics defined in Figure 2.4. The rule does induction on the transition $R \xrightarrow{\alpha} R'$ to prove the predicate *Prop*. Each inductive case shares the name of the semantic rule which it represents.

be encoded as inductively defined predicates. As an example, the induction rule for the semantics of the process algebra defined in chapter 2 can be found in Figure 5.1.

This induction rule does induction over the transition $Prop\ R\ \alpha\ R'$ to prove the logical proposition *Prop*, which takes two agents and an action as argument. One inductive case is given for every rule of the semantics. The occurrence of *Prop* in the assumptions of these cases denotes the induction hypothesis.

When using this induction rule, any assumptions of the goal will be present for each instance of the induction hypothesis as well – in order to use the induction hypothesis, these assumptions must first be proven. The following lemma demonstrates.

lemma *freshDerivative*:
fixes $P :: toy$ **and** $\alpha :: act$ **and** $P' :: toy$ **and** $y :: name$
assumes $P \xrightarrow{\alpha} P'$ **and** $y \sharp P$
shows $y \sharp P'$
using *assms*

Both assumptions are needed for the induction. The transition $P \xrightarrow{\alpha} P'$ is what we are doing induction over, and $y \sharp P$ is an extra assumption for the induction. We begin by applying the induction rule using the following Isabelle command:

apply(*induct rule*: *semantics'.induct*)

One subgoal is created for every inductive case, note the occurrence of the induction hypothesis $y \sharp P \Longrightarrow y \sharp P'$ in each case. The COMM-case (case 4) has two instances of the induction hypothesis as two transitions are present in the premise of the rule.

1. $\bigwedge \alpha\ P.\ y \sharp \alpha.P \Longrightarrow y \sharp P$
2. $\bigwedge P\ \alpha\ P'\ Q.\ [\![P \xrightarrow{\alpha} P'; y \sharp P \Longrightarrow y \sharp P'; y \sharp P \mid Q]\!] \Longrightarrow y \sharp P' \mid Q$
3. $\bigwedge Q\ \alpha\ Q'\ P.\ [\![Q \xrightarrow{\alpha} Q'; y \sharp Q \Longrightarrow y \sharp Q'; y \sharp P \mid Q]\!] \Longrightarrow y \sharp P \mid Q'$
4. $\bigwedge P\ \alpha\ P'\ Q\ Q'.$
   $[\![P \xrightarrow{\alpha} P'; y \sharp P \Longrightarrow y \sharp P'; Q \xrightarrow{\alpha} Q'; y \sharp Q \Longrightarrow y \sharp Q'; y \sharp P \mid Q]\!] \Longrightarrow y \sharp P' \mid Q'$
5. $\bigwedge P\ \alpha\ P'\ x.\ [\![P \xrightarrow{\alpha} P'; y \sharp P \Longrightarrow y \sharp P'; x \sharp \alpha; y \sharp (\nu x)P]\!] \Longrightarrow y \sharp (\nu x)P'$

apply(*auto simp add*: *abs-fresh*)

Isabelle manages to solve all of the subgoals automatically with a heuristic called *auto*. Most Isabelle heuristics can be augmented with specific rules. In this case, the rule *abs-fresh* takes care of possible cases for the assumption $y \sharp (\nu x)P$ in subgoal 5 – either $y = x$, or $y \neq x$ and $y \sharp P$.

No subgoals!

done

5.2.3 *Inversion proofs*

Proofs by case analysis are commonly referred to as inversion proofs. Given a transition $P \xrightarrow{\alpha} P'$, an inversion rule will look at what transitions are actually possible by looking at the structure of P. They differ from induction rules in that they there is no induction hypothesis, even if an operator occurs in the premise of an inference rule, and the question they answer is: What set of inference rule could have made the transition $P \xrightarrow{\alpha} P'$ possible?

In this thesis, the most common use of inversion rules will be in the congruence proofs – that bisimulation is preserved by all operators of the calculus.

The following lemma proves that simulation is preserved by the action prefix.

lemma *simActPres*:
fixes P :: *toy* **and** Q :: *toy* **and** α :: *act* **and** $\mathscr{R}$ :: (*toy* × *toy*) *set*

assumes $(P, Q) \in \mathscr{R}$

shows $\alpha.P \hookrightarrow_{\mathscr{R}} \alpha.Q$

The first step is to unfold the definition of simulation

using *assms*
apply(*auto simp add*: *simulation-def*)

1. $\bigwedge \beta\ R.\ [\![(P, Q) \in \mathscr{R};\ \alpha.Q \xrightarrow{\beta} R]\!] \Longrightarrow \exists P'.\ \alpha.P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

The next step is to do inversion over the transition $\alpha.Q \xrightarrow{\beta} R$ and see what possible transitions it can do.

apply(*erule semantics'.cases*)

1. $\bigwedge \beta\ R\ \gamma\ T.$
$[\![(P, Q) \in \mathscr{R};\ \alpha.Q = \gamma.T;\ \beta = \gamma;\ R = T]\!] \Longrightarrow \exists P'.\ \alpha.P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

2. $\bigwedge \beta\ R\ Q_1\ \gamma\ Q_1'\ Q_2.$
$[\![(P, Q) \in \mathscr{R};\ \alpha.Q = Q_1 \mid Q_2;\ \beta = \gamma;\ R = Q_1' \mid Q_2;\ Q_1 \xrightarrow{\gamma} Q_1']\!]$
$\Longrightarrow \exists P'.\ \alpha.P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

3. $\bigwedge \beta\ R\ Q_2\ \gamma\ Q_2'\ Q_1.$
$[\![(P, Q) \in \mathscr{R};\ \alpha.Q = Q_1 \mid Q_2;\ \beta = \gamma;\ R = Q_1 \mid Q_2';\ Q_2 \xrightarrow{\gamma} Q_2']\!]$
$\Longrightarrow \exists P'.\ \alpha.P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

4. $\bigwedge \beta\ R\ Q_1\ \gamma\ Q_1'\ Q_2\ Q_2'.$

$$\left[\begin{array}{l}
R \xrightarrow{\beta} R' \\
\\
\bigwedge \alpha\ P.\ \dfrac{R = \alpha\,.\,P \qquad \beta = \alpha \qquad R' = P}{Prop}\ \textsc{Act} \\
\\
\bigwedge P\ \alpha\ P'\ Q.\ \dfrac{R = P \mid Q \qquad \beta = \alpha \qquad R' = P' \mid Q \qquad P \xrightarrow{\alpha} P'}{Prop}\ \textsc{Par1} \\
\\
\bigwedge Q\ \alpha\ Q'\ P.\ \dfrac{R = P \mid Q \qquad \beta = \alpha \qquad R' = P \mid Q' \qquad Q \xrightarrow{\alpha} Q'}{Prop}\ \textsc{Par2} \\
\\
P\ \alpha\ P'\ Q\ Q'.\ \dfrac{\begin{pmatrix} R = P \mid Q & \beta = \tau & R' = P' \mid Q' \\ P \xrightarrow{\alpha} P' & Q \xrightarrow{\alpha} Q' & \end{pmatrix}}{Prop}\ \textsc{Sync} \\
\\
P\ \alpha\ P'\ x.\ \dfrac{\begin{pmatrix} R = (\nu x)P & \beta = \alpha & R' = (\nu x)P' \\ P \xrightarrow{\alpha} P' & x \sharp \alpha & \end{pmatrix}}{Prop}\ \textsc{Scope}
\end{array}\right]$$
$$\overline{\qquad Prop \qquad}$$

Figure 5.2: The inversion rule for the semantics of the process algebra described in Figure 2.4. The rule does inversion on the transition $R \xrightarrow{\alpha} R'$ to prove the predicate *Prop*. Each inversion case introduces equality constraints for R, α, and R' such that their structure matches the requirements of the semantic rules. As for the induction rule, the inversion cases share the names of the semantic rules which they represent.

$[\![(P, Q) \in \mathscr{R}; \alpha . Q = Q_1 \mid Q_2 ; \beta = \tau; R = {Q_1}' \mid {Q_2}' ; Q_1 \xrightarrow{\gamma} {Q_1}'; Q_2 \xrightarrow{\gamma} {Q_2}']\!]$

$\Longrightarrow \exists P'. \alpha . P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

5. $\bigwedge \beta\ R\ T\ \gamma\ T'\ x.$

$[\![(P, Q) \in \mathscr{R}; \alpha . Q = (\nu x) T; \beta = \gamma; R = (\nu x) T'; T \xrightarrow{\gamma} T'; x \sharp \gamma]\!]$

$\Longrightarrow \exists P'. \alpha . P \xrightarrow{\beta} P' \wedge (P', R) \in \mathscr{R}$

The inversion rule provides five cases, but only case number *1* is applicable since the other cases have trivially false equality constraints in the assumptions.

apply(*auto simp add*: *toy.inject*)

1. $(P, Q) \in \mathscr{R} \Longrightarrow \exists P'. \alpha . P \xrightarrow{\alpha} P' \wedge (P', Q) \in \mathscr{R}$

The lemma *toy.inject* proves and disproves equality of agents, and it removes four of the five cases, leaving the ACT-case which is provable from the ACT-rule and the assumption that $(P, Q) \in \mathscr{R}$

apply(*blast intro*: *Action*)

No subgoals!

done

This inversion rule works very well, as long as no terms in the case analysis contain binders. Consider the SCOPE-case in the rule in Figure 5.2 if the term R has the form $(\nu y)Q$. The equality constraint in the assumption will then be $(\nu y)Q = (\nu x)P$, and the axioms of nominal logic will provide two cases – one where $x = y$ and one where $x \neq y$ and $Q = (x\ y) \cdot P$.

To do these alpha-equivalence cases in every proof will quickly boil down to tediously detailed proofs which push name swappings back and forth, and would not provide the abstraction level needed when doing proofs. How Nominal Isabelle circumvents this problem will be described in section 5.3.

5.2.4 Coinductive proofs

All bisimulations in this thesis are defined using coinductive definitions. When proving that two agents are bisimilar, one needs to find a candidate relation containing the two agents, and there must be no way that any two members of this candidate set can fall out of the set by applying the rules that form the coinductive definition. One coinduction rule for bisimulation looks as follows:

$$\frac{(P, Q) \in \mathscr{X} \qquad \bigwedge P\, Q.\ \dfrac{(P, Q) \in \mathscr{X}}{P \hookrightarrow_{\mathscr{X}} Q}\ \textsc{Simulation} \qquad \bigwedge P\, Q.\ \dfrac{(P, Q) \in \mathscr{X}}{(Q, P) \in \mathscr{X}}\ \textsc{Symmetry}}{P \sim Q}$$

In order to prove that P and Q are bisimilar, one must find a candidate relation $\mathscr{X}$ such that $\mathscr{X}$ is symmetric, and that for all agents P and Q in $\mathscr{X}$ there is no way for P to simulate Q such that the derivatives fall outside of $\mathscr{X}$.

Note that the SYMMETRY case can also be written as $\mathscr{X} \subseteq \mathscr{X}^{-}$. This is shorter, easier to read, but a bit inconvenient to work with in a theorem prover – when applying the coinduction rule, the first thing that Isabelle will do is to unfold the definition of $\subseteq$ to become what is declared in the SYMMETRY case. When writing rules for a theorem prover it is generally good practice to write them in such a way that the theorem prover's automatic heuristics cannot simplify the rule any further.

The following lemma illustrates, by proving that bisimulation is reflexive.

lemma *bisimReflexive*:
 fixes P :: *toy*
 shows $P \sim P$
apply(*coinduct rule*: *bisimCoinduct*[**where** *X*=*Id*])

By setting $\mathscr{X}$ to the identity relation, we get the following three subgoals.

1. $(P, P) \in Id$
2. $\bigwedge P\, Q.\ (P, Q) \in Id \Longrightarrow P \hookrightarrow_{Id} Q$
3. $\bigwedge P\, Q.\ (P, Q) \in Id \Longrightarrow (Q, P) \in Id$

Case 1 and 3 follow trivially from the definition of the identity relation, and case 2 follows from the definition of $\hookrightarrow$.

apply(*auto simp add*: *simulation-def*)

No subgoals!

done

There are many variants of coinductive proofs, and sometimes more powerful rules than the one presented above are necessary. These rules will be presented throughout the thesis as they are needed.

5.3 Nominal logic in Isabelle

Isabelle/HOL-Nominal is an extension of Isabelle/HOL, the most developed logical theory in Isabelle. When writing proofs in HOL-Nominal, often referred to as Nominal Isabelle, the user has access to all the logical infrastructure present in HOL.

This section will describe how Nominal Isabelle is used to reason about calculi with binders, and also provide the theoretical connection from Chapter 4.

5.3.1 Atom swapping and permutations

Nominal Isabelle provides support for creating nominal datatypes. A nominal datatype can be viewed as a nominal set, described in Section 4.1, and Nominal Isabelle uses type classes to add the requirements of nominal sets to the standard Isabelle datatypes. It is also possible to define new nominal datatypes. At the core of the formalisation is the atom type. An atom type contains a countably infinite number of atoms, which are used as building blocks when building nominal datatypes, and which can be bound using a binding construct.

For each atom type, a swapping function *swap* is defined which is defined exactly as in nominal logic. A permutation is then defined as a list of pairs of names – a permutation function is then created which recurses over the list and applies the pairs as swappings one pair at a time. Since permutations are lists, we can use the standard library functions on lists to reason about them. More specifically, the empty list [] is the empty permutation, a swapping can be appended to a permutation using the #-operator, and two permutations can be appended using the @-operator.

For all members c of an atom type $\mathscr{A}$ the following must hold:

$$[] \cdot c = c$$
$$(a, b) \# p \cdot c = swap\ (a, b)\ (p \cdot c)$$
$$swap\ (a, b)\ c = (\mathit{if}\ a = c\ \mathit{then}\ b\ \mathit{else\ if}\ b = c\ \mathit{then}\ a\ \mathit{else}\ c)$$
$$\text{infinite}\ \mathscr{A}$$

These axioms dictate that the empty list must be the identity permutation, that permutations and name swappings operate in the desired way, and that the number of atoms have to be countably infinite.

There is also notion of permutation equality.

Definition 5.1. *Two permutations p and q are said to be equivalent if when applied to any atom, have the same effect.*

$$p \cong q \stackrel{\text{def}}{=} \forall a.\ p \cdot a = q \cdot a$$

A type is a permutation type if all of its members x meet the following constraints.

$$[] \cdot x = x$$
$$(p @ q) \cdot x = p \cdot q \cdot x$$
$$p \cong q \Longrightarrow p \cdot x = q \cdot x$$

To check whether or not a type is a permutation type, a permutation function is introduced, which typically distributes over all the components of a type, until it reaches the atoms, and performs the permutations there. Most of the commonly used Isabelle datatypes have been proven to be permutation types. Isabelle's support for function overloading makes this process seamless.

The inverse of a permutation p^- is obtained by reversing the permutation list p.

Lemma 5.2. *A permutation p is canceled by its inverse* p^-.

$$p \cdot p^- \cdot x = x$$
$$p^- \cdot p \cdot x = x$$

5.3.2 Support and freshness

With the permutations in place, support and freshness can be defined just as they are for nominal logic.

Definition 5.3 (Support)**.** *The support of the term T is denoted supp T.*

$$supp\ T \stackrel{\text{def}}{=} \{a : infinite\ \{b : (a\ b) \cdot T \neq T\}\}$$

Note that every atom type will yield its own support function. Also, when working with support in lemmas it is often necessary to provide proper type annotation. Isabelle will not automatically be able to determine the atom type of an expression such as *supp x*, unless it is made clear from another part of the proof context.

Freshness is then defined in the standard way.

Definition 5.4 (Freshness)**.** *The name x fresh for the term T is denoted* $x \sharp T$.

$$x \sharp T \stackrel{\text{def}}{=} x \notin supp\ T$$

5.3.3 Atom abstraction

In nominal logic, the existence of an atom abstraction function is axiomatised, in Isabelle an abstraction function is defined which given an atom and a permutation type creates an atom abstraction.

The constructors *nSome* and *nNone* are constructors for a local *option* datatype in Nominal Isabelle. as Nominal Isabelle needs to be able to distinguish these instances of the option type from the ones provided by the user.

Definition 5.5 (Atom abstraction)**.** *A name a bound in the term T is denoted* $[x].T$

$$[a].T \stackrel{\text{def}}{=}$$
$$\lambda b.\ \textit{if } b = a \textit{ then nSome } T \textit{ else if } b \sharp T \textit{ then nSome } (a\ b) \cdot T \textit{ else nNone}$$

From this definition, the axioms for nominal logic can be derived and need not be postulated.

Lemma 5.6.

$$p \cdot [a].x = [(p \cdot a)].(p \cdot x)$$
$$([a].x = [b].y) = (a = b \wedge x = y \vee a \neq b \wedge x = (a\ b) \cdot y \wedge a \sharp y)$$

5.3.4 Nominal datatypes

Isabelle/HOL has support for creating inductively defined datatypes. The nominal package expands these datatypes to include types with binders. A user must declare the atom types needed for the formalisation. Isabelle will then automatically create swapping functions, and a host of lemmas designed to reason about freshness, support, and permutations. Nominal datatypes are defined in much the same way as regular datatypes in functional programming languages. The binding occurrences of atom types are enclosed by « and ».

```
nominal_datatype act = Action name
                     | Tau

nominal_datatype toy = ToyNil
                     | Action act toy
                     | Par toy toy
                     | Res "«name» toy"
```

From this definition, Isabelle will automatically create rules to reason about freshness and injectivity of the nominal datatype.

Lemma 5.7. *The following equalities are proven automatically about freshness of the process algebra.*

$$\begin{aligned} a \sharp \mathbf{0} &\Leftrightarrow \mathit{True} \\ a \sharp \alpha.P &\Leftrightarrow a \sharp \alpha \wedge a \sharp P \\ a \sharp P \mid Q &\Leftrightarrow a \sharp P \wedge a \sharp Q \\ a \sharp (\nu x)P &\Leftrightarrow a \sharp [x].P \end{aligned}$$

Note how the freshness for terms with binders simplify to their corresponding atom abstraction.

Lemma 5.8. *The following are the injectivity lemmas proven for the nominal datatype.*

$$\begin{aligned} \alpha.P = \beta.Q &\Leftrightarrow \alpha = \beta \wedge P = Q \\ P \mid R = Q \mid S &\Leftrightarrow P = Q \wedge R = S \\ (\nu x)P = (\nu y)Q &\Leftrightarrow [x].P = [y].Q \end{aligned}$$

Again, the terms with binders are simplified to their corresponding atom abstractions. From there, any further proof regarding alpha-equivalence or alpha-conversions can be done.

5.3.5 Induction rules

Isabelle automatically creates induction and inversion rules for inductively defined datatypes, predicates and sets. It will do this for terms with binders as well, but with the drawback that any new binders introduced by the induction rule will not necessarily be sufficiently fresh according to the Barendregt variable convention; a name will just be as fresh as the rule dictates, and all possible proof contexts cannot be known beforehand.

Nominal Isabelle introduces the notion of avoiding contexts of atoms. When applying an induction rule in Nominal Isabelle the user can provide a finite set of atoms with which any newly occurring bound name may not clash; we say that the bound names avoid this context. The automatically generated induction rule for the process calculus defined in chapter 2 can be found in Figure 5.3.

The avoiding context $\mathscr{C}$ denotes the names that freshly generated bound names must not clash with; in this case the bound name x in the SCOPE-rule. Without this style of rules, a multitude of tedious manual alpha-conversions would have to be made in inductive proofs.

An example of an application of this rule can be found in the next section.

$$\frac{\left[\begin{array}{l} R \xrightarrow{\beta} R' \\ \\ \bigwedge \alpha\ P\ \mathscr{C}.\ \dfrac{}{Prop\ \mathscr{C}\ (\alpha\,.\,P)\ \alpha\ P}\ \textsc{Act} \\ \\ \bigwedge P\ \alpha\ P'\ Q\ \mathscr{C}.\ \dfrac{P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P'}{Prop\ \mathscr{C}\ (P \mid Q)\ \alpha\ (P' \mid Q)}\ \textsc{Par1} \\ \\ \bigwedge Q\ \alpha\ Q'\ P\ \mathscr{C}.\ \dfrac{Q \xrightarrow{\alpha} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ \alpha\ Q'}{Prop\ \mathscr{C}\ (P \mid Q)\ \alpha\ (P \mid Q')}\ \textsc{Par2} \\ \\ \bigwedge P\ \alpha\ P'\ Q\ Q'\ \mathscr{C}.\ \dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P' \\ Q \xrightarrow{\alpha} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ \alpha\ Q' \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ \tau\ (P' \mid Q')}\ \textsc{Sync} \\ \\ \bigwedge P\ \alpha\ P'\ x\ \mathscr{C}.\ \dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P' \\ x \sharp \alpha \qquad x \sharp \mathscr{C} & \end{pmatrix}}{Prop\ \mathscr{C}\ ((\nu x)P)\ \alpha\ ((\nu x)P')}\ \textsc{Scope} \end{array}\right]}{Prop\ \mathscr{C}\ R\ \beta\ R'}$$

Figure 5.3: The nominal induction rule for the semantics described in Figure 2.4

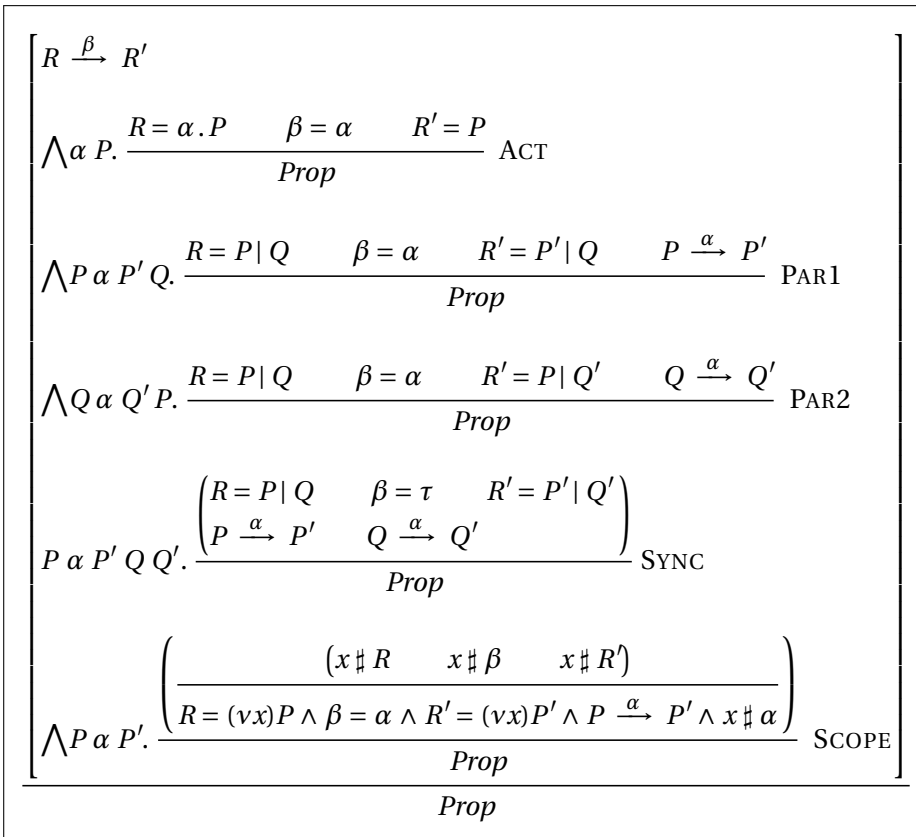


Figure 5.4: The nominal inversion rule for the semantics described in Figure 2.4

5.3.6 *Inversion rules*

As was shown in section 5.2.3, the regular inversion rules provided by Isabelle do not handle binders very well, as the equality constraints with binders lead to case explosions for the different alpha-variants. The nominal datatype package has support for an alternative inversion rule which handles binders more fluently. When inversion is done on a term with a binder, this binder cannot be chosen to be sufficiently fresh. It has already been fixed, and any freshness conditions must already have been established. The power of nominal induction rules, such as the one in Figure 5.3, is that any new bound name which is introduced can be chosen in such a way that it is sufficiently fresh, but in an inversion on a transition with binders, say $(\nu x)P \xrightarrow{\alpha} P'$, the binder x is already present in the proof context and must be manually alpha-converted if it is not sufficiently fresh.

The nominal inversion rule provided by Isabelle for our simple process calculus can be found in Figure 5.4. The case of interest is the SCOPE-case.

The binder x is universally quantified by the entire rule, which requires it to be instantiated by the user prior to invoking the inversion rule. A set of freshness conditions are imposed on this binder – it may not clash with the originating process, its action nor its derivative.

The following lemma proves that simulation is preserved by restriction.

lemma *simResPres*:
fixes $P :: toy$
and $Q :: toy$
and $\mathscr{R} :: (toy \times toy)\ set$
and $x :: name$

assumes $P \hookrightarrow_{\mathscr{R}} Q$
and $\bigwedge P\ Q\ x.\ (P, Q) \in \mathscr{R} \Longrightarrow ((\nu x)P, (\nu x)Q) \in \mathscr{R}$

shows $(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)Q$

The first step is to unfold the definition of simulation, but only in the conclusion of the goal

apply(*simp add*: *simulation-def*, *auto*)

1. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$

In order to use the inversion rule, sufficient freshness conditions of x must be known. These are assumed for now, and will be proven later.

apply(*subgoal-tac* $x \sharp \alpha \wedge x \sharp Q'$)

1. $\bigwedge \alpha\ Q'.\ [\![(\nu x)Q \xrightarrow{\alpha} Q';\ x \sharp \alpha \wedge x \sharp Q']\!] \Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$
2. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp \alpha \wedge x \sharp Q'$

The inversion rule can now be used, with the binder set to x

apply(*erule-tac semanticsCases*[**where** x=x])

1. $\bigwedge \alpha\ Q'\ \beta\ R.$
   $[\![x \sharp \alpha \wedge x \sharp Q';\ (\nu x)Q = \beta\,.\,R;\ \alpha = \beta;\ Q' = R]\!]$
   $\Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$
2. $\bigwedge \alpha\ Q'\ Q_1\ \beta\ Q_1{}'\ Q_2.$
   $[\![x \sharp \alpha \wedge x \sharp Q';\ (\nu x)Q = Q_1 \mid Q_2\ ;\ \alpha = \beta;\ Q' = Q_1{}' \mid Q_2\ ;\ Q_1 \xrightarrow{\beta} Q_1{}']\!]$
   $\Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$
3. $\bigwedge \alpha\ Q'\ Q_2\ \beta\ Q_2{}'\ Q_1.$
   $[\![x \sharp \alpha \wedge x \sharp Q';\ (\nu x)Q = Q_1 \mid Q_2\ ;\ \alpha = \beta;\ Q' = Q_1 \mid Q_2{}';\ Q_2 \xrightarrow{\beta} Q_2{}']\!]$
   $\Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$
4. $\bigwedge \alpha\ Q'\ Q_1\ \beta\ Q_1{}'\ Q_2\ Q_2{}'.$

$[\![x \sharp \alpha \wedge x \sharp Q'; (\nu x)Q = Q_1 \mid Q_2\,; \alpha = \tau; Q' = {Q_1}' \mid {Q_2}'\,; Q_1 \xrightarrow{\beta} {Q_1}'; Q_2 \xrightarrow{\beta} {Q_2}']\!]$
$\Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$

5. $\bigwedge \alpha\ Q'\ R\ \beta\ R'.$
$[\![x \sharp \alpha \wedge x \sharp Q';$
$[\![x \sharp (\nu x)Q; x \sharp \alpha; x \sharp Q']\!]$
$\Longrightarrow (\nu x)Q = (\nu x)R \wedge \alpha = \beta \wedge Q' = (\nu x)R' \wedge R \xrightarrow{\beta} R' \wedge x \sharp \beta]\!]$
$\Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}$
6. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp \alpha \wedge x \sharp Q'$

The inversion rule provides five cases, where only the SCOPE-case is applicable as the other ones have false equality constraints.

apply(*auto simp add*: *abs-fresh alpha toy.inject*)
thm *toy.inject*

1. $\bigwedge \beta\ R'.\ [\![Q \xrightarrow{\beta} R'; x \sharp \beta]\!] \Longrightarrow \exists P'.\ (\nu x)P \xrightarrow{\beta} P' \wedge (P', (\nu x)R') \in \mathscr{R}$
2. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp \alpha$
3. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp Q'$

The injectivity rules remove all but one case, and the two assumptions that were previously postulated.

using *assms*
apply(*simp add*: *simulation-def*, *blast intro*: *Res*)

1. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp \alpha$
2. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp Q'$

By unfolding the definition of simulation in the assumptions, the case is proven with the SCOPE-rule and the *C1* assumption of the lemma. The remaining postulated freshness conditions are discharged using lemma 2.9.

apply(*force dest*: *freshDerivative'* *simp add*: *abs-fresh*)

1. $\bigwedge \alpha\ Q'.\ (\nu x)Q \xrightarrow{\alpha} Q' \Longrightarrow x \sharp Q'$

apply(*force dest*: *freshDerivative'* *simp add*: *abs-fresh*)

No subgoals!

done

This proof concludes the coverage of the Isabelle apply scripts. They will not be used any more in this thesis, but they provide an insight in the inner workings of Isabelle and how the proofs are done. Their main advantage is that writing small proofs using them takes very little time – the above proofs took a few minutes to write – but they do not scale well. Isabelle has

support for the structured proof language *Isar*, which will be covered in the next section.

5.3.7 Equivariance properties

In order to create the tailor made induction and inversion rules for inductively defined predicates, all predicates in their construction must be equivariant. The reason for this is that when alpha-conversions are done, atom swappings appear under the scope of the binders, and it must be possible to propagate these swappings through other parts of the definition where the terms under the binder occur. Isabelle has automatic support for proving equivariance of inductively defined predicates, but the user can also create equivariance lemmas when needed. These will be stored in a separate class of lemmas which Isabelle can use internally when needed.

5.4 Writing human readable proofs

Writing proofs with apply scripts has several drawbacks. First of all the proofs are hard to read. A large proof can easily have several hundred lines of code, the extreme ones even thousands, and figuring out the proof state in the middle of the proof just by looking at a list of **apply**-commands is nearly impossible. Secondly, apply scripts are not very robust. Isabelle is a tool in constant development, and it is not uncommon for the automatic heuristics to perform slightly differently between different releases. If a heuristic in a newer version of Isabelle suddenly proves more than previously, the proof state mid script will not be what was originally anticipated, and the proof will fail. Finally, as the complexity of proofs increase, so does their search space as more information is available. The automatic heuristics in Isabelle will quickly grind to a halt exploring dead ends in the proof tree.

To circumvent this problem, the proof language Isar was introduced, which tackles all of these issues. Firstly, the proofs are more readable. A well written Isar proof can be read and understood without running Isabelle in the background. Secondly, it divides the proof into manageable chunks. Even though every subgoal in an Isar proof can be proven using apply scripts in the standard way, the idea is that all subgoals should be proven by one line proofs making them far less likely to break in the version changes. Finally, as the subgoals are much smaller than the complete proof state, the automatic heuristics have a much more manageable task to handle, making them more effective.

The rest of this section will redo most of the proofs in this chapter with Isar. Even though these proofs are simple, and can be proved using one line

by Isabelle, they demonstrate the readability of Isar. The following is an Isar proof of the lemma *prefixComm*, found on page 53, and proves that $a.P$ and $a.Q$ can synchronise.

lemma *prefixCommIsar*:
 fixes a :: *act* **and** P :: *toy* **and** Q :: *toy*

 shows $a.P \mid a.Q \xrightarrow{\tau} P \mid Q$
proof −
 have $a.P \xrightarrow{a} P$ **by**(*rule semantics.Action*)
 moreover have $a.Q \xrightarrow{a} Q$ **by**(*rule semantics.Action*)
 ultimately show $a.P \mid a.Q \xrightarrow{\tau} P \mid Q$ **by**(*rule semantics.Comm*)
qed

The keyword **have** tells Isabelle what to prove, and the **by** command how to prove it. The **by**-command can be replaced by an apply script, but as previously mentioned should be a one line proof or a very short apply script.

The **moreover** keyword collects what was proven on the previous line and when the command **ultimately** is found, all the collected propositions are used as assumptions for the current proof.

Isar Code	Meta-logic formula when proving D
have A **by** ...	
moreover ...	
⋮	
have B **by** ...	
moreover ...	$[\![A; B; C]\!] \Longrightarrow D$
⋮	
have C **by** ...	
ultimately have/**show** D	
by ...	

Note that other things can be proven between the **moreover** commands, where the vertical dotted lines are, but they will not be added to the assumption chain.

The keyword **show** tells Isabelle that the goal being proven now is the actual main goal of the lemma. Isabelle will check that what is written actually corresponds to what Isabelle expects to be proving, and that no illegal assumptions have been made along the way. Instead of writing out the predicate to be proven, the keyword *?thesis* can be used.

5.4.1 Inductive proofs

When doing inductive proofs in Isar the user gets access to each inductive case separately. As was seen earlier, Isabelle can do the following proof with one line, but doing it without any automation nicely demonstrates most of the Isar constructs that will be used throughout the thesis. The apply script proof can be found on page 55.

lemma *freshDerivativeIsar*:
 fixes P :: *toy* **and** α :: *act* **and** P' :: *toy* **and** y :: *name*
 assumes $P \xrightarrow{\alpha} P'$ **and** $y \sharp P$
 shows $y \sharp P'$
using *assms*
proof(*nominal-induct avoiding*: *y rule*: *semantics'.strong-inducts*)
 case(*Action* α P)
 from ‹$y \sharp \alpha . P$› **show** $y \sharp P$ **by** *simp*
next
 case(*Par1* P α P' Q)
 from ‹$y \sharp P \mid Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *auto*
 from ‹$y \sharp P$› **have** $y \sharp P'$ **by**(*rule Par1*)
 with ‹$y \sharp Q$› **show** $y \sharp P' \mid Q$ **by** *simp*
next
 case(*Par2* Q α Q' P)
 from ‹$y \sharp P \mid Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *auto*
 from ‹$y \sharp Q$› **have** $y \sharp Q'$ **by**(*rule Par2*)
 with ‹$y \sharp P$› **show** $y \sharp P \mid Q'$ **by** *simp*
next
 case(*Comm* P α P' Q Q')
 from ‹$y \sharp P \mid Q$› **have** $y \sharp P$ **and** $y \sharp Q$ **by** *auto*
 from ‹$y \sharp P$› **have** $y \sharp P'$ **by**(*rule Comm*)
 moreover from ‹$y \sharp Q$› **have** $y \sharp Q'$ **by**(*rule Comm*)
 ultimately show $y \sharp P' \mid Q'$ **by** *simp*
next
 case(*Res* P α P' x y)
 from ‹$y \sharp (\nu x)P$› ‹$x \sharp y$› **have** $y \sharp P$
  by(*simp add*: *abs-fresh at-fresh*[*OF at-name-inst*])
 hence $y \sharp P'$ **by**(*rule Res*)
 thus $y \sharp (\nu x)P'$ **by**(*simp add*: *abs-fresh*)
qed

A few new keywords were introduced in this example. First of all, when applying the induction rule, Isabelle is told to avoid the name *y*. The result of this can be found in the SCOPE-case where the predicate $x \sharp y$ can be found in the assumptions.

The **hence** and the **thus** keywords are variants of **from** and **show** respectively, but they add the previously proved predicate to the assumptions of

$$\frac{\alpha.P \xrightarrow{\beta} P' \qquad Prop\ \alpha\ P}{Prop\ \beta\ P'}\ \textsc{Act}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{\alpha} R \\ \bigwedge P'.\ \dfrac{P \xrightarrow{\alpha} P'}{Prop\ \alpha\ (P' \mid Q)} \qquad \bigwedge Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{Prop\ \alpha\ (P \mid Q')} \\ \bigwedge \alpha\ P'\ Q'.\ \dfrac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha} Q'}{Prop\ \tau\ (P' \mid Q')} \end{array}\right]}{Prop\ \alpha\ R}\ \textsc{Par}$$

$$\frac{\left[(\nu x)P \xrightarrow{\alpha} R \qquad \bigwedge P'.\ \dfrac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{Prop\ ((\nu x)P')}\right]}{Prop\ R}\ \textsc{Scope}$$

Figure 5.5: Three inversion rules, one rule for every operator

the current subgoal. The **with** keyword works like **hence** except that it not only adds the previously proved predicate, but also a list of predicates to the assumptions of the current subgoal.

Finally, as in the previous proofs, the predicates to be proven are explicitly spelled out after a **show** or **thus** command. If the goal is long and complex, the keyword *?case* can be used instead, but when possible it will be spelled out for clarity. In short, *?case* is the main goal of the current inductive case of an inductive proof, whereas *?thesis* is the main goal of a regular proof.

5.4.2 Inversion proofs

The inversion rules provided by Isabelle simplify case analysis of the transition systems significantly, but they do not lend themselves well to Isar style proofs. When doing inductive proofs, Isabelle provides support for instantiating the different cases, and giving shorthand notation to access subgoals and assumptions. This only works if the induction rule is the only rule applied. If the user applies any of Isabelle's automatic heuristics to the subgoals the shorthand notation, and the ability to retrieve the inductive cases by name, are lost. The reason for this is that it is not possible to determine

how the automatic heuristics will affect the subgoals in the general case; cases may be split into several sub cases, be simplified, or be proven completely.

This style of proofs do work for inversion as well, but the structure of the inversion rules requires the user to manually reason about the equality constraints provided for each case, giving unwieldy proofs. Isabelle's automatic heuristics can unify these constraints, but in doing so, the case instantiations and shorthand notations are lost. Moreover, in the nominal case all binders for every rule must be instantiated before the rule is applied, even the ones which are not present in the transition being analysed.

The solution to this problem is to create specific inversion rules for the transitions that are of interest. There is generally one rule per operator in the calculus. The rules in Figure 5.5 are directly derived from the inversion rule in Figure 5.4, all by one line proofs. By using these rules as induction rules, the infrastructure for doing inductive proofs is made available for inversion proofs as well.

The following is the Isar version of *simActPres* lemma, found on page 56, and proves that simulation is preserved by the action prefix.

lemma *isarSimActPres*:
 fixes $P :: toy$ **and** $Q :: toy$ **and** $\mathscr{R} :: (toy \times toy)\ set$ **and** $\alpha :: act$

 assumes $(P, Q) \in \mathscr{R}$

 shows $\alpha.P \hookrightarrow_{\mathscr{R}} \alpha.Q$
proof(*auto simp add*: *simulation-def*)
 fix $\alpha' Q'$
 assume $\alpha.Q \xrightarrow{\alpha'} Q'$
 thus $\exists P'.\ \alpha.P \xrightarrow{\alpha'} P' \wedge (P', Q') \in \mathscr{R}$
 proof(*induct rule*: *actionCases*)
  case *cAction*
  have $\alpha.P \xrightarrow{\alpha} P$ **by**(*rule Action*)
  thus $\exists P'.\ \alpha.P \xrightarrow{\alpha} P' \wedge (P', Q) \in \mathscr{R}$ **using** ‹$(P, Q) \in \mathscr{R}$›
   by *blast*
 qed
qed

At the end of this proof the **using** command is used to add the predicate $(P, Q) \in \mathscr{R}$ to the assumptions of the subgoal. It works similarly to the **with** keyword in that it adds a list of predicates to the assumptions of a goal.

In the following proof, the concept of a label is introduced. In previous proofs, when list of predicates have been given as arguments to a goal, using the **using** or **with** commands for example, this has been done inline with the whole predicates written explicitly in the list. For short predicates this adds to readability, but for larger ones it detracts from it, and it would also

be difficult to write and maintain proofs if all large predicates were to be retyped every time they were used. When a predicate is proved, it can be preceded by an alphanumeric label and a colon. This label can then be used later in the proof to represent the predicate in a compact way.

The following proof is an Isar version of the *simResPres* lemma, found on page 66, which proves that simulation is preserved by restriction.

lemma *isarSimResPres*:
 fixes $P :: toy$ **and** $Q :: toy$ **and** $\mathscr{R} :: (toy \times toy)\ set$ **and** $x :: name$

 assumes *PSimQ*: $P \hookrightarrow_{\mathscr{R}} Q$
 and *A1*: $\bigwedge P\ Q\ x.\ (P, Q) \in \mathscr{R} \Longrightarrow ((\nu x)P, (\nu x)Q) \in \mathscr{R}'$

 shows $(\nu x)P \hookrightarrow_{\mathscr{R}'} (\nu x)Q$
proof(*auto simp add*: *simulation-def*)
 fix $\alpha\ Q''$
 assume $(\nu x)Q \xrightarrow{\alpha} Q''$
 thus $\exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', Q'') \in \mathscr{R}'$
 proof(*induct rule*: *resCases*)
  case(*cRes Q'*)
  from *PSimQ* $\langle Q \xrightarrow{\alpha} Q' \rangle$
  obtain P' **where** *PTrans*: $P \xrightarrow{\alpha} P'$ **and** $(P', Q') \in \mathscr{R}$
   by(*auto simp add*: *simulation-def*)
  from *PTrans* $\langle x \sharp \alpha \rangle$ **have** $(\nu x)P \xrightarrow{\alpha} (\nu x)P'$ **by**(*rule Res*)
  moreover from $\langle (P', Q') \in \mathscr{R} \rangle$ **have** $((\nu x)P', (\nu x)Q') \in \mathscr{R}'$ **by**(*rule A1*)
  ultimately show $\exists P'.\ (\nu x)P \xrightarrow{\alpha} P' \wedge (P', (\nu x)Q') \in \mathscr{R}'$
   by *blast*
 qed
qed

The preservation properties of simulation are usually the most involved proofs when formalising process algebras, and they will be given much attention in this thesis. Their general structure is the presented here.

5.4.3 Coinductive proofs

Before moving on to an actual coinductive proof we introduce the notion of a *block*. A block can intuitively be thought of as an inline lemma. It is always a good idea to factorise proofs in such a way so that lemmas can be used in a variety of proof contexts. Nevertheless, it is sometimes desirable to create inline lemmas when these lemmas are very specific to the proof at hand.

A proof block declares parameters, assumptions and conclusion of a logical predicate in the following manner:

Isar code	meta-logic formula
{	
fix $a::\tau_1$ $b::\tau_2$ $c::\tau_3$	
⋮	
assume A	
⋮	$\bigwedge a::\tau_1\ b::\tau_2\ c::\tau_3.\ [\![A; B]\!] \Longrightarrow C$
assume B	
⋮	
have C **by** ...	
}	

The **fix** keyword declares the names, and if necessary the types of the arguments for the lemma. Every occurrence of the **assume** keyword adds a premise to the block, and the final proved predicate of the block is its conclusion.

In the last section it was proved that bisimulation it reflexive. Proving that it is also transitive is slightly more involved.

lemma *bisimTransitive*:
 fixes P :: *toy* **and** Q :: *toy* **and** R :: *toy*
 assumes $P \sim Q$ **and** $Q \sim R$
 shows $P \sim R$
proof −
 let $?X = \sim \circ \sim$
  — set the candidate relation to the relational composition of bisimulation with itself
 from *assms* **have** $(P, R) \in ?X$ **by** *auto*
 thus $P \sim R$
 proof(*coinduct rule*: *bisimCoinduct*)
  case(*cSim P R*)
  {
   fix *P Q R*
   assume $P \sim Q$
   hence $P \hookrightarrow_\sim Q$ **by**(*rule bisim.cases*) *auto*
   moreover assume $Q \sim R$
   hence $Q \hookrightarrow_\sim R$ **by**(*rule bisim.cases*) *auto*
   ultimately have $P \hookrightarrow_{?X} R$ **by**(*simp add*: *simulation-def*) *blast*
  }
  — This block provides the following sub-lemma:
  — $\bigwedge P\ Q\ R.\ [\![P \sim Q; Q \sim R]\!] \Longrightarrow P \hookrightarrow_{?X} R$
  with ⟨$(P, R) \in ?X$⟩ **show** $P \hookrightarrow_{?X} R$ **by** *auto*
 next

case(*cSym P R*)
thus (*R*, *P*) ∈ *?X* **by**(*blast intro*: *bisim.cases*)
qed
qed

5.5 Set comprehension

Set comprehension will be used extensively in this thesis, primarily to define candidate relations for bisimulations. Sets are generally defined by constraining values over a logical predicate. For instance, the set

$$\{(x, y) : Prop\ x\ y\}$$

represents the pair of all terms x and y such that the predicate $Prop\ x\ y$ holds. There are a few special cases. The set

$$\{(x, y) : True\}$$

represents all possible pairings of the terms x and y as the predicate generating the set is always true. This is not the same as the set

$$\{(x, y)\}$$

which is just the singleton set with the pair (x, y).

There are two cases where not all the arguments to the predicate are present in the resulting set – either the extra arguments are already fixed in the proof environment, or they are quantified within the set comprehension. The set

$$\{(x, y) : Prop\ x\ y\ z\}$$

represents the set of all pairs of x and y such that the predicate $Prop\ x\ y\ z$ holds for a term z which is already fixed in the proof environment. If z is quantified within the set comprehension, this is done explicitly as

$$\{(x, y) : \forall z.\ Prop\ x\ y\ z\}$$

or

$$\{(x, y) : \exists z.\ Prop\ x\ y\ z\}.$$

5.6 Concluding remarks

This chapter is a crash course in Isabelle to enable the reader to parse the proofs provided in this thesis. It is not complete, but designed to give a rough idea of how Isabelle operates. The proofs are legible even without a deeper knowledge of Isabelle, and even though it may not always be clear *how* Isabelle internally derives a proof, it should be clear *what* is being proved, and which proof structure is used. Any Isabelle proof in this thesis is designed to be transferable to a pen-and-paper counterpart, with the aid of the information provided in this chapter.

Part II:
The calculus of communicating systems

6. The Calculus of Communicating Systems

The Calculus of Communicating Systems (CCS) is one of the oldest process calculus. It was designed around 1980 by Robin Milner and has been an important stepping stone for later, more advanced process calculi.

A CCS agent communicates with its environment through the use of *actions* and *coactions*. Given an infinite set of action names $\mathcal{N}$, the set of coactions $\overline{\mathcal{N}}$ is defined as $\{\overline{a}.\ a \in \mathcal{N}\}$. We extend complementation to include all actions, such that $\overline{\overline{a}} = a$. Additionally, we add a separate internal action, denoted by τ, which represents an internal action within an agent.

Labels in CCS are either actions, coactions or τ-actions.

Definition 6.1 (Actions)**.**

$$\alpha \stackrel{\text{def}}{=} \tau \mid a \mid \overline{a}$$

CCS processes are defined in the following way:

Definition 6.2 (Agents)**.**

$$\begin{array}{lll} P \stackrel{\text{def}}{=} & \mathbf{0} & \textit{Nil} \\ & \alpha.P & \textit{Prefix} \\ & P + Q & \textit{Sum} \\ & P \mid Q & \textit{Parallel} \\ & (\nu x)P & \textit{Restriction} \\ & !P & \textit{Replication} \end{array}$$

An agent which has no actions, often called the deadlocked or the empty agent, is denoted $\mathbf{0}$. The agent $\alpha.P$ can do the action α and reach the state P. Two agents P and Q running in parallel are represented by $P \mid Q$ and $P + Q$ represents an agent that can do either P or Q nondeterministically. A name x can be locally bound in an agent P through the use of Restriction, written $(\nu x)P$. The agent $!P$ represents an arbitrary number of instances of the agent P running in parallel.

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \textsc{Action} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \textsc{Sum1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}\ \textsc{Sum2}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2}$$

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q' \qquad a \neq \tau}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Comm} \qquad \frac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ \textsc{Scope}$$

$$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}\ \textsc{Repl}$$

Figure 6.1: The operational semantics for CCS. A semantics without structural congruence needs symmetric versions of the SUM and PAR rules. Note that the COMM-rule does not require a symmetric version as $\overline{\overline{a}} = a$.

6.1 Operational semantics

The operational semantics of CCS follows the standard pattern, and the notation $P \xrightarrow{\alpha} P'$ denotes that the agent P can do the action α and reach the state P'. The complete semantics, without structural congruence, can be found in Figure 6.1.

Two agents running in parallel can synchronise. Consider the following two agents.

$$P = a.\mathbf{0} \quad \text{and} \quad Q = \overline{a}.\mathbf{0}$$

The agents P and Q have the transitions $P \xrightarrow{a} \mathbf{0}$ and $Q \xrightarrow{\overline{a}} \mathbf{0}$ respectively. By putting these two agents in parallel P and Q can still do their actions individually but they can also synchronise.

$$\frac{P \xrightarrow{a} \mathbf{0}}{P \mid Q \xrightarrow{a} \mathbf{0} \mid Q}\ \textsc{Par1} \qquad \frac{Q \xrightarrow{\overline{a}} \mathbf{0}}{P \mid Q \xrightarrow{\overline{a}} P \mid \mathbf{0}}\ \textsc{Par2}$$

$$\frac{P \xrightarrow{a} \mathbf{0} \qquad Q \xrightarrow{\overline{a}} \mathbf{0}}{P \mid Q \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}}\ \textsc{Comm}$$

An environment can interact with the above agents over the action a. It is often desirable to restrict access to the agents to ensure that they only interact with each other and not with some other agent in the environment. This is achieved by using Restriction.

$$\dfrac{\dfrac{P \xrightarrow{a} \mathbf{0} \qquad Q \xrightarrow{\overline{a}} \mathbf{0}}{P \mid Q \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}}\ \textsc{Comm}}{(\nu a)(P \mid Q) \xrightarrow{\tau} (\nu a)(\mathbf{0} \mid \mathbf{0})}\ \textsc{Scope}$$

By restricting the name a inside the agents P and Q the outside environment can no longer communicate with P or Q using a.

Sum is used to encode nondeterministic choice. The agent $P + Q$ can behave either as P or Q yielding the following transitions:

$$\dfrac{P \xrightarrow{a} \mathbf{0}}{P + Q \xrightarrow{a} \mathbf{0}}\ \textsc{Sum1} \qquad \dfrac{Q \xrightarrow{\overline{a}} \mathbf{0}}{P + Q \xrightarrow{\overline{a}} \mathbf{0}}\ \textsc{Sum2}$$

Replication spawns an arbitrary number copies of an agent and runs them in parallel. The agent $P + Q$ cannot by itself communicate as it behaves either as P or Q but by spawning multiple copies they can communicate. Replication produces an infinite number of possible transitions, the following example illustrates one possible trace.

$$\dfrac{\dfrac{\dfrac{P \xrightarrow{a} \mathbf{0}}{P + Q \xrightarrow{a} \mathbf{0}}\ \textsc{Sum1} \qquad \dfrac{\dfrac{\dfrac{Q \xrightarrow{\overline{a}} \mathbf{0}}{P + Q \xrightarrow{\overline{a}} \mathbf{0}}\ \textsc{Sum2}}{(P + Q) \mid !(P + Q) \xrightarrow{\overline{a}} \mathbf{0} \mid !(P + Q)}\ \textsc{Par1}}{!(P + Q) \xrightarrow{\overline{a}} \mathbf{0} \mid !(P + Q)}\ \textsc{Repl}}{(P + Q) \mid !(P + Q) \xrightarrow{\tau} \mathbf{0} \mid (\mathbf{0} \mid !(P + Q))}\ \textsc{Comm}}{!(P + Q) \xrightarrow{\tau} \mathbf{0} \mid (\mathbf{0} \mid !(P + Q))}\ \textsc{Repl}$$

Note that to restrict a in this example, we have to put the ν-binder outside Replication, i.e. $!(\nu x)(P + Q)$. In the agent $(\nu x)!(P + Q)$ the name a is local to each copy of $P + Q$ making synchronisation between the copies impossible.

6.2 Nominal infrastructure

From a formalisation point of view, CCS is not much more difficult than the example process calculi presented in chapter 2. There are a few more constructors, and the notion of actions and coactions need to be modeled.

$$\left[\begin{array}{l}
R \xrightarrow{\beta} R' \\[2ex]
\bigwedge \alpha\ P\ \mathscr{C}.\ \dfrac{}{Prop\ \mathscr{C}\ \alpha.P\ \alpha\ P}\ \textsc{Act} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q\ \mathscr{C}.\ \dfrac{P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P'}{Prop\ \mathscr{C}\ (P + Q)\ \alpha\ P'}\ \textsc{Sum1} \\[2ex]
\bigwedge Q\ \alpha\ Q'\ P\ \mathscr{C}.\ \dfrac{Q \xrightarrow{\alpha} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ \alpha\ Q'}{Prop\ \mathscr{C}\ (P + Q)\ \alpha\ Q'}\ \textsc{Sum2} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q\ \mathscr{C}.\ \dfrac{P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P'}{Prop\ \mathscr{C}\ (P \mid Q)\ \alpha\ (P' \mid Q)}\ \textsc{Par1} \\[2ex]
\bigwedge Q\ \alpha\ Q'\ P\ \mathscr{C}.\ \dfrac{Q \xrightarrow{\alpha} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ \alpha\ Q'}{Prop\ \mathscr{C}\ (P \mid Q)\ \alpha\ (P \mid Q')}\ \textsc{Par2} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q\ Q'\ \mathscr{C}.\ \dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P' \\ Q \xrightarrow{\overline{\alpha}} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ \overline{\alpha}\ Q' \qquad \alpha \neq \tau \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (\tau)\ (P' \mid Q')}\ \textsc{Comm} \\[2ex]
\bigwedge P\ \alpha\ P'\ x\ \mathscr{C}.\ \dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ \alpha\ P' \\ x \sharp \alpha \qquad x \sharp \mathscr{C} & \end{pmatrix}}{Prop\ \mathscr{C}\ ((\nu x)P)\ \alpha\ ((\nu x)P')}\ \textsc{Scope} \\[2ex]
\bigwedge P\ \alpha\ P'\ \mathscr{C}.\ \dfrac{P \mid !P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (P \mid !P)\ \alpha\ P'}{Prop\ \mathscr{C}\ !P\ \alpha\ P'}\ \textsc{Repl}
\end{array}\right]$$

$$\overline{Prop\ \mathscr{C}\ R\ \beta\ R'}$$

Figure 6.2: An induction rule of the semantics defined in Figure 6.1. Induction is done on the transition $R \xrightarrow{\beta} R'$ to prove the predicate *Prop*. The freshness context $\mathscr{C}$ represents the finite set of terms with which the bound names of the SCOPE-rule may not clash. Each inductive case shares the name of the semantic rule which it represents.

CCS only has one type of atom, and that is names.

atom_decl *name*

Actions do not contain any binders, but are created as nominal datatypes to automatically generate auxiliary lemmas regarding freshness, support and equivariance.

nominal_datatype *act* = *Action name*
| *CoAction name*
| *Tau*

CCS agents contain binders through the ν-operator. This binding occurrence is declared when creating the nominal datatype.

nominal_datatype *ccs* = *CCSNil*
| *Action act ccs*
| *Sum ccs ccs*
| *Par ccs ccs*
| *Res "«name» ccs"*
| *Bang ccs*

From these definitions, Isabelle creates the following injectivity rules:

Lemma 6.3. *Injectivity rules for actions and agents.*

$$\begin{aligned}
\alpha.P = \beta.Q &\Leftrightarrow \alpha = \beta \wedge P = Q \\
P + R = Q + S &\Leftrightarrow P = Q \wedge R = S \\
P \mid R = Q \mid S &\Leftrightarrow P = Q \wedge R = S \\
(\nu x)P = (\nu y)Q &\Leftrightarrow [x].P = [y].Q \\
!P = !Q &\Leftrightarrow P = Q
\end{aligned}$$

The only special case is the one for determining equality of agents with binders, where the atom abstraction defined in Chapters 4 and 5 is used.

This injectivity lemma can be used to prove the following alpha-equivalence lemma.

Lemma 6.4. *If* $y \sharp P$ *then* $(\nu x)P = (\nu y)([(x, y)] \cdot P)$.

Proof. Follows from Lemma 6.3 and the definition of atom abstraction. □

This lemma enables alpha-conversion directly at the level of the agents, rather than using the underlying nominal layer.

6.3 Induction rules

The induction rule generated by Isabelle is displayed in Figure 6.2. Given a transition $P \xrightarrow{\alpha} P'$, the rule will prove a four place predicate of the form $Prop\,\mathscr{C}\,P\,\alpha\,P'$, where $\mathscr{C}$ is the context of names to be avoided by any freshly introduced bound names. The only rule which includes bound names in CCS is SCOPE.

The induction rule can then be used to prove the following lemma:

Lemma 6.5.

$$\textit{If } P \xrightarrow{\alpha} P' \textit{ and } x \sharp P \textit{ then } x \sharp \alpha.$$
$$\textit{If } P \xrightarrow{\alpha} P' \textit{ and } x \sharp P \textit{ then } x \sharp P'.$$

Proof. By induction on the transition $P \xrightarrow{\alpha} P'$ using the induction rule in Figure 6.2. Isabelle manages to prove all of the inductive cases using its automatic heuristics. □

6.4 Inversion rules

The inversion rule that Isabelle automatically generates can be found in Figure 6.4. Given a transition $R \xrightarrow{\beta} R'$ and a proposition *Prop* to prove, the inversion rule will generate one case for every semantic rule, with equivalence constraints to unify R, β, and R' with the required agents for each rule.

The SCOPE case behaves differently from the others as it has to reason about the bound name x. This bound name is not instantiated by the inversion case, as it is not universally quantified along with P, α and P', but by the user before invoking the rule. The constraints set on x is that it is fresh for R, β and R'. In this case, the facts $x \sharp R$ and $x \sharp R'$ will be trivially true since $R = (\nu x)P$ and $R' = (\nu x)P'$ for some P and P'.

This rule handles well for apply scripts as Isabelle's automatic tactics will handle unification nicely. It does not work as well when working with Isar-style proofs as the unifications will then either be done by hand, or by Isabelle with the tradeoff that much convenient proof infrastructure is lost in the process.

From this rule it is straightforward to derive custom tailored inversion rules for the different transition cases of interest. The derived inversion rules can be found in figure 6.4. The proof for each rule is a one line proof using the standard inversion rule, with the exception of the SCOPE case which requires a proof that $x \sharp \alpha$ using Lemma 6.5.

$$\left[\begin{array}{l}
R \xrightarrow{\beta} R' \\[2ex]
\bigwedge \alpha\ P.\ \dfrac{R = \alpha.P \qquad \beta = \alpha \qquad R' = P}{Prop}\ \textsc{Act} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q.\ \dfrac{R = P + Q \qquad \beta = \alpha \qquad R' = P' \qquad P \xrightarrow{\alpha} P'}{Prop}\ \textsc{Sum1} \\[2ex]
\bigwedge Q\ \alpha\ Q'\ P.\ \dfrac{R = P + Q \qquad \beta = \alpha \qquad R' = Q' \qquad Q \xrightarrow{\alpha} Q'}{Prop}\ \textsc{Sum2} \\[2ex]
\bigwedge P\ \alpha\ P'\ Q.\ \dfrac{R = P \mid Q \qquad \beta = \alpha \qquad R' = P' \mid Q \qquad P \xrightarrow{\alpha} P'}{Prop}\ \textsc{Par1} \\[2ex]
\bigwedge Q\ \alpha\ Q'\ P.\ \dfrac{R = P \mid Q \qquad \beta = \alpha \qquad R' = P \mid Q' \qquad Q \xrightarrow{\alpha} Q'}{Prop}\ \textsc{Par2} \\[2ex]
P\ \alpha\ P'\ Q\ Q'.\ \dfrac{\begin{pmatrix} R = P \mid Q & \beta = \tau & R' = P' \mid Q' \\ P \xrightarrow{\alpha} P' & Q \xrightarrow{\overline{\alpha}} Q' & \alpha \neq \tau \end{pmatrix}}{Prop}\ \textsc{Comm} \\[2ex]
\bigwedge P\ \alpha\ P'.\ \dfrac{\left(\dfrac{(x \sharp R \qquad x \sharp \beta \qquad x \sharp R')}{R = (\nu x)P \wedge \beta = \alpha \wedge R' = (\nu x)P' \wedge P \xrightarrow{\alpha} P' \wedge x \sharp \alpha}\right)}{Prop}\ \textsc{Scope} \\[2ex]
\bigwedge P\ \alpha\ P'.\ \dfrac{R = !P \qquad \beta = \alpha \qquad R' = P' \qquad P \mid !P \xrightarrow{\alpha} P'}{Prop}\ \textsc{Repl}
\end{array}\right]$$
$$\overline{Prop}$$

Figure 6.3: An inversion rule of the semantics defined in Figure 6.1. Inversion is done on the transition $R \xrightarrow{\beta} R'$ to prove the predicate *Prop*. The bound name x in the SCOPE-case is quantified for the entire lemma, and must be proven to be fresh for R, β, and R' in order to use the inversion rule. Each inductive case shares the name of the semantic rule which it represents.

$$\frac{\alpha.P \xrightarrow{\beta} P' \qquad Prop\ \alpha\ P}{Prop\ \beta\ P'}\ \textsc{Action}$$

$$\frac{\left[P + Q \xrightarrow{\alpha} R \qquad \bigwedge P'.\ \frac{P \xrightarrow{\alpha} P'}{Prop\ P'} \qquad \bigwedge Q'.\ \frac{Q \xrightarrow{\alpha} Q'}{Prop\ Q'}\right]}{Prop\ R}\ \textsc{Sum}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{\alpha} R \\ \bigwedge P'.\ \frac{P \xrightarrow{\alpha} P'}{Prop\ \alpha\ (P' \mid Q)} \qquad \bigwedge Q'.\ \frac{Q \xrightarrow{\alpha} Q'}{Prop\ \alpha\ (P \mid Q')} \\ \bigwedge P'\ Q'\ a.\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q' \qquad a \neq \tau \qquad \alpha = \tau}{Prop\ (\tau)\ (P' \mid Q')} \end{array}\right]}{Prop\ \alpha\ R}\ \textsc{Par}$$

$$\frac{\left[(\nu x)P \xrightarrow{\alpha} P' \qquad \bigwedge P'.\ \frac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{Prop\ ((\nu x)P')}\right]}{Prop\ P'}\ \textsc{Scope}$$

Figure 6.4: Inversion rules for the operational semantics

$$\frac{\left[\begin{array}{l} !P \xrightarrow{\beta} R \\ \\ \bigwedge \alpha\ P'\ \mathscr{C}.\ \dfrac{P \xrightarrow{\alpha} P'}{Prop\ \mathscr{C}\ (P \mid !P)\ \alpha\ (P' \mid !P)}\ \textsc{Par1} \\ \\ \bigwedge \alpha\ P'\ \mathscr{C}.\ \dfrac{!P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ !P\ \alpha\ P'}{Prop\ \mathscr{C}\ (P \mid !P)\ \alpha\ (P \mid P')}\ \textsc{Par2} \\ \\ \bigwedge \alpha\ P'\ P''\ \mathscr{C}.\ \dfrac{\begin{pmatrix} P \xrightarrow{\alpha} P' \qquad !P \xrightarrow{\overline{\alpha}} P'' \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ !P\ \overline{\alpha}\ P'' \qquad \alpha \neq \tau \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau)\ (P' \mid P'')}\ \textsc{Comm} \\ \\ \bigwedge \alpha\ P'\ \mathscr{C}.\ \dfrac{P \mid !P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (P \mid !P)\ \alpha\ P'}{Prop\ \mathscr{C}\ !P\ \alpha\ P'}\ \textsc{Repl} \end{array}\right]}{Prop\ \mathscr{C}\ !P\ \beta\ R}$$

Figure 6.5: Custom induction rule for Replication. Induction is done on the transition $!P \xrightarrow{\alpha} P'$ to prove the predicate *Prop*. The rule has one case for Replication, and one case for every semantic rule for Parallel.

6.5 Induction on replicated agents

Replication is the only operator which appears in the premise of its inference rule. Therefore, even though it would be possible to generate an inversion rule for it, that rule would not be very useful. An inversion on the transition $!P \xrightarrow{\alpha} P'$ would yield that this transition was derived from $P \mid !P \xrightarrow{\alpha} P'$, where inversion on this transitions provides three cases, one for PAR1, PAR2 and COMM respectively. The cases for PAR2 and COMM have the transition $!P \xrightarrow{\alpha} P'$ in their derivations, causing a circularity.

Proofs involving Replication typically are by induction on the length of the inference chain, rather than structural inversion of the transitions. A custom made induction rule for Replication can be found in Figure 6.5.

7. Strong bisimilarity

Bisimilarity for CCS is defined in the same manner as in Section 2.4. Two agents P and Q are bisimilar if for every action α that P can do, Q can mimic that action and their resulting derivatives are bisimilar, and vice versa. A general strategy to determine whether or not two agents are bisimilar is to find a relation which contains the two agents, and for every action that one of the agent can do, the other agent must be able to mimic that action and the derivatives should be in that relation.

7.1 Definitions

Strong simulation for CCS is defined in the standard way. An agent P simulates an agent Q preserving the relation $\mathscr{R}$, written $P \hookrightarrow_{\mathscr{R}} Q$, if for every action P can do, Q can do the same action, and their derivatives are in $\mathscr{R}$. More formally, we have the following definition:

Definition 7.1 (Simulation)**.** *An agent P simulating an agent Q preserving $\mathscr{R}$ is denoted $P \hookrightarrow_{\mathscr{R}} Q$.*

$$P \hookrightarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall\, a\, Q'.\ Q \xrightarrow{a} Q' \longrightarrow (\exists P'.\ P \xrightarrow{a} P' \wedge (P', Q') \in \mathscr{R})$$

We want to coinductively define bisimilarity as the largest symmetric relation $\sim$ s.t. whenever $P \sim Q$ it holds that $P \hookrightarrow_{\sim} Q$. In order to coinductively define a relation, the function that generates it needs to be monotonic. More precisely, the following lemma is needed.

Lemma 7.2. *If $P \hookrightarrow_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \hookrightarrow_{\mathscr{R}'} Q$.*

Proof. Follows from Definition 7.1. □

We can now define bisimilarity.

Definition 7.3 (Bisimilarity)**.** *Bisimilarity, denoted $\sim$, is defined coinductively as the largest relation satisfying:*

$$\begin{aligned} P \sim Q \Longrightarrow P &\hookrightarrow_{\sim} Q && \textsc{Simulation} \\ \wedge\ Q &\sim P && \textsc{Symmetry} \end{aligned}$$

7.1.1 Primitive inference rules

From the definitions of simulation and bisimilarity we can derive the following introduction and elimination rules.

Lemma 7.4. *Introduction and elimination rules for simulation*

$$\frac{\bigwedge \alpha\ Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}}{P \hookrightarrow_{\mathscr{R}} Q}\ \hookrightarrow\text{-I}$$

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}\ \hookrightarrow\text{-E}$$

Proof. Follows immediately from Definition 7.1. □

Lemma 7.5. *Introduction and elimination rules for bisimilarity.*

$$\frac{P \hookrightarrow_{\sim} Q \qquad Q \sim P}{P \sim Q}\ \sim\text{-I} \qquad \frac{P \sim Q}{P \hookrightarrow_{\sim} Q}\ \sim\text{-E1} \qquad \frac{P \sim Q}{P \hookrightarrow_{\sim} Q}\ \sim\text{-E2}$$

Lemma 7.6. *Coinduction rule for bisimilarity.*

$$\frac{\begin{array}{l}(P, Q) \in \mathscr{X} \\ \bigwedge R\ S.\ \dfrac{(R, S) \in \mathscr{X}}{R \hookrightarrow_{\mathscr{X} \cup \sim} S}\ \textsc{Simulation} \\ \bigwedge R\ S.\ \dfrac{(R, S) \in \mathscr{X}}{(S, R) \in \mathscr{X}}\ \textsc{Symmetry}\end{array}}{P \sim Q}$$

Proof. Derived from the coinduction rule that Isabelle provides from Definition 7.3. □

This lemma will be implicitly used in coinductive proofs. To prove that two agents are bisimilar, a symmetric candidate relation $\mathscr{X}$ must be chosen which contains the two agents and where all member pairs of $\mathscr{X}$ are simulations preserving $\mathscr{X} \cup \sim$. The following coinduction rule can be derived:

Simulations are parametrised on an arbitrary relation $\mathscr{R}$. Each operator in CCS is provided with a set of constraints such that the operator preserves $\mathscr{R}$. This set should be kept as small as possible as each constraint will have to be proven when we establish preservation properties of bisimilarity. This

section covers all proofs that are needed to show that a relation is preserved by all operators.

In the cases where a simulation occurs both in the premise of a lemma and its conclusions, such as the proof for transitivity (Lemma 7.8 bellow) or preservation of Parallel (Lemma 7.17 bellow), the simulation relation used in premise and conclusion are not the same. The reason for this will be made clearer from the bisimilarity proofs, but suffice to say this makes the lemmas more general.

7.2 Bisimulation is an equivalence relation

We first establish lemmas for reflexivity and transitivity. In order for a simulation relation to be reflexive, it has to at least contain the identity relation.

Lemma 7.7. *If* $Id \subseteq \mathscr{R}$ *then* $P \hookrightarrow_{\mathscr{R}} P$.

Proof. Follows immediately from Definition 7.1. Since the simulating relation contains the identity relation, any derivative of P will be in it. □

Lemma 7.8. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *and* $Q \hookrightarrow_{\mathscr{R}'} R$ *and* $\mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}''$ *then* $P \hookrightarrow_{\mathscr{R}''} R$.

Proof. Follows from Definition 7.1. The derivatives of P and Q are in $\mathscr{R}$ and the derivatives of Q and R are in $\mathscr{R}'$. The assumption $\mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}''$ then ensures that the derivatives of P and R are in $\mathscr{R}''$. □

Lemma 7.9. *Bisimulation is an equivalence relation*

Proof. **Reflexivity:** By coinduction with $\mathscr{X}$ set to the identity relation. Lemma 7.7 discharges the simulation case.

Symmetry: Follows immediately from ∼-E2.

Transitivity: By coinduction with $\mathscr{X}$ set to $\sim \circ \sim$. Lemma 7.8 discharges the simulation case.

□

7.3 Preservation properties

Bisimilarity is a congruence, i.e. it is preserved by all operators.

7.3.1 Prefix

The only requirement of the candidate relation needed here is that the agents under the prefix are in the relation.

Lemma 7.10. *If* $(P, Q) \in \mathscr{R}$ *then* $\alpha.P \hookrightarrow_{\mathscr{R}} \alpha.Q$.

Proof. Follows from Definition 7.1 and the fact that $a.P$ and $a.Q$ can each only do an a-action (Figure 6.4 ACTION) and $(P, Q) \in \mathscr{R}$. □

Lemma 7.11. *If* $P \sim Q$ *then* $\alpha.P \sim \alpha.Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(a.P,\ a.Q) \mid P \sim Q\}$ and Lemma 7.10. □

7.3.2 Sum

In order for simulation to be preserved by Sum, the candidate relation must include the identity relation for the case where R does an action.

Lemma 7.12. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *and* $Id \subseteq \mathscr{R}'$ *then* $P + R \hookrightarrow_{\mathscr{R}'} Q + R$.

Proof. Follows from Definition 7.1, the SUM inversion rule in Figure 6.4, and the SUM1 and SUM2 rules from Figure 6.1. □

Lemma 7.13. *If* $P \sim Q$ *then* $P + R \sim Q + R$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(P + R,\ Q + R) \mid P \sim Q\}$, Lemma 7.12 and the fact that $\sim$ is reflexive. □

7.3.3 Restriction

In order to prove that simulation is preserved by Restriction, the same must hold for the candidate relation.

Lemma 7.14.

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad \bigwedge R\, S\, y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}'}}{(\nu x)P \hookrightarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from Definition 7.1, the SCOPE inversion rule in Figure 6.4, and the SCOPE-rule from Figure 6.1. □

Lemma 7.15. *If* $P \sim Q$ *then* $(\nu x)P \sim (\nu x)Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)P,\ (\nu x)Q) \mid P \sim Q\}$ and Lemma 7.14. □

7.3.4 Parallel

In order to prove that bisimilarity is preserved by Parallel we will start by proving a more general lemma for composing simulations with Parallel.

Lemma 7.16.

$$\frac{\begin{array}{l} P \hookrightarrow_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \\ R \hookrightarrow_{\mathscr{R}'} T \qquad (R, T) \in \mathscr{R}' \\ \bigwedge P'\, Q'\, R'\, T'.\ \dfrac{(P', Q') \in \mathscr{R} \qquad (R', T') \in \mathscr{R}'}{(P' \mid R', Q' \mid T') \in \mathscr{R}''} \end{array}}{P \mid R \hookrightarrow_{\mathscr{R}''} Q \mid T}$$

Proof. The Isabelle proof of this lemma can be found in Figure 7.1. □

This lemma states that if P simulates Q preserving $\mathscr{R}$, and R simulates T preserving $\mathscr{R}'$, then $P \mid R$ simulates $Q \mid T$ preserving $\mathscr{R}''$ as long as long as any pair in $\mathscr{R}$ composed by Parallel with pairs in $\mathscr{R}'$ are in $\mathscr{R}''$.

This lemma is more general than strictly necessary to prove that bisimilarity is preserved by Parallel. However, it will be useful when we prove that bisimilarity is preserved by Replication. The lemma needed for parallel preservation is easily derivable.

Lemma 7.17.

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge S\, T\, U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U, T \mid U) \in \mathscr{R}'}}{P \mid R \hookrightarrow_{\mathscr{R}'} Q \mid R}$$

Proof. Follows from Lemma 7.16 by setting its relations $\mathscr{R}''$ to $\mathscr{R}'$ and $\mathscr{R}'$ to the identity relation. □

We can now prove that bisimilarity is preserved by the Parallel.

Lemma 7.18. *If $P \sim Q$ then $P \mid R \sim Q \mid R$.*

Proof. The Isabelle proof of this lemma can be found in Figure 7.2. □

lemma parPres:
fixes P :: ccs **and** Q :: ccs
and $\mathscr{R}$ :: (ccs × ccs) set **and** $\mathscr{R}'$ :: (ccs × ccs) set **and** $\mathscr{R}''$ :: (ccs × ccs) set

assumes $P \hookrightarrow_{\mathscr{R}} Q$ **and** (P, Q) ∈ $\mathscr{R}$
and $R \hookrightarrow_{\mathscr{R}'} T$ **and** (R, T) ∈ $\mathscr{R}'$
and C1: $\bigwedge$P′ Q′ R′ T′. ⟦(P′, Q′) ∈ $\mathscr{R}$; (R′, T′) ∈ $\mathscr{R}'$⟧ ⟹
(P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$

shows $P \mid R \hookrightarrow_{\mathscr{R}''} Q \mid T$
proof(induct rule: simI) — *Apply introduction rule* ↪-I
case(Sim α U)
from ‹Q | T $\xrightarrow{\alpha}$ U›
show ?case
proof(induct rule: parCases) — *Apply* PAR *inversion rule from Figure 6.4*

PAR1 *case*
Given that $Q \xrightarrow{\alpha} Q'$ *prove that there exists an S such that*
$P \mid R \xrightarrow{\alpha} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(cPar1 Q′)
from ‹$P \hookrightarrow_{\mathscr{R}} Q$› ‹Q $\xrightarrow{\alpha}$ Q′› **obtain** P′ **where** P $\xrightarrow{\alpha}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(rule simE)
from ‹P ⟼α ≺ P′› **have** P | R $\xrightarrow{\alpha}$ P′ | R **by**(rule Par1)
moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R, T) ∈ $\mathscr{R}'$› **have** (P′ | R, Q′ | T) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\alpha}$ S ∧ (S, Q′ | T) ∈ $\mathscr{R}''$ **by** blast
next

PAR*2 case*
Given that $T \xrightarrow{\alpha} T'$ *prove that there exists an S such that*
$P \mid R \xrightarrow{\alpha} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T′)
from ‹$R \hookrightarrow_{\mathscr{R}'} T$› ‹T $\xrightarrow{\alpha}$ T′›
obtain R′ **where** R $\xrightarrow{\alpha}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(rule simE)
from ‹R $\xrightarrow{\alpha}$ R′› **have** P | R $\xrightarrow{\alpha}$ P | R′ **by**(rule Par2)
moreover from ‹(P, Q) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$› **have** (P | R′, Q | T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\alpha}$ S ∧ (S, Q | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM *case*

Given that $Q \xrightarrow{a} Q'$ *and* $T \xrightarrow{\overline{a}} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm Q′ T′ a)
from ⟨$P \hookrightarrow_{\mathscr{R}} Q$⟩ ⟨Q $\xrightarrow{a}$ Q′⟩
obtain P′ **where** P $\xrightarrow{a}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$ **by**(rule simE)
from ⟨$R \hookrightarrow_{\mathscr{R}'} T$⟩ ⟨T $\xrightarrow{\overline{a}}$ T′⟩
obtain R′ **where** R $\xrightarrow{\overline{a}}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$ **by**(rule simE)
from ⟨P $\xrightarrow{a}$ P′⟩ ⟨R $\xrightarrow{\overline{a}}$ R′⟩ ⟨a ≠ τ⟩ **have** P | R $\xrightarrow{\tau}$ P′ | R′
by(rule Comm)
moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R′, T′) ∈ $\mathscr{R}'$⟩
have (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\tau}$ S ∧ (S, Q′ | T′) ∈ $\mathscr{R}''$ **by** blast
qed
qed

Figure 7.1: The Isar proof for that simulation is preserved by Parallel. Note that both the introduction rule simI and the inversion rule parCases are used as induction rules.

```
lemma bisimParPres:
 fixes P :: ccs and Q :: ccs and R :: ccs
 assumes P ~ Q
 shows P | R ~ Q | R
proof –
 let ?X = {(S, T) | P Q R S T. P ~ Q ∧ S = P | R ∧ T = Q | R}
 from ‹P ~ Q› have (P | R, Q | R) ∈ ?X by auto
 thus P | R ~ Q | R
 proof(coinduct rule: bisimCoinduct) — Apply coinduction using Lemma 7.6
      SIMULATION case
      Given that (S, T) ∈ ?X, prove that S ↪_{?X ∪ ~} T.
  case(cSim S T)
  {
   fix P Q R
   assume P ~ Q
   moreover hence P ↪_~ Q by(rule bisimE)
   moreover have ⋀P Q R. P ~ Q ⟹ (P | R, Q | R) ∈ ?X by auto
   ultimately have P | R ↪_{?X} Q | R
    by(rule-tac simParPres)
   hence P | R ↪_{?X ∪ ~} Q | R
    by(rule-tac monotonic) auto
  }
      This block proves that for all P, Q, and R, if P ~ Q then
      P | R ↪_{?X ∪ ~} Q | R.
  thus S ↪_{?X ∪ ~} T using ‹(S, T) ∈ ?X› by auto
 next
      SYMMETRY case
      Given that (S, T) ∈ ?X, prove that (T, S) ∈ ?X.
  case(cSym S T)
  from ‹(S, T) ∈ ?X› show (T, S) ∈ ?X
   by(auto dest: strongBisim.symmetric)
 qed
qed
```

Figure 7.2: The Isabelle proof that strong bisumlation is preserved by the Parallel.

7.3.5 Replication

Proving that bisimilarity is preserved by the Replication is more involved than the preservation proofs covered so far. The difficulty is to choose the right candidate relation. The approach is to inductively define a candidate relation, and prove that it is a bisimulation. As Replication spawns an arbitrary number of agents running in parallel, any candidate relation must support the constraints required to prove that bisimilarity is preserved by Parallel, as well as being preserved by Replication.

Definition 7.19 (*bangRel*)**.** *The bangRel relation is parametrised with a relation* $\mathscr{R}$*.*

If $(P, Q) \in \mathscr{R}$ *then* $(!P, !Q) \in \mathit{bangRel}\,\mathscr{R}$*.*
If $(R, T) \in \mathscr{R}$ *and* $(P, Q) \in \mathit{bangRel}\,\mathscr{R}$ *then* $(R \mid P, T \mid Q) \in \mathit{bangRel}\,\mathscr{R}$*.*

The predicate *bangRel* takes a relation $\mathscr{R}$ as an argument, and returns a relation which is closed by Replication and by Parallel. Moreover, the agents appearing on the right hand side of the |-operator, are members of *bangRel* $\mathscr{R}$; the intuition is that as with Replication, the *bangRel* predicate can be unfolded, adding new parallel agents an arbitrary number of times.

The next step is to prove what is required of a relation $\mathscr{R}$ for a simulation to preserve *bangRel* $\mathscr{R}$.

Lemma 7.20. *Simulation is preserved by Replication.*

$$\frac{(P, Q) \in \mathscr{R} \qquad \bigwedge R\,S.\ \dfrac{(R, S) \in \mathscr{R}}{R \hookrightarrow_{\mathscr{R}} S}}{!P \hookrightarrow_{\mathit{bangRel}\,\mathscr{R}} !Q}$$

Proof. By induction using the induction rule for Replication from Figure 6.5 on the transitions that $!Q \xrightarrow{\alpha} Q'$. □

Lemma 7.21. *If* $P \sim Q$ *then* $!P \sim !Q$*.*

Proof. By coinduction with X set to *bangRel* $\sim$. The candidate relation is symmetric since $\sim$ is symmetric. The simulation cases are resolved by Lemma 7.20 for the !-case, and Lemma 7.16 for the | -case. □

7.4 Bisimilarity is a congruence

We now have the lemmas we need to prove that bisimilarity is a congruence.

Theorem 7.1. *Strong bisimilarity is a congruence.*

Proof. That strong bisimulation is an equivalence relation follows from Lemma 7.9, and that it is preserved by all operators follows from lemmas 7.11, 7.13, 7.15, 7.18, and 7.21. □

8. Structural congruence

In this chapter, we will prove that all structurally congruent agents are also bisimilar. The laws of structural congruence for CCS can be found in Figure 8.1.

8.1 Abelian monoid laws for parallel

All of these proofs are one line proofs, modulo choosing the coinductive relations for the bisimulation proofs.

8.1.1 Parallel is commutative

The agents $P \mid Q$, and $Q \mid P$, are structurally equal. As such, only one simulation lemma is required. However, in order for them to simulate each other, the candidate relation containing the derivatives must include all parallel commutative pairs of agents.

Lemma 8.1.

$$\frac{\bigwedge R\ T.\ (R \mid T,\ T \mid R) \in \mathscr{R}}{P \mid Q \hookrightarrow_{\mathscr{R}} Q \mid P}$$

Proof. By the definition of $\hookrightarrow$ and case analysis using the PAR rule from Fig. 6.4. The cases are then discharged using the PAR1, PAR2 and COMM rules from the operational semantics. □

From this lemma, the bisimulation follows.

Lemma 8.2. $P \mid Q \sim Q \mid P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(P \mid Q,\ Q \mid P) : \mathit{True}\},$$

and Lemma 8.1. □

The structural congruence ≡ is defined as the smallest congruence satisfying the following laws:

1. The abelian monoid laws for Parallel: commutativity $P \mid Q \equiv Q \mid P$, associativity $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and Nil as unit $P \mid \mathbf{0} \equiv P$; and the same laws for Sum.
2. The unfolding law $!P \equiv P \mid !P$
3. The scope extension laws

$$\begin{array}{rcll} (\nu x)\mathbf{0} & \equiv & \mathbf{0} & \\ (\nu x)(P \mid Q) & \equiv & P \mid (\nu x)Q & \text{if } x \sharp P \\ (\nu x)(P + Q) & \equiv & P + (\nu x)Q & \text{if } x \sharp P \\ (\nu x)\alpha.P & \equiv & \alpha.(\nu x)P & \text{if } x \sharp \alpha \\ (\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P & \end{array}$$

Figure 8.1: The definition of structural congruence.

8.1.2 Parallel is associative

These two simulation lemmas require that their candidate relation contain all pairs of parallel left associative, and right associative agents respectively.

Lemma 8.3.

$$\frac{\bigwedge S\ T\ U.\ ((S \mid T) \mid U, S \mid (T \mid U)) \in \mathscr{R}}{(P \mid Q) \mid R \hookrightarrow_{\mathscr{R}} P \mid (Q \mid R)}$$

$$\frac{\bigwedge S\ T\ U.\ (S \mid (T \mid U), (S \mid T) \mid U) \in \mathscr{R}}{P \mid (Q \mid R) \hookrightarrow_{\mathscr{R}} (P \mid Q) \mid R}$$

Proof. By the definition of $\hookrightarrow$ and case analysis on the possible transitions, using the PAR inversion rule from Fig. 6.4. The individual cases are then discharged using the PAR1, PAR2 and COMM rules from the operational semantics. □

When proving the bisimulation lemma, the candidate relation is chosen so that it is symmetric and meets the individual requirements of the simulation lemmas in Lemma 8.3.

Lemma 8.4. $(P \mid Q) \mid R \sim P \mid (Q \mid R)$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((P \mid Q) \mid R, P \mid (Q \mid R)) : \mathit{True}\} \cup$$
$$\{(P \mid (Q \mid R), (P \mid Q) \mid R) : \mathit{True}\},$$

and Lemma 8.3. □

8.1.3 Parallel has Nil as unit

Lemma 8.5.

$$\frac{\bigwedge Q.\ (Q \mid \mathbf{0},\ Q) \in \mathscr{R}}{P \mid \mathbf{0} \hookrightarrow_{\mathscr{R}} P} \qquad \frac{\bigwedge Q.\ (Q,\ Q \mid \mathbf{0}) \in \mathscr{R}}{P \hookrightarrow_{\mathscr{R}} P \mid \mathbf{0}}$$

Proof. By the definition of $\hookrightarrow$, case analysis using the PAR from Fig. 6.4 and the PAR1 rule from the operational semantics. □

Lemma 8.6. $P \mid \mathbf{0} \sim P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(P \mid \mathbf{0},\ P) : \mathit{True}\} \cup \{(P,\ P \mid \mathbf{0}) : \mathit{True}\},$$

and Lemma 8.5. □

8.2 Abelian monoid laws for Sum

The abelian monoid laws for sum are significantly easier to prove than their counterparts for Parallel. The main reason for this is that whenever an agent does a choice, the rest of the agent is discarded leaving only the derivative of the chosen agent.

8.2.1 Sum is commutative

As for the corresponding proof for the Parallel, only one simulation lemma is needed to prove that Sum is commutative. However, the only requirement on the candidate relation is that it is reflexive. The reason for this is that whichever agent in $P + Q$ does an action, only its derivative will remain, and $Q + P$ can mimic with the same action.

Lemma 8.7.

$$\frac{\mathit{Id} \subseteq \mathscr{R}}{P + Q \hookrightarrow_{\mathscr{R}} Q + P}$$

Proof. By the definition of $\hookrightarrow$, case analysis using the SUM rule from Fig. 6.4 and the SUM1 and SUM2 rules from the operational semantics. □

The candidate relation for the bisimulation proof is a symmetric binary set which contains only the original agents. The coinduction rule will require the derivatives to be in either this set, or that they are bisimilar. Since bisimulation is reflexive, it meets the only constraint that the simulation lemma imposes.

Lemma 8.8. $P + Q \sim Q + P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(P + Q, Q + P), (Q + P, P + Q)\},$$

reflexivity of bisimilarity, and Lemma 8.7. □

8.2.2 Sum is associative

As for the corresponding proof for Parallel, two simulation lemmas are required. However, for the same reasons as for the commutative case for sum, the only requirement needed on the candidate relation is that it is reflexive.

Lemma 8.9.

$$\frac{Id \subseteq \mathscr{R}}{(P + Q) + R \hookrightarrow_{\mathscr{R}} P + (Q + R)} \qquad \frac{Id \subseteq \mathscr{R}}{P + (Q + R) \hookrightarrow_{\mathscr{R}} (P + Q) + R}$$

Proof. By the definition of $\hookrightarrow$, case analysis using the SUM inversion rule from Fig. 6.4 and the SUM1 and SUM2 rules from the operational semantics. □

The bisimulation proof is then proven in a similar manner as Lemma 8.8.

Lemma 8.10. $(P + Q) + R \sim P + (Q + R)$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((P + Q) + R, P + (Q + R)), (P + (Q + R), (P + Q) + R)\},$$

reflexivity of bisimilarity, and Lemma 8.9. □

8.2.3 Sum has Nil as unit

Lemma 8.11.

$$\frac{Id \subseteq \mathscr{R}}{P + \mathbf{0} \hookrightarrow_{\mathscr{R}} P} \qquad \frac{Id \subseteq \mathscr{R}}{P \hookrightarrow_{\mathscr{R}} P + \mathbf{0}}$$

Proof. By the definition of $\hookrightarrow$, case analysis using the SUM from Fig. 6.4 and the SUM1 rule from the operational semantics. □

Lemma 8.12. $P + \mathbf{0} \sim P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(P + \mathbf{0}, P), (P, P + \mathbf{0})\},$$

reflexivity of bisimilarity, and Lemma 8.5. □

8.3 Scope extension laws

Lemma 8.13.

$$(\nu x)\mathbf{0} \hookrightarrow_{\mathscr{R}} \mathbf{0} \qquad\qquad \mathbf{0} \hookrightarrow_{\mathscr{R}} (\nu x)\mathbf{0}$$

Proof. Follows from the definition of $\hookrightarrow$. Since neither $(\nu x)\mathbf{0}$ nor $\mathbf{0}$ has any transitions no constraints need to be set on $\mathscr{R}$. □

Lemma 8.14. $(\nu x)\mathbf{0} \sim \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)\mathbf{0}, \mathbf{0}), (\mathbf{0}, (\nu x)\mathbf{0})\},$$

and Lemma 8.13. □

8.3.1 Scope extension for parallel

Lemma 8.15.

$$\frac{x \sharp P \qquad \bigwedge y\, R\, T.\ \dfrac{y \sharp R}{((\nu y)(R \mid T), R \mid (\nu y)T) \in \mathscr{R}}}{(\nu x)(P \mid Q) \hookrightarrow_{\mathscr{R}} P \mid (\nu x)Q}$$

$$\frac{x \sharp P \qquad \bigwedge y\, R\, T.\ \dfrac{y \sharp R}{(R \mid (\nu y)T, (\nu y)(R \mid T)) \in \mathscr{R}}}{P \mid (\nu x)Q \hookrightarrow_{\mathscr{R}} (\nu x)(P \mid Q)}$$

Proof. Follows from the definition of $\hookrightarrow$, The SCOPE and PAR inversion rules from Figure 6.4, Lemma 6.5 and the SCOPE PAR and COMM rules from the operational semantics. □

Lemma 8.16. *If $x \sharp P$ then $(\nu x)(P \mid Q) \sim P \mid (\nu x)Q$.*

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)(P \mid Q), P \mid (\nu x)Q) : x \sharp P\} \cup \{(P \mid (\nu x)Q, (\nu x)(P \mid Q)) : x \sharp P\}$$

and Lemma 8.15. □

Using this lemma we can derive another very useful rule which states that binding a name in an agent where it does not occur does nothing.

Lemma 8.17. *If $x \sharp P$ then $(\nu x)P \sim P$.*

Proof. The proof is derivable from the structural congruence rules proven so far. The numbers of the lemmas used are displayed for every rewrite step.

$$\begin{array}{lllllll}
(\nu x)P & \overset{8.6,\,7.9}{\sim} & (\nu x)P \mid \mathbf{0} & \overset{8.2}{\sim} & \mathbf{0} \mid (\nu x)P & \overset{8.16,\,7.9}{\sim} & (\nu x)(\mathbf{0} \mid P) \\
 & \overset{7.15,\,8.2}{\sim} & (\nu x)(P \mid \mathbf{0}) & \overset{8.16}{\sim} & P \mid (\nu x)\mathbf{0} & \overset{8.2}{\sim} & (\nu x)\mathbf{0} \mid P \\
 & \overset{7.18,\,8.14}{\sim} & \mathbf{0} \mid P & \overset{8.2}{\sim} & P \mid \mathbf{0} & \overset{8.6}{\sim} & P
\end{array}$$

□

8.3.2 Scope extension for sum

Lemma 8.18.

$$\frac{x \sharp P \qquad \bigwedge y\, R.\ \dfrac{y \sharp R}{((\nu y)R, R) \in \mathscr{R}} \qquad Id \subseteq \mathscr{R}}{(\nu x)(P + Q) \hookrightarrow_{\mathscr{R}} P + (\nu x)Q}$$

$$\frac{x \sharp P \qquad \bigwedge y\, R.\ \dfrac{y \sharp R}{(R, (\nu y)R) \in \mathscr{R}} \qquad Id \subseteq \mathscr{R}}{P + (\nu x)Q \hookrightarrow_{\mathscr{R}} (\nu x)(P + Q)}$$

Proof. Follows from the definition of $\hookrightarrow$, The SCOPE and SUM inversion rules from Figure 6.4 and the SCOPE and SUM rules from the operational semantics. □

Lemma 8.19. *If $x \sharp P$ then $(\nu x)(P + Q) \sim P + (\nu x)Q$.*

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)(P + Q), P + (\nu x)Q), (P + (\nu x)Q, (\nu x)(P + Q))\},$$

reflexivity and symmetry of bisimilarity, and lemmas 8.17 and 8.18. □

8.3.3 Scope extension of prefixes

Lemma 8.20.

$$\frac{x \sharp \alpha \qquad Id \subseteq \mathscr{R}}{(\nu x)\alpha.P \hookrightarrow_{\mathscr{R}} \alpha.(\nu x)P} \qquad \frac{x \sharp \alpha \qquad Id \subseteq \mathscr{R}}{\alpha.(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)\alpha.P}$$

Proof. Follows from the definition of $\hookrightarrow$, the ACTION and SCOPE inversion rules from Figure 6.4 and the ACTION and SCOPE rules from the operational semantics. □

Lemma 8.21. *If* $x \sharp \alpha$ *then* $(\nu x)\alpha.P \sim \alpha.(\nu x)P$.

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)\alpha.P,\ \alpha.(\nu x)P),\ (\alpha.(\nu x)P,\ (\nu x)\alpha.P)\},$$

reflexivity of bisimilarity, and Lemma 8.20. □

8.3.4 Restriction is commutative

Lemma 8.22.

$$\frac{\bigwedge Q.\ ((\nu x)(\nu y)Q,\ (\nu y)(\nu x)Q) \in \mathscr{R}}{(\nu x)(\nu y)P \hookrightarrow_{\mathscr{R}} (\nu y)(\nu x)P}$$

Proof. Follows from the definition of $\hookrightarrow$, the SCOPE inversion rule from Figure 6.4, and the SCOPE rule from the operational semantics. □

Lemma 8.23. $(\nu x)(\nu y)P \sim (\nu y)(\nu x)P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)(\nu y)P,\ (\nu y)(\nu x)P) : \mathit{True}\}$$

and Lemma 8.22. □

8.4 The unfolding law

Lemma 8.24.

$$\frac{Id \subseteq \mathscr{R}}{P \mid !P \hookrightarrow_{\mathscr{R}} !P} \qquad \frac{Id \subseteq \mathscr{R}}{!P \hookrightarrow_{\mathscr{R}} P \mid !P}$$

Proof. Follows from the definition of $\hookrightarrow$, the REPL and PAR rules from Figure 6.4 and the REPL an PAR rules from the operational semantics. □

Lemma 8.25. $!P \sim P \mid !P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(!P,\ P \mid !P),\ (P \mid !P, !P)\}.$$

The goal follows immediately from Lemma 8.24 and reflexivity of bisimilarity. □

8.5 Bisimilarity includes structural congruence

The main structural congruence theorem follows from the combined lemmas in this section.

Theorem 8.1. *If* $P \equiv Q$ *then* $P \sim Q$.

Proof.

Abelian monoid laws for the Parallel: Lemmas 8.2, 8.4, and 8.6.
Abelian monoid laws for the Sum: Lemmas 8.8, 8.10, and 8.12.
Scope extension laws: Lemmas 8.14, 8.16, 8.19, 8.21, and 8.23.
Unfolding law: Lemma 8.25.

□

9. Weak Bisimilarity

The two main weak equivalences in CCS are weak bisimilarity and weak congruence, where weak congruence, as the name suggests, is a congruence. The difference between the two is in simulating τ-actions - in weak bisimilarity, an agent can mimic a τ-action by doing nothing whereas for weak congruence, at least one τ-action must be taken to mimic the original one. Other than that, they behave the same in that an action is mimicked by a corresponding action preceded and followed by a τ-chain.

In this chapter we will define weak bisimilarity, prove that it is an equivalence relation and preserved by all operators except Sum. In the next chapter we will define weak congruence, and prove that it is a congruence.

9.1 τ-chains

We define τ-chains in the standard way as the reflexive transitive closure of τ-actions.

Definition 9.1 (τ-chain)**.**

$$P \Longrightarrow P' \stackrel{\text{def}}{=} (P, P') \in \{(P, P') : P \xrightarrow{\tau} P'\}^*$$

9.1.1 Core lemmas

Isabelle provides its own induction rules for sets generated by transitive closure. To simplify proofs on τ-chains a corresponding induction rule is derived.

Lemma 9.2.

$$\frac{P \Longrightarrow R \qquad Prop\ P \qquad \bigwedge P'\ P''.\ \dfrac{P \Longrightarrow P' \qquad P' \xrightarrow{\tau} P'' \qquad Prop\ P'}{Prop\ P''}}{Prop\ R}$$

Proof. Follows from the definition of $\Longrightarrow$ and Isabelle's induction rule for reflexive transitive closure. □

$$\frac{P \Longrightarrow P' \qquad P \neq P'}{P + Q \Longrightarrow P'}\ \textsc{Sum1} \qquad \frac{Q \Longrightarrow Q' \qquad Q \neq Q'}{P + Q \Longrightarrow Q'}\ \textsc{Sum2}$$

$$\frac{P \Longrightarrow P'}{P \mid Q \Longrightarrow P' \mid Q}\ \textsc{Par1} \qquad \frac{Q \Longrightarrow Q'}{P \mid Q \Longrightarrow P \mid Q'}\ \textsc{Par2}$$

$$\frac{P \Longrightarrow P'}{(\nu x)P \Longrightarrow (\nu x)P'}\ \textsc{Scope} \qquad \frac{P \mid !P \Longrightarrow P' \qquad P' \neq P \mid !P}{!P \Longrightarrow P'}\ \textsc{Repl}$$

Figure 9.1: Lifting semantics for τ-chains.

Lemma 9.3.

$$\textit{If } P \Longrightarrow P' \textit{ and } P' \xrightarrow{\tau} P'' \textit{ then } P \Longrightarrow P''.$$
$$\textit{If } P' \xrightarrow{\tau} P'' \textit{ and } P \Longrightarrow P' \textit{ then } P \Longrightarrow P''.$$

Proof. Follows from the definition of $\Longrightarrow$. □

Lemma 9.4. *If $P \Longrightarrow P'$ and $P' \Longrightarrow P''$ then $P \Longrightarrow P''$.*

Proof. By induction on $P' \Longrightarrow P''$. □

9.1.2 Lifting τ-chains

As described in Section 2.5 it is convenient to lift the operational semantics to the weak level. A stepping stone for this is to derive rules for how τ-chains behave for the different operators. The rules in Figure 9.1 have been derived from the operational semantics.

These rules are only applicable to the cases where τ-chains appear both in the assumptions and in the conclusions of the rules. It is therefore not possible to lift all operational rules. The ACTION-rule cannot be lifted as actions can be other than τ. The COMM-rule cannot be lifted as non τ-actions by necessity appear in the assumptions of the rule.

The SUM and REPL-rules are a bit more delicate. They cannot be lifted since a τ-chain can be empty. In the SUM1-rule the agent P actually has to perform at least one action for a τ-chain to be available. The SUM-rules in Figure 9.1 therefore only holds if the τ-chain in the assumption is nonempty. The same restriction is required for the REPL-rule.

$$\frac{}{\alpha.P \overset{\widehat{\alpha}}{\Longrightarrow} P}\ \textsc{Action} \qquad \frac{P \overset{\widehat{\alpha}}{\Longrightarrow} P'}{P \mid Q \overset{\widehat{\alpha}}{\Longrightarrow} P' \mid Q}\ \textsc{Par1} \qquad \frac{Q \overset{\widehat{\alpha}}{\Longrightarrow} Q'}{P \mid Q \overset{\widehat{\alpha}}{\Longrightarrow} P \mid Q'}\ \textsc{Par2}$$

$$\frac{P \overset{\widehat{\alpha}}{\Longrightarrow} P' \qquad Q \overset{\widehat{\bar{\alpha}}}{\Longrightarrow} Q' \qquad \alpha \neq \tau}{P \mid Q \overset{\widehat{\tau}}{\Longrightarrow} P' \mid Q'}\ \textsc{Comm} \qquad \frac{P \overset{\widehat{\alpha}}{\Longrightarrow} P' \qquad x \sharp \alpha}{(\nu x)P \overset{\widehat{\alpha}}{\Longrightarrow} (\nu x)P'}\ \textsc{Scope}$$

Figure 9.2: Lifted semantics for weak transitions.

9.2 Weak semantics

Two agents P and Q are weakly bisimilar if for every action P can do, Q can mimic that action but is allowed to do an arbitrary number of τ-actions before and afterwards, and conversely for Q and P. Moreover, a τ-action can be mimicked by doing nothing.

A weak τ-respecting transition is defined in the same manner as in Section 2.5 – an action is preceded and proceeded by a τ-chain.

Definition 9.5 (Weak τ-respecting transitions)**.**

$$P \overset{\alpha}{\Longrightarrow} P' \overset{\text{def}}{=} \exists P'' P'''.\ P \Longrightarrow P'' \wedge P'' \xrightarrow{\alpha} P''' \wedge P''' \Longrightarrow P'$$

A strong transition implies a weak τ-respecting one.

Lemma 9.6. *If* $P \xrightarrow{\alpha} P'$ *then* $P \overset{\alpha}{\Longrightarrow} P'$.

Proof. Follows from Definition 9.5 by instantiating the τ-chains with empty chains. □

Another type of weak transition, denoted $P \overset{\widehat{\alpha}}{\Longrightarrow} P'$, which may be empty if $\alpha = \tau$ defined as follows:

Definition 9.7 (Weak transition)**.**

$$P \overset{\widehat{\alpha}}{\Longrightarrow} P' \overset{\text{def}}{=} P \overset{\alpha}{\Longrightarrow} P' \vee \alpha = \tau \wedge P = P'$$

Strong transitions also imply this type of transition.

Lemma 9.8. *If* $P \xrightarrow{\alpha} P'$ *then* $P \overset{\widehat{\alpha}}{\Longrightarrow} P'$.

Proof. Follows from Definition 9.7 and Lemma 9.6. □

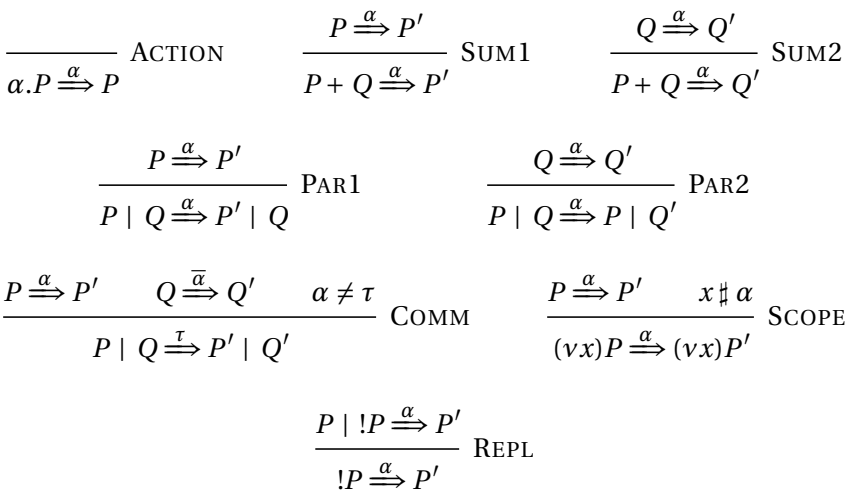


Figure 9.3: Lifted semantics for weak transitions.

9.2.1 *Lifted semantics*

In order to reuse the proof strategies developed for the proofs in Chapter 7, we lift the operational semantics from Figure 6.1, as described in Section 2.5, to include both types of weak transitions.

Lifting the semantic rules where the derivatives maintain the structure of the originating agent, like PAR1 or SCOPE, is reasonably straightforward. The transition in the assumption of the rule can be split into its components, and the corresponding rules from Figure 9.1 and the operational semantics can be used for the τ-chains and the transition respectively. This works for both types of weak transitions.

A bit more work is required when lifting the semantics of the rules where the structure is not maintained, such as SUM1 or REPL. Since the structure changes the first time a transition is made, the proof needs to handle the cases when the transition is done by the preceding τ-chain or, in the case the τ-chain is empty, by the action itself. Since these rules require at least one transition to be liftable, the transitions cannot have the form $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$, as it can be empty when $\alpha = \tau$.

The lifted semantics for the two types of weak transitions can can be found in figures 9.2 and 9.3.

9.3 Weak Bisimilarity

Simulation is defined in a similar way to its strong counterpart.

Definition 9.9 (Weak simulation)**.**

$$P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall a\, Q'.\ Q \xrightarrow{a} Q' \longrightarrow (\exists P'.\ P \stackrel{\widehat{a}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R})$$

In order to coinductively define weak bisimilarity we must prove that weak simulation is monotonic.

Lemma 9.10. *If $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} Q$.*

Proof. Follows directly for the definition of $\widehat{\rightsquigarrow}$. □

Definition 9.11 (Weak bisimilarity)**.** *Weak bisimilarity, denoted $\dot{\approx}$, is defined coinductively as the largest relation satisfying:*

$$\begin{array}{rll} P \dot{\approx} Q \Longrightarrow P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q & & \textsc{Simulation} \\ \wedge\ Q \dot{\approx} P & & \textsc{Symmetry} \end{array}$$

9.3.1 Primitive inference rules

From the definitions of weak simulation and bisimilarity the following introduction and elimination rules are derivable.

Lemma 9.12. *Introduction and elimination rules for weak simulation.*

$$\frac{\bigwedge \alpha\ Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}}{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\ \widehat{\rightsquigarrow}\text{-I}$$

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E}$$

Lemma 9.13. *Introduction and elimination rules for weak bisimilarity.*

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q \qquad Q \dot{\approx} P}{P \dot{\approx} Q}\ \dot{\approx}\text{-I} \qquad \frac{P \dot{\approx} Q}{P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q}\ \dot{\approx}\text{-E1} \qquad \frac{P \dot{\approx} Q}{Q \dot{\approx} P}\ \dot{\approx}\text{-E2}$$

The coinduction rule for weak congruence is derived similarly to strong bisimulation.

Lemma 9.14.

$$\frac{(P, Q) \in \mathscr{X} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} S}\ \textsc{Simulation} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{(S, R) \in \mathscr{X}}\ \textsc{Symmetry}}{P \mathrel{\dot{\approx}} Q}$$

Proof. Follows from the coinductive rule provided by Isabelle. □

9.3.2 Weak bisimilarity includes strong bisimilarity

All strongly bisimilar agents are also weakly bisimilar. To prove this, we first require an auxiliary lemma for simulation.

Lemma 9.15. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *then* $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$.

Proof. Follows from the definitions of $\hookrightarrow$ and $\widehat{\rightsquigarrow}$ and Lemma 9.8 which proves that all strong transitions imply a corresponding weak one. □

Lemma 9.16. *If* $P \sim Q$ *then* $P \mathrel{\dot{\approx}} Q$.

Proof. By coinduction using Lemma 9.14 with $\mathscr{X}$ set to $\sim$. The candidate relation $\mathscr{X}$ is symmetric as bisimulation is symmetric. Moreover, since $P \sim Q$ we have by $\sim$-E1 that $P \hookrightarrow_{\sim} Q$, and hence $P \mathrel{\widehat{\rightsquigarrow}}_{\sim} Q$ by Lemma 9.15, and finally that $P \mathrel{\widehat{\rightsquigarrow}}_{\sim \cup \dot{\approx}} Q$ since $\widehat{\rightsquigarrow}$ is monotonic. □

9.3.3 Structural congruence

Structurally congruent agents are weakly bisimilar since strongly bisimilar agents are also weakly bisimilar.

Theorem 9.1. *If* $P \equiv Q$ *then* $P \mathrel{\dot{\approx}} Q$.

Proof. Follows immediately from Theorem 8.1 and Lemma 9.16. □

9.4 Weak bisimulation is an equivalence relation

The requirements of the candidate relations in order for weak simulation to be proved reflexive and transitive are the same as for regular simulation. However, the proof for transitivity is more involved. We first establish that weak simulation is reflexive.

Lemma 9.17. *If* $Id \subseteq \mathscr{R}$ *then* $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} P$.

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$. □

The proof that weak bisimilarity is transitive is more involved than its strong counterpart. Consider the case where P is weakly bisimilar to Q, and Q is weakly bisimilar to R. When P does an action, Q does a corresponding weak action, and it is this action that R must mimic, not a strong one.

The first lemma which needs to be proven is how to mimic a τ-chain.

Lemma 9.18.

$$\frac{Q \Longrightarrow Q' \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S}}{\exists P'.\ P \Longrightarrow P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. By induction on $Q \Longrightarrow Q'$. In the base case where $Q = Q'$, P' is set to P finishing the case since $P \Longrightarrow P$ and $(P, Q) \in \mathscr{R}$. The inductive step provides a τ-chain $Q \Longrightarrow Q''$ and a transition $Q'' \xrightarrow{\tau} Q'$. The induction hypothesis gives a P'' such that $P \Longrightarrow P''$ and $(P'', Q'') \in \mathscr{R}$. From $(P'', Q'') \in \mathscr{R}$ it follows from the assumptions that $P'' \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q''$ and hence by $Q'' \xrightarrow{\tau} Q'$ and Lemma 9.12 that there exists an agent P' s.t. $P'' \stackrel{\tau}{\Longrightarrow} P'$ and $(P', Q') \in \mathscr{R}$. The proof is concluded by appending $P \Longrightarrow P''$ to $P'' \stackrel{\tau}{\Longrightarrow} P'$ and instantiating the existential quantifier to P'. □

This lemma states that if P weakly simulates Q and Q can do a τ-chain then P can do a corresponding τ-chain and the derivatives will still be in the relation. The assumption that states that all members of $\mathscr{R}$ are also simulations allow simulations to be derived after parsing the τ-chains.

A similar lemma needs to be proven for weak transitions.

Lemma 9.19.

$$\frac{(P, Q) \in \mathscr{R} \qquad Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q' \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S}}{\exists P'.\ P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. By case analysis on $Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$.
Staying case ($Q = Q'$ and $\alpha = \tau$): Instantiate the existential quantifier to P, and solve the goal with the assumption $(P, Q) \in \mathscr{R}$.
Moving case ($Q \stackrel{\alpha}{\Longrightarrow} Q'$):
From the definition of $Q \stackrel{\alpha}{\Longrightarrow} Q'$, we obtain a Q'' and a Q''' s.t. $Q \Longrightarrow Q'''$, $Q''' \stackrel{\alpha}{\longrightarrow} Q''$ and $Q'' \Longrightarrow Q'$. From $Q \Longrightarrow Q'''$ and the assumptions, we use Lemma 9.18 to obtain a P''' s.t. $P \Longrightarrow P'''$ and $(P''', Q''') \in \mathscr{R}$. From (P''', Q''') $\in \mathscr{R}$ and the assumptions we get that $P''' \rightsquigarrow_{\mathscr{R}} Q'''$, and with "$Q''' \stackrel{\alpha}{\longrightarrow} Q''$ we obtain a P'' s.t. $P''' \stackrel{\widehat{\alpha}}{\Longrightarrow} P''$ and $(P'', Q'') \in \mathscr{R}$, using Lemma 9.12. From $(P'', Q'') \in \mathscr{R}$, $Q'' \Longrightarrow Q'$ and the assumptions we obtain a P' s.t. $P'' \Longrightarrow P'$ and $(P', Q') \in \mathscr{R}$, again using Lemma 9.18. Finally, we append $P \Longrightarrow P'''$, $P''' \stackrel{\widehat{\alpha}}{\Longrightarrow} P''$ and $P'' \Longrightarrow P'$ to $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$ and solve the goal by instantiating the existential quantifier to P'. □

It is now possible to prove what is required of the candidate relation for simulation to be transitive.

Lemma 9.20.

$$\frac{(P, Q) \in \mathscr{R} \qquad Q \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} R \qquad \mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}'' \qquad \bigwedge S\ T.\ \dfrac{(S, T) \in \mathscr{R}}{S \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} T}}{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}''} R}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$ and Lemma 9.19 to simulate the weak transitions. □

Proving that bisimulation is an equivalence relation is now straightforward.

Lemma 9.21. *Weak bisimualtion is an equivalence relation.*

Proof. **Reflexivity:** By coinduction with $\mathscr{X}$ set to the identity relation, and Lemma 9.17.

Symmetry: Follows immediately from $\dot{\approx}$-E2.

Transitivity: Follows by coinduction and setting $\mathscr{X}$ to $\dot{\approx} \circ \dot{\approx}$, and Lemma 9.20.

□

9.5 Preservation properties

Weak bisimilarity is not preserved by Sum. The following counterexample is taken from [55].

Consider the following three agents:

$$P \stackrel{\text{def}}{=} a.\mathbf{0} \qquad Q \stackrel{\text{def}}{=} \tau.a.\mathbf{0} \qquad R \stackrel{\text{def}}{=} b.\mathbf{0}$$

Agents P and Q are weakly bisimilar. If P does an a-action, Q can mimic by doing a τ and then an a-action. Similarly, if Q does a τ-action, P can mimic by doing nothing. However, it is not the case that $P + R$ is weakly bisimilar to $Q + R$. If $Q + R$ does a τ-action ending up in $a.\mathbf{0}$, the only choice that $P + R$ has to mimic that action is to do nothing, but the derivatives $a.\mathbf{0}$ and $P + R$ are not bisimilar.

The fact that the semantics is lifted allows us to reason about weak transitions in a similar way that we reason about strong ones. As a result, the lemmas in this section follow the ones in Section 7.1.1 almost completely, with the only difference being which lemmas are being used.

9.5.1 Prefix

Lemma 9.22. *If* $(P, Q) \in \mathscr{R}$ *then* $\alpha.P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} \alpha.Q$.

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$ and the fact that $a.P$ and $a.Q$ can each only do an a-action (Figure 6.4 ACTION) and $(P, Q) \in \mathscr{R}$. □

Lemma 9.23. *If* $P \mathrel{\dot{\approx}} Q$ *then* $\alpha.P \mathrel{\dot{\approx}} \alpha.Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(a.P,\ a.Q) \mid P \sim Q\}$ and Lemma 9.22. □

9.5.2 Restriction

Lemma 9.24.

$$\frac{P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q \qquad \bigwedge R\,S\,y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}'}}{(\nu x)P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}'}} (\nu x)Q}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$, the SCOPE inversion rule in Figure 6.4, and the SCOPE-rule from the lifted semantics. □

Lemma 9.25. *If* $P \mathrel{\dot{\approx}} Q$ *then* $(\nu x)P \mathrel{\dot{\approx}} (\nu x)Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)P,\ (\nu x)Q) : P \stackrel{\cdot}{\approx} Q\}$ and Lemma 9.24. □

9.5.3 Parallel

As with strong bisimilarity, we prove a general simulation lemma, which will be used in the proof that weak bisimilarity is preserved by replication.

Lemma 9.26.

$$\frac{\begin{array}{l} P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \\ R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} T \qquad (R, T) \in \mathscr{R}' \\ \bigwedge P'\, Q'\, R'\, T'.\ \dfrac{(P', Q') \in \mathscr{R} \qquad (R', T') \in \mathscr{R}'}{(P' \mid R', Q' \mid T') \in \mathscr{R}''} \end{array}}{P \mid R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}''} Q \mid T}$$

Proof. The Isabelle proof of this lemma can be found in Figure 9.4. □

The following lemma is then easily derivable.

Lemma 9.27.

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge S\, T\, U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U, T \mid U) \in \mathscr{R}'}}{P \mid R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} Q \mid R}$$

Proof. Follows immediately from Lemma 9.26 and setting $\mathscr{R}''$ to $\mathscr{R}'$ and $\mathscr{R}'$ to the identity relation. □

9.6 Bisimulation up-to techniques

As the complexity of the proofs for bisimilarity increase, so does the complexity of their candidate relations. The relations we have seen so far have been relatively simple, generally of the form that they contain the pair that is to be proven bisimilar, and a few standard requisites such as reflexivity, or that the relation is compositional.

In the coinduction rule for weak bisimilarity, Lemma 9.14, the derivatives of any agents in the candidate relation must either be in the candidate relation, or bisimilar. Bisimulation up-to techniques allow for a more general treatment of the derivative agents – they must be equivalent to agents either in the candidate relation or agents which are bisimilar. The equivalence, here denoted $\doteq$, differs for each proof, but in the general case any

derivatives of the members of a candidate relation $\mathscr{X}$ must be in the relation $\dot{=} \circ (\mathscr{X} \cup \dot{\approx}) \circ \dot{=}$.

It is generally desirable for the equivalence relation to be as large as possible – the more general the relation, the less specific the requirements of the candidate relation. However, bisimulation up-to techniques are not applicable for all equivalence classes. In the original printing of Milner's book Communication and Concurrency [55], Milner proposed

$$\mathscr{Y} = \dot{\approx} \circ (\mathscr{X} \cup \dot{\approx}) \circ \dot{\approx},$$

i.e. bisimulation up to weak bisimilarity. However, this technique equates agents which should not be bisimilar. The following counterexample was discovered by Sjödin and Jonsson. Consider the agents $\tau.P$ and $\mathbf{0}$, where P is any agent *not* bisimilar to $\mathbf{0}$. Clearly these agents are not weakly bisimilar, but they are bisimilar up to weak bisimilarity. Set the candidate relation $\mathscr{X}$ to $\{(\tau.P, \mathbf{0}), (\mathbf{0}, \tau.P)\}$. The only possible transition is $\tau.P \xrightarrow{\tau} P$, which $\mathbf{0}$ can mimic by doing nothing, and the derivatives P and $\mathbf{0}$ are in $\mathscr{Y}$; the agents P and $\tau.P$ are weakly bisimilar, hence P can be rewritten to $\tau.P$ using the up to techniques, and $\tau.P$ and $\mathbf{0}$ are in $\mathscr{X}$.

Bisimulation up to techniques which are available are bisimulation up to structural congruence, and strong bisimilarity. Moreover, a variant of bisimulation up to weak bisimilarity is attainable.

Lemma 9.28.

$$\frac{(P, Q) \in \mathscr{Y} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{Y}}{R \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx} \circ (\mathscr{Y} \cup \dot{\approx}) \circ \sim} S} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{Y}}{(S, R) \in \mathscr{Y}}}{P \dot{\approx} Q}$$

Proof. By coinduction with $\mathscr{X}$ set to $\dot{\approx} \circ (\mathscr{Y} \cup \dot{\approx}) \circ \dot{\approx}$. □

This coinductive lemma requires the derivatives to be in the relation $\dot{\approx} \circ (\mathscr{Y} \cup \dot{\approx}) \circ \sim$, and therefore the counterexample above does not hold – even though $\tau.P$ is weakly bisimilar to P, they are not strongly bisimilar, and the rewriting technique used in the counter-example does not work.

With these techniques in place, we can prove that weak bisimilarity is preserved by replication.

Lemma 9.29. *If $P \dot{\approx} Q$ then $P \mid R \dot{\approx} Q \mid R$.*

Proof. The Isabelle proof for this lemma can be found in Figure 9.5. □

The remaining operator which bisimilarity is preserved by is Parallel. In order to prove this we need to use bisimulation up-to techniques.

lemma weakSimParPres:
fixes P :: ccs **and** Q :: ccs **and** R :: ccs **and** T :: ccs
and $\mathscr{R}$ :: (ccs × ccs) set **and** $\mathscr{R}'$ :: (ccs × ccs) set **and** $\mathscr{R}''$ :: (ccs × ccs) set

assumes P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q **and** (P, Q) ∈ $\mathscr{R}$
and R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T **and** (R, T) ∈ $\mathscr{R}'$
and C1: $\bigwedge$P′ Q′ R′ T′. ⟦(P′, Q′) ∈ $\mathscr{R}$; (R′, T′) ∈ $\mathscr{R}'$⟧ ⟹
(P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$

shows P | R $\widehat{\rightsquigarrow}_{\mathscr{R}''}$ Q | T
proof(induct rule: weakSimI) — *Apply introduction rule* $\widehat{\rightsquigarrow}$-I
case(Sim α U)
from ⟨Q | T $\xrightarrow{\alpha}$ U⟩
show ?case
proof(induct rule: parCases) — *Apply* PAR *inversion rule from Figure 6.4*

PAR1 *case*
Given that $Q \xrightarrow{\alpha} Q'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\alpha}}{\Longrightarrow} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(cPar1 Q′)
from ⟨P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q⟩ ⟨Q $\xrightarrow{\alpha}$ Q′⟩ **obtain** P′ **where** P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(rule weakSimE)
from ⟨P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′⟩ **have** P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′ | R **by**(rule weakPar1)
moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R, T) ∈ $\mathscr{R}'$⟩ **have** (P′ | R, Q′ | T) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S ∧ (S, Q′ | T) ∈ $\mathscr{R}''$ **by** blast
next

PAR2 *case*
Given that $T \xrightarrow{\alpha} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\alpha}}{\Longrightarrow} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T′)
from ⟨R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{\alpha}$ T′⟩ **obtain** R′ **where** R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(rule weakSimE)
from ⟨R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′⟩ **have** P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P | R′ **by**(rule weakPar2)
moreover from ⟨(P, Q) ∈ $\mathscr{R}$⟩ ⟨(R′, T′) ∈ $\mathscr{R}'$⟩
have (P | R′, Q | T′) ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S ∧ (S, Q | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM *case*

Given that $Q \xrightarrow{a} Q'$ *and* $T \xrightarrow{\overline{a}} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm Q′ T′ α)

from ‹P $\widehat{\leadsto}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\alpha}$ Q′› **obtain** P′ **where** P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(rule weakSimE)

from ‹R $\widehat{\leadsto}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{\alpha}}$ T′› **obtain** R′ **where** R $\stackrel{\widehat{\overline{\alpha}}}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(rule weakSimE)

from ‹P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′› ‹R $\stackrel{\widehat{\overline{\alpha}}}{\Longrightarrow}$ R′› ‹$\alpha \neq \tau$› **have** P | R $\stackrel{\tau}{\Longrightarrow}$ P′ | R′
by(auto intro: weakCongSync simp add: weakTrans-def)

hence P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ P′ | R′ **by**(simp add: weakTrans-def)

moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$›

have (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$ **by**(rule C1)

ultimately show ∃S. P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ S ∧ (S, Q′ | T′) ∈ $\mathscr{R}''$ **by** blast

qed

qed

Figure 9.4: The Isabelle proof that simulation is preserved by the Parallel. It is very similar to the corresponding proof for strong simulation, found in Figure 7.2, but the lifted semantic rules are used instead of their regular semantic counterparts.

```
lemma weakBisimParPres:
 fixes P :: ccs and Q :: ccs and R :: ccs
 assumes P ≈ Q
 shows P | R ≈ Q | R
proof −
 let ?X = {(P | R, Q | R) | P Q R. P ≈ Q}
 from assms have (P | R, Q | R) ∈ ?X by auto
 thus P | R ≈ Q | R
 proof(coinduct rule: weakBisimCoinduct)
  case(cSim S T)
  {
   fix P Q R
   assume P ≈ Q
   moreover hence P ⇝̂ (≈ ∪ .) Q
    by(rule weakBisimulationE)
   moreover have ⋀P Q R. P ≈ Q ⟹ (P | R, Q | R) ∈ ?X by auto
   ultimately have P | R ⇝̂ ?X Q | R
    by(rule-tac weakSimPres.parPres)
   hence P | R ⇝̂ ?X ∪ (≈ ∪ .) Q | R
    by(rule-tac weakSimMonotonic) auto
  }
  thus S ⇝̂ ?X ∪ (≈ ∪ .) T using ‹(S, T) ∈ ?X› by auto
 next
  case(cSym PR QR)
  from ‹(PR, QR) ∈ ?X› show (QR, PR) ∈ ?X
   by(blast dest: weakBisimulation.symmetric)
 qed
qed
```

Figure 9.5: The proof that weak bisimulation is preserved by Parallel.

9.6.1 Replication

The preservation properties for replication are a bit more complicated than for its strong counterparts. The reason for this is that the simulating agent has the possibility of doing nothing when mimicking a τ-action. As the lifted semantics rule in Figure 9.3 dictates, the REPL rule may only be used if an action is actually performed. This puts special constraints on the candidate relations for bisimulation, which is demonstrated by the simulation lemmas.

Lemma 9.30.

$$(P, Q) \in \mathscr{R} \qquad \mathit{bangRel}\,\mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge R\,S.\ \frac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} S}$$

$$\bigwedge R\,S\,T\,U.\ \frac{(R, S) \in \mathscr{R} \qquad (T, U) \in \mathscr{R}'}{(R \mid T, S \mid U) \in \mathscr{R}'} \qquad \bigwedge R\,S.\ \frac{(R \mid\ !R, S) \in \mathscr{R}'}{(!R, S) \in \mathscr{R}'}$$

$$!P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}'}} !Q$$

Proof. Similar to the corresponding proof for strong simulation, Lemma 7.20. However, in the replication case the REPL rule from Figure 9.3 can only be used when the mimicking agent mimics at least one τ-action; if it does nothing, the last assumption from the lemma is used to ensure that the derivatives remain in $\mathscr{R}'$. □

Lemma 9.31. *If $P \mathrel{\dot{\approx}} Q$ then $!P \mathrel{\dot{\approx}} !Q$.*

Proof. Follows the same pattern as the corresponding lemma for strong bisimilarity, Lemma 7.21, but by coinduction up to weak bisimilarity with $\mathscr{Y}$ set to $\mathit{bangRel}\,\dot{\approx}$. The simulation case is discharged by Lemma 9.30, where its final requisite is proven by the fact that $!P \mathrel{\dot{\approx}} P \mid !P$, by Theorem 9.1. □

With this lemma in place, we can prove that the main preservation theorem for weak bisimilarity.

Theorem 9.2. *Weak bisimulation is preserved by all operators except the Sum.*

Proof. Follows from lemmas 9.23, 9.25, 9.29, and 9.31. □

10. Weak congruence

As shown in the previous chapter, weak bisimilarity is not a congruence since it is not preserved by Sum. Let us recapitulate the counter-example:

$$P \stackrel{\text{def}}{=} a.\mathbf{0} \qquad Q \stackrel{\text{def}}{=} \tau.a.\mathbf{0} \qquad R \stackrel{\text{def}}{=} b.\mathbf{0}$$

The problem is that even though P and Q are bisimilar, if $Q + R$ does a τ-action, the only option $P + R$ has is to do nothing, and the derivatives are not bisimilar. An alternative bisimulation is one where a simulating agent is required to do *at least* the action the simulated agent did, even if this was a τ-action. This change is enough to obtain a coungruence. Moreover, only the initial step is required to mimic a τ-action, and the derivatives need only be weakly bisimilar. In this scenario, P and Q are not bisimilar as P cannot mimic the τ-action done by Q.

This bisimulation relation is called *weak congruence* and will be formally defined in the next sections.

10.1 Definitions

Simulation for weak congruence, which we will call τ-simulation, is defined in the same manner as for weak bisimilarity (Definition 9.9), with the exception that a an action must be mimicked by at least that action, and any number of internal actions. As mentioned previously, the derivatives of weak congruence need only be weakly bisimilar and not weakly congruent. This distinction is not visible in the definition of simulation, but will become apparent when weak congruence is defined. Nevertheless, some changes in the simulation proofs are needed to handle this discrepancy.

Definition 10.1 (τ-simulation)**.** *An agent P which τ-simulates an agent Q preserving $\mathscr{R}$ is written $P \rightsquigarrow_{\mathscr{R}} Q$.*

$$P \rightsquigarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall\, a\, Q'.\ Q \xrightarrow{a} Q' \longrightarrow (\exists\, P'.\ P \stackrel{a}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R})$$

All bisimulations up until now have been defined using coinductive definitions. This has made sense since all derivatives of the simulated agents have been required to remain in the same relation as they start in.

For weak congruence this is not the case – the derivatives are required to be weakly bisimilar, and not weakly congruent. As a result, the definition for weak congruence is not coinductive – it is just a conjunction of two τ-simulations.

Definition 10.2 (Weak congruence)**.**

$$P \approx Q \stackrel{\text{def}}{=} P \rightsquigarrow_{\dot{\approx}} Q \wedge Q \rightsquigarrow_{\dot{\approx}} P$$

10.1.1 Primitive inference rules

Introduction and elimination rules for τ-simulation and weak congruence are derived in the standard way.

Lemma 10.3. *Introduction and elimination rules for τ-simulation*

$$\frac{\bigwedge \alpha\ Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \stackrel{\alpha}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}}{P \rightsquigarrow_{\mathscr{R}} Q}\ \rightsquigarrow\text{-I}$$

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \stackrel{\alpha}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}\ \rightsquigarrow\text{-E}$$

Proof. Follows immediately from Definition 10.1. □

Lemma 10.4. *Introduction and elimination rules for weak congruence.*

$$\frac{P \rightsquigarrow_{\dot{\approx}} Q \qquad Q \rightsquigarrow_{\dot{\approx}} P}{P \approx Q}\ \approx\text{-I} \qquad\qquad \frac{P \approx Q}{P \rightsquigarrow_{\dot{\approx}} Q}\ \approx\text{-E}$$

Proof. Follows immediately from Definition 10.2. □

The coinductive rules for bisimilarity are defined such that it is only required to prove the simulations one way, as long as the candidate relation is symmetric. By only using the definition of weak congruence, we would have to prove the τ-simulations both ways for all of our proofs. The following lemma lets us adopt the proof strategy that we use for bisimilarites for weak congruence as well.

Lemma 10.5. *Symmetric introduction rule for weak congruence.*

$$\frac{Prop\ P\ Q \qquad \bigwedge P\ Q.\ \dfrac{Prop\ P\ Q}{Prop\ Q\ P} \qquad \bigwedge P\ Q.\ \dfrac{Prop\ P\ Q}{F\ P \rightsquigarrow_{\dot{\approx}} F\ Q}}{F\ P \approx F\ Q}$$

Proof. Follows immediately from the definition of $\approx$ □

This rule has one predicate *Prop* and one function *F*. The predicate *Prop* takes two agents as arguments and represents an assumption required to prove the weak congruence. It must be symmetric, since weak congruence is symmetric, and any two agents *P* and *Q* that validate *Prop* must also form a τ-simulation. Moreover, the function *F* maps one agent to another, allowing the proof goal to depend on the arguments of *Prop*. A lemma which uses this introduction rule is one which proves that all strongly bisimilar agents are also weakly congruent, Lemma 10.7 below.

10.1.2 Weak congruence includes strong bisimilarity

In the same manner as was done for weak bisimilarity, we can translate strong simulations to τ-simulations.

Lemma 10.6. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *then* $P \rightsquigarrow_{\mathscr{R}} Q$.

Proof. Follows from the definitions of $\hookrightarrow$ and $\rightsquigarrow$ and Lemma 9.6 which converts a strong transition to a weak one. □

Lemma 10.7. *If* $P \sim Q$ *then* $P \approx Q$.

Proof. We use the introduction rule, Lemma 10.5, and instantiate *Prop* to $\lambda P\ Q.\ P \sim Q$ and *F* to the identity function. Since strong bisimilarity is symmetric (Lemma 7.9), the symmetry case is discharged leaving the simulation case. For any *P* and *Q* s.t. $P \sim Q$ we have that $P \hookrightarrow_{\sim} Q$, by the definition of $\sim$. Since $\hookrightarrow$ is monotonic (Lemma 7.2) and $\sim \subseteq \dot{\approx}$ (Lemma 9.16), we have that $P \hookrightarrow_{\dot{\approx}} Q$, and hence $P \rightsquigarrow_{\dot{\approx}} Q$, by Lemma 10.6. □

10.1.3 Weak bisimularity includes weak congruence

Lemma 10.8. *If* $P \approx Q$ *then* $P \dot{\approx} Q$.

Proof. By coinduction, using Lemma 9.14, and setting the candidate relation $\mathscr{X}$ to $\{(P, Q) : P \approx Q\}$. The set $\mathscr{X}$ is trivially symmetric. For every pair $(P, Q) \in \mathscr{X}$ we get that $P \approx Q$ and thus $P \rightsquigarrow_{\dot{\approx}} Q$, from the definition of $\approx$. By monotonicity of $\rightsquigarrow$, the proof is concluded by proving that $P \rightsquigarrow_{\mathscr{X} \cup \dot{\approx}} Q$. □

10.1.4 Structural congruence

The main structural congruence theorem for weak bisimilarity and weak congruence can now be proven.

Theorem 10.1.

$$\textit{If } P \equiv Q \textit{ then } P \dot{\approx} Q.$$
$$\textit{If } P \equiv Q \textit{ then } P \approx Q.$$

Proof. Follows immediately from Theorem 8.1 and lemmas 9.16 and 10.7. □

10.2 Weak congruence is an equivalence relation

The reflexivity proof for τ-simulations has the same structure as its counterpart for weak bisimulation.

Lemma 10.9. *If* $Id \subseteq \mathscr{R}$ *then* $P \rightsquigarrow_{\mathscr{R}} P$.

Proof. Follows from Lemma 10.3. Any transitions $P \xrightarrow{\alpha} P'$ can, by Lemma 9.6, be mimicked by $P \stackrel{\alpha}{\Longrightarrow} P'$, and $(P', P') \in \mathscr{R}$. □

The proof strategy for proving transitivity of τ-simulations differs slightly from its counterpart for weak simulations (Lemma 9.20). The reason is the different requirements for the derivatives in the two bisimulation relations. For weak bisimilarity, the derivatives need to be weakly bisimilar.

When proving that weak congruence is transitive we reuse lemmas from the corresponding proof from weak bisimilarity. This is possible as the derivatives of weakly congruent agents are weakly bisimilar.

Lemma 10.10.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad Q \stackrel{\alpha}{\Longrightarrow} Q' \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S}}{\exists P'.\ P \stackrel{\alpha}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. By unfolding $Q \stackrel{\alpha}{\Longrightarrow} Q'$ we obtain a Q'' and a Q''' s.t. $Q \Longrightarrow Q'''$, $Q''' \xrightarrow{\alpha} Q''$, and $Q'' \Longrightarrow Q'$. The proof is a proof by cases on $Q \Longrightarrow Q'''$.

case $(Q = Q''')$: The transition $Q''' \xrightarrow{\alpha} Q''$ is simulated using Lemma 10.3, and the trailing τ-chain by Lemma 9.18.

case $(Q \neq Q''')$: Since $Q \neq Q'''$ there exists a Q'''' s.t. $Q \xrightarrow{\tau} Q''''$ and $"Q'''' \Longrightarrow Q'''$. The transition $Q \xrightarrow{\tau} Q''''$ can then be simulated using Lemma 10.3, and the remaining τ-chains and transitions by Lemmas 9.12 and 9.18. The

important observation is that even if $\alpha = \tau$, and the simulating agents for weak bisimilarity has the option of doing nothing, at least one τ-action has been forced by the initial simulation. □

The proof that simulation for weak congruence is transitive can now be completed.

Lemma 10.11. *The simulation for weak congruence is transitive.*

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad Q \rightsquigarrow_{\mathscr{R}'} R \qquad \mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}'' \qquad \bigwedge S\ T.\ \dfrac{(S, T) \in \mathscr{R}}{S \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} T}}{P \rightsquigarrow_{\mathscr{R}''} R}$$

Proof. Follows from the definition of $\rightsquigarrow$ and Lemma 10.10 to simulate the weak actions. □

We can now prove that weak congruence is an equivalence relation.

Lemma 10.12.

Proof. **Reflexivity:** Follows from the definition of $\approx$, reflexivity of weak bisimilarity (Lemma 9.21), and Lemma 10.9.

Symmetry: Follows immediately from the definition of $\approx$.

Transitivity: The symmetric introduction rule (Lemma 10.5) is instantiated with *Prop* set to $\lambda P\, R.\ \exists Q.\ P \approx Q \wedge Q \approx R$ and F set to the identity function.
Symetry case: Follows immediately from the fact that weak congruence is symmetric.
Simulation case: From *Prop P R* we obtain a Q s.t. $P \approx Q$ and $Q \approx R$, and hence $P \rightsquigarrow_{\dot{\approx}} Q$ and $Q \rightsquigarrow_{\dot{\approx}} R$. Moreover, since weak bisimulation is transitive (Lemma 9.21), we have that $\dot{\approx} \circ \dot{\approx} \subseteq \dot{\approx}$, and hence by Lemma 9.20 that $P \rightsquigarrow_{\dot{\approx}} R$.

□

10.3 Preservation properties

The next step is to prove that weak congruence actually is a congruence. All congruence lemmas have a typical form – if two agents are weakly congruent, then two other agents containing the original ones must be weakly congruent as well. The following adaptation to the symmetric introduction rule (Lemma 10.5) will be useful.

Lemma 10.13. *A version of Lemma 10.5 where Prop is weak congruence.*

$$\frac{P \approx Q \qquad \bigwedge P\,Q.\ \dfrac{P \approx Q}{F\,P \rightsquigarrow_{\dot{\approx}} F\,Q}}{F\,P \approx F\,Q}$$

Proof. Follows immediately from Lemma 10.5 with *Prop* set to $\approx$, and the fact that weak congruence is symmetric. □

10.3.1 Prefix

Lemma 10.14. *If* $(P, Q) \in \mathscr{R}$ *then* $\alpha.P \rightsquigarrow_{\mathscr{R}} \alpha.Q$.

Proof. Follows from the definition of $\rightsquigarrow$ and the fact that $a.P$ and $a.Q$ can each only do an a-action (Figure 6.4 ACTION) and $(P, Q) \in \mathscr{R}$. □

Lemma 10.15. *If* $P \approx Q$ *then* $\alpha.P \approx \alpha.Q$.

Proof. Follows from Lemma 10.13, with F set to *Action* α. Since $P \approx Q$, we know that $P \rightsquigarrow_{\dot{\approx}} Q$, by $\approx$-I, and that $P \mathrel{\dot{\approx}} Q$, by Lemma 10.8. The simulation can then be proven using Lemma 10.14. □

10.3.2 Sum

Lemma 10.16.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad Id \subseteq \mathscr{R}'}{P + R \rightsquigarrow_{\mathscr{R}'} Q + R}$$

Proof. Follows from the definition of $\rightsquigarrow$, the SUM inversion rule in Figure 6.4, and the SUM1 and SUM2 rules from the lifted semantics. In the case where R does an action, Lemma 9.6 is used to convert the strong action to a weak one. □

Lemma 10.17. *If* $P \approx Q$ *then* $P + R \approx Q + R$.

Proof. Follows from Lemma 10.13, with F set to $\lambda P.\ P + R$. Since $P \approx Q$, we know that $P \rightsquigarrow_{\dot{\approx}} Q$, by $\approx$-I. The simulation can then be proven using Lemma 10.16, and the fact that weak bisimulation is reflexive (Lemma 9.21). □

```
lemma parPres:
  fixes P :: ccs and Q :: ccs and R :: ccs
  assumes P ≅ Q
  shows P | R ≅ Q | R
using assms
proof(induct rule: weakCongISym2)
  case(cSim P Q)
  from ‹P ≅ Q› have P ⇝≳ Q
    by(rule weakCongruenceE) — Lemma ≈-E1
  moreover from ‹P ≅ Q› have P ≈ Q
    by(rule weakCongruenceWeakBisimulation) — Lemma 10.8
  ultimately show P | R ⇝≳ Q | R
    using weakBisimulationPres.parPres — Lemma 9.29
    by(rule weakCongSimPres.parPres) — Lemma 10.18
qed
```

Figure 10.1: The proof that weak congruence is preserved by Parallel.

10.3.3 Parallel

Lemma 10.18.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge S\ T\ U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U, T \mid U) \in \mathscr{R}'}}{P \mid R \rightsquigarrow_{\mathscr{R}'} Q \mid R}$$

Proof. The Isabelle proof of this lemma is similar to the one found in Figure 7.1. □

Lemma 10.19. *If $P \approx Q$ then $P \mid R \approx Q \mid R$.*

Proof. The Isabelle proof of this lemma can be found in Figure 10.1. □

10.3.4 Restriction

Lemma 10.20.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \bigwedge R\ S\ y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}'}}{(\nu x)P \rightsquigarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from the definition of $\rightsquigarrow$, the SCOPE inversion rule in Figure 6.4, and the SCOPE from the lifted semantics. □

Lemma 10.21. *If $P \approx Q$ then $(\nu x)P \approx (\nu x)Q$.*

Proof. Follows from Lemma 10.13, with F set to $\nu\, x$. Since $P \approx Q$, we know that $P \rightsquigarrow_{\dot{\approx}} Q$, by $\approx$-I. The simulation can then be proven using Lemma 10.20, and the fact that weak bisimilarity is preserved by restriction (Lemma 9.25). □

10.3.5 Replication

The proof that weak congruence is preserved by replication is simpler than its counterpart for weak bisimilarity.

Lemma 10.22.

$$\frac{(P, Q) \in \mathscr{R} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \rightsquigarrow_{\mathscr{R}'} S} \qquad \mathscr{R} \subseteq \mathscr{R}'}{!P \rightsquigarrow_{bangRel\ \mathscr{R}'} !Q}$$

Proof. Follows the same structure as its counterpart for strong bisimilarity, Lemma 7.20, but with the lifted semantics from Figure 9.3. □

Before proving that weak congruence is preserved by replication, we need the following auxiliary lemma.

Lemma 10.23. $bangRel\ \dot{\approx} \subseteq \dot{\approx}$

Proof. By induction on $bangRel\ \dot{\approx}$.

Parallel case: We have that $R \dot{\approx} T$, and from the induction hypothesis that $P \dot{\approx} Q$, and we must prove that $R \mid P \dot{\approx} T \mid Q$, which follows from the preservation properties of weak bisimilarity, and the structural congruence laws.

Replication case: From $P \dot{\approx} Q$ we have to prove that $!P \dot{\approx} !Q$ which follows directly from Lemma 9.31.

□

We also need to prove that weak congruence simulation is monotonic.

Lemma 10.24. *If $P \rightsquigarrow_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \rightsquigarrow_{\mathscr{R}'} Q$.*

Proof. Follows immediately from the definition of $\rightsquigarrow$. □

Lemma 10.25. *If $P \approx Q$ then $!P \approx !Q$.*

Proof. The proof uses the symmetric introduction rule $\approx$-I2. From $P \approx Q$, the definition of $\rightsquigarrow$, and Lemma 10.8 we have that $!P \rightsquigarrow_{bangRel\ \dot{\approx}} !Q$ by Lemma 10.22. We can then prove that $!P \rightsquigarrow_{\dot{\approx}} !Q$ with Lemmas 10.24 and 10.23. □

10.4 Weak congruence is a congruence

It is now possible to prove the congruence theorem.

Theorem 10.2. *Weak congruence is a congruence*

Proof. That weak congruence is an equivalence relation follows from Lemma 10.12, and that it is preserved by all operators follows from lemmas 10.15, 10.17, 10.19, and 10.21. □

11. Conclusions

This part of the thesis demonstrates how to formalise a substantial part of the meta theory of CCS in Isabelle. We encode the semantics, using nominal logic to reason about the binding constructs (Chapter 6), we define strong bisimilarity and prove that it is a congruence (Chapter 7), and we define the structural congruence laws and prove that all bisimilar agents are also structurally congruent (Chapter 8). Moreover, we define weak bisimilarity and prove that it is preserved by all operators except Sum (Chapter 9), and we define weak congruence and prove that it is a congruence (Chapter 10). We also prove that weak congruence includes weak bisimilarity which includes strong bisimilarity.

The time it took to write the original formalisation was approximately six months, with no prior Isabelle knowledge. At that time Nominal Isabelle did not exist, and no proofs required that agents are treated up to alpha-equivalence. For this thesis the formalisation was adapted to include support for replication, and use nominal logic. This took about one day.

A description of the different parts of the formalisation and their size can be found in Figure 11.1. The part for agents include the nominal datatype for CCS-agents, and supporting lemmas for actions and coactions. The part for the semantics include the induction and inversion rules required for the rest of the formalisation. The part for strong bisimilarity includes the definitions for strong simulation and bisimilarity, as well as the congruence results. The part for structural congruence includes the definitions of structural congruence, and the proof that all strongly bisimilar agents are also structurally congruent. The formalisation for weak congruence and weak bisimilarity are very interdependent.

11.1 Reusing results

In order to reuse strategies for strong bisimilarity to weak bisimilarity and weak congruence we lifted the strong semantics to weak counterparts – one for weak bisimilarity, and one for weak congruence. After this the proofs map to their strong counterparts almost completely. A valid question is if it is not easier to create a bisimilarity framework which given a semantics proves the congruence results for the resulting bisimulation relations. After all, the semantics in Figure 6.1 and Figure 9.3 are identical modulo the

Part	Lines of code
Agents	46
Semantics	230
Strong bisimilarity	517
Structural congruence	470
Weak bisimilarity	1044
Weak Congruence	748
Total	3055

Figure 11.1: The size of the different parts of the Isabelle formalisation of the CCS meta-theory.

transition arrows used. If a general framework were available, it would be possible to instantiate it with a semantics and be done – no proof duplication would be necessary. The answer is that yes, this is possible, but it is questionable if its worth the effort.

The main problem with a general framework is that weak bisimilarities rarely map as neatly onto their stronger counterparts as weak congruence does. There are usually some discrepancies or special cases as a result of ignoring the internal actions that have to be taken into consideration, and a uniform framework would have a hard time catching these. There are also cases where it is not possible to lift the semantics. The only non liftable rules for CCS are the SUM rules for weak transitions. But even if the rule is not liftable, it is possible to derive simulation lemmas.

Lemma 11.1. *Weak simulations preserved by Sum.*

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad Id \subseteq \mathscr{R}' \qquad \bigwedge S\ T\ U.\ \dfrac{(S, T) \in \mathscr{R}}{(S + U, T) \in \mathscr{R}'}}{P + R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} Q + R}$$

Weak bisimilarity is not preserved by Sum, yet this lemma proves that a weak simulation is preserved by Sum if for all P, Q and R, if P and Q are in $\mathscr{R}$, then $P + R$, and Q must be in $\mathscr{R}'$. Intuitively, this requirement is needed when Q does a τ-action to a Q' and P mimics by doing nothing – hence the derivatives P and Q' are in $\mathscr{R}$. The only way for $P + R$ to mimic the action of $Q + R$ is to do nothing, but then the derivatives $P + R$ and Q' must be in $\mathscr{R}'$. The corresponding part of the bisimilarity proof fails. A reasonable candidate relation would be $\mathscr{X} = \{(P + R, Q + R) : P \mathrel{\dot{\approx}} Q\}$, with $\mathscr{R}$ set to $\mathrel{\dot{\approx}}$ and $\mathscr{R}'$ set to $\mathscr{X}$ in the simulation lemma. To prove the requisites of the simulation lemma, we would need a lemma which states that $P + R \mathrel{\dot{\approx}} P$,

which is clearly not true. There is of course no guarantee that just because this strategy fails, the lemma is false – for that we have the counter example from Section 9.5.

In short, creating a general framework for process calculi which proves congruences is not trivial, and even though there is a bit of cut and pasting of proofs in this formalisation, the question is how much is actually gained by making a general framework if it can only be used for a few congruences. As the complexity of the calculi increase, so does the complexity of their proofs. In the next two parts where pi- and psi-calculi are described there will be examples of weak bisimilarites which differ significantly from their strong counterparts. The techniques used in this CCS formalisation are applicable and provide useful infrastructure for reasoning about weak equivalences.

Part III:
The pi-calculus

12. Introduction

In the pi-calculus, designed by Milner, Parrow, and Walker in 1992 [58], agents send communication channels to each other – channels which can later be used for further communication effectively changing the topology of the network.

Prefixes in the pi-calculus are either input, output or τ-actions.

Definition 12.1 (Actions)**.**

$$\begin{array}{lll} \alpha \stackrel{\text{def}}{=} & a(x) & \textit{Input} \\ & \overline{a}b & \textit{Output} \\ & \tau & \textit{Tau} \end{array}$$

The τ-prefix denotes the standard internal action; the prefix $a(x)$ is an input prefix, where the agent $a(x).P$ can receive a name along a, and the name x is a place-holder for the name received, and binding into P; finally, the prefix $\overline{a}b$ is an output prefix which sends the b over a.

Pi-calculus agents are defined in the following way:

Definition 12.2 (Agents)**.**

$$\begin{array}{lll} P \stackrel{\text{def}}{=} & \mathbf{0} & \textit{Nil} \\ & \alpha.P & (\textit{Input}, \textit{Output}, \textit{Tau}) \\ & [a{=}b]P & \textit{Match} \\ & [a{\neq}b]P & \textit{Mismatch} \\ & P + Q & \textit{Sum} \\ & P \mid Q & \textit{Parallel} \\ & (\nu x)P & \textit{Restriction} \\ & !P & \textit{Replication} \end{array}$$

The syntactic differences to CCS are the prefixes, and the match and mismatch operators. Intuitively, the agent $[a{=}b]P$ behaves as P if a and b are equal, and the agent $[a{\neq}b]P$ behaves like P if they are not.

In the pi-calculus names can both represent data being sent between agents, and communication channels which the agents can use. For instance in the system

$$\overline{a}b.P \mid a(x).\overline{x}c.Q$$

the agent $\overline{a}b.P$ can send the name b along a. The agent $a(x).\overline{x}c.Q$ can then receive b, also along a, and then output c along b. The transitions

$$\overline{a}b.P \mid a(x).\overline{x}c.Q \xrightarrow{\tau} P \mid \overline{b}c.Q\{^b/_x\} \xrightarrow{\overline{b}c} P \mid Q\{^b/_x\}$$

describe this derivation, where the agent $Q\{^b/_x\}$ represents the agent Q with all occurrences of x replaced by b.

As in CCS, it is possible to restrict channels such that they are local only to that agent. This is achieved through Restriction. For instance, restricting the name a in the system above would ensure that only the two parallel agents can use it for communication. The transitions

$$(\nu a)(\overline{a}b.P \mid a(x).\overline{x}c.Q) \xrightarrow{\tau} (\nu a)(P \mid \overline{b}c.Q\{^b/_x\}) \xrightarrow{\overline{b}c} (\nu a)(P \mid Q\{^b/_x\})$$

describe this derivation. Even though restricted channels cannot be used by agents outside the scope of Restriction, restricted names can be sent out of their scopes, which then grow to include the receiver. In the system

$$(\nu b)\overline{a}b.P \mid a(x).Q$$

the name b is restricted in the agent $(\nu b)\overline{a}b.P$ By outputting b along a, through the transition

$$(\nu b)\overline{a}b.P \mid a(x).Q \xrightarrow{\tau} (\nu b)(P \mid Q\{^b/_x\})$$

the scope of the ν-binder changes, and the name b is bound in both P and Q. Note that all occurrences of x in Q have been replaced by b.

12.1 Part outline

This part of the thesis will formalise a substantial amount of the meta theory for the pi-calculus. There are two styles of semantics – early and late operational semantics, and we will prove results for both. In the next chapter we will introduce the Isabelle formalisation of pi-calculus agents, as well as the early operational semantics and its induction and inversion rules. In Chapter 14 we will define strong bisimilarity, which contrary to CCS is not a congruence. We also define strong equivalence which is a congruence. In Chapter 15 we define weak bisimilarity. As for CCS weak bisimilarity for the pi-calculus is not a congruence – it is not preserved by Sum

for the same reasons. In Chapter 16 we define weak congruence in a similar way as is done for CCS, and prove that it is a congruence. In Chapter 17 we define the late operational semantics, we define the same bisimulation relations that were defined for the early semantics and prove the same results. In Chapter 18 we define structural congruence and prove that all late bisimilar agents are also structurally congruent. In Chapter 19 we define an axiomatisation of strong late bisimilarity of the finite pi-calculus (without Replication) and prove that it is sound and complete. In Chapter 20 we prove that all late bisimilar agents are also early bisimilar, allowing us to prove that all early bisimilar agents are also structurally congruent. Chapter 21 concludes.

13. Formalising the pi-calculus

As for CCS, pi-calculus agents utilise a single atom type *name* for its binding constructs. The nominal datatype for pi-calculus agents differs from that of CCS mainly in that the input prefix contains a binder.

Isabelle code	Syntax
nominal_datatype *pi* =	
PiNil	**0**
Output name name pi	$\overline{a}b.P$
Input name "«name»pi"	$a(x).P$
Match name name pi	$[a{=}b]P$
Mismatch name name pi	$[a{\neq}b]P$
Sum pi pi	$P + Q$
Par pi pi	$P \mid Q$
Res "«name»pi"	$(\nu x)P$
Bang pi	$!P$

From this definition, Isabelle automatically derives the standard injectivity rules. As for CCS, agents with binders are equated using the binding construct when determining equality.

Lemma 13.1. *Injectivity rules for pi-calculus agents.*

$$\begin{aligned}
\overline{a}b.P = \overline{c}d.Q &\Leftrightarrow a = c \wedge b = d \wedge P = Q \\
\tau.P = \tau.Q &\Leftrightarrow P = Q \\
a(x).P = b(y).Q &\Leftrightarrow a = b \wedge [x].P = [y].Q \\
[a{=}b]P = [c{=}d]Q &\Leftrightarrow a = c \wedge b = d \wedge P = Q \\
[a{\neq}b]P = [c{\neq}d]Q &\Leftrightarrow a = c \wedge b = d \wedge P = Q \\
P + Q = R + S &\Leftrightarrow P = R \wedge Q = S \\
P \mid Q = R \mid S &\Leftrightarrow P = R \wedge Q = S \\
(\nu x)P = (\nu y)Q &\Leftrightarrow [x].P = [y].Q \\
!P = !Q &\Leftrightarrow P = Q
\end{aligned}$$

Lemma 13.2.

$$\textit{If } y \sharp P \textit{ then } a(x).P = a(y).(x\,y) \cdot P.$$
$$\textit{If } y \sharp P \textit{ then } (\nu x)P = (\nu y)(x\,y) \cdot P.$$

Proof. Follows immediately from Lemmas 13.1 and 5.6. □

13.1 Substitution

In the pi-calculus, agents communicate by sending names. When an agent $a(x).P$ receives a name b all instances of x in P will be replaced by x. In Isabelle, this is modeled by creating two substitution functions – one for names, and one for agents. In both cases, the term $T\{^a/_x\}$ will denote a name or an agent T with all instances of x replaced by a.

The first function defines substitution on names.

Definition 13.3 (Name substitution)**.**

$$a\{^c/_b\} \stackrel{\text{def}}{=} \textit{if } a = b \textit{ then } c \textit{ else } a$$

Substitutions for agents is a function substituting names for names making sure that any binders do not clash with any of the names being substituted.

Definition 13.4 (Substitution)**.** *An agent P with all non binding occurrences of c replaced by d is denoted* $P\{^d/_c\}$.

$\mathbf{0}\{^d/_c\} = \mathbf{0}$
$(\tau.P)\{^d/_c\} = \underline{\tau}.P\{^d/_c\}$
$\overline{a}b.P\{^d/_c\} = \overline{a\{^d/_c\}}b\{^d/_c\}.P\{^d/_c\}$
$\textit{If } x \neq a \textit{ and } x \neq c \textit{ and } x \neq d \textit{ then } a(x).P\{^d/_c\} = a\{^d/_c\}(x).P\{^d/_c\}.$
$([a{=}b]P)\{^d/_c\} = [a\{^d/_c\}{=}b\{^d/_c\}]P\{^d/_c\}$
$([a{\neq}b]P)\{^d/_c\} = [a\{^d/_c\}{\neq}b\{^d/_c\}]P\{^d/_c\}$
$(P + Q)\{^d/_c\} = P\{^d/_c\} + Q\{^d/_c\}$
$(P \mid Q)\{^d/_c\} = P\{^d/_c\} \mid Q\{^d/_c\}$
$\textit{If } x \neq c \textit{ and } x \neq d \textit{ then } ((\nu x)P)\{^d/_c\} = (\nu x)P\{^d/_c\}.$
$(!P)\{^d/_c\} = !P\{^d/_c\}$

The side conditions for the Input case are stronger than what is required for the substitution to avoid the binder. They require that the name a is disjoint from x in the agent $a(x).P$ even though a is outside the scope of x. The reason for this has to do with how Isabelle handles inductive functions for nominal datatypes, where the binders are assumed to be as fresh as possible for the automatic heuristics. There are two ways to get around this. The

first is to not use the heuristics provided by Isabelle. The second is to derive an alternate simplification rule for the Input case where x and a can potentially be the same. We chose the latter option.

Lemma 13.5. *If $x \neq b$ and $x \neq c$ then $a(x).P\{^c/_b\} = a\{^c/_b\}(x).P\{^c/_b\}$.*

Proof. By picking a suitably fresh name y such that $y \sharp (a, b, c, P, P\{^b/_c\})$, alpha-converting x for y and the Input case from Definition 13.4. □

13.1.1 Lemmas for substitution

In order to reason efficiently about substitution in the following proofs, a few auxiliary lemmas need to be created. The first lemma that needs to be proven for any nominal function is equivariance.

Lemma 13.6. $p \cdot a\{^c/_b\} = (p \cdot a)\{^{p \cdot c}/_{p \cdot b}\}$
$p \cdot P\{^b/_a\} = (p \cdot P)\{^{p \cdot b}/_{p \cdot a}\}$

Proof. Substitution for names follows immediately from Definition 13.3. The proof that substitution on agents is equivariant follows from induction on P avoiding a and b, and Definition 13.4. □

The next step is to prove the freshness lemmas needed for the rest of the formalisation.

Lemma 13.7. *If $a \sharp P$ and $a \neq c$ then $a \sharp P\{^c/_b\}$.*
If $a \neq b$ then $a \sharp P\{^b/_a\}$.
If $a \sharp P$ then $P\{^b/_a\} = P$.

Proof. The exact proof strategy for the different sub-lemmas varies, but they are all proven by nominal induction over P avoiding a, b and c. □

Substituting a name for a name which does not already exists in an agent is equivalent to a permutation.

Lemma 13.8. *If $b \sharp P$ then $(a\ b) \cdot P = P\{^b/_a\}$.*

Proof. By induction on P avoiding a and b. □

13.2 Early operational semantics

Transitions in the pi-calculus have the form $P \xrightarrow{\alpha} P'$, where P is an agent, α is an action and P' is the α-derivative of P. At a first glance, this looks identical to the way transitions are defined in CCS. In the pi-calculus the action α

$$\frac{x \neq a \qquad x \neq u}{a(x).P \xrightarrow{au} P\{^{u}/_{x}\}}\ \textsc{Input} \qquad \frac{}{\overline{a}b.P \xrightarrow{\overline{a}b} P}\ \textsc{Output} \qquad \frac{}{\tau.P \xrightarrow{\tau} P}\ \textsc{Tau}$$

$$\frac{P \longmapsto V}{[b{=}b]P \longmapsto V}\ \textsc{Match} \qquad \frac{P \longmapsto V \qquad a \neq b}{[a{\neq}b]P \longmapsto V}\ \textsc{Mismatch}$$

$$\frac{P \longmapsto V}{P + Q \longmapsto V}\ \textsc{Sum1} \qquad \frac{Q \longmapsto V}{P + Q \longmapsto V}\ \textsc{Sum2} \qquad \frac{P \xrightarrow{\overline{a}b} P' \qquad a \neq b}{(\nu b)P \xrightarrow{a(\nu b)} P'}\ \textsc{Open}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad y \sharp \alpha}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'}\ \textsc{ScopeF}$$

$$\frac{P \xrightarrow{a(\nu x)} P' \qquad y \neq a \qquad y \neq x \qquad x \sharp P \qquad x \neq a}{(\nu y)P \xrightarrow{a(\nu x)} (\nu y)P'}\ \textsc{ScopeB}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1F}$$

$$\frac{P \xrightarrow{a(\nu x)} P' \qquad x \sharp P \qquad x \sharp Q \qquad x \neq a}{P \mid Q \xrightarrow{a(\nu x)} P' \mid Q}\ \textsc{Par1B}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2F}$$

$$\frac{Q \xrightarrow{a(\nu x)} Q' \qquad x \sharp P \qquad x \sharp Q \qquad x \neq a}{P \mid Q \xrightarrow{a(\nu x)} P \mid Q'}\ \textsc{Par2B}$$

$$\frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Comm1} \qquad \frac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Comm2}$$

$$\frac{P \xrightarrow{ax} P' \qquad Q \xrightarrow{a(\nu x)} Q' \qquad x \sharp P \qquad x \sharp Q \qquad x \neq a}{P \mid Q \xrightarrow{\tau} (\nu x)(P' \mid Q')}\ \textsc{Close1}$$

$$\frac{P \xrightarrow{a(\nu x)} P' \qquad Q \xrightarrow{ax} Q' \qquad x \sharp P \qquad x \sharp Q \qquad x \neq a}{P \mid Q \xrightarrow{\tau} (\nu x)(P' \mid Q')}\ \textsc{Close2}$$

$$\frac{P \mid !P \longmapsto V}{!P \longmapsto V}\ \textsc{Bang}$$

Figure 13.1: The inductive definition of the operational semantics for the pi-calculus. Every bound name is declared to be fresh for everything outside its scope to allow Isabelle to automatically generate induction and inversion rules.

may bind a name, in the case of a bound output, and the scope of this binding extends into P'. This observation is made already in the original presentation of the pi-calculus [58] where lemmas concerning variants of transitions are spelled out. In his tutorial on the polyadic pi-calculus [56] Milner uses "commitments" rather than labeled transitions. A transition here corresponds to a pair consisting of an agent and a commitment where the latter may have binders and contains both the action and derivative agent. Thus, there is a discrepancy between a more traditional syntax for transitions (looking like 3-tuples) and the intended semantics (that action and derivative in reality is one construct with names that can be bound in both). In many presentations of the pi-calculus this issue is glossed over, and if alpha-conversions are not defined rigorously the three-element syntax for transitions works fine. But here it poses a problem — it would require that the rules for changing the bound variables are explicitly stated, and the otherwise smooth treatment of α-variants provided by the nominal framework would be lost. Moreover no appeals can be made to the Barendregt variable convention. In Section 3.1.1 we showed an example for when the Barendregt variable convention is inconsistent. This example corresponds to the OPEN rule, and a three-element syntax for transitions will be unsound for the same reasons. Therefore, the formalisations in this thesis follows [56], with a slight change of notation to avoid confusion of prefixes and commitments, and defines a *residual*-datatype which contains both action and derivative. A binder in the action has the whole residual as scope, meaning that it also binds into the derivative. A similar technique was used by Gabbay when formalising the pi-calculus in FM set theory [38].

Definition 13.9 (Residuals)**.** *The residual datatype*

nominal_datatype *freeRes* =
InputR name name
| *OutputR name name*
| *Tau*

nominal_datatype *residual* =
BoundOutputR name «*name*»*pi*
| *FreeR freeRes pi*

Both these datatypes in the definition above are nominal datatypes, even though only the *residual*-datatype contains binders. The reason for this is that the nominal package provides equivariance and freshness lemmas only for nominal datatypes. A user is either required to do these by hand, or declare all datatypes as nominal datatypes.

The downside to this approach is that much of the infrastructure for regular datatypes is unavailable for nominal ones, most notably case distinc-

tion. Since a nominal datatype has infinitely many alpha-equivalent terms, case analysis is not trivial. The nominal package provides case-rules for inductively defined predicates, see Figure 6.4 for an example, but not for nominal datatypes. For nominal datatypes without binders, this restriction is reasonably mild since deriving case-rules for them is unproblematic.

As functions over nominal datatypes cannot depend on bound names. This poses a problem since traditionally, some of the operational rules have conditions on the bound names. An example of this is the PAR-rule, which in the standard operational semantics states that the transition $P \mid Q \xrightarrow{\alpha} P' \mid Q$ can occur only if $P \xrightarrow{\alpha} P'$ and $\text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset$. A function such as bn does not exist in nominal logic and thus cannot be created using the nominal datatype package. An easy solution is to split the operational rules which have these types of conditions into two rules — one for the transitions with bound names, and one for the ones without. Doing this creates extra proof obligations, but most proofs have to consider bound and free transitions separately anyway. In effect the complexity remains the same.

Definition 13.10 (Operational semantics)**.**

1. *A transition is written as $P \longmapsto V$ where P is an agent and V is a residual.*
2. *$P \xrightarrow{a(\nu x)} P'$ denotes a bound output transition with the bound name x in the action. The residual by itself is written $a(\nu y) \prec P'$*
3. *$P \xrightarrow{\alpha} P'$ denotes a transition without bound names. Note that α is of type freeRes. The residual by itself is written $\alpha \prec P'$.*

As mentioned in Chapter 5, Isabelle provides good support for automatically creating induction rules for inductively defined predicates containing nominal terms provided the bound names are sufficiently fresh. When defining the semantics, we therefore make sure that the bound names are fresh for everything outside of their scope. This has the extra advantage that the induction rule is more versatile, as all of these conditions will be available when doing induction on the inference of transitions. The disadvantage is that to apply the introduction rules of the predicate, these freshness conditions must be known – what is won at induction, is lost at introduction. Later in this section a solution for this will be discussed.

The operational semantics is defined by the introduction rules in Figure 13.1. It contains the split rules for PAR and RES and all bound names are fresh for everything outside their scope, more specifically fresh for everything outside their scope.

Isabelle automatically derives an equivarience lemma for the operational semantics.

Lemma 13.11. *If $P \longmapsto V$ then $(p \cdot P) \longmapsto (p \cdot V)$.*

Proof. By induction on $P \longmapsto V$ □

13.3 Nominal induction rules

In order to mimic the Barendregt variable convention, any rule with bound names in the assumptions must ensure that these bound names are sufficiently fresh. This is achieved by instantiating the induction rules with a freshness context $\mathscr{C}$ for which all bound names are fresh. Isabelle will automatically create a nominal induction rule with this property, found in Figure 13.2. The rule is quite large, and in induction over transitions of the form $P \longmapsto V$ there are a total of 19 inductive cases.

- One for Input, Output, and Tau respectively.
- One for Match and Mismatch respectively.
- Two for Sum.
- Eight for Parallel – the ParF, ParB, Comm and Close-rules and their symetric counterparts.
- Three for Restriction – one for Open, one for ResF and one for ResB
- One for Replication.

The induction rule looks like a general induction rule apart from the extra argument to the predicate to be proven. That extra argument $\mathscr{C}$ is the freshness conditions for the bound names.

As long as the complexity of the proofs is manageable, a big induction rule does not make the formalisation more difficult as Isabelle will manage large portions of the proofs automatically. Later in this thesis this induction rule will be used more extensively, but for now its main use will be to prove some straightforward lemmas.

The freshness conditions are actually stronger than strictly necessary. Not all of them are needed to derive the transitions since many of them can be inferred.

Lemma 13.12.

If $P \xrightarrow{\overline{a}b} P'$ *and* $c \sharp P$ *then* $c \neq a$.
If $P \xrightarrow{\overline{a}b} P'$ *and* $c \sharp P$ *then* $c \neq b$.
If $P \xrightarrow{\overline{a}b} P'$ *and* $c \sharp P$ *then* $c \sharp P'$.
If $P \xrightarrow{ab} P'$ *and* $c \sharp P$ *then* $c \neq a$.
If $P \xrightarrow{a(\nu x)} P'$ *and* $c \sharp P$ *then* $c \neq a$.
If $P \xrightarrow{au} P'$ *and* $c \sharp P$ *and* $c \neq u$ *then* $c \sharp P'$.
If $P \xrightarrow{a(\nu x)} P'$ *and* $c \sharp P$ *and* $c \neq x$ *then* $c \sharp P'$.
If $P \xrightarrow{\tau} P'$ *and* $c \sharp P$ *then* $c \sharp P'$.

Proof. All cases are proven by induction on their corresponding transitions. □

This lemma states that if a name is fresh for an agent and the bound names of its action, then it is also fresh for the action and any derivative of that action.

It is now possible to derive weaker introduction rules. The resulting semantics can be found in Figure 13.3. The rules are derived by starting with as few freshness conditions for the binders as possible and alpha-converting the binders to be as fresh as required by the old introduction rules.

Other useful lemmas relate the support of the agents in a transition. If an agent does an output action, no new names can be introduced in the derivative.

Lemma 13.13. *If* $P \xrightarrow{\overline{a}b} P'$ *then* $supp\, P' \subseteq supp\, P$.

Proof. By induction on $P \xrightarrow{\overline{a}b} P'$ □

Bound output actions open a binder, and a derivative can potentially contain the bound names of an action.

Lemma 13.14. *If* $P \xrightarrow{a(\nu x)} P'$ *and* $x \sharp P$ *then* $supp\, P' - \{x\} \subseteq supp\, P$.

Proof. By induction on $P \xrightarrow{a(\nu x)} P'$. In the OPEN-case the name x can appear in the derivative. □

As an input action receives a new name, any name received can potentially be in the derivative.

Lemma 13.15. *If* $P \xrightarrow{ax} P'$ *then* $supp\, P' - \{x\} \subseteq supp\, P$.

Proof. By induction on $P \xrightarrow{ax} P'$. In the INPUT-case the name x can be received and substituted into the derivative. □

Lemma 13.16. *If* $P \xrightarrow{\tau} P'$ *then* $supp\, P' \subseteq supp\, P$.

Proof. By induction on $P \xrightarrow{\tau} P'$. The relevant cases are the COMM and CLOSE cases, where lemmas 13.14 and 13.15 are used do determine the support of the derivatives of the communicating agents. In the COMM cases, the input name is already in the support of the agent outputting the same name. In the CLOSE cases, the bound output name is restricted in the derivative, and hence not a member of the support. □

$$R \longmapsto W$$

$$\bigwedge P\, \mathscr{C}.\ \frac{}{Prop\ \mathscr{C}\ (\tau.P)\ (\tau \prec P)}\ \textsc{Tau}$$

$$\bigwedge x\, a\, u\, P\, \mathscr{C}.\ \frac{x \neq a \qquad x \neq u \qquad x \sharp \mathscr{C}}{Prop\ \mathscr{C}\ a(x).P\ (au \prec P\{^{u}/_{x}\})}\ \textsc{Input}$$

$$\bigwedge a\, b\, P\, \mathscr{C}.\ \frac{}{Prop\ \mathscr{C}\ \overline{a}b.P\ (\overline{a}b \prec P)}\ \textsc{Output}$$

$$\bigwedge P\, V\, b\, \mathscr{C}.\ \frac{P \longmapsto V \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ V}{Prop\ \mathscr{C}\ ([b{=}b]P)\ V}\ \textsc{Match}$$

$$\bigwedge P\, V\, a\, b\, \mathscr{C}.\ \frac{P \longmapsto V \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ V \qquad a \neq b}{Prop\ \mathscr{C}\ ([a{\neq}b]P)\ V}\ \textsc{Mismatch}$$

$$\bigwedge P\, a\, b\, P'\, \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{\overline{a}b} P' \qquad Prop\ \mathscr{C}\ P\ (\overline{a}b \prec P') \\ a \neq b \qquad b \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ ((\nu b)P)\ (a(\nu b) \prec P')}\ \textsc{Open}$$

$$\bigwedge P\, V\, Q\, \mathscr{C}.\ \frac{P \longmapsto V \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ V}{Prop\ \mathscr{C}\ (P + Q)\ V}\ \textsc{Sum1}$$

$$\bigwedge Q\, V\, P\, \mathscr{C}.\ \frac{Q \longmapsto V \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ V}{Prop\ \mathscr{C}\ (P + Q)\ V}\ \textsc{Sum2}$$

$$\bigwedge P\, a\, x\, P'\, Q\, \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{a(\nu x)} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (a(\nu x) \prec P') \\ x \sharp P \qquad x \sharp Q \qquad x \neq a \qquad x \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (a(\nu x) \prec P' \mid Q)}\ \textsc{Par1B}$$

$$\bigwedge P\, \alpha\, P'\, Q\, \mathscr{C}.\ \frac{P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (\alpha \prec P')}{Prop\ \mathscr{C}\ (P \mid Q)\ (\alpha \prec P' \mid Q)}\ \textsc{Par1F}$$

$$\bigwedge Q\, a\, x\, Q'\, P\, \mathscr{C}.\ \frac{\begin{pmatrix} Q \xrightarrow{a(\nu x)} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (a(\nu x) \prec Q') \\ x \sharp P \qquad x \sharp Q \qquad x \neq a \qquad x \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (a(\nu x) \prec P \mid Q')}\ \textsc{Par2B}$$

$$\bigwedge Q\ \alpha\ Q'\ P\ \mathscr{C}.\ \frac{Q \xrightarrow{\alpha} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (\alpha \prec Q')}{Prop\ \mathscr{C}\ (P \mid Q)\ (\alpha \prec P \mid Q')}\ \textsc{Par2F}$$

$$\bigwedge P\ a\ b\ P'\ Q\ Q'\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{ab} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (ab \prec P') \\ Q \xrightarrow{\overline{a}b} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (\overline{a}b \prec Q') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (\tau \prec P' \mid Q')}\ \textsc{Comm1}$$

$$\bigwedge P\ a\ b\ P'\ Q\ Q'\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{\overline{a}b} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (\overline{a}b \prec P') \\ Q \xrightarrow{ab} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (ab \prec Q') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (\tau \prec P' \mid Q')}\ \textsc{Comm2}$$

$$\bigwedge P\ a\ x\ P'\ Q\ Q'\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{ax} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (ax \prec P') \\ Q \xrightarrow{a(vx)} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (a(vx) \prec Q') \\ x \sharp P & x \sharp Q \qquad x \neq a \qquad x \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (\tau \prec (vx)(P' \mid Q'))}\ \textsc{Close1}$$

$$\bigwedge P\ a\ x\ P'\ Q\ Q'\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{a(vx)} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (a(vx) \prec P') \\ Q \xrightarrow{ax} Q' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ Q\ (ax \prec Q') \\ x \sharp P & x \sharp Q \qquad x \neq a \qquad x \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid Q)\ (\tau \prec (vx)(P' \mid Q'))}\ \textsc{Close2}$$

$$\bigwedge P\ a\ y\ P'\ x\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{a(vy)} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (a(vy) \prec P') \\ x \neq a \qquad x \neq y & x \sharp \mathscr{C} \\ y \sharp P \qquad y \neq a & y \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ ((vx)P)\ (a(vy) \prec (vx)P')}\ \textsc{ScopeB}$$

$$\bigwedge P\ \alpha\ y\ P'\ x\ \mathscr{C}.\ \frac{\begin{pmatrix} P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ P\ (\alpha \prec P') \\ x \sharp \alpha \qquad x \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ ((vx)P)\ (\alpha \prec (vx)P')}\ \textsc{ScopeF}$$

$$\bigwedge P\ V\ \mathscr{C}.\ \frac{P \mid !P \longmapsto V \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (P \mid !P)\ V}{Prop\ \mathscr{C}\ (!P)\ V}\ \textsc{Repl}$$

$$Prop\ \mathscr{C}\ R\ W$$

Figure 13.2: Nominal inductive rule for doing induction over a transition of the form $R \longmapsto W$. The 19 cases are named after their corresponding semantic rule, and any newly introduced bound name will avoid the freshness context $\mathscr{C}$.

$$a(x).P \xrightarrow{au} P\{^u/_x\} \text{ INPUT} \qquad \frac{}{\overline{a}b.P \xrightarrow{\overline{a}b} P} \text{ OUTPUT} \qquad \frac{}{\tau.P \xrightarrow{\tau} P} \text{ TAU}$$

$$\frac{P \longmapsto V}{[b{=}b]P \longmapsto V} \text{ MATCH} \qquad \frac{P \longmapsto V \qquad a \neq b}{[a{\neq}b]P \longmapsto V} \text{ MISMATCH}$$

$$\frac{P \longmapsto V}{P + Q \longmapsto V} \text{ SUM1} \qquad \frac{Q \longmapsto V}{P + Q \longmapsto V} \text{ SUM2}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \text{ PAR1F} \qquad \frac{P \xrightarrow{a(vx)} P' \qquad x \sharp Q}{P \mid Q \xrightarrow{a(vx)} P' \mid Q} \text{ PAR1B}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \text{ PAR2F} \qquad \frac{Q \xrightarrow{a(vx)} Q' \qquad x \sharp P}{P \mid Q \xrightarrow{a(vx)} P \mid Q'} \text{ PAR2B}$$

$$\frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \text{ COMM1} \qquad \frac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \text{ COMM2}$$

$$\frac{P \xrightarrow{ax} P' \qquad Q \xrightarrow{a(vx)} Q' \qquad x \sharp P}{P \mid Q \xrightarrow{\tau} (vx)(P' \mid Q')} \text{ CLOSE1}$$

$$\frac{P \xrightarrow{a(vx)} P' \qquad Q \xrightarrow{ax} Q' \qquad x \sharp Q}{P \mid Q \xrightarrow{\tau} (vx)(P' \mid Q')} \text{ CLOSE2}$$

$$\frac{P \xrightarrow{\overline{a}b} P' \qquad a \neq b}{(vb)P \xrightarrow{a(vb)} P'} \text{ OPEN}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad y \sharp \alpha}{(vy)P \xrightarrow{\alpha} (vy)P'} \text{ SCOPEF} \qquad \frac{P \xrightarrow{a(vx)} P' \qquad y \neq a \qquad y \neq x}{(vy)P \xrightarrow{a(vx)} (vy)P'} \text{ SCOPEB}$$

$$\frac{P \mid !P \longmapsto V}{!P \longmapsto V} \text{ REPL}$$

Figure 13.3: The derived operational semantics rule for the pi-calculus. These rules have been derived from the ones in Figure 13.1, and the freshness conditions for each rule have been trimmed using Lemma 13.12 to include only the ones which are strictly necessary.

13.4 Inversion rules

Isabelle derives a custom inversion rule for nominal datatypes in the same manner as described in Chapter 5. This rule turns out to be very cumbersome to work with in practice, but it is very useful for deriving more applicable tailor made inversion rules.

13.4.1 Nominal inversion

The automatically generated inversion rule for the operational semantics can be found in Figure 13.4. It works in the same way as the inversion rule for CCS (Figure 6.4) in that the bound names in the different cases of the rule are free for the whole inversion rule. The intuition is that the user chooses which bound names are to be used, and as long as these bound names are sufficiently fresh, inversion can be applied. The number of bound names for the pi-calculus is quite large. Whereas CCS had one for the SCOPE-case, the pi-calculus has nine bound names distributed among the different cases.

This rule is cumbersome to work with. Whenever inversion is applied to a transition, all bound names must be instantiated, even the ones which do not match the inversion pattern, and are simplified away. The reason is that the rule is designed in such a way that the freshness conditions must be satisfied. Another problem, which holds for Isabelle's inversion rules in general, is that unification of equivalent terms is not performed in the predicate which is to be proven. This results in a proof state with lots of equivalence assumptions on terms. These can be discharged by applying a suitable automated heuristic in Isabelle, but in so doing, the Isar shorthand commands for cases, assumptions, and goal state are lost. Briefly put, Isabelle inversion rules work well for apply-scripts, but not for structured proofs.

The inversion rule is still useful to infer inversion rules for the individual cases.

13.4.2 Ensuring freshness of new bound names

Inversion on a transition can introduce new bound names which are not part of the transition being inverted. Consider inverting the transition $P \xrightarrow{\tau} P'$. The six applicable cases are the PAR, COMM and CLOSE-rules, and their symmetric versions, but the COMM and CLOSE rules introduce the new bound names x, and x and y respectively which do not occur in the original transition. Just as with induction, it must be possible to instantiate these names with names which are sufficiently fresh, depending on the current proof state.

$$R \longmapsto W$$

$$\bigwedge P.\ \frac{R = \tau.P \qquad W = \tau \prec P}{\mathit{Prop}}\ \textsc{Tau} \qquad \bigwedge a\ b\ P.\ \frac{\begin{pmatrix} R = \overline{a}b.P \\ W = \overline{a}b \prec P \end{pmatrix}}{\mathit{Prop}}\ \textsc{Output}$$

$$\bigwedge a\ u\ P.\ \frac{\left(\dfrac{x_1 \sharp R \qquad x_1 \sharp W}{R = a(x_1).P \wedge W = au \prec P\{^{u}/_{x_1}\} \wedge x_1 \neq a \wedge x_1 \neq u}\right)}{\mathit{Prop}}\ \textsc{Input}$$

$$\bigwedge P\ V\ b.\ \frac{R = [b{=}b]P \qquad W = V \qquad P \longmapsto V}{\mathit{Prop}}\ \textsc{Match}$$

$$\bigwedge P\ V\ Q.\ \frac{\begin{pmatrix} R = P + Q \\ W = V \\ P \longmapsto V \end{pmatrix}}{\mathit{Prop}}\ \textsc{Sum1} \qquad \bigwedge P\ V\ Q.\ \frac{\begin{pmatrix} R = P + Q \\ W = V \\ Q \longmapsto V \end{pmatrix}}{\mathit{Prop}}\ \textsc{Sum1}$$

$$\bigwedge P\ a\ P'\ Q.\ \frac{\left(\dfrac{x_2 \sharp R \qquad x_2 \sharp W}{\begin{array}{l} R = P \mid Q \wedge W = a(\nu x_2) \prec P' \mid Q \wedge P \xrightarrow{a(\nu x_2)} P' \wedge \\ x_2 \sharp P \wedge x_2 \sharp Q \wedge x_2 \neq a \end{array}}\right)}{\mathit{Prop}}\ \textsc{Par1B}$$

$$\bigwedge P\ \alpha\ P'\ Q.\ \frac{R = P \mid Q \qquad W = \alpha \prec P' \mid Q \qquad P \xrightarrow{\alpha} P'}{\mathit{Prop}}\ \textsc{Par1F}$$

$$\bigwedge P\ a\ P'\ Q.\ \frac{\left(\dfrac{x_3 \sharp R \qquad x_3 \sharp W}{\begin{array}{l} R = P \mid Q \wedge W = a(\nu x_3) \prec P \mid Q' \wedge Q \xrightarrow{a(\nu x_3)} Q' \wedge \\ x_3 \sharp P \wedge x_3 \sharp Q \wedge x_3 \neq a \end{array}}\right)}{\mathit{Prop}}\ \textsc{Par2B}$$

$$\bigwedge Q\ \alpha\ Q'\ P.\ \frac{R = P \mid Q \qquad W = \alpha \prec P \mid Q' \qquad Q \xrightarrow{\alpha} Q'}{\mathit{Prop}}\ \textsc{Par2F}$$

$$\bigwedge P\ a\ b\ P'\ Q\ Q'.\ \frac{\begin{pmatrix} R = P \mid Q & W = \tau \prec P' \mid Q' \\ P \xrightarrow{ab} P' & Q \xrightarrow{\overline{a}b} Q' \end{pmatrix}}{\mathit{Prop}}\ \textsc{Comm1}$$

$$\left[\begin{array}{l}
\bigwedge P\ a\ b\ P'\ Q\ Q'.\ \dfrac{\begin{pmatrix} R = P \mid Q & W = \tau \prec P' \mid Q' \\ P \xrightarrow{\overline{a}b} P' & Q \xrightarrow{ab} Q' \end{pmatrix}}{Prop}\ \textsc{Comm2} \\[2ex]
\bigwedge P\ a\ P'\ Q\ Q'.\ \dfrac{\left(\begin{array}{c} x_4 \sharp R \qquad x_4 \sharp W \\ \hline R = P \mid Q \wedge W = \tau \prec (\nu x_4)(P' \mid Q') \wedge x_4 \neq a \wedge \\ P \xrightarrow{ax_4} P' \wedge Q \xrightarrow{a(\nu x_4)} Q' \wedge x_4 \sharp P \wedge x_4 \sharp Q \end{array}\right)}{Prop}\ \textsc{Close1} \\[2ex]
\bigwedge P\ a\ P'\ Q\ Q'.\ \dfrac{\left(\begin{array}{c} x_5 \sharp R \qquad x_5 \sharp W \\ \hline R = P \mid Q \wedge W = \tau \prec (\nu x_5)(P' \mid Q') \wedge x_5 \neq a \wedge \\ P \xrightarrow{a(\nu x_5)} P' \wedge Q \xrightarrow{ax_5} Q' \wedge x_5 \sharp P \wedge x_5 \sharp Q \end{array}\right)}{Prop}\ \textsc{Close2} \\[2ex]
\bigwedge P\ a\ P'.\ \dfrac{\left(\begin{array}{c} x_6 \sharp R \qquad x_6 \sharp W \\ \hline R = (\nu x_6)P \wedge W = a(\nu x_6) \prec P' \wedge P \xrightarrow{\overline{a}x_6} P' \wedge a \neq x_6 \end{array}\right)}{Prop}\ \textsc{Open} \\[2ex]
\bigwedge P\ a\ P'.\ \dfrac{\left(\begin{array}{c} x_7 \neq y_1 \qquad x_7 \sharp R \qquad x_7 \sharp W \qquad y_1 \sharp R \qquad y_1 \sharp W \\ \hline R = (\nu y_1)P \wedge W = a(\nu x_7) \prec (\nu y_1)P' \wedge P \xrightarrow{a(\nu x_7)} P' \wedge \\ y_1 \neq a \wedge y_1 \neq x_7 \wedge x_7 \sharp P \wedge x_7 \neq a \end{array}\right)}{Prop}\ \textsc{ScopeB} \\[2ex]
\bigwedge P\ \alpha\ P'.\ \dfrac{\left(\begin{array}{c} x_8 \sharp R \qquad x_8 \sharp W \\ \hline R = (\nu x_8)P \wedge W = \alpha \prec (\nu x_8)P' \wedge P \xrightarrow{\alpha} P' \wedge x_8 \sharp \alpha \end{array}\right)}{Prop}\ \textsc{ScopeF} \\[2ex]
\bigwedge P\ V.\ \dfrac{R = !P \qquad W = V \qquad P \mid !P \longmapsto V}{Prop}\ \textsc{Repl}
\end{array}\right]$$

$$Prop$$

Figure 13.4: The automatically generated inversion rule for the operational semantics. The inversion is done on transitions of the form $R \longmapsto W$. Note that the bound names $x_1 \ldots x_8$ and y_1 are not quantified by their respective case, but globally for the whole inversion rule. Before applying the rule these bound names must be instantiated and made sufficiently fresh to satisfy the freshness conditions for their corresponding case.

$$\frac{\tau.P \xrightarrow{\alpha} P' \qquad Prop\,(\tau)\;P}{Prop\;\alpha\;P'}\;\textsc{Tau} \qquad \frac{\overline{a}b.P \xrightarrow{\alpha} P' \qquad Prop\,(\overline{a}b)\;P}{Prop\;\alpha\;P'}\;\textsc{Output}$$

$$\frac{a(x).P \xrightarrow{\alpha} P' \qquad \bigwedge u.\; Prop\;au\;(P\{^{u}/_{x}\})}{Prop\;\alpha\;P'}\;\textsc{Input}$$

$$\frac{\left[[a{=}b]P \longmapsto V \qquad \dfrac{P \longmapsto V}{F\;a\;a}\right]}{F\;a\;b}\;\textsc{Match}$$

$$\frac{\left[[a{\neq}b]P \longmapsto V \qquad \dfrac{P \longmapsto V \qquad a \neq b}{F\;a\;b}\right]}{F\;a\;b}\;\textsc{Mismatch}$$

$$\frac{\left[(\nu x)P \xrightarrow{\alpha} Q \qquad x \sharp \alpha \qquad \bigwedge P'.\; \dfrac{P \xrightarrow{\alpha} P'}{F\;((\nu x)P')}\right]}{F\;Q}\;\textsc{ScopeF}$$

$$\frac{\left[\begin{array}{l}(\nu y)P \xrightarrow{a(\nu x)} Q \qquad x \neq y \\ \\ \bigwedge P'.\; \dfrac{P \xrightarrow{\overline{a}y} P' \qquad a \neq y}{F\;((x\;y)\cdot P')} \qquad \bigwedge P'.\; \dfrac{P \xrightarrow{a(\nu x)} P' \qquad y \neq a}{F\;((\nu y)P')}\end{array}\right]}{F\;Q}\;\textsc{ScopeB}$$

$$\frac{\left[!P \longmapsto V \qquad \dfrac{P \mid !P \longmapsto V}{Prop}\right]}{Prop}\;\textsc{Repl}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{\alpha} R \\ \bigwedge P'.\ \dfrac{P \xrightarrow{\alpha} P'}{F\,\alpha\,(P' \mid Q)} \qquad \bigwedge Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{F\,\alpha\,(P \mid Q')} \\ \bigwedge P'\,Q'\,a\,b.\ \dfrac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{F\,(\tau)\,(P' \mid Q')} \\ \bigwedge P'\,Q'\,a\,b.\ \dfrac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{ab} Q'}{F\,(\tau)\,(P' \mid Q')} \\ \bigwedge P'\,Q'\,a\,x.\ \dfrac{P \xrightarrow{ax} P' \qquad Q \xrightarrow{a(vx)} Q' \qquad x \sharp P \qquad x \sharp \mathscr{C}}{F\,(\tau)\,((\nu x)(P' \mid Q'))} \\ \bigwedge P'\,Q'\,a\,x.\ \dfrac{P \xrightarrow{a(vx)} P' \qquad Q \xrightarrow{ax} Q' \qquad x \sharp Q \qquad x \sharp \mathscr{C}}{F\,(\tau)\,((\nu x)(P' \mid Q'))} \end{array}\right]}{F\,\alpha\,R}\ \textsc{ParF}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{a(vx)} R \\ \bigwedge P'.\ \dfrac{P \xrightarrow{a(vx)} P' \qquad x \sharp Q}{F\,(P' \mid Q)} \qquad \bigwedge Q'.\ \dfrac{Q \xrightarrow{a(vx)} Q' \qquad x \sharp P}{F\,(P \mid Q')} \end{array}\right]}{F\,R}\ \textsc{ParB}$$

Figure 13.5: The derived inversion rules for the pi-calculus, one rule for each operator.

This is achieved by picking arbitrary names which are sufficiently fresh to discharge the freshness conditions of the applicable case-rules, but also fresh for any supplied freshness context.

13.4.3 Rules with multiple binders

In the pi-calculus there is an extra case which needs to be considered about inversion on transitions which contain multiple bound names, such as $(\nu x)P \xrightarrow{a(\nu y)} P'$. This transition could have been derived either from the ScopeB, or the Open-rule. The ScopeB-rule requires that x and y are disjoint, whereas the Open-rule requires that they are equal, but things are not that simple. Even if x and y are disjoint, the transition could still have been derived by the Open-rule as shown by the following inference.

$$\dfrac{\dfrac{(\nu x)P \xrightarrow{\overline{a}b} (x\,y)\cdot P' \qquad x \neq a}{(\nu x)P \xrightarrow{a(\nu x)} (x\,y)\cdot P'}\,\textsc{Open} \qquad x \sharp P'}{(\nu x)P \xrightarrow{a(\nu y)} P'}\,\textsc{alpha-equivalence}$$

In this inference, the Open-rule is applied and the residual is then alpha-converted. This is still a valid transition as all terms are equal up to alpha-equivalence. This gives the somewhat unintuitive result that given a transition inferred by Open, $(\nu x)P \xrightarrow{a(\nu y)} P'$, x and y are not necessarily equal.

When creating a general inversion rule this poses a slight problem. Even though the intuitive view of the Open-rule is that the bound names are the same, the inversion rule must cover the cases where the names are disjoint. The solution is to always require all bound names to be disjoint. The nominal package provides infrastructure for ensuring that bound names are sufficiently fresh, and in particular they are fresh for any other bound name in consideration. The downside is that by requiring the bound names to be disjoint in the Open-rule, the alpha-converting permutation must explicitly be stated in the case-rule, as the above inference demonstrates.

13.5 Induction on replicated agents

Replication is the only operator which appears in the premise of its inference rule. Even though it would be possible to generate an inversion rule for it, that rule would not be very useful – most lemmas involving Replication require induction rather than inversion, as discussed in Section 6.5; we must hence create a special induction rule for Replication.

$$\frac{\left[\begin{array}{l} !P \longmapsto V \\[2ex] \bigwedge a\ x\ P'\ \mathscr{C}.\ \dfrac{P \xrightarrow{a(vx)} P' \qquad x \sharp P \qquad x \sharp \mathscr{C}}{Prop\ \mathscr{C}\ (P \mid !P)\ (a(vx) \prec P' \mid !P)}\ \textsc{Par1B} \\[2ex] \bigwedge \alpha\ P'\ \mathscr{C}.\ \dfrac{P \xrightarrow{\alpha} P'}{Prop\ \mathscr{C}\ (P \mid !P)\ (\alpha \prec P' \mid !P)}\ \textsc{Par1F} \\[2ex] \bigwedge a\ x\ P'\ \mathscr{C}\ \dfrac{\begin{pmatrix} !P \xrightarrow{a(vx)} P' \qquad x \sharp P \qquad x \sharp \mathscr{C} \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (a(vx) \prec P') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau \prec P' \mid P'')}\ \textsc{Par2B} \\[2ex] \bigwedge \alpha\ P'\ \mathscr{C}.\ \dfrac{!P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (\alpha \prec P')}{Prop\ \mathscr{C}\ (P \mid !P)\ (\alpha \prec P \mid P')}\ \textsc{Par2F} \\[2ex] \bigwedge a\ b\ P'\ P''\ \mathscr{C}\ \dfrac{\begin{pmatrix} !P \xrightarrow{ab} P' \qquad !P \xrightarrow{\overline{a}b} P'' \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (\overline{a}b \prec P'') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau \prec P' \mid P'')}\ \textsc{Comm1} \\[2ex] \bigwedge a\ b\ P'\ P''\ \mathscr{C}\ \dfrac{\begin{pmatrix} !P \xrightarrow{\overline{a}b} P' \qquad !P \xrightarrow{ab} P'' \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (ab \prec P'') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau \prec P' \mid P'')}\ \textsc{Comm2} \\[2ex] \bigwedge a\ x\ P'\ P''\ \mathscr{C}\ \dfrac{\begin{pmatrix} !P \xrightarrow{ax} P' \qquad !P \xrightarrow{a(vx)} P'' \qquad x \sharp P \qquad x \sharp \mathscr{C} \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (a(vx) \prec P'') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau \prec (vx)(P' \mid P''))}\ \textsc{Close1} \\[2ex] \bigwedge a\ x\ P'\ P''\ \mathscr{C}\ \dfrac{\begin{pmatrix} !P \xrightarrow{a(vx)} P' \qquad !P \xrightarrow{ax} P'' \qquad x \sharp P \qquad x \sharp \mathscr{C} \\ \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (!P)\ (ax \prec P'') \end{pmatrix}}{Prop\ \mathscr{C}\ (P \mid !P)\ (\tau \prec (vx)(P' \mid P''))}\ \textsc{Close2} \\[2ex] \bigwedge W\ \mathscr{C}.\ \dfrac{P \mid !P \longmapsto W \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (P \mid !P)\ W}{Prop\ \mathscr{C}\ (!P)\ W}\ \textsc{Repl} \end{array}\right]}{Prop\ \mathscr{C}\ (!P)\ V}$$

Figure 13.6: Custom induction rule for Replication. Induction is done on the transition $!P \longmapsto V$ to prove the predicate *Prop*. The rule has one case for replication, and one case for every semantic rule for Parallel.

14. Strong bisimilarity

When defining bisimilarity between two agents in the pi-calculus, extra care has to be taken with respect to bound names in actions. Consider the following agents:

$$P = a(u).(\nu b)\overline{b}x.\mathbf{0}$$
$$Q = a(x).\mathbf{0}$$

The agents P and Q should be bisimilar since they both can do only one input-action along a channel a and then nothing more. But since x occurs free in P, P cannot be alpha-converted into $a(x).(\nu b)\overline{b}x.\mathbf{0}$, which means P cannot mimic any action from Q. However, since agents have finite support, there exists a name w which is fresh in both P and Q and after alpha-converting both agents, they can mimic each other's actions. Hence, we only consider actions whose bound names are fresh for both P and Q.

This chapter will follow the same outline as Chapter 7. We will define strong bisimilarity for the pi-calculus, and prove that it is an equivalence relation. We also want an equivalence which is a congruence, but it turns out that a standard version of bisimilarity, as the one in Chapter 7, is not a congruence for the pi-calculus. The reason is that the communicating capabilities of pi-calculus agents allows interaction in such a way that two agents are equal for one set of inputs, but not for others. A counter-example is presented in [67]. Consider the following two agents, assuming that $a \neq b$.

$$P = a(x).\mathbf{0} \mid \overline{b}c.\mathbf{0} \qquad\qquad Q = a(x).\overline{b}c.\mathbf{0} + \overline{b}c.a(x).\mathbf{0}$$

At a first glance, these agents appear to be equal – since $a \neq b$, the agents themselves cannot communicate. Hence, whichever action P chooses to do, Q can follow by choosing the appropriate sub-agent, and vice versa. However, in the presence of substitutions, these agents are not equal. If both agents have all occurrences of b substituted for a, P can perform a τ-action, as the parallel sub-agents have the same subject, whereas Q cannot. In order for an equivalence to be preserved by Input, it must also be closed under substitutions – if two agents are to be considered equal, they must be equal for all possible inputs they can receive.

We will define strong bisimilarity in the standard way, prove that it is an equivalence relation and that it is preserved by all operators except by

Input. We will then close that relation under substitutions, to get a congruence.

14.1 Definitions

The coinductive definition of strong bisimilarity is done in exactly the same way as for CCS, making many of the bisimilarity proofs similar. However, the definition for simulation differs, and the scope migrating capabilities of the pi-calculus make the lemmas which reason about simulation more involved than for CCS.

The operational semantics for the pi-calculus defined in Figure 13.1 requires that some semantic rules are split into two cases – one where binders occur on the label, and one where they do not. When defining simulation, the same must be done, and two corresponding simulation cases are defined. an agent P is said to simulate an agent Q preserving the relation $\mathscr{R}$ if for every action Q can do, P can mimic that action and the derivatives are in $\mathscr{R}$.

Definition 14.1 (Simulation)**.** *An agent P simulating an agent Q preserving $\mathscr{R}$ is denoted $P \hookrightarrow_{\mathscr{R}} Q$*

$$
\begin{aligned}
&P \hookrightarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \\
&(\forall\, a\, y\, Q'.\ Q \xrightarrow{a(vy)} Q' \longrightarrow y \sharp P \longrightarrow (\exists P'.\ P \xrightarrow{a(vy)} P' \wedge (P', Q') \in \mathscr{R})) \wedge \\
&(\forall\, \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}))
\end{aligned}
$$

Before defining bisimilarity through coinduction, a monotonicity lemma for simulation is required.

Lemma 14.2. *If $P \hookrightarrow_{\mathscr{R}} P'$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \hookrightarrow_{\mathscr{R}'} P'$.*

Proof. Follows from the definition of $\hookrightarrow$. The assumption $\mathscr{R} \subseteq \mathscr{R}'$ ensures that any derivatives of P and Q in $\mathscr{R}$ also are in $\mathscr{R}'$. □

Definition 14.3 (Bisimilarity)**.** *Bisimilarity, denoted $\dot{\sim}$, is defined coinductively as the greatest fixpoint satisfying:*

$$
\begin{aligned}
P \dot{\sim} Q \Longrightarrow P \hookrightarrow_{\dot{\sim}} Q \quad & \textsc{Simulation} \\
\wedge\ Q \dot{\sim} P \quad & \textsc{Symmetry}
\end{aligned}
$$

14.1.1 Primitive inference rules

It is often necessary to ensure that the bound name in the actions is fresh for some freshness context $\mathscr{C}$ supplied by the user. Otherwise, there will often be a massive case analysis on whether or not bound names of actions occur free in any other terms in the proof. In a similar way as for induction and inversion, this tedium is bypassed by deriving a better introduction rule for simulation.

Lemma 14.4. *Introduction rule for simulation*

$$\frac{\begin{array}{c}\mathit{eqvt}\ \mathscr{R} \\ \bigwedge a\, y\, Q'.\ \dfrac{Q \xrightarrow{a(\nu y)} Q' \qquad y \sharp P \qquad y \sharp Q \qquad y \sharp \mathscr{C}}{\exists P'.\ P \xrightarrow{a(\nu y)} P' \wedge (P', Q') \in \mathscr{R}} \\ \bigwedge \alpha\, Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}\end{array}}{P \hookrightarrow_{\mathscr{R}} Q}\ \hookrightarrow\text{-I}$$

Proof. Follows from the definition of $\hookrightarrow$. The bound names in the actions of the transitions are alpha-converted to avoid P, Q, a and $\mathscr{C}$ and the fact that $\mathscr{R}$ is equivariant allows the alpha-converting permutations to be applied to the derivatives in $\mathscr{R}$. □

This introduction rule is used extensively in the upcoming proofs. It ensures that whenever bound names appear in the proof context, these bound names do not clash with other names and give rise to manual alpha-conversions. As a result, alpha-conversions are reduced to the instances where they would be required in a careful manual proof.

Note that the extra requirement that the simulation relation is equivariant is needed. The reason is that if the relation is not closed under permutations, the agents cannot be alpha-converted, as the derivatives would then fall outside the relation. As it turns out, all relations of interest turn out to be equivariant and the proofs of this trivial.

The elimination rules for simulation are derived in the standard way.

Lemma 14.5. *Elimination rules for simulation.*

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{a(\nu x)} Q' \qquad x \sharp P}{\exists P'.\ P \xrightarrow{a(\nu x)} P' \wedge (P', Q') \in \mathscr{R}}\ \hookrightarrow\text{-E1}$$

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}\ \hookrightarrow\text{-E2}$$

Proof. Follows from the definition of $\rightarrowtail$. □

Corresponding introduction and elimination rules for bisimilarity follow immediately from its definition.

Lemma 14.6.

$$\frac{P \rightarrowtail_{\dot\sim} Q \qquad Q \dot\sim P}{P \dot\sim Q}\ \dot\sim\text{-I} \qquad \frac{P \dot\sim Q}{P \rightarrowtail_{\dot\sim} Q}\ \dot\sim\text{-E1} \qquad \frac{P \dot\sim Q}{Q \dot\sim P}\ \dot\sim\text{-E2}$$

Proof. Follows from Definition 14.3. □

To prove that two agents are bisimilar, a symmetric candidate relation $\mathscr{X}$ is chosen containing the agents where all agent pairs simulate each other preserving $\mathscr{X} \cup \dot\sim$.

Lemma 14.7. *Coinduction rule for bisimilarity.*

$$\frac{\begin{array}{ll}(P, Q) \in \mathscr{X} & \\ \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{R \rightarrowtail_{\mathscr{X} \cup \dot\sim} S} & \textsc{Simulation} \\ \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{(S, R) \in \mathscr{X}} & \textsc{Symmetry}\end{array}}{P \dot\sim Q}$$

Proof. Follows from the coinduction rule that Isabelle provides from Definition 14.3 □

For the rest of this chapter, whenever a proof is done by coinduction, this will be the coinduction principle used.

14.1.2 Equivariance properties

The introduction rule for simulation has the that the relations under consideration are equivariant. Such relations will often derived from bisimilarity. Therefore, it is important that bisimilarity is equivariant. The first step is to prove this property for simulation.

Lemma 14.8.

If $P \rightarrowtail_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *and eqvt* $\mathscr{R}'$ *then* $p \cdot P \rightarrowtail_{\mathscr{R}'} p \cdot Q$.

Proof. Follows from the definition of $\hookrightarrow$. The fact that the transition system is equivariant (Lemma 13.11) makes it possible to cancel the permutation p by applying its inverse and perform the simulation. The proof can then be finished by applying the permutation p to the transition to cancel the inverse, and the assumption $eqvt\ \mathscr{R}'$ ensures that the derivatives are still in $\mathscr{R}'$. □

With the equivariance lemma for simulation in place, it is possible to prove that bisimilarity is equivariant.

Lemma 14.9. *If* $P \dot{\sim} Q$ *then* $(p \cdot P) \dot{\sim} (p \cdot Q)$.

Proof. The proof is done in two steps. Since Lemma 14.8 requires the simulating relation in the conclusion to be equivariant, coinduction cannot be used directly.

The first step is done by coinduction setting $\mathscr{X}$ to $\{(p \cdot P, p \cdot Q) : P \dot{\sim} Q\}$, and Lemma 14.8 is used to prove that P simulates Q with the simulation relation $\mathscr{X}$, which is trivially equivariant. The second step proves that P simulates Q with the simulating relation $\mathscr{X} \cup bisim$ using the monotonicity lemma 14.2. □

14.2 Bisimulation is an equivalence relation

Simulations are parametrised on an arbitrary relation $\mathscr{R}$. As for CCS every simulation lemma is augmented with constraints for that relation such that the lemma is provable.

Bisimilarity is symmetric by definition, but reflexivity and transitivity must be proven. These lemmas correspond closely to the corresponding proofs for CCS, but in the transitivity case, care must be taken so that any bound names on the labels of the transitions do not clash with the intermediate agent.

Lemma 14.10. *If* $Id \subseteq \mathscr{R}$ *then* $P \hookrightarrow_{\mathscr{R}} P$.

Proof. Follows from the definition of $\hookrightarrow$ and the fact that $Id \subseteq \mathscr{R}$. □

Lemma 14.11. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *and* $Q \hookrightarrow_{\mathscr{R}'} R$ *and* $eqvt\ \mathscr{R}''$ *and* $\mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}''$ *then* $P \hookrightarrow_{\mathscr{R}''} R$.

Proof. The case where R does a bound output is the complicated one. We use $\hookrightarrow$-I to ensure that any bound names of the action are fresh for P and Q. We hence have to prove that for all a, x and R' such that $R \xrightarrow{a(\nu x)} R'$ there exists an S such that $P \xrightarrow{a(\nu x)} S$ and $(S, R') \in \mathscr{R}''$.

- From $Q \hookrightarrow_{\mathscr{R}'} R$, $R \xrightarrow{a(\nu x)} R'$, and $x \sharp Q$ we obtain a Q' such that $Q \xrightarrow{a(\nu x)} Q'$ and $(Q', R') \in \mathscr{R}'$.
- From $P \hookrightarrow_{\mathscr{R}} Q$, $Q \xrightarrow{a(\nu x)} Q'$, and $x \sharp P$ we obtain a P' such that $P \xrightarrow{a(\nu x)} P'$ and $(P', Q') \in \mathscr{R}$.
- From $\mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}''$, $(P', Q') \in \mathscr{R}$' and $(Q', R') \in \mathscr{R}'$ we have that $(P', R') \in \mathscr{R}''$
- We can then prove the goal by setting S to P'.

The case where the action contains no bound names follow the same pattern. □

Lemma 14.12. *Bisimulation is an equivalence relation*

Proof. **Reflexivity:** $P \dot{\sim} P$

Follows by coinduction and setting $\mathscr{X}$ to Id and Lemma 14.10.

Symmetry: *If* $P \dot{\sim} Q$ *then* $Q \dot{\sim} P$.

Follows immediately from the definition of $\dot{\sim}$.

Transitivity: *If* $P \dot{\sim} Q$ *and* $Q \dot{\sim} R$ *then* $P \dot{\sim} R$.

Follows by coinduction and setting $\mathscr{X}$ to $\dot{\sim} \circ \dot{\sim}$, Lemma 14.11 and the fact that bisimilarity is equivariant.

□

We now proceed to prove that bisimilarity is preserved by all operators except Input.

14.3 Preservation properties

The preservation lemmas are similar to those for CCS, with some exceptions. Most notably, as bound names can be communicated between the agents with the OPEN and CLOSE-rules, any rule involving Parallel must take into account that the scope of the binders can change.

14.3.1 Output and Tau

The proofs that bisimilarity is preserved Output and Tau are straightforward.

Lemma 14.13. *If* $(P, Q) \in \mathscr{R}$ *then* $\tau.P \hookrightarrow_{\mathscr{R}} \tau.Q$.

Proof. Follows from the definition of $\hookrightarrow$, the fact that $\tau.P$ and $\tau.Q$ can each only do a τ-action and $(P, Q) \in \mathscr{R}$. □

Lemma 14.14. *If* $P \dot{\sim} Q$ *then* $\tau.P \dot{\sim} \tau.Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{(\tau.P, \tau.Q), (\tau.Q, \tau.P)\}$, and Lemma 14.13. □

Lemma 14.15. *If* $(P, Q) \in \mathscr{R}$ *then* $\overline{a}b.P \hookrightarrow_{\mathscr{R}} \overline{a}b.Q$.

Proof. Follows from the definition of $\hookrightarrow$, the fact that $\overline{a}b.P$ and $\overline{a}b.Q$ can each only do an output-action and $(P, Q) \in \mathscr{R}$. □

Lemma 14.16. *If* $P \dot{\sim} Q$ *then* $\overline{a}b.P \dot{\sim} \overline{a}b.Q$.

Proof. Follows by coinduction, setting $\mathscr{X}$ to $\{(\overline{a}b.P, \overline{a}b.Q), (\overline{a}b.Q, \overline{a}b.P)\}$, and Lemma 14.15. □

14.3.2 Match and Mismatch

Lemma 14.17. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $[a{=}b]P \hookrightarrow_{\mathscr{R}'} [a{=}b]Q$.

Proof. Follows from the definition of $\hookrightarrow$, the MATCH inversion rule from Figure 13.4.2 and the MATCH-rule from the operational semantics. The inversion rule ensures that $a = b$ and hence the agents can do exactly the same transitions. □

Lemma 14.18. *If* $P \dot{\sim} Q$ *then* $[a{=}b]P \dot{\sim} [a{=}b]Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{([a{=}b]P, [a{=}b]Q), ([a{=}b]Q, [a{=}b]P)\}$, and Lemma 14.17. □

Lemma 14.19. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $[a{\neq}b]P \hookrightarrow_{\mathscr{R}'} [a{\neq}b]Q$.

Proof. Follows from the definition of $\hookrightarrow$, the MISMATCH inversion rule from Figure 13.4.2 and the MISMATCH-rule from the operational semantics. The inversion rule ensures that $a \neq b$ and therefore, the agents can do exactly the same transitions. □

Lemma 14.20. *If* $P \dot{\sim} Q$ *then* $[a{\neq}b]P \dot{\sim} [a{\neq}b]Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{([a{\neq}b]P, [a{\neq}b]Q), ([a{\neq}b]Q, [a{\neq}b]P)\}$, and Lemma 14.19. □

14.3.3 Sum

Lemma 14.21.
If $P \hookrightarrow_{\mathscr{R}} Q$ *and* $Id \subseteq \mathscr{R}'$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $P + R \hookrightarrow_{\mathscr{R}'} Q + R$.

Proof. Follows from the definition of $\hookrightarrow$, the SUM inversion rule from Figure 13.4.2 and the SUM1 and SUM2-rule from the operational semantics. In the case where R does a transition, the assumption $Id \subseteq \mathscr{R}'$ is used to ensure that the derivatives remain in $\mathscr{R}'$. □

Lemma 14.22. *If* $P \mathrel{\dot{\sim}} Q$ *then* $P + R \mathrel{\dot{\sim}} Q + R$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{(P + R, Q + R), (Q + R, P + R)\}$, and Lemma 14.21. □

14.3.4 Restriction

Lemma 14.23.

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad \bigwedge R\, S\, y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}'} \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}'}{(\nu x)P \hookrightarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from the definition of $\hookrightarrow$, the SCOPE inversion rule and the OPEN, SCOPEF and SCOPEB-rules from the operational semantics. The assumption $\bigwedge R\, S\, y.\ (R, S) \in \mathscr{R} \Longrightarrow ((\nu y)R, (\nu y)S) \in \mathscr{R}'$ is used in the OPEN-case, as the restricted names are dropped from the derivatives. □

Lemma 14.24. *If* $P \mathrel{\dot{\sim}} Q$ *then* $(\nu x)P \mathrel{\dot{\sim}} (\nu x)Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{((\nu x)P, (\nu x)Q) : P \mathrel{\dot{\sim}} Q\}$, and Lemma 14.23. □

14.3.5 Parallel

The scope migrating capabilities of the pi-calculus make the proofs for Parallel more involved than their CCS counterparts. The first step is to prove what is required for simulation to be preserved by Parallel. We will start by proving a more general lemma, in which two simulations are composed.

Lemma 14.25.

$$\frac{\begin{array}{ll} P \hookrightarrow_{\mathscr{R}} Q & (P, Q) \in \mathscr{R} \\ R \hookrightarrow_{\mathscr{R}'} S & (R, S) \in \mathscr{R}' \\ \bigwedge P' Q' R' S'.\ \dfrac{(P', Q') \in \mathscr{R} \qquad (R', S') \in \mathscr{R}'}{(P' \mid R', Q' \mid S') \in \mathscr{R}''} & \end{array}}{P \mid R \hookrightarrow_{\mathscr{R}''} Q \mid S}$$

Proof. Follows from the $\hookrightarrow$-I introduction rule, the ParF and ParB inversion rule and the Par, Comm and Close-rules from the operational semantics. The requirement that $\mathscr{R}''$ is closed under restriction is used in the Close-cases to allow for scope migration. The Isabelle proof can be found in Figure 14.1. □

This lemma is more general than strictly necessary to prove that bisimilarity is preserved by Parallel. It will also be useful when we prove that bisimilarity is preserved by Replication. The lemma needed for parallel preservation is easily derivable.

Lemma 14.26.

$$\frac{\begin{array}{ll} P \hookrightarrow_{\mathscr{R}} Q & (P, Q) \in \mathscr{R} \\ \bigwedge S\ T\ U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U, T \mid U) \in \mathscr{R}'} & \bigwedge S\ T\ x.\ \dfrac{(S, T) \in \mathscr{R}'}{((\nu x)S, (\nu x)T) \in \mathscr{R}'} \end{array}}{P \mid R \hookrightarrow_{\mathscr{R}'} Q \mid R}$$

Proof. Follows from Lemma 14.25 by setting its relations $\mathscr{R}''$ to $\mathscr{R}'$ and $\mathscr{R}'$ to the identity relation. □

Before moving on to the bisimilarity part of this proof, we need to introduce the concept of a binding sequence.

The binders that have been discussed thus far have all been single binders in the sense that they are declared and bound one at a time. This turns out to be enough for most cases, but in bisimilarity proofs on parallel agents it becomes necessary to reason about sequences of binders. The reason for this is that the scope of binders can change using the Open and Close-rules.

A binding sequence is a finite, possibly empty, list of restrictions.

Definition 14.27 (Binding sequences)**.** *A sequence of names $\tilde{y}$ binding into an agent P is denoted $(\nu\tilde{y})P$.*

$$(\nu[])P = P \qquad\qquad (\nu(x{\cdot}\tilde{y}))P = (\nu x)(\nu\tilde{y})P$$

Figure 14.1: Isabelle proof that simulation is preserved by Parallel.

lemma parCompose:
fixes P :: pi **and** Q :: pi **and** R :: pi **and** T :: pi
and $\mathscr{R}$ :: (pi × pi) set **and** $\mathscr{R}'$ :: (pi × pi) set **and** $\mathscr{R}''$:: (pi × pi) set
assumes $P \hookrightarrow_{\mathscr{R}} Q$ **and** $R \hookrightarrow_{\mathscr{R}'} T$ **and** $(P, Q) \in \mathscr{R}$ **and** $(R, T) \in \mathscr{R}'$
and Par: $\bigwedge P' Q' R' T'.\ [\![(P', Q') \in \mathscr{R}; (R', T') \in \mathscr{R}']\!] \Longrightarrow (P' \mid R', Q' \mid T') \in \mathscr{R}''$
and Res: $\bigwedge P' Q' x.\ (P', Q') \in \mathscr{R}'' \Longrightarrow ((\nu x)P', (\nu x)Q') \in \mathscr{R}''$
shows $P \mid R \hookrightarrow_{\mathscr{R}''} Q \mid T$
proof(induct rule: simCases) — *Apply introduction rule* $\hookrightarrow$-I
case(Bound a x U)
from ‹$x \sharp (P \mid R)$› **have** $x \sharp P$ **and** $x \sharp R$ **by** simp+
from ‹$Q \mid T \xrightarrow{a(\nu x)} U$› **show** $\exists S.\ P \mid R \xrightarrow{a(\nu x)} S \wedge (S, U) \in \mathscr{R}''$
proof(induct rule: parCasesB) — *Apply* ParB *inversion rule from Figure 13.4.2*

ParB1 *case*
Given that $Q \xrightarrow{a(\nu x)} Q'$ *prove that there exists an S such that* $P \mid R \xrightarrow{a(\nu x)} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(cPar1 Q')
from ‹$P \hookrightarrow_{\mathscr{R}} Q$› ‹$Q \xrightarrow{a(\nu x)} Q'$› ‹$x \sharp P$› **obtain** P'
where PTrans: $P \xrightarrow{a(\nu x)} P'$ **and** $P' \mathscr{R} Q'$: $(P', Q') \in \mathscr{R}$ **by**(blast dest: elim)
from PTrans ‹$x \sharp R$› **have** $P \mid R \xrightarrow{a(\nu x)} P' \mid R$ **by**(rule Par1B)
moreover from $P' \mathscr{R} Q'$ ‹$(R, T) \in \mathscr{R}'$› **have** $(P' \mid R, Q' \mid T) \in \mathscr{R}''$ **by**(rule Par)
ultimately show $\exists PR'.\ P \mid R \xrightarrow{\overline{a}(\nu x)} PR' \wedge (PR', Q' \mid T) \in \mathscr{R}''$ **by** blast
next

ParB2 *case*
Given that $T \xrightarrow{a(\nu x)} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{a(\nu x)} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T')
from ‹$R \hookrightarrow_{\mathscr{R}'} T$› ‹$T \xrightarrow{a(\nu x)} T'$› ‹$x \sharp R$› **obtain** R'
where RTrans: $R \xrightarrow{a(\nu x)} R'$ **and** $R' \mathscr{R}' T'$: $(R', T') \in \mathscr{R}'$ **by**(blast dest: elim)
from RTrans ‹$x \sharp P$› **have** $P \mid R \xrightarrow{a(\nu x)} P \mid R'$ **by**(rule Par2B)
moreover from ‹$(P, Q) \in \mathscr{R}$› $R' \mathscr{R}' T'$ **have** $(P \mid R', Q \mid T') \in \mathscr{R}''$ **by**(rule Par)
ultimately show $\exists PR'.\ P \mid R \xrightarrow{\overline{a}(\nu x)} PR' \wedge (PR', Q \mid T') \in \mathscr{R}''$ **by** blast
qed

next
case(Free α QT′)
from ‹Q | T $\xrightarrow{\alpha}$ QT′› **show** ∃PR′. P | R $\xrightarrow{\alpha}$ PR′ ∧ (PR′, QT′) ∈ $\mathscr{R}''$
proof(induct rule: parCasesF[**where** C=(P, R)]) — ParF *inversion rule*

Par1 *case*
Given that Q $\xrightarrow{\alpha}$ *Q′ prove that there exists an S such that*
P | R $\xrightarrow{\alpha}$ *S and* (*S*, *Q′ | T*) ∈ $\mathscr{R}''$.

case(cPar1 Q′)
from ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\alpha}$ Q′› **obtain** P′
where PTrans: P $\xrightarrow{\alpha}$ P′ **and** P$\mathscr{R}$: (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: elim)
from PTrans **have** P | R $\xrightarrow{\alpha}$ P′ | R **by**(rule Par1F)
moreover from P$\mathscr{R}$ ‹(R, T) ∈ $\mathscr{R}$′› **have** (P′ | R, Q′ | T) ∈ $\mathscr{R}''$ **by**(rule Par)
ultimately show ∃PR′. P | R $\xrightarrow{\alpha}$ PR′ ∧ (PR′, Q′ | T) ∈ $\mathscr{R}''$ **by** blast
next

Par2 *case*
Given that T $\xrightarrow{\alpha}$ *T′ prove that there exists an S such that*
P | R $\xrightarrow{\alpha}$ *S and* (*S*, *Q | T′*) ∈ $\mathscr{R}''$.

case(cPar2 T′)
from ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\alpha}$ T′› **obtain** R′
where RTrans: R $\xrightarrow{\alpha}$ R′ **and** R$\mathscr{R}$: (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: elim)
from RTrans **have** P | R $\xrightarrow{\alpha}$ P | R′ **by**(rule earlySemantics.Par2F)
moreover from ‹(P, Q) ∈ $\mathscr{R}$› R$\mathscr{R}$ **have** (P | R′, Q | T′) ∈ $\mathscr{R}''$ **by**(rule Par)
ultimately show ∃PR′. P | R $\xrightarrow{\alpha}$ PR′ ∧ (PR′, Q | T′) ∈ $\mathscr{R}''$ **by** blast
next

Comm1 *case*
Given that Q $\xrightarrow{ab}$ *Q′ and T* $\xrightarrow{a\,[b]}$ *T′ prove that there exists an S*
such that P | R $\xrightarrow{\tau}$ *S and* (*S*, *Q′ | T′*) ∈ $\mathscr{R}''$.

case(cComm1 Q′T′ a b)
from ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{ab}$ Q′› **obtain** P′
where PTrans: P $\xrightarrow{ab}$ P′ **and** P′$\mathscr{R}$Q′: (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: elim)
from ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}b}$ T′› **obtain** R′
where RTrans: R $\xrightarrow{\overline{a}b}$ R′ **and** R$\mathscr{R}$T′: (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: elim)
from PTrans RTrans **have** P | R $\xrightarrow{\tau}$ P′ | R′ **by**(rule Comm1)
moreover from P′$\mathscr{R}$Q′ R$\mathscr{R}$T′ **have** (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$ **by**(rule Par)
ultimately show ∃PR′. P | R $\xrightarrow{\tau}$ PR′ ∧ (PR′, Q′ | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM2 *case*
Given that $Q \xrightarrow{a\,[b]} Q'$ *and* $T \xrightarrow{ab} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm2 Q′ T′ a b)
from ‹P ↪$_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}b}$ Q'› **obtain** P′
where PTrans: P $\xrightarrow{\overline{a}b}$ P′ **and** P′$\mathscr{R}$Q′: (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: elim)
from ‹R ↪$_{\mathscr{R}'}$ T› ‹T $\xrightarrow{ab}$ T'› **obtain** R′
where RTrans: R $\xrightarrow{ab}$ R′ **and** R′$\mathscr{R}'$T′: (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: elim)
from PTrans RTrans **have** P | R $\xrightarrow{\tau}$ P′ | R′ **by**(rule Comm2)
moreover from P′$\mathscr{R}$Q′ R′$\mathscr{R}'$T′ **have** (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$ **by**(rule Par)
ultimately show ?case **by** blast

next

CLOSE1 *case*
Given that $Q \xrightarrow{ax} Q'$ *and* $T \xrightarrow{a(vx)} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, (vx)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose1 Q′ T′ a x) **from** ‹x ♯ (P, R)› **have** x ♯ P **and** x ♯ R **by** simp+
from ‹P ↪$_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{ax}$ Q'› **obtain** P′
where PTrans: P $\xrightarrow{ax}$ P′ **and** P′$\mathscr{R}$Q′: (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: elim)
from ‹R ↪$_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}(vx)}$ T'› ‹x ♯ R› **obtain** R′
where RTrans: R $\xrightarrow{\overline{a}(vx)}$ R′ **and** R′$\mathscr{R}'$T′: (R′, T′) ∈ $\mathscr{R}'$
by(blast dest: elim)
from PTrans RTrans ‹x ♯ P› **have** P | R $\xrightarrow{\tau}$ (vx)(P′ | R′) **by**(rule Close1)
moreover from P′$\mathscr{R}$Q′ R′$\mathscr{R}'$T′ **have** ((vx)(P′ | R′), (vx)(Q′ | T′)) ∈ $\mathscr{R}''$
by(blast intro: Par Res)
ultimately show ?case **by** blast

next

CLOSE2 *case*

Given that $Q \xrightarrow{a(vx)} Q'$ *and* $T \xrightarrow{ax} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, (\nu x)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose2 Q′ T′ a x) **from** ‹x ♯ (P, R)› **have** x ♯ P **and** x ♯ R **by** simp+
from ‹P ↪$_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}(vx)}$ Q′› ‹x ♯ P› **obtain** P′
where PTrans: P $\xrightarrow{\overline{a}(vx)}$ P′ **and** P′$\mathscr{R}$Q′: (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: elim)
from ‹R ↪$_{\mathscr{R}'}$ T› ‹T $\xrightarrow{ax}$ T′› **obtain** R′
where RTrans: R $\xrightarrow{ax}$ R′ **and** R′$\mathscr{R}'$T′: (R′, T′) ∈ $\mathscr{R}'$
by(blast dest: elim)
from PTrans RTrans ‹x ♯ R› **have** P | R $\xrightarrow{\tau}$ (νx)(P′ | R′) **by**(rule Close2)
moreover from P′$\mathscr{R}$Q′ R′$\mathscr{R}'$T′ **have** $((\nu x)(P' \mid R'), (\nu x)(Q' \mid T')) \in \mathscr{R}''$
by(blast intro: Par Res)
ultimately show ?case **by** blast
qed
qed

Binding sequences recursively bind the names of a sequence to an agent.

A lemma which will be used extensively in the upcoming proofs is the following, which states what is required for a simulation to be closed under a binding sequence.

Lemma 14.28.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge R\ S\ x.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu x)R, (\nu x)S) \in \mathscr{R}} \qquad P \hookrightarrow_{\mathscr{R}} Q}{(\nu \tilde{y})P \hookrightarrow_{\mathscr{R}} (\nu \tilde{y})Q}$$

Proof. By induction on $\tilde{y}$.

Base case ($\tilde{y} = []$)**:** Follows immediately from the assumption $P \hookrightarrow_{\mathscr{R}} Q$.

Inductive step ($\tilde{y} = x\tilde{x}$)**:** From the induction hypothesis we get that $(\nu \tilde{x})P \hookrightarrow_{\mathscr{R}} (\nu \tilde{x})Q$, and hence by Lemma 14.23 that $(\nu x)(\nu \tilde{x})P \hookrightarrow_{\mathscr{R}} (\nu x)(\nu \tilde{x})Q$

□

The intuition behind the lemma is quite simple. If a simulation relation $\mathscr{R}$ is preserved by the Restriction and P simulates Q preserving $\mathscr{R}$, then since $\mathscr{R}$ is preserved by restriction and thus $(\nu x)P$ simulates $(\nu x)Q$ preserving $\mathscr{R}$ for an arbitrary name x, then by induction $(\nu x)P$ must simulate $(\nu x)Q$

preserving $\mathscr{R}$ where $\tilde{x}$ is an arbitrary chain of restricted names. The fact that the proof is an inductive proof requires the candidate relation $\mathscr{R}$ to be the same in the assumptions and the conclusion. This is a general lemma which is used repeatedly when proving bisimilarites involving Parallel.

This section has touched briefly on the notion of binding sequences, and what is needed for this formalisation of the pi-calculus. Most notably, any reasoning about alpha-equivalence of agents with binding sequences has been omitted. Binding sequences is a research area in its own right and will be covered extensively in Part IV where they form an integral part of the calculi being formalised.

We can now prove that bisimilarity is preserved by Parallel.

Lemma 14.29. *If* $P \nsim Q$ *then* $P \mid R \nsim Q \mid R$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to $\{((\nu x)(P \mid R), (\nu x)(Q \mid R)) : P \nsim Q\}$, and Lemmas 14.26 and 14.28. □

14.3.6 Replication

As for CCS, in order to prove that bisimilarity is preserved by the Replication, we inductively define a candidate relation which is preserved by the Replication. The scope migrating capabilities of the pi-calculus require that any candidate relation involving Parallel is preserved by restriction as well.

Definition 14.30 (bangRel)**.** *The bangRel relation is parametrised with a relation* $\mathscr{R}$.

If $(P, Q) \in \mathscr{R}$ *then* $(!P, !Q) \in bangRel\,\mathscr{R}$.
If $(R, T) \in \mathscr{R}$ *and* $(P, Q) \in bangRel\,\mathscr{R}$ *then* $(R \mid P, T \mid Q) \in bangRel\,\mathscr{R}$.
If $(P, Q) \in bangRel\,\mathscr{R}$ *then* $((\nu a)P, (\nu a)Q) \in bangRel\,\mathscr{R}$.

The predicate *bangRel* takes a relation as an argument, and returns a relation which is closed by Replication, Parallel, and Restriction. Moreover, the agents appearing on the right hand side of the |-operator are members of $bangRel\,\mathscr{R}$; the intuition is that as with Replication the *bangRel* predicate can be unfolded, adding new parallel agents an arbitrary number of times.

The next step is to prove what is required of a relation $\mathscr{R}$ for a simulation to preserve $bangRel\,\mathscr{R}$.

Lemma 14.31. *Simulation is preserved by Replication.*

$$\frac{(P, Q) \in \mathscr{R} \qquad \bigwedge R\,S.\ \dfrac{(R, S) \in \mathscr{R}}{R \hookrightarrow_{\mathscr{R}} S} \qquad eqvt\ \mathscr{R}}{!P \hookrightarrow_{bangRel\ \mathscr{R}} !Q}$$

Proof. By induction using the induction rule for the Replication from Figure 13.6 on the transitions that $!Q \longmapsto U$. □

Lemma 14.32. *If* $P \dot{\sim} Q$ *then* $!P \dot{\sim} !Q$.

Proof. By coinduction with X set to $bangRel\ \dot{\sim}$. The candidate relation is symmetric since $\dot{\sim}$ is symmetric. The simulation cases are resolved by Lemma 14.31 for the $!$-case, Lemma 14.25 for the $|$-case, and Lemma 14.23 for the ν-case. □

With these lemmas in place, we can prove that bisimilarity is preserved by all operators except Input.

Theorem 14.1. *Bisimilarity is preserved by all operators, except Input.*

Proof. Follows from lemmas 14.16, 14.14, 14.18, 14.20, 14.22, 14.29, 14.24, and 14.32. □

14.4 Strong equivalence

Strong bisimilarity is not preserved by Input, since the environment can change the behaviour of an agent by sending names to it. The intuition is that to have a congruence, two equal agents must behave the same regardless of any names received from the environment. To achieve this, bisimilarity is closed under all possible substitutions, and a congruence is obtained. The first step to formalise this is to introduce the notion of sequential substitutions.

14.4.1 Sequential substitution

Sequential substitutions are sequences of single substitutions.

Definition 14.33 (Sequential substitution)**.** *Sequential substitution is denoted* $a\sigma$ *for names, and* $P\sigma$ *for agents.*

$$\begin{array}{lllll} a[] & = a & & P[] & = P \\ a(x{\cdot}\sigma) & = a\{^{snd\ x}/_{fst\ x}\}\sigma & & P(x{\cdot}\sigma) & = P\{^{snd\ x}/_{fst\ x}\}\sigma \end{array}$$

Note that sequential substitution will continue substitution until σ is empty. It is therefore possible for a name to be substituted several times with one sequential substitution. This differentiates it from parallel substitution, where substitution is aborted when the first match was found. The reason for using sequential substitution and not parallel is that it is easier to define, the proofs of its behaviour are simpler, and it is

enough for this formalisation. Parallel substitution will be discussed in detail in Part IV.

To allow sequential substitution to function similarly as single substitutions, the laws from the substitution function are derived for sequential substitutions.

Lemma 14.34. *Sequential substitution distributes over the agents.*

$$\begin{aligned}
&\mathbf{0}\sigma = \mathbf{0}\\
&(\tau.P)\sigma = \tau.P\sigma\\
&\textit{If } x \sharp \sigma \textit{ then } a(x).P\sigma = a\sigma(x).P\sigma.\\
&\overline{a}b.P\sigma = \overline{a\sigma}b\sigma.P\sigma\\
&(P + Q)\sigma = P\sigma + Q\sigma\\
&(P \mid Q)\sigma = P\sigma \mid Q\sigma\\
&\textit{If } x \sharp \sigma \textit{ then } ((\nu x)P)\sigma = (\nu x)P\sigma.\\
&(!P)\sigma = !P\sigma
\end{aligned}$$

Proof. By induction over σ. □

Finally, sequential substitution is equivariant.

Lemma 14.35. $p \cdot P\sigma = (p \cdot P)(p \cdot \sigma)$

Proof. By induction over σ. □

14.4.2 Closure under substitution

A relation is closed under substitutions if the result of applying any sequential substitution to any pair in the relation is also in that relation.

Definition 14.36 (substClosed)**.** *Closing a relation $\mathscr{R}$ under substitutions is denoted $\mathscr{R}^s$.*

$$\mathscr{R}^s \stackrel{\text{def}}{=} \{(P, Q) : \forall \sigma.\ (P\sigma, Q\sigma) \in \mathscr{R}\}$$

Closing a relation under substitution preserves equivariance.

Lemma 14.37. *If eqvt $\mathscr{R}$ then eqvt $\mathscr{R}^s$.*

Proof. By the definition of *eqvt* we must prove that for all P and Q, if $(P, Q) \in \mathscr{R}^s$ then $(p \cdot P, p \cdot Q) \in \mathscr{R}^s$. By Definition 14.36 we must prove that for all σ, $((p \cdot P)\sigma, (p \cdot Q)\sigma) \in \mathscr{R}$.

- Since $(P,\ Q) \in \mathscr{R}^s$ we have that $(P(p^- \cdot \sigma),\ P(p^- \cdot \sigma)) \in \mathscr{R}$ by Definition 14.36.
- Hence $(p \cdot P(p^- \cdot \sigma),\ p \cdot P(p^- \cdot \sigma)) \in \mathscr{R}$ since $\mathscr{R}$ is equivariant, and hence $((p \cdot P)\sigma,\ (p \cdot Q)\sigma) \in \mathscr{R}$ by equivariance.

□

Lemma 14.38. $\mathscr{R}^s \subseteq \mathscr{R}$

Proof. Follows immediately since the empty substitution is a substitution. □

14.4.3 Strong equivalence

By closing bisimilarity under substitutions a congruence is obtained. The notation $\sim$ will be used for $\dot{\sim}^s$, called strong equivalence. That strong equivalence is a subset of strong bisimilarity follows trivially.

Lemma 14.39. *If* $P \sim Q$ *then* $P \dot{\sim} Q$.

Proof. Follows from Lemma 14.38. □

Most proofs for the preservation properties of strong equivalence follow from their corresponding proofs for strong bisimilarity, and the definition for closure under substitutions. The exceptions are agents containing binders. As Lemma 14.34 shows, in order for a sequential substitution to be pushed into a term with a binder, that binder must not occur free in the substitution. Since substitution closure require that all possible substitutions must be taken into account, a name clash will invariably occur and a manual alpha-conversion has to be done. This is a typical example of where omitting alpha-conversions in a pen-and-paper proof can be dangerous.

The remaining result, that strong equivalence is preserved by Input, is not entirely straightforward. The first step is to prove an auxiliary lemma which states that two agents $a(x).P$ and $a(x).Q$ are bisimilar if P and Q are bisimilar for all possible substitutions. As with all bisimilarity proofs, we first prove a simulation lemma.

Lemma 14.40.
If $\forall\, y.\ (P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R}$ *and* $eqvt\ \mathscr{R}$ *then* $a(x).P \hookrightarrow_{\mathscr{R}} a(x).Q$.

Proof. Follows from the definition of $\hookrightarrow$ and the fact that $a(x).P$ and $a(x).Q$ can each only do an input-action. The assumption $\forall\, y.\ (P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R}$ ensures that whatever input name the agents receive along a, the derivatives will still be in $\mathscr{R}$. □

The corresponding lemma for bisimilarity is:

Lemma 14.41. *If* $\forall y.\ P\{^y/_x\} \dot{\sim} Q\{^y/_x\}$ *then* $a(x).P \dot{\sim} a(x).Q$.

Proof. Follows from coinduction with $\mathscr{X}$ set to $\{(a(x).P, a(x).Q) : \forall y.\ P\{^y/_x\} \dot{\sim} Q\{^y/_x\}\}$ and Lemma 14.40.

□

We can now prove that strong equality is preserved by Input.

Lemma 14.42. *If* $P \sim Q$ *then* $a(x).P \sim a(x).Q$.

Proof. We must prove that $a(x).P\sigma \dot{\sim} a(x).Q\sigma$ for all σ. We alpha-convert x to y such that y does not clash with σ, and push σ past the binder. Since $P \sim Q$, we know that $P\sigma\{^z/_y\} \dot{\sim} Q\sigma\{^z/_y\}$ for all z, and hence the lemma is proven from Lemma 14.41. The Isabelle proof can be found in Figure 14.2. □

To prove that strong equivalence is a congruence we must prove that it is an equivalence relation preserved by all operators.

Lemma 14.43. *Strong equivalence is an equivalence relation.*

Proof. Follows immediately from Lemma 14.12 and Definition 14.36. □

Lemma 14.44. *Strong equivalence is closed under all operators.*

Proof. All cases, except the one for Input, follow directly from their counterparts for bisimilarity.

Preserved by Output: *If* $P \sim Q$ *then* $\overline{a}b.P \sim \overline{a}b.Q$.

Follows from lemmas 14.16, 14.34 and Definition 14.36.

Preserved by Tau: *If* $P \sim Q$ *then* $\tau.P \sim \tau.Q$.

Follows from lemmas 14.14, 14.34 and Definition 14.36.

Preserved by Input: *If* $P \sim Q$ *then* $a(x).P \sim a(x).Q$.

The full Isabelle proof is presented in Figure 14.2.

Preserved by Match: *If* $P \sim Q$ *then* $[a{=}b]P \sim [a{=}b]Q$.

Follows from lemmas 14.18, 14.34 and Definition 14.36.

Preserved by Mismatch: *If* $P \sim Q$ *then* $[a{\neq}b]P \sim [a{\neq}b]Q$.

Follows from lemmas 14.20, 14.34 and Definition 14.36.

Preserved by Sum: *If* $P \sim Q$ *then* $P + R \sim Q + R$.

Follows from lemmas 14.22, 14.34 and Definition 14.36.

Preserved by Parallel: *If* $P \sim Q$ *then* $P \mid R \sim Q \mid R$.

Follows from lemmas 14.29, 14.34 and Definition 14.36.

Preserved by Restriction: *If* $P \sim Q$ *then* $(\nu x)P \sim (\nu x)Q$.

Follows from lemmas 14.24, 14.34 and Definition 14.36. Alpha-conversion of the binder x is done to not clash with the substitutions.

Preserved by Replication: *If* $P \sim Q$ *then* $!P \sim !Q$.

Follows from lemmas 14.32, 14.34 and Definition 14.36.

□

The main congruence theorem can now be proven.

Theorem 14.2. *Stronge equivalence is a congruence*

Proof. Follows immediately from lemmas 14.43 and 14.44. □

```
lemma eqInputPres:
 fixes P :: pi and Q :: pi and a :: name and x :: name
 assumes P ∼ Q
 shows a(x).P ∼ a(x).Q
proof(auto simp add: substClosed-def)
 fix σ :: (name × name) list
 {
  fix P Q a x σ
  assume P ∼ Q
  then have ∀ y. P(σ@[(x, y)]) ≁ Q(σ@[(x, y)])
   by(unfold substClosed-def) blast
  hence ∀ y. (Pσ){y/x} ≁ (Qσ){y/x} by simp
  hence aσ(x).(Pσ) ≁ aσ(x).(Qσ) by(rule bisimInputPres)
  moreover assume x ♯ σ
  ultimately have (a(x).P)σ ≁ (a(x).Q)σ by simp
 }
 note Goal = this

 obtain y::name where y ♯ P and y ♯ Q and y ♯ σ
  by(generate-fresh name) auto
 from ‹P ∼ Q› have ([(x, y)] · P) ∼ ([(x, y)] · Q) by(rule eqvts)
 hence (a(y).([(x, y)] · P))σ ≁ (a(y).([(x, y)] · Q))σ
  using ‹y ♯ σ› by(rule Goal)
 moreover from ‹y ♯ P› ‹y ♯ Q›
 have a(x).P = a(y).([(x, y)] · P) and a(x).Q = a(y).([(x, y)] · Q)
  by(simp add: alphaInput)+

 ultimately show (a(x).P)σ ≁ (a(x).Q)σ by simp
qed
```

Figure 14.2: The proof that strong equivalence is preserved by Input.

15. Weak bisimilarity

The definition of weak bisimilarity follows that of CCS very closely. The layout of this chapter will follow that of its CCS counterpart, Chapter 9. We will first define τ-chains as the reflexive transitive closure of τ-actions. We will then define two types of transitions – one which is contains at least the action on the label, and another which can do nothing if the label is a τ. The early operational semantics defined in Figure 13.3 will then be lifted as much as possible to include both types of weak transitions.

15.1 τ-chains

We define τ-chains in the standard way, as the reflexive transitive closure of τ-actions.

Definition 15.1 (τ-chains)**.**

$$P \Longrightarrow P' \stackrel{\text{def}}{=} (P, P') \in \{(P, P') : P \longmapsto \tau \prec P'\}^*$$

Lemma 15.2. *Rule for induction over the length of a τ-chain.*

$$\frac{P \Longrightarrow P' \qquad \mathit{Prop}\ P \qquad \bigwedge P''\ P'''.\ \dfrac{P \Longrightarrow P'' \qquad P'' \xrightarrow{\tau} P''' \qquad \mathit{Prop}\ P''}{\mathit{Prop}\ P'''}}{\mathit{Prop}\ P'}$$

Proof. Follows directly from Isabelle's induction rule for reflexive transitive closures. □

This induction lemma can be used to prove that τ-chains are equivariant.

Lemma 15.3. *If $P \Longrightarrow P'$ then $(p \cdot P) \Longrightarrow (p \cdot P')$.*

Proof. By induction on the length of $P \Longrightarrow P'$

Base case ($P = P'$) Follows immediately since $(p \cdot P) \Longrightarrow (p \cdot P)$.

Inductive step From the induction hypothesis we get that $(p \cdot P) \Longrightarrow (p \cdot P')$. Moreover, since $P' \xrightarrow{\tau} P''$, we have that $p \cdot P' \xrightarrow{\tau} p \cdot P''$, by Lemma 13.11, and hence $(p \cdot P) \Longrightarrow (p \cdot P'')$.

□

$$\frac{P \Longrightarrow P' \qquad P \neq P'}{[b{=}b]P \Longrightarrow P'}\ \textsc{Match} \qquad \frac{P \Longrightarrow P' \qquad a \neq b \qquad P \neq P'}{[a{\neq}b]P \Longrightarrow P'}\ \textsc{Mismatch}$$

$$\frac{P \Longrightarrow P' \qquad P \neq P'}{P + Q \Longrightarrow P'}\ \textsc{Sum1F} \qquad \frac{Q \Longrightarrow Q' \qquad Q \neq Q'}{P + Q \Longrightarrow Q'}\ \textsc{Sum2F}$$

$$\frac{P \Longrightarrow P'}{P \mid Q \Longrightarrow P' \mid Q}\ \textsc{Par1F} \qquad \frac{Q \Longrightarrow Q'}{P \mid Q \Longrightarrow P \mid Q'}\ \textsc{Par1F}$$

$$\frac{P \Longrightarrow P'}{(\nu a)P \Longrightarrow (\nu a)P'}\ \textsc{ResF}$$

Figure 15.1: The lifted operational semantics rules for τ-chains. Note that the MATCH, MISMATCH, and the SUM rules require that the τ-chain does at least one τ-action. The COMM and CLOSE rules are not included as they have non τ-actions in their premises.

We also need to be able to infer freshness conditions of the derivative of τ-chains.

Lemma 15.4. *If $P \Longrightarrow P'$ and $x \sharp P$ then $x \sharp P'$.*

Proof. By induction on the length of $P \Longrightarrow P'$.

Base case ($P = P'$) Follows from the assumption that. $x \sharp P$.

Inductive step From the induction hypothesis we get that $x \sharp P'$, and hence, since $P' \xrightarrow{\tau} P''$, we have that $x \sharp P''$ using Lemma 13.12. □

Finally, the semantic rules from Figure 13.3 need to be lifted to support τ-chains. Rules whose derivatives are not of the same syntactic structure as the original agent, such as the SUM or the MATCH rules, need an extra premise that at least one action is performed. A full list of the lifted rules can be found in Figure 15.1. Note that these rules are inferred from the original semantics.

15.2 Weak Semantics

The semantics for the pi-calculus has two types of transitions – with binders on the label, and without. We extend the notation used for the strong se-

mantics and write, $P \xRightarrow{\overline{a}(vx)} P'$ for a weak bound output action, and $P \xRightarrow{\alpha} P'$ for transitions with no binders.

Definition 15.5 (Weak τ-respecting transitions)**.**

$$P \xRightarrow{\overline{a}(vx)} P' \stackrel{\text{def}}{=} \exists P''' P''.\, P \Longrightarrow P''' \wedge P''' \xrightarrow{a(vx)} P'' \wedge P'' \Longrightarrow P'$$

$$P \xRightarrow{\alpha} P' \stackrel{\text{def}}{=} \exists P''' P''.\, P \Longrightarrow P''' \wedge P''' \xrightarrow{\alpha} P'' \wedge P'' \Longrightarrow P'$$

A weak τ-respecting transitions is a transition with a preceding and succeeding τ-chain.

Note that even though the transition $P \xRightarrow{\overline{a}(vx)} P'$ appears to contain a binder into P', in reality it does not. The binder occurs inside the definition, where x binds into P''. The agent P'' then does a τ-chain to P', which x does not bind into, unless $P'' = P'$. Formally, one can still reason about x as a binder. Consider the following lemma:

Lemma 15.6. *If $P \Longrightarrow P'$ then $supp\, P' \subseteq supp\, P$.*

Proof. By induction on $P \Longrightarrow P'$, and Lemma 13.16. □

This lemma proves that there is no way that any new names can be introduced by a τ-chain; the name x can be communicated within the agent, but if so it occurs free in an output-prefix in P. We will call the binder x of a transition $P \xRightarrow{\overline{a}(vx)} P'$ a virtual binder since it is not a binder by definition; we create lemmas so that we can reason about virtual binders as if they were a real one.

The following lemma alpha-converts weak bound output transitions.

Lemma 15.7. *If $P \xRightarrow{\overline{a}(vx)} P'$ and $y \sharp P$ then $P \xRightarrow{\overline{a}(vy)} (x\,y) \cdot P'$.*

Proof. The proof is initially done by case analysis whether or not $x = y$. If so, the permutation cancels out, and the proof is done. We now prove the case where $x \neq y$.

From $P \xRightarrow{\overline{a}(vx)} P'$, and Definition 15.5, we obtain a P''' and a P'' such that $P \Longrightarrow P'''$, $P''' \xrightarrow{a(vx)} P''$, and $P'' \Longrightarrow P'$. The proof is then structured as follows:

- From $P \Longrightarrow P'''$, and $y \sharp P$, we have that $y \sharp P'''$, by Lemma 15.4.
- From $P''' \xrightarrow{a(vx)} P''$, $y \sharp P'''$, and $x \neq y$, we have that $y \sharp P''$, by Lemma 13.12.
- From $P''' \xrightarrow{a(vx)} P''$ and $y \sharp P''$ we can alpha-convert the transition to $P''' \xrightarrow{a(vy)} (x\,y) \cdot P''$.
- From $P'' \Longrightarrow P'$ we have that $(x\,y) \cdot P'' \Longrightarrow (x\,y) \cdot P'$, using Lemma 15.3.

$\tau.P \stackrel{\tau}{\Longrightarrow} P$ TAU $\qquad a(x).P \stackrel{au}{\Longrightarrow} P\{^{u}/_{x}\}$ INPUT $\qquad \overline{a}b.P \stackrel{\overline{a}b}{\Longrightarrow} P$ OUTPUT

$$\frac{P \stackrel{\overline{b}(\nu x)}{\Longrightarrow} P'}{[a{=}a]P \stackrel{\overline{b}(\nu x)}{\Longrightarrow} P'}\ \textsc{MatchB} \qquad \frac{P \stackrel{\alpha}{\Longrightarrow} P'}{[a{=}a]P \stackrel{\alpha}{\Longrightarrow} P'}\ \textsc{MatchF}$$

$$\frac{P \stackrel{\overline{c}(\nu x)}{\Longrightarrow} P' \qquad a \neq b}{[a{\neq}b]P \stackrel{\overline{c}(\nu x)}{\Longrightarrow} P'}\ \textsc{MismatchF} \qquad \frac{P \stackrel{\alpha}{\Longrightarrow} P' \qquad a \neq b}{[a{\neq}b]P \stackrel{\alpha}{\Longrightarrow} P'}\ \textsc{MismatchB}$$

$$\frac{P \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P'}{P + Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P'}\ \textsc{Sum1B} \qquad \frac{P \stackrel{\alpha}{\Longrightarrow} P'}{P + Q \stackrel{\alpha}{\Longrightarrow} P'}\ \textsc{Sum1F}$$

$$\frac{Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} Q'}{P + Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} Q'}\ \textsc{Sum2B} \qquad \frac{Q \stackrel{\alpha}{\Longrightarrow} Q'}{P + Q \stackrel{\alpha}{\Longrightarrow} Q'}\ \textsc{Sum2F}$$

$$\frac{P \stackrel{\alpha}{\Longrightarrow} P'}{P \mid Q \stackrel{\alpha}{\Longrightarrow} P' \mid Q}\ \textsc{Par1F} \qquad \frac{P \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P' \qquad x \sharp Q}{P \mid Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P' \mid Q}\ \textsc{Par1B}$$

$$\frac{Q \stackrel{\alpha}{\Longrightarrow} Q'}{P \mid Q \stackrel{\alpha}{\Longrightarrow} P \mid Q'}\ \textsc{Par2F} \qquad \frac{Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} Q' \qquad x \sharp P}{P \mid Q \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P \mid Q'}\ \textsc{Par2B}$$

$$\frac{P \overset{ab}{\Longrightarrow} P' \qquad Q \overset{\overline{a}b}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'}\ \textsc{Comm1} \qquad \frac{P \overset{\overline{a}b}{\Longrightarrow} P' \qquad Q \overset{ab}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'}\ \textsc{Comm2}$$

$$\frac{P \overset{ax}{\Longrightarrow} P' \qquad Q \overset{\overline{a}(vx)}{\Longrightarrow} Q' \qquad x \sharp P}{P \mid Q \overset{\tau}{\Longrightarrow} (vx)(P' \mid Q')}\ \textsc{Close1}$$

$$\frac{P \overset{\overline{a}(vx)}{\Longrightarrow} P' \qquad Q \overset{ax}{\Longrightarrow} Q' \qquad x \sharp Q}{P \mid Q \overset{\tau}{\Longrightarrow} (vx)(P' \mid Q')}\ \textsc{Close2} \qquad \frac{P \overset{\overline{a}b}{\Longrightarrow} P' \qquad a \neq b}{(vb)P \overset{\overline{a}(vb)}{\Longrightarrow} P'}\ \textsc{Open}$$

$$\frac{P \overset{\alpha}{\Longrightarrow} P' \qquad x \sharp \alpha}{(vx)P \overset{\alpha}{\Longrightarrow} (vx)P'}\ \textsc{ResF} \qquad \frac{P \overset{\overline{a}(vx)}{\Longrightarrow} P' \qquad y \neq a \qquad y \neq x}{(vy)P \overset{\overline{a}(vx)}{\Longrightarrow} (vy)P'}\ \textsc{ResB}$$

$$\frac{P \mid !P \overset{\overline{a}(vx)}{\Longrightarrow} P'}{!P \overset{\overline{a}(vx)}{\Longrightarrow} P'}\ \textsc{ReplB} \qquad \frac{P \mid !P \overset{\alpha}{\Longrightarrow} P'}{!P \overset{\alpha}{\Longrightarrow} P'}\ \textsc{ReplF}$$

Figure 15.2: The lifted weak early operational semantics for the pi-calculus. Note that in addition to the PAR and the RES-rules, the SUM and the REPL rules need to be duplicated for bound and free transitions respectively.

- Finally, from $P \Longrightarrow P'''$, $P''' \xrightarrow{a(\nu y)} (x\ y) \cdot P''$, and $(x\ y) \cdot P'' \Longrightarrow (x\ y) \cdot P'$, we have that $P \xRightarrow{\overline{a}(\nu y)} (x\ y) \cdot P'$, by Definition 15.5.

□

The main difference from this lemma and the general alpha-converting lemmas is that the fresh name is required to be fresh for the originating agent, and not for the derivative. As the proof above demonstrates, the freshness conditions are propagated through the chain until the alpha-conversion is done on the internal single transition. With this slight discrepancy, the virtual binders of weak τ-respecting transitions can be alpha-converted in a manner similar to their strong counterparts.

15.2.1 Lifting the semantics

Lifting the semantics to the weak level is relatively straightforward, with one observation – weak τ-respecting transitions are by definition split into two kinds, one for free and one for bound transitions, and thus all rules which reason about arbitrary transitions must be duplicated. This differs from the strong semantics in Figure 13.2 which only splits the rules that explicitly needs to reason about the binding structure of a transition, such as the PAR and the SCOPE rules. Hence the weak semantics must in addition have duplicated rules for the SUM and REPL cases. The lifted semantics can be found in Figure 15.2.

In order to define weak bisimilarity, an agent must have the option of doing nothing when mimicking a τ-action. The weaker transition $P \xRightarrow{\widehat{\alpha}} P'$ is defined in the standard way.

Definition 15.8 (Weak transition)**.**

$$P \xRightarrow{\widehat{\alpha}} P' \stackrel{\text{def}}{=} P \xRightarrow{\alpha} P' \vee \alpha = \tau \wedge P = P'$$

A transition $P \xRightarrow{\widehat{\alpha}} P'$ either does a weak transition $P \xRightarrow{\alpha} P'$ or, if $\alpha = \tau$, the agent P can do nothing and hence $P = P'$.

These transitions will be used when defining weak bisimilarity. We can derive lifted rules also for this kind of transition. The relevant cases are the ones where a τ-chain is empty, see Figure 15.3, the other cases have already lifted by the semantics defined in Figure 15.2.

15.3 Weak bisimilarity

Formally, weak bisimilarity is defined in a similar way to the previous bisimularities in this thesis.

$$\frac{P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'}{P \mid Q \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \mid Q}\ \textsc{Par1F} \qquad \frac{Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'}{P \mid Q \stackrel{\widehat{\alpha}}{\Longrightarrow} P \mid Q'}\ \textsc{Par2F}$$

$$\frac{P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \qquad x \sharp \alpha}{(\nu x)P \stackrel{\widehat{\alpha}}{\Longrightarrow} (\nu x)P'}\ \textsc{ResF}$$

Figure 15.3: The lifted semantics for weak transitions with possibly empty τ-chains.

The proof strategies for the rest of this chapter follow that of strong bisimilarity very closely, as the lifted semantics allow us to reuse most of its proof heuristics.

Weak simulation is defined in the same way as its strong counterpart, with the exception that the mimicking action can do an arbitrary number of τ-actions prior to and after the visible action. In case the visible action is a τ-action, the mimicking agent has the option of doing nothing. After both transitions are made, the derivatives must be in a provided candidate relation.

Definition 15.9 (Weak simulation)**.** *An agent P weakly simulating an agent Q preserving $\mathscr{R}$ is denoted $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$.*

$P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \stackrel{\text{def}}{=}$

$(\forall\, a\, x\, Q'.\ Q \xrightarrow{a(\nu x)} Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P \stackrel{\overline{a}(\nu x)}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R})) \wedge$

$(\forall\, \alpha\, Q'.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}))$

In order to coinductively define weak bisimilarity, the generating function must be monotonic – in this case, weak simulation. The following lemma proves this.

Lemma 15.10. *If $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} P'$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} P'$.*

Proof. Follows directly from the definition of $\widehat{\rightsquigarrow}$. □

Weak bisimilarity can then be defined coinductively in the standard way.

Definition 15.11 (Weak bisimilarity)**.** *Weak bisimilarity, denoted $\dot{\approx}$, is defined coinductively as the greatest fixpoint satisfying:*

$$\begin{aligned} P \dot{\approx} Q \Longrightarrow P &\mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q && \textsc{Simulation} \\ \wedge\ Q &\dot{\approx} P && \textsc{Symmetry} \end{aligned}$$

15.3.1 Primitive inference rules

As for strong simulation, weak simulation is divided into two cases – one where a bound output action, containing a binder, is mimicked, and one where free actions are mimicked. Moreover, a special introduction rule ensures that any bound name occurring in the bound output action is fresh for an arbitrary context $\mathscr{C}$.

Lemma 15.12.

$$\frac{\bigwedge a\, x\, Q'.\ \dfrac{Q \xrightarrow{a(vx)} Q' \qquad x \sharp P \qquad x \sharp Q \qquad x \neq a \qquad x \sharp \mathscr{C}}{\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R}} \qquad \bigwedge \alpha\, Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}}}{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\ \widehat{\rightsquigarrow}\text{-I}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$. The bound names of the actions are alpha-converted to avoid P, Q, a, and the freshness context $\mathscr{C}$. □

The corresponding elimination rules follow immediately from the definition.

Lemma 15.13. *Elimination rules for weak simulation*

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{a(vx)} Q' \qquad x \sharp P}{\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E1}$$

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E2}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$. □

Lemma 15.14. *Introduction and elimination rules for weak bisimilarity.*

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q \qquad Q \dot{\approx} P}{P \dot{\approx} Q}\ \dot{\approx}\text{-I} \qquad \frac{P \dot{\approx} Q}{P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q}\ \dot{\approx}\text{-E1} \qquad \frac{P \dot{\approx} Q}{Q \dot{\approx} P}\ \dot{\approx}\text{-E2}$$

Proof. Follows from the definition of $\dot{\approx}$. □

Lemma 15.15. *Coinduction rule for weak bisimilarity.*

$$\dfrac{\begin{array}{l}(P, Q) \in \mathscr{X} \\ \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} S} \quad \textsc{Simulation} \\ \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{(S, R) \in \mathscr{X}} \quad \textsc{Symmetry}\end{array}}{P \dot{\approx} Q}$$

Proof. Follows from the coinduction rule that Isabelle derives from Definition 15.11. □

15.3.2 Equivariance

To prove that weak bisimilarity is equivariant we have to prove the corresponding result for τ-chains, weak transitions, and weak simulation.

Lemma 15.16. *τ-chains are equivariant.*

$$\textit{If } P \Longrightarrow P' \textit{ then } (p \cdot P) \Longrightarrow (p \cdot P').$$

Proof. By induction on $P \Longrightarrow P'$ and Lemma 13.11. □

Lemma 15.17. *Weak transitions are equivariant*

$$\textit{If } P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \textit{ then } p \cdot P \stackrel{\widehat{p \cdot \alpha}}{\Longrightarrow} p \cdot P'.$$

Proof. Follows from Definitions 15.8 and 15.5, and Lemmas 15.16 and 13.11. □

Lemma 15.18. *Weak simulation is equivariant.*

$$\textit{If } P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \textit{ and } \mathscr{R} \subseteq \mathscr{R}' \textit{ and } \mathrm{eqvt}\, \mathscr{R}' \textit{ then } p \cdot P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} p \cdot Q.$$

Proof. Similar to Lemma 14.8 but uses Lemma 15.17 to infer equivariance of the mimicking transitions. □

Lemma 15.19. *Weak bisimilarity is equivariant.*

$$\textit{If } P \dot{\approx} Q \textit{ then } p \cdot P \dot{\approx} p \cdot Q.$$

Proof. Similar to Lemma 14.9 but uses Lemma 15.18 to infer equivariance of the weak simulations. □

15.3.3 Weak bisimilarity includes strong bisimilarity

Lemma 15.20.

$$\textit{If } P \xrightarrow{a(vx)} P' \textit{ then } P \overset{\overline{a}(vx)}{\Longrightarrow} P'.$$
$$\textit{If } P \xrightarrow{\alpha} P' \textit{ then } P \overset{\widehat{\alpha}}{\Longrightarrow} P'.$$

Proof. Follows from Definitions 15.5 and 15.8 by using empty τ-chains. □

Lemma 15.21. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *then* $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$.

Follows from Definitions 14.1, 15.9, and Lemma 15.20.

Lemma 15.22. *If* $P \mathrel{\dot{\sim}} Q$ *then* $P \mathrel{\dot{\approx}} Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\dot{\sim}$. The candidate relation is symmetric since $\mathscr{X}$ is symmetric, and the simulation case follows directly from Lemma 15.21. □

15.4 Weak bisimulation is an equivalence relation

The proofs required for simulation are reflexivity and transitivity, in order to prove that weak bisimulation is an equivalence relation. That weak bisimilarity is symmetric follows from its definition.

Lemma 15.23. *If* $Id \subseteq \mathscr{R}$ *then* $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} P$.

Proof. Follows immediately from the definition of $\widehat{\rightsquigarrow}$. □

The proof that a weak simulation is transitive follows the corresponding proof for CCS very closely. We must prove that the simulating agent can mimic a weak action, and not just a strong one. The following lemma shows how τ-chains are mimicked.

Lemma 15.24.

$$\frac{Q \Longrightarrow Q' \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S}}{\exists P'.\ P \Longrightarrow P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. Proved in the same way as Lemma 9.18.

□

The next step is to prove how weak transitions are mimicked. As weak simulation is split into two cases, one for a bound output, and one for free actions, the following lemma is split into two cases.

Lemma 15.25.

$$\frac{\bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S} \qquad \mathit{eqvt}\ \mathscr{R} \qquad (P, Q) \in \mathscr{R} \qquad Q \xRightarrow{\overline{a}(vx)} Q' \qquad x \sharp P}{\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R}}$$

$$\frac{\bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S} \qquad \mathit{eqvt}\ \mathscr{R} \qquad (P, Q) \in \mathscr{R} \qquad Q \xRightarrow{\widehat{\alpha}} Q'}{\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. The second case, where a free transition is mimicked, is proved in the same way as Lemma 9.19, but for the first case the extra requirement that $x \sharp P$ must be handled.

From the definition of $Q \xRightarrow{\overline{a}(vx)} Q'$, we obtain a Q'' and a Q''' s.t. $Q \Longrightarrow Q'''$, $Q''' \xrightarrow{a(vx)} Q''$ and $Q'' \Longrightarrow Q'$. From $Q \Longrightarrow Q'''$ and the assumptions, we use Lemma 15.24 to obtain a P''' s.t. $P \Longrightarrow P'''$ and $(P''', Q''') \in \mathscr{R}$. From $P \Longrightarrow P'''$, and $x \sharp P$, we get that $x \sharp P'''$, using Lemma 15.4. From $(P''', Q''') \in \mathscr{R}$ and the assumptions we get that $P''' \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q'''$, and with $Q''' \xrightarrow{a(vx)} Q''$ and $x \sharp P'''$ we obtain a P'' s.t. $P''' \xRightarrow{\overline{a}(vx)} P''$ and $(P'', Q'') \in \mathscr{R}$, using Lemma 15.13. From $(P'', Q'') \in \mathscr{R}$, $Q'' \Longrightarrow Q'$ and the assumptions we obtain a P' s.t. $P'' \Longrightarrow P'$ and $(P', Q') \in \mathscr{R}$, again using Lemma 15.24. Finally, we append $P \Longrightarrow P'''$, $P''' \xRightarrow{\overline{a}(vx)} P''$ and $P'' \Longrightarrow P'$ to $P \xRightarrow{\overline{a}(vx)} P'$ and solve the goal by instantiating the existential quantifier to P'.

□

The transitivity result for weak simulations is:

Lemma 15.26.

$$\frac{\begin{array}{lll} Q \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} R & \mathit{eqvt}\ \mathscr{R} & \\ \mathit{eqvt}\ \mathscr{R}'' & \mathscr{R} \circ \mathscr{R}' \subseteq \mathscr{R}'' & \bigwedge S\, T.\ \dfrac{(S, T) \in \mathscr{R}}{S \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} T} \qquad (P, Q) \in \mathscr{R} \end{array}}{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}''} R}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$ and Lemma 15.25 to simulate the weak transitions. □

We can now prove that weak bisimulation is an equivalence relation.

Lemma 15.27. *Weak bisimulation is an equivalence relation*

Proof.

Reflexivity: $P \mathrel{\dot{\approx}} P$

Follows by coinduction and setting $\mathscr{X}$ to *Id* and Lemma 15.23.

Symmetry: *If* $P \dot{\approx} Q$ *then* $Q \dot{\approx} P$.

Follows immediately from the definition of $\dot{\approx}$

Transitivity: *If* $P \dot{\approx} Q$ *and* $Q \dot{\approx} R$ *then* $P \dot{\approx} R$.

Follows by coinduction and setting $\mathscr{X}$ to $\dot{\approx} \circ \dot{\approx}$, Lemma 15.26 equivariance of weak bisimilarity.

□

We move on to the remaining preservation properties for weak simulation.

15.5 Preservation properties

Weak bisimilarity is preserved by all operators except Input and Sum.

15.5.1 Output and Tau

The proofs that bisimilarity is preserved by Output and Tau follow their counterparts for strong bisimilarity very closely – in particular, the requisites on the simulation relations are identical.

Lemma 15.28.

$$\frac{(P, Q) \in \mathscr{R}}{\overline{a}b.P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} \overline{a}b.Q}$$

Proof. The only action that $\overline{a}b.P$ can perform is an output action, which $\overline{a}b.Q$ can mimic with a weak action using the OUTPUT-rule from the lifted semantics in Figure 15.2. The derivatives P and Q are in $\mathscr{R}$ by the assumption. □

Lemma 15.29. *If* $P \dot{\approx} Q$ *then* $\overline{a}b.P \dot{\approx} \overline{a}b.Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{(\overline{a}b.P, \overline{a}b.Q), (\overline{a}b.Q, \overline{a}b.P)\},$$

and Lemma 15.28. □

Lemma 15.30. .

$$\frac{(P, Q) \in \mathscr{R}}{\tau.P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} \tau.Q}$$

Proof. The only action that $\tau.P$ can perform is an output action, which $\tau.Q$ can mimic with a weak action using the TAU-rule from the lifted semantics in Figure 15.2. The derivatives P and Q are in $\mathscr{R}$ by the assumption. □

Lemma 15.31. *If $P \dot{\approx} Q$ then $\tau.P \dot{\approx} \tau.Q$.*

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{(\tau.P, \tau.Q), (\tau.Q, \tau.P)\},$$

and Lemma 15.30. □

15.5.2 Match and Mismatch

The preservation properties for the Match and Mismatch are a bit more complicated than for their strong counterparts. The reason for this is that the simulating agent has the possibility of doing nothing when mimicking a τ-action. As the lifted semantics rule in Figure 15.2 dictate, the MATCH and MISMATCH rules may only be used if an action is actually performed. This puts special constraints on the candidate relations for bisimulation, which is demonstrated by the simulation lemmas.

Lemma 15.32.

$$\frac{P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge R\, S\, c.\ \dfrac{(R, S) \in \mathscr{R}}{([c{=}c]R, S) \in \mathscr{R}}}{[a{=}b]P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}'}} [a{=}b]Q}$$

Proof. The relevant case is where $[a{=}b]Q$ does a free action, in which case the only possible antecedent, according to the MATCH inversion rule in Figure 13.4.2, is if $a = b$ and $Q \xrightarrow{\alpha} Q'$. Since $P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q$, we obtain a P' such that $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$ and $(P', Q') \in \mathscr{R}$. The proof is then done by case analysis on $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$.

- $P \stackrel{\alpha}{\Longrightarrow} P'$: Since $a = b$ we have by the MATCHF-rule in Figure 15.2, that $[a{=}b]P \stackrel{\alpha}{\Longrightarrow} P'$. Finally, from $(P', Q') \in \mathscr{R}$ and $\mathscr{R} \subseteq \mathscr{R}'$, we have that $(P', Q') \in \mathscr{R}'$
- $P = P'$ and $\alpha = \tau$: Since the agent P does nothing, the only option available to $[a{=}b]P$ is to do the same. Therefore, the agents $[a{=}b]P$ and Q' must be in $\mathscr{R}$. This follows from the assumptions since $(P', Q') \in \mathscr{R}$ and $P = P'$.

In the case where the agent does a bound output, the proof follows the structure of the first case. □

Lemma 15.33. *If $P \dot{\approx} Q$ then $[a{=}b]P \dot{\approx} [a{=}b]Q$.*

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{([a{=}b]P, [a{=}b]Q), ([a{=}b]Q, [a{=}b]P)\},$$

and Lemma 15.32. The requirement of the candidate relation when the mimicking agent does nothing requires a structural congruence law, namely that $[a{=}a]P \dot{\approx} P$, which will be proved in Chapter 20. □

The proof that weak simulation is preserved by Mismatch follows the same structure.

Lemma 15.34.

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge R\, S\, c\, d.\ \dfrac{(R, S) \in \mathscr{R} \qquad c \neq d}{([c{\neq}d]R, S) \in \mathscr{R}}}{[a{\neq}b]P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} [a{\neq}b]Q}$$

Proof. The proof strategy is the same as for Lemma 15.32, but with the MISMATCH inversion rule from Figure 13.4.2 used to derive the case needed to be simulated, and the MISMATCH rules from Figure 15.2 to derive the mimicking weak actions. □

Lemma 15.35. *If $P \dot{\approx} Q$ then $[a{\neq}b]P \dot{\approx} [a{\neq}b]Q$.*

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{([a{\neq}b]P, [a{\neq}b]Q), ([a{\neq}b]Q, [a{\neq}b]P)\},$$

and Lemma 15.34. The requirement of the candidate relation when the mimicking agent does nothing requires a lemma stating that $[a{\neq}b]P \dot{\approx} P$ if $a \neq b$. This is not a structural congruence law, as it is not preserved by substitution, but it will be proven in Chapter 20. □

15.5.3 Restriction

The proof that weak simulation is preserved by Restriction does not require any extra assumptions of the candidate relation, as the SCOPEF-rule is lifted to a weak semantics.

Lemma 15.36. .

$$\frac{\begin{array}{l} P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \\ \bigwedge R\, S\, y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}'} \end{array} \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \mathit{eqvt}\ \mathscr{R} \qquad \mathit{eqvt}\ \mathscr{R}'}{(\nu x)P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$, the SCOPE inversion rule and the OPEN, SCOPEF and SCOPEB-rules from the lifted operational semantics. The assumption $\bigwedge R\ S\ y.\ (R, S) \in \mathscr{R} \Longrightarrow ((\nu y)R, (\nu y)S) \in \mathscr{R}'$ is used in the OPEN-case, as the restricted names are dropped from the derivatives. □

Lemma 15.37. *If* $P \mathrel{\dot{\approx}} Q$ *then* $(\nu x)P \mathrel{\dot{\approx}} (\nu x)Q$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{((\nu x)P, (\nu x)Q) : P \mathrel{\dot{\approx}} Q\},$$

and Lemma 15.36. □

15.5.4 Parallel

The semantic rules for parallel as also lifted to a weak counterpart. This lemma too follows its strong counterpart very closely.

Lemma 15.38. .

$$\frac{P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge S\ T\ U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U\ ,\ T \mid U) \in \mathscr{R}'} \qquad \bigwedge S\ T\ x.\ \dfrac{(S, T) \in \mathscr{R}'}{((\nu x)S, (\nu x)T) \in \mathscr{R}'}}{P \mid R \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}'}} Q \mid R}$$

Proof. The Isabelle proof can be found in Figure 15.4. □

We also have to prove that weak simulations are preserved by binding sequences, as was discussed in Section 14.3.5.

Lemma 15.39.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge R\ S\ y.\ \dfrac{(R, S) \in \mathscr{R}}{((\nu y)R, (\nu y)S) \in \mathscr{R}} \qquad P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q}{(\nu \tilde{y})P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} (\nu \tilde{y})Q}$$

Proof. By induction on $\tilde{y}$ and Lemma 15.36. □

Lemma 15.40. *If* $P \mathrel{\dot{\approx}} Q$ *then* $P \mid R \mathrel{\dot{\approx}} Q \mid R$.

Proof. Follows by coinduction and setting $\mathscr{X}$ to

$$\{((\nu x)(P \mid R), (\nu x)(Q \mid R)) : P \mathrel{\dot{\approx}} Q\},$$

and lemmas 15.38 and 15.39. The Isabelle proof can be found in Figure 15.5 □

Figure 15.4: The Isabelle proof that weak simulation is preserved by Parallel

lemma parCompose:
 fixes P :: pi **and** Q :: pi **and** R :: pi **and** T :: pi
 and $\mathscr{R}$ :: (pi × pi) set **and** $\mathscr{R}'$ :: (pi × pi) set **and** $\mathscr{R}''$ :: (pi × pi) set
 assumes P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q **and** (P, Q) ∈ $\mathscr{R}$
 and R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T **and** (R, T) ∈ $\mathscr{R}'$
 and C1: $\bigwedge$P′ Q′ R′ T′. ⟦(P′, Q′) ∈ $\mathscr{R}$; (R′, T′) ∈ $\mathscr{R}'$⟧ ⟹ (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$
 and C2: $\bigwedge$P′ Q′ x. (P′, Q′) ∈ $\mathscr{R}''$ ⟹ ((νx)P′, (νx)Q′) ∈ $\mathscr{R}''$
 shows P | R $\widehat{\rightsquigarrow}_{\mathscr{R}''}$ Q | T
proof(induct rule: weakSimI) — *Apply introduction rule* $\widehat{\rightsquigarrow}$-I
 case(Bound U a x)
 from ‹x ♯ P | R› **have** x ♯ P **and** x ♯ R **by** simp+
 from ‹Q | T $\xrightarrow{\overline{a}(\nu x)}$ U›
 show ∃S. P | R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ S ∧ (S, U) ∈ $\mathscr{R}''$
 proof(induct rule: parCasesB) — *Apply* PARB *inversion rule from Figure 13.4.2*

PAR1 *case*
Given that $Q \xrightarrow{a(\nu x)} Q'$ *prove that there exists an S such that* $P \mid R \overset{\overline{a}(\nu x)}{\Longrightarrow} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

  case(cPar1 Q′)
  from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{a(\nu x)}$ Q′› ‹x ♯ P›
  obtain P′ **where** P $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: weakSimE)
  from ‹P $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ P′› ‹x ♯ R› **have** P | R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ P′ | R **by**(rule Par1B)
  moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R, T) ∈ $\mathscr{R}'$› **have** (P′ | R, Q′ | T) ∈ $\mathscr{R}''$
   by(rule C1)
  ultimately show ∃S. P | R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ S ∧ (S, Q′ | T) ∈ $\mathscr{R}''$ **by** blast
 next

PAR2 case
Given that $T \xrightarrow{a(\nu x)} T'$ prove that there exists an *S* such that $P \mid R \overset{\overline{a}(\nu x)}{\Longrightarrow} S$ and $(S, Q \mid T') \in \mathscr{R}''$.

  case(cPar2 T′)
  from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{a(\nu x)}$ T′› ‹x ♯ R›
  obtain R′ **where** R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: weakSimE)
  from ‹R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ R′› ‹x ♯ P› **have** P | R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ P | R′ **by**(rule Par2B)
  moreover from ‹(P, Q) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$› **have** (P | R′, Q | T′) ∈ $\mathscr{R}''$
   by(rule C1)
  ultimately show ∃S. P | R $\overset{\overline{a}(\nu x)}{\Longrightarrow}$ S ∧ (S, Q | T′) ∈ $\mathscr{R}''$ **by** blast
 qed

next
case(Free U α)
from ‹Q | T $\xrightarrow{\alpha}$ U›
show $\exists$S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S $\wedge$ (S, U) $\in \mathscr{R}''$
proof(induct rule: parCasesF[**where** C = (P, R)]) — *Apply* ParF *inversion rule*

ParF1 *case*
Given that $Q \xrightarrow{\alpha} Q'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\alpha}}{\Longrightarrow} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(cPar1 Q′)
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\alpha}$ Q′› **obtain** P′ **where** P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′ **and** (P′, Q′) $\in \mathscr{R}$ **by**(blast dest: weakSimE)
from ‹P $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P′›**have** P | R $\Longrightarrow^{\wedge}$ $\alpha \prec$ P′ | R **by**(rule Par1F)
moreover from ‹(P′, Q′) $\in \mathscr{R}$› ‹(R, T) $\in \mathscr{R}$′› **have** (P′ | R, Q′ | T) $\in \mathscr{R}''$ **by**(rule C1)
ultimately show $\exists$S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S $\wedge$ (S, Q′ | T) $\in \mathscr{R}''$ **by** blast
next

ParF2 *case*
Given that $T \xrightarrow{\alpha} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\alpha}}{\Longrightarrow} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T′)
with ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› **obtain** R′ **where** R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′ **and** (R′, T′) $\in \mathscr{R}'$ **by**(blast dest: weakSimE)
from ‹R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′› **have** P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P | R′ **by**(rule Par2F)
moreover from ‹(P, Q) $\in \mathscr{R}$› ‹(R′, T′) $\in \mathscr{R}$′› **have** (P | R′, Q | T′) $\in \mathscr{R}''$ **by**(rule C1)
ultimately show $\exists$S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S $\wedge$ (S, Q | T′) $\in \mathscr{R}''$ **by** blast

next

COMM1 *case*
Given that $Q \xrightarrow{ab} Q'$ *and* $T \xrightarrow{a\,[b]} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\hat{\tau}}{\Longrightarrow} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm1 Q′ T′ a b)
from ⟨P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q⟩ ⟨Q $\xrightarrow{ab}$ Q′⟩ **obtain** P′ **where** P $\stackrel{ab}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ⟨R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{\overline{a}b}$ T′⟩ **obtain** R′ **where** R $\stackrel{\overline{a}b}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ⟨P $\stackrel{ab}{\Longrightarrow}$ P′⟩ ⟨R $\stackrel{\overline{a}b}{\Longrightarrow}$ R′⟩ **have** P | R $\stackrel{\tau}{\Longrightarrow}$ P′ | R′ **by**(rule Comm1)
hence P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ P′ | R′
by(rule weakTransitionI)
moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R′, T′) ∈ $\mathscr{R}'$⟩ **have** (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ S ∧ (S, Q′ | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM2 *case*
Given that $Q \xrightarrow{a\,[b]} Q'$ *and* $T \xrightarrow{ab} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\hat{\tau}}{\Longrightarrow} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm2 Q′ T′ a b)
from ⟨P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q⟩ ⟨Q $\xrightarrow{\overline{a}b}$ Q′⟩ **obtain** P′ **where** P $\stackrel{\overline{a}b}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ⟨R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{ab}$ T′⟩ **obtain** R′ **where** R $\stackrel{ab}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ⟨P $\stackrel{\overline{a}b}{\Longrightarrow}$ P′⟩ ⟨R $\stackrel{ab}{\Longrightarrow}$ R′⟩ **have** P | R $\stackrel{\tau}{\Longrightarrow}$ P′ | R′ **by**(rule Comm2)
hence P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ P′ | R′
by(rule weakTransitionI)
moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R′, T′) ∈ $\mathscr{R}'$⟩ **have** (P′ | R′, Q′ | T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ S ∧ (S, Q′ | T′) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE1 *case*
Given that $Q \xrightarrow{ax} Q'$ *and* $T \xrightarrow{a(vx)} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\hat{\tau}}{\Longrightarrow} S$ *and* $(S, (\nu x)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose1 Q′ T′ a x)
from ‹x ♯ (P, R)› **have** x ♯ P **and** x ♯ R **by** auto
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{ax}$ Q'›
obtain P′ **where** P $\stackrel{ax}{\Longrightarrow}$ P′ **and** (P′, Q') ∈ $\mathscr{R}$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}(vx)}$ T'› ‹x ♯ R›
obtain R′ **where** R $\stackrel{\overline{a}(vx)}{\Longrightarrow}$ R′ **and** (R′, T') ∈ $\mathscr{R}'$ **by**(blast dest: weakSimE)
from ‹P $\stackrel{ax}{\Longrightarrow}$ P'› ‹R $\stackrel{\overline{a}(vx)}{\Longrightarrow}$ R'› ‹x ♯ P› **have** P | R $\stackrel{\tau}{\Longrightarrow}$ (νx)(P′ | R')
by(rule Close1)
hence P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ (νx)(P′ | R') **by**(rule weakTransitionI)
moreover from ‹(P′, Q') ∈ $\mathscr{R}$› ‹(R′, T') ∈ $\mathscr{R}'$›
have ((νx)(P′ | R'), (νx)(Q′ | T')) ∈ $\mathscr{R}''$ **by**(blast intro: C1 C2)
ultimately show ∃S. P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ S ∧ (S, (νx)(Q′ | T')) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE2 *case*
Given that $Q \xrightarrow{a(vx)} Q'$ *and* $T \xrightarrow{ax} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\hat{\tau}}{\Longrightarrow} S$ *and* $(S, (\nu x)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose2 Q′ T′ a x)
from ‹x ♯ (P, R)› **have** x ♯ R **and** x ♯ P **by** auto
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}(vx)}$ Q'› ‹x ♯ P›
obtain P′ **where** P $\stackrel{\overline{a}(vx)}{\Longrightarrow}$ P′ **and** (P′, Q') ∈ $\mathscr{R}$ **by**(blast dest: weakSimE)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{ax}$ T'› **obtain** R′ **where** R $\stackrel{ax}{\Longrightarrow}$ R′ **and** (R′, T') ∈ $\mathscr{R}'$
by(fastsimp dest: weakSimE simp add: weakFreeTransition-def)
from ‹P $\stackrel{\overline{a}(vx)}{\Longrightarrow}$ P'› ‹R $\stackrel{ax}{\Longrightarrow}$ R'› ‹x ♯ R› **have** P | R $\stackrel{\tau}{\Longrightarrow}$ (νx)(P′ | R')
by(rule Close2)
hence P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ (νx)(P′ | R') **by**(rule weakTransitionI)
moreover from ‹(P′, Q') ∈ $\mathscr{R}$› ‹(R′, T') ∈ $\mathscr{R}'$›
have ((νx)(P′ | R'), (νx)(Q′ | T')) ∈ $\mathscr{R}''$ **by**(blast intro: C1 C2)
ultimately show ∃S. P | R $\stackrel{\hat{\tau}}{\Longrightarrow}$ S ∧ (S, (νx)(Q′ | T')) ∈ $\mathscr{R}''$ **by** blast
qed
qed

15.5.5 Replication

As for CCS, in order to prove that weak bisimilarity is preserved by Replication we need to use bisimulation up-to techniques. We derive the following coinduction law.

Lemma 15.41.

$$\frac{(P,\,Q) \in \mathscr{Y} \qquad \mathit{eqvt}\,\mathscr{Y} \qquad \bigwedge R\,S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{R \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx} \circ (\mathscr{Y} \cup \dot{\approx}) \circ \dot{\sim}} S} \qquad \bigwedge R\,S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{(S,\,R) \in \mathscr{Y}}}{P \mathrel{\dot{\approx}} Q}$$

Proof. By coinduction using Lemma 15.15 setting $\mathscr{X}$ to $\dot{\approx} \circ (\mathscr{Y} \cup \dot{\approx}) \circ \dot{\approx}$. □

This lemma is nearly identical to its counterpart for CCS, except that it requires the candidate relation to be equivariant.

Lemma 15.42.

$$\frac{(P,\,Q) \in \mathscr{R} \qquad \mathit{bangRel}\,\mathscr{R} \subseteq \mathscr{R}' \qquad \mathit{eqvt}\,\mathscr{R}' \qquad \bigwedge R\,S.\ \dfrac{(R,\,S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S} \qquad \bigwedge R\,S\,T\,U.\ \dfrac{(R,\,S) \in \mathscr{R} \qquad (T,\,U) \in \mathscr{R}'}{(R \mid T\,,\, S \mid U) \in \mathscr{R}'} \qquad \bigwedge R\,S\,x.\ \dfrac{(R,\,S) \in \mathscr{R}'}{((\nu x)R,\,(\nu x)S) \in \mathscr{R}'} \qquad \bigwedge R\,S.\ \dfrac{(R \mid !R\,,\, S) \in \mathscr{R}'}{(!R,\,S) \in \mathscr{R}'}}{!P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} !Q}$$

Proof. Similar to the corresponding proof for strong simulation, Lemma 14.31. However, in the replication case the REPL rule from Figure 15.2 can only be used when the mimicking agent mimics at least one τ-action, if it does nothing, the last assumption from the lemma is used to ensure that the derivatives remain in $\mathscr{R}'$. □

Lemma 15.43. *If $P \mathrel{\dot{\approx}} Q$ then $!P \mathrel{\dot{\approx}} !Q$.*

Proof. Follows the same pattern as the corresponding lemma for strong bisimilarity, Lemma 14.32, but by coinduction up-to weak bisimilarity with $\mathscr{Y}$ set to $\mathit{bangRel}\,\dot{\approx}$. The simulation case is discharged by Lemma 15.42, where its final requisite is proven by the fact that $!P \mathrel{\dot{\approx}} P \mid !P$, the proof of which is deferred to Chapter 20. □

With this lemma in place we can prove the main preservation theorem for weak bisimilarity.

Theorem 15.1. *Weak bisimilarity is preserved by all operators except Sum and Input.*

Proof. Follows from lemmas 15.29, 15.31, 15.37, 15.40, and 15.43. □

Figure 15.5: Isabelle proof of Lemma 15.40 which proves that weak bisimilarity is preserved by Parallel.

```
lemma weakBisimParPres:
  fixes P :: pi and Q :: pi and R :: pi
  assumes P ≈ Q
  shows P | R ≈ Q | R
proof −
  let ?X = {((νx̃)(P | R), (νx̃)(Q | R)) | x̃ P R Q. P ≈ Q}
  have BC: ⋀P Q. P | Q = (νε)(P | Q) by auto
  from ‹P ≈ Q› have (P | R, Q | R) ∈ ?X by(blast intro: BC)
  thus ?thesis
  proof(coinduct rule: weakBisimCoinduct)
    case(cSim S T)
    { fix P Q R x̃
      assume P ≈ Q
      moreover hence P ⌢→≳ Q by(rule weakBisimE)
      moreover have ⋀P Q R. P ≈ Q ⟹ (P | R, Q | R) ∈ ?X using BC
        by blast
      moreover {
        fix S T x
        assume (S, T) ∈ ?X
        then obtain x̃ P Q R where P ≈ Q
          and A: S = (νx̃)(P | R) and B: T = (νx̃)(Q | R) by auto
        from A have (νx)S = (νxx̃)(P | R) by auto
        moreover from B have (νx)T = (νxx̃)(Q | R) by auto
        ultimately have ((νx)S, (νx)T) ∈ ?X using ‹P ≈ Q› by blast }
      note Res = this
      moreover have eqvt weakBisim by simp
      ultimately have P | R ⌢→?X Q | R
        by(rule-tac weakEarlySimPres.parPres)
      moreover have eqvt ?X
        by(auto simp add: eqvt-def) (blast intro: eqvts)
      ultimately have (νx̃)(P | R) ⌢→?X (νx̃)(Q | R) using Res
        by(rule-tac weakEarlySimPres.resChainI)
      hence (νx̃)(P | R) ⌢→?X ∪ ≳ (νx̃)(Q | R) using Res
        by(force intro: weakEarlySim.monotonic)
    }
    with ‹(S, T) ∈ ?X› show S ⌢→?X ∪ ≳ T
      by blast
  next
    case(cSym S T)
    from ‹(S, T) ∈ ?X› show (T, S) ∈ ?X by(blast dest: weakBisimSymmetric)
  qed
qed
```

16. Weak congruence

Weak bisimilarity is not preserved by Sum nor by Input, for the same reasons as for weak bisimilarity for CCS and strong bisimilarity for the pi-calculus. To define a relation which is preserved by Sum we use the same technique as in Chapter 10 for CCS – we require that a τ-action is mimicked by at least one τ-action, disallowing that an agent mimics a τ-action by doing nothing. We call this bisimilarity τ-bisimilarity. A weak congruence is then obtained by closing τ-bisimilarity under substitutions in the same way as is done in Section 14.4 for strong equivalence.

16.1 τ-bisimilarity

As with CCS, we define a version of weak bisimilarity where agents mimicking τ-actions at least have to do one τ-action – they do not have the option of doing nothing.

Definition 16.1 (τ-simulation)**.** *An agent P τ-simulating an agent Q preserving $\mathscr{R}$ is denoted $P \rightsquigarrow_{\mathscr{R}} Q$.*
$P \rightsquigarrow_{\mathscr{R}} Q \equiv$
$(\forall Q'\, a\, x.\ Q \xrightarrow{a(vx)} Q' \longrightarrow x \sharp P \longrightarrow (\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R})) \wedge (\forall Q'$
$\alpha.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \xRightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}))$

Agents which are τ-bisimilar do not require their derivatives to be τ-bisimilar, but only weakly bisimilar. We define τ-bisimilarity in a similar way as weak congruence for CCS, as the conjunction of two τ-simulations.

Definition 16.2. *Two agents P and Q are τ-bisimilar, denoted $P \cong Q$, if they τ-simulate each other.*

$$P \cong Q \stackrel{\text{def}}{=} P \rightsquigarrow_{\dot{\approx}} Q \wedge Q \rightsquigarrow_{\dot{\approx}} P$$

16.1.1 Primitive inference rules

The introduction rule for τ-simulation follows the standard pattern, allowing the user to choose a freshness context $\mathscr{C}$ for which any bound names of the simulation actions must be fresh.

Lemma 16.3. *Introduction and elimination rules for τ-simulation.*

$$\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge a\ x\ Q'.\ \frac{x \sharp \mathscr{C} \qquad Q \xrightarrow{a(vx)} Q'}{\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R}}$$

$$\frac{\bigwedge \alpha\ Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}}{P \rightsquigarrow_{\mathscr{R}} Q}\ \rightsquigarrow\text{-I}$$

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{a(vx)} Q' \qquad x \sharp P}{\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R}}\ \rightsquigarrow\text{-E1}$$

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}\ \rightsquigarrow\text{-E2}$$

Proof. Follows from Definition 16.1. For the introduction rule, Lemma 15.7 is used to alpha-convert the bound name x such that it is fresh for $\mathscr{C}$. □

In order for two agents to be τ-bisimilar, they have to τ-simulate each other. We derive the following introduction rules, which are similar to the ones created for weak congruence for CCS.

Lemma 16.4.

$$\frac{\mathit{Prop}\ P\ Q \qquad \bigwedge R\ S.\ \dfrac{\mathit{Prop}\ R\ S}{\mathit{Prop}\ S\ R} \qquad \bigwedge R\ S.\ \dfrac{\mathit{Prop}\ R\ S}{F\ R \rightsquigarrow_{\dot{\approx}} F\ S}}{F\ P \cong F\ Q}\ \cong\text{-I}$$

$$\frac{P \cong Q \qquad \bigwedge R\ S.\ \dfrac{R \cong S}{F\ R \rightsquigarrow_{\dot{\approx}} F\ S}}{F\ P \cong F\ Q}\ \cong\text{-I2}$$

Proof. Follows directly from Definition 16.2. □

We also derive the elimination rules.

Lemma 16.5.

$$\frac{P \cong Q}{P \rightsquigarrow_{\dot{\approx}} Q}\cong\!-E1 \qquad\qquad \frac{P \cong Q}{Q \rightsquigarrow_{\dot{\approx}} P}\cong\!-E2$$

16.1.2 τ-bisimilarity includes strong bisimilarity

Lemma 16.6. *If* $P \xrightarrow{\alpha} P'$ *then* $P \stackrel{\alpha}{\Longrightarrow} P'$.

Proof. Follows from Definitions 15.5 by using empty τ-chains. □

Lemma 16.7. *If* $P \hookrightarrow_{\mathscr{R}} Q$ *then* $P \rightsquigarrow_{\mathscr{R}} Q$.

Proof. Follows from Definitions 14.1, 16.1, and Lemma 16.6. □

We additionally require that τ-simulation is monotonic.

Lemma 16.8. *If* $P \rightsquigarrow_{\mathscr{R}} P'$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $P \rightsquigarrow_{\mathscr{R}'} P'$.

Proof. Follows from the Definition 16.1. □

We can now prove that all strongly bisimilar agents are also τ-bisimilar.

Lemma 16.9. *If* $P \dot{\sim} Q$ *then* $P \cong Q$.

Proof. We use the symmetric introduction rule $\cong$-I with *Prop* set to $\dot{\sim}$ and F set to the identity function. *Prop* is symmetric, since bisimilarity is symmetric. The simulation case is proven in the following way:

- From $P \dot{\sim} Q$ we have that $P \hookrightarrow_{\dot{\sim}} Q$ by $\dot{\sim}$-E1.
- Hence $P \rightsquigarrow_{\dot{\sim}} Q$ by Lemma 16.7.
- Hence $P \rightsquigarrow_{\dot{\approx}} Q$ by Lemmas 15.22 and 16.8.

□

16.1.3 Weak bisimilarity includes τ-bisimilarity

Lemma 16.10. *If* $P \rightsquigarrow_{\mathscr{R}} Q$ *then* $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$.

Proof. Follows directly from Definitions 15.9, 16.1, and 15.8. □

Lemma 16.11. *If* $P \cong Q$ *then* $P \dot{\approx} Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(P, Q) : P \cong Q\}$. The relation is symmetric, and the simulation case is proved in the following way:

- from $P \cong Q$ we have that $P \rightsquigarrow_{\dot{\approx}} Q$ by $\cong$-E1.
- Hence $P \rightsquigarrow_{\mathscr{X} \cup \dot{\approx}} Q$ by Lemma 16.8.
- Hence $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} Q$ by Lemma 16.10.

□

16.2 τ-bisimilarity is an equivalence relation

Lemma 16.12.

$$\textit{If } Id \subseteq \mathscr{R} \textit{ then } P \rightsquigarrow_{\mathscr{R}} P.$$

Proof. Follows from $\rightsquigarrow$-I. Any transition $P \xrightarrow{\alpha} P'$ can be mimicked by a weak transition $P \stackrel{\alpha}{\Longrightarrow} P'$, and $(P', P') \in \mathscr{R}$. □

τ-bisimilarity is symmetric. To prove it is transitive, we must first prove two auxiliary lemmas.

Lemma 16.13.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{R}}{R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S} \qquad \textit{eqvt } \mathscr{R} \qquad (P, Q) \in \mathscr{R} \qquad Q \stackrel{\alpha}{\Longrightarrow} Q'}{\exists P'.\ P \stackrel{\alpha}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}}$$

Proof. Proved in a similar way as Lemma 10.10. □

Lemma 16.14.

$$\frac{\begin{array}{lll} P \rightsquigarrow_{\mathscr{R}} Q & Q \rightsquigarrow_{\mathscr{R}} R & \textit{eqvt } \mathscr{R} \\ \textit{eqvt } \mathscr{R}'' & \mathscr{R} \circ \mathscr{R} \subseteq \mathscr{R}'' & \bigwedge S\, T.\ \dfrac{(S, T) \in \mathscr{R}}{S \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} T} \qquad (P, Q) \in \mathscr{R} \end{array}}{P \rightsquigarrow_{\mathscr{R}''} R}$$

Proof. Follows from $\rightsquigarrow$-I, and lemmas 15.25 and 16.13 to simulate the weak actions. □

Lemma 16.15. *τ-bisimulation is an equivalence relation*

Proof. **Reflexivity:** Follows from the definition of $\cong$, reflexivity of weak bisimilarity (Lemma 15.27), and Lemma 16.12.

Symmetry: τ-bisimilarity is symmetric by definition.

Transitivity: The symmetric introduction rule $\cong$-I is instantiated with *Prop* set to $\lambda P\, R.\ \exists Q.\ P \cong Q \wedge Q \cong R$.

Symmetry case: Follows immediately since τ-bisimilarity is symmetric.

Simulation case: From *Prop P R* we obtain a Q such that $P \cong Q$ and $Q \cong R$. Hence $P \rightsquigarrow_{\dot{\approx}} Q$ and $Q \rightsquigarrow_{\dot{\approx}} R$ by $\cong$-E2. Moreover, since weak bisimilarity is transitive (Lemma 15.27) and equivariant we have that $\dot{\approx} \circ \dot{\approx} \subseteq \dot{\approx}$, and hence by Lemma 16.14 that $P \rightsquigarrow_{\dot{\approx}} R$.

□

16.3 Preservation properties

In this section we prove that τ-bisimilarity is preserved by all operators except Input.

16.3.1 Output and Tau

Lemma 16.16.

$$\textit{If } (P, Q) \in \mathscr{R} \textit{ then } \overline{a}b.P \rightsquigarrow_{\mathscr{R}} \overline{a}b.Q.$$

Proof. Follows from $\rightsquigarrow$-I and the fact that $\overline{a}b.P$ and $\overline{a}b.Q$ can each only do an $\overline{a}b$-action, and $(P, Q) \in \mathscr{R}$. □

Lemma 16.17. *If $P \cong Q$ then $\overline{a}b.P \cong \overline{a}b.Q$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda R.\, \overline{a}b.R$. Since $P \cong Q$ we know that $P \rightsquigarrow_{\dot{\approx}} Q$ by $\cong$-E1, and that $P \dot{\approx} Q$ by Lemma 16.11. The simulation can then be proved using Lemma 16.16. □

Lemma 16.18.

$$\textit{If } (P, Q) \in \mathscr{R} \textit{ then } \tau.P \rightsquigarrow_{\mathscr{R}} \tau.Q.$$

Proof. Follows from $\rightsquigarrow$-I and the fact that $\tau.P$ and $\tau.Q$ can each only do a τ-action and $(P, Q) \in \mathscr{R}$. □

Lemma 16.19. *If $P \cong Q$ then $\tau.P \cong \tau.Q$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda R.\, \tau.R$. Since $P \cong Q$ we know that $P \rightsquigarrow_{\dot{\approx}} Q$ by $\cong$-E1, and that $P \dot{\approx} Q$ by Lemma 16.11. The simulation can then be proved using Lemma 16.16. □

16.3.2 Match and Mismatch

The proofs that τ-bisimilarity is preserved by Match and Mismatch are simpler than the ones for weak bisimilarity – the reason for this is that we disallow agents to mimic τ-actions by doing nothing. These lemmas therefore correspond more closely to their counterparts for strong bisimilarity.

Lemma 16.20.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}'}{[a{=}b]P \rightsquigarrow_{\mathscr{R}'} [a{=}b]Q}$$

Proof. Follows from $\rightsquigarrow$-I, the MATCH inversion rule from Figure 13.4.2, and the MATCHF and MATCHB rules from the weak operational semantics from Figure 15.3. The fact that $\mathscr{R} \subseteq \mathscr{R}'$ ensures that the derivatives of P and Q are in $\mathscr{R}'$. □

Lemma 16.21. *If $P \cong Q$ then $[a{=}b]P \cong [a{=}b]Q$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda R.\ [a{=}b]R$. Since $P \cong Q$ we have that $P \rightsquigarrow_{\dot{\approx}} Q$, and hence that $[a{=}b]P \rightsquigarrow_{\dot{\approx}} [a{=}b]Q$ by Lemma 16.20. □

Lemma 16.22.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}'}{[a{\neq}b]P \rightsquigarrow_{\mathscr{R}'} [a{\neq}b]Q}$$

Proof. Follows from $\rightsquigarrow$-I, the MISMATCH inversion rule from Figure 13.4.2, and the MISMATCHF and MISMATCHB rules from the weak operational semantics from Figure 15.3. The fact that $\mathscr{R} \subseteq \mathscr{R}'$ ensures that the derivatives of P and Q are in $\mathscr{R}'$. □

Lemma 16.23. *If $P \cong Q$ then $[a{\neq}b]P \cong [a{\neq}b]Q$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda R.\ [a{\neq}b]R$. Since $P \cong Q$ we have that $P \rightsquigarrow_{\dot{\approx}} Q$, and hence that $[a{\neq}b]P \rightsquigarrow_{\dot{\approx}} [a{\neq}b]Q$ by Lemma 16.22. □

16.3.3 Sum

We know that τ-bisimilarity is preserved by Sum, as an agent must mimic a τ-action with at least one τ-action.

Lemma 16.24.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad Id \subseteq \mathscr{R}'}{P + R \rightsquigarrow_{\mathscr{R}'} Q + R}$$

Proof. Follows from $\rightsquigarrow$-I, the SUM inversion rule from Figure 13.4.2 and the SUM1 and SUM2-rules from the weak operational semantics from Figure 15.3. In the case where R does a transition, the assumption $\mathscr{R} \subseteq \mathscr{R}'$ is used to ensure that the derivatives remain in $\mathscr{R}'$. □

Lemma 16.25. *If $P \cong Q$ then $P + R \cong Q + R$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda T.T{+}R$. Since $P \cong Q$ we know that $P \rightsquigarrow_{\dot{\approx}} Q$ by $\cong$-E1. The simulation is then proved using Lemma 16.24. □

16.3.4 Restriction

Lemma 16.26.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad \bigwedge R\,S\,x.\ \dfrac{(R,\,S) \in \mathscr{R}}{((\nu x)R,\,(\nu x)S) \in \mathscr{R}'} \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}'}{(\nu x)P \rightsquigarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from $\rightsquigarrow$-I, the SCOPE inversion rule from Figure 13.4.2 and the SCOPE and OPEN-rules from the weak operational semantics from Figure 15.3. In the case where Q opens the bound name x, the assumption $\mathscr{R} \subseteq \mathscr{R}'$ is used to ensure that the derivatives remain in $\mathscr{R}'$. □

Lemma 16.27. *If $P \cong Q$ then $(\nu x)P \cong (\nu x)Q$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda R.((\nu x))R$. Since $P \cong Q$ we know that $P \rightsquigarrow_{\dot{\approx}} Q$ by $\cong$-E1. The simulation is then proved using Lemma 16.26. □

16.3.5 Parallel

Lemma 16.28.

$$\frac{P \rightsquigarrow_{\mathscr{R}} Q \qquad (P, Q) \in \mathscr{R} \qquad \bigwedge S\,T\,U.\ \dfrac{(S, T) \in \mathscr{R}}{(S \mid U\ ,\ T \mid U) \in \mathscr{R}'} \qquad \bigwedge S\,T\,x.\ \dfrac{(S, T) \in \mathscr{R}'}{((\nu x)S, (\nu x)T) \in \mathscr{R}'}}{P \mid R \rightsquigarrow_{\mathscr{R}'} Q \mid R}$$

Proof. Follows from $\rightsquigarrow$-I, the PAR inversion rule from Figure 13.4.2 and the PAR, COMM, and CLOSE-rules from the weak operational semantics from Figure 15.3. The assumptions of the lemma ensure that any derivatives of P and Q are in the relation $\mathscr{R}'$. The Isabelle proof is very similar to the one found in Figure 15.4 modulo which lifted semantic rules are used. □

Lemma 16.29. *If $P \cong Q$ then $P \mid R \cong Q \mid R$.*

Proof. Follows from $\cong$-I2 with F set to $\lambda T.T \mid R$. Since $P \cong Q$ we know that $P \rightsquigarrow_{\dot{\approx}} Q$ and $P \mathrel{\dot{\approx}} Q$, by $\cong$-E1 and Lemma 16.11. The simulation is then be proved using Lemma 16.28, where Lemmas 15.40 and 15.37 proves the constraints required of $\mathscr{R}$ and $\mathscr{R}'$. □

16.3.6 Replication

Lemma 16.30.

$$\frac{(P, Q) \in \mathscr{R} \qquad \bigwedge R\,S.\ \dfrac{(R, S) \in \mathscr{R}}{R \rightsquigarrow_{\mathscr{R}'} S} \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad eqvt\ \mathscr{R}'}{!P \rightsquigarrow_{bangRel\ \mathscr{R}'} !Q}$$

Proof. Follows the same structure as its counterpart for strong bisimilarity, Lemma 7.20, but with the lifted semantics from Figure 15.2. □

As for CCS, we need a few auxiliary lemmas to prove that τ-bisimilarity is preserved by Replication.

Lemma 16.31. *$bangRel \dot{\approx} \subseteq \dot{\approx}$*

Proof. By induction on $bangRel \dot{\approx}$.

Parallel case: We have that $R \dot{\approx} T$, and from the induction hypothesis that $P \dot{\approx} Q$, and we must prove that $R \mid P \dot{\approx} T \mid Q$, which follows from the preservation properties of weak bisimilarity, and the structural congruence laws (which will be proven in Chapter 18).

Restriction case: From $P \dot{\approx} Q$ we have to prove that $(\nu x)P \dot{\approx} (\nu x)Q$, which follows from Lemma 15.37.

Replication case: From $P \dot{\approx} Q$ we have to prove that $!P \dot{\approx} !Q$ which follows directly from Lemma 15.43.

□

Lemma 16.32. *If $P \cong Q$ then $!P \cong !Q$.*

Proof. The proof uses the symmetric introduction rule $\cong$-I2. From $P \cong Q$, the definition of $\rightsquigarrow$, and Lemma 16.11. We have that $!P \rightsquigarrow_{bangRel \dot{\approx}} !Q$ by Lemma 16.30. We can then prove that $!P \rightsquigarrow_{\dot{\approx}} !Q$ with Lemmas 16.8 and 16.31. □

We can now prove the main preservation theorem for τ-bisimilarity.

Theorem 16.1. *τ-bisimilarity is preserved by all operators except Input.*

Proof. Follows from lemmas 16.17, 16.19, 16.21, 16.23, 16.25, 16.27, 16.29, and 16.32. □

16.4 Weak congruence

As for strong equivalence, we obtain a congruence by closing τ-bisimilarity under all possible substitutions.

Definition 16.33. *Weak congruence, denoted $\approx$ is obtained by closing τ-bisimilarity under substitutions.*

$$P \approx Q \stackrel{\text{def}}{=} \forall \sigma . P\sigma \cong Q\sigma$$

Weak congruence is, as the name suggests, a congruence. The proof that it is preserved by all operators except Input follows directly from their corresponding lemmas for τ-bisimilarity, by distributing the substitutions over the operators of the agents. To prove that weak congruence is preserved by Input, we use the same techniques as for strong bisimilarity in Section 14.4.

16.4.1 Input

We must first prove that two agents $a(x).P$ and $a(x).Q$ are weakly bisimilar if the agents P and Q are weakly bisimilar for all possible substitutions. We begin with the simulation proof.

Lemma 16.34.

$$\frac{\forall\, y.\ (P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R} \qquad eqvt\ \mathscr{R}}{a(x).P\ \widehat{\rightsquigarrow}_{\mathscr{R}}\ a(x).Q}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$ and the fact that $a(x).P$ and $a(x).Q$ can each only do an input-action. The assumption $\forall\, y.\ (P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R}$ ensures that whatever input name the agents receive along a, the derivatives will still be in $\mathscr{R}$. □

Lemma 16.35. *If* $\forall\, y.\ P\{^y/_x\} \mathrel{\dot{\approx}} Q\{^y/_x\}$ *then* $a(x).P \mathrel{\dot{\approx}} a(x).Q$.

Proof. Follows from coinduction with $\mathscr{X}$ set to $\{(a(x).P,\ a(x).Q) : \forall\, y.\ P\{^y/_x\} \mathrel{\dot{\approx}} Q\{^y/_x\}\}$ and Lemma 16.34. □

In order to prove that weak congruence is preserved by Input, we must prove a corresponding lemma for Lemma 16.34 for τ-simulation as well.

Lemma 16.36.

$$\frac{\forall\, y.\ (P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R} \qquad eqvt\ \mathscr{R}}{a(x).P \rightsquigarrow_{\mathscr{R}}\ a(x).Q}$$

Proof. Similar to Lemma 16.34 as an input action cannot be mimicked by doing nothing. □

Lemma 16.37. *If* $P \approx Q$ *then* $a(x).P \approx a(x).Q$.

Proof. We must prove that $a(x).P\sigma \cong a(x).Q\sigma$ for all σ. We alpha-convert x to y such that y does not clash with σ, and push σ past the binder. Since $P \approx Q$, we know that $P\sigma\{^z/_y\} \cong Q\sigma\{^z/_y\}$ for all z, and hence that $P\sigma\{^z/_y\} \mathrel{\dot{\approx}} Q\sigma\{^z/_y\}$ by Lemma 16.11. The lemma is then proved using Lemma 16.36. □

16.4.2 Weak congruence is a congruence

We can now prove that weak congruence is a congruence.

Theorem 16.2. *Weak congruence is a congruence.*

Proof. That weak congruence is preserved by Input is proved by Lemma 16.37, that it is an equivalence relation and preserved by all other operators follows from Lemma 16.15, Theorem 16.1, and the definition of $\approx$, where any binders are alpha-converted not to clash with the substitution. □

17. Late operational semantics

In the original pi-calculus paper [58], the authors describe a *late operational semantics*. The difference between the two semantics lies in when an agent instantiates an input. As the names of the semantics hint at, the early semantics does this as early as possible, and the late one as late as possible, when inferring an action. More formally, the INPUT-rules of the semantics differ in the following way:

$$\text{Early INPUT}\quad \frac{}{a(x).P \xrightarrow{au} P\{^{u}/_{x}\}} \qquad \text{Late INPUT}\quad \frac{}{a(x).P \xrightarrow{a(x)} P}$$

In the early case, the agent $a(x).P$ can evolve to an infinite number of derivatives – one for every possible name u it can receive along a. In the late case, no substitution is done, but the bound name x moves up to the label of the transition. Hence, in the late semantics, the object of an input action is binding, and the bound name x on the label binds into the derivative in the same way as a bound output. An input label can hence be alpha-converted in the standard way. Moreover, the syntax differs in that the bound name is enclosed in parentheses in the late transitions, similarly to other binders.

The substitution in the late semantics is in the COMM and the CLOSE rules.

$$\text{Early COMM1}\quad \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{Late COMM1}\quad \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P'\{^{b}/_{x}\} \mid Q'}$$

$$\text{Early CLOSE1}\quad \frac{P \xrightarrow{ax} P' \qquad Q \xrightarrow{\overline{a}(vx)} Q' \qquad x \sharp P}{P \mid Q \xrightarrow{\tau} (vx)(P' \mid Q')}$$

$$\text{Late CLOSE1}\quad \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}(vy)} Q' \qquad y \sharp P}{P \mid Q \xrightarrow{\tau} (vy)(P'\{^{y}/_{x}\} \mid Q')}$$

Intuitively, the bound name on a late input action can be thought of as a place-holder for the name it will receive in a communication. Input actions are like functions where the bound name is the place-holder for the argument, which is substituted when the function is invoked.

In the next few chapters we will only reason about the late semantics, and all transitions and equivalences are assumed to be late ones. In Chapter 20, we will compare the two semantics, at which point a distinguishing syntax will be used.

17.1 Formalising the semantics

The formalisation of the late semantics for the pi-calculus follows the same pattern as before, but the input actions behave differently. This also impacts how the equivalences work. The first step is to create a residual datatype.

17.1.1 The residual datatype

As for the early semantics, a residual datatype is required in order to maintain the binding structure of the bound names on the label and the derivatives. The late operational semantics has two types of actions with binders – the input actions, and the bound output actions. To model this, the residual datatype can consists of either a free action and a derivative, or of a bound action where a subject datatype tells if the bound action is an input action or a bound output action. This allows the semantic rules to treat both types of transitions uniformly.

$$\frac{}{a(x).P \xrightarrow{a(x)} P}\ \textsc{Input} \qquad \frac{}{\overline{a}b.P \xrightarrow{\overline{a}b} P}\ \textsc{Output} \qquad \frac{}{\tau.P \xrightarrow{\tau} P}\ \textsc{Tau}$$

$$\frac{P \longmapsto Rs}{[b{=}b]P \longmapsto Rs}\ \textsc{Match} \qquad \frac{P \longmapsto Rs \qquad a \neq b}{[a{\neq}b]P \longmapsto Rs}\ \textsc{Mismatch}$$

$$\frac{P \longmapsto Rs}{P + Q \longmapsto Rs}\ \textsc{Sum1} \qquad \frac{Q \longmapsto Rs}{P + Q \longmapsto Rs}\ \textsc{Sum2} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1F}$$

$$\frac{P \xrightarrow{a\langle\langle x\rangle\rangle} P' \qquad x \sharp Q}{P \mid Q \xrightarrow{a\langle\langle x\rangle\rangle} P' \mid Q}\ \textsc{Par1B} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2F}$$

$$\frac{Q \xrightarrow{a\langle\langle x\rangle\rangle} Q' \qquad x \sharp P}{P \mid Q \xrightarrow{a\langle\langle x\rangle\rangle} P \mid Q'}\ \textsc{Par2B} \qquad \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P'\{^b/_x\} \mid Q'}\ \textsc{Comm1}$$

$$\frac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{^b/_x\}}\ \textsc{Comm2}$$

$$\frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{\overline{a}(\nu y)} Q' \qquad y \sharp P}{P \mid Q \xrightarrow{\tau} (\nu y)(P'\{^y/_x\} \mid Q')}\ \textsc{Close1}$$

$$\frac{P \xrightarrow{\overline{a}(\nu y)} P' \qquad Q \xrightarrow{a(x)} Q' \qquad y \sharp Q}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q'\{^y/_x\})}\ \textsc{Close2}$$

$$\frac{P \xrightarrow{\overline{a}b} P' \qquad a \neq b}{(\nu b)P \xrightarrow{\overline{a}(\nu b)} P'}\ \textsc{Open} \qquad \frac{P \xrightarrow{\alpha} P' \qquad y \sharp \alpha}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'}\ \textsc{ScopeF}$$

$$\frac{P \xrightarrow{a\langle\langle x\rangle\rangle} P' \qquad y \sharp a \qquad y \neq x}{(\nu y)P \xrightarrow{a\langle\langle x\rangle\rangle} (\nu y)P'}\ \textsc{ScopeB} \qquad \frac{P \mid !P \longmapsto Rs}{!P \longmapsto Rs}\ \textsc{Bang}$$

Figure 17.1: The late operational semantics for the pi-calculus. The rules that differ from the early semantics described in Figure 13.3 are the INPUT, the COMM and the CLOSE rules.

Definition 17.1. *The residual datatype*

nominal_datatype *subject* =
  InputS name
| *BoundOutputS name*

nominal_datatype *freeRes* =
  OutputR name name
| *TauR*

nominal_datatype *residual* =
  BoundR subject «*name*»*pi*
| *FreeR freeRes pi*

Definition 17.2 (Late operational semantics)**.**

1. *A transition can be written as* $P \longmapsto V$ *where* P *is an agent and* V *is a residual. The semantics is defined in Figure 17.1.*
2. $P \xrightarrow{a\langle\langle x\rangle\rangle} P'$ *denotes a bound transition with the bound name* x *in the action. Note that* a *is of type subject, which determines the type of the action. The residual by itself is written* $a\langle\langle x\rangle\rangle \prec P'$*. Individually, an input residual is written* $a(x) \prec P'$*, and a bound output residual is written* $\overline{a}\,(\nu\, x) \prec P'$*.*
3. $P \xrightarrow{\alpha} P'$ *denotes a transition without bound names. Note that* α *is of type freeRes. The residual by itself is written* $\alpha \prec P'$*. Individually, an output residual is written* $\overline{a}b \prec P'$*, and a tau-residual is written* $\tau \prec P'$*.*

At a first glance, the late residuals seem more complicated than their early counterparts, but they are actually easier to work with. In the early semantics, an input action is free, and the INPUT action in the semantics allows arbitrary names to be introduced in both the action and the derivative, which requires special infrastructure on how to reason about freshness.

Lemma 17.3. *Derived freshness rules for actions and derivatives.*

$$\begin{aligned}
&\textit{If } P \longmapsto Rs \textit{ and } x \sharp P \textit{ then } x \sharp Rs.\\
&\textit{If } P \xrightarrow{\alpha} P' \textit{ and } y \sharp P \textit{ then } y \sharp \alpha \textit{ and } y \sharp P'.\\
&\textit{If } P \xrightarrow{a\langle\langle x\rangle\rangle} P' \textit{ and } y \sharp P \textit{ then } y \sharp a.\\
&\textit{If } P \xrightarrow{a\langle\langle x\rangle\rangle} P' \textit{ and } y \sharp P \textit{ and } y \neq x \textit{ then } y \sharp P'.
\end{aligned}$$

Proof. By induction on the length of the derivation of the transitions. □

These freshness conditions state that any names fresh for a process are fresh for its residual. Moreover if they are not equal to the bound name in the residual, they are also fresh for the derivative. The corresponding lemma for the early semantics, Lemma 13.12, is more complicated as it handles the input cases separately.

17.1.2 Defining the semantics

In order for Nominal Isabelle to derive induction and inversion rules, the bound names must be fresh for everything outside their scope. These freshness conditions will hence be available in inductions on the length of a derivation.

From the introduction rules of the semantics, and the freshness rules of Lemma 17.3, the standard operational semantics, with minimal freshness conditions, is derived in Fig. 17.1.

17.1.3 Inversion rules

Isabelle automatically derives a general inversion rule for the operational semantics. As for the early semantics, this rule is cumbersome to work with on its own, but deriving custom inversion rules is straightforward. The custom inversion rules for the late operational semantics can be found in Figure 17.2.

The late operational semantics contains more binders than the early one. Moreover, the INPUT-rule suffers from the same problem as the SCOPEB rule for the bound output case – two binders are present in the rule. These must be assumed to be disjoint, and an explicit alpha-converting permutation is introduced at inversion on agents with Input as their topmost operator.

17.2 Bisimilarity

Simulation for the late semantics differs from the ones we have encountered so far. The reason is that the input-action contains a place-holder for a name that can later be substituted.

As for early simulation, late simulation is split into two cases – one where a free action is mimicked, and one where a bound action is mimicked. In the case of an input action, all derivatives obtained by substituting the bound name for any name must be in the candidate relation.

Definition 17.4 (Simulation)**.** *An agent P simulating an agent Q preserving $\mathscr{R}$ is denoted $P \hookrightarrow_{\mathscr{R}} Q$.*

derivative $P\ Q\ a\ x\ \mathscr{R} \stackrel{\text{def}}{=}$
case a *of InputS* $b \Rightarrow \forall\, u.\ (P\{^u/_x\},\ Q\{^u/_x\}) \in \mathscr{R}$
$\mid$ *BoundOutputS* $b \Rightarrow (P,\ Q) \in \mathscr{R}$

$P \hookrightarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=}$
$(\forall\, a\ x\ Q'.\ Q \xrightarrow{a\langle\langle x\rangle\rangle} Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P \xrightarrow{a\langle\langle x\rangle\rangle} P' \wedge \mathit{derivative}\ P'\ Q'\ a\ x\ \mathscr{R})) \wedge$

$$\frac{\left[\begin{array}{l} a(x).P \xrightarrow{b\langle\langle y\rangle\rangle} P' \qquad y \neq a \qquad y \neq x \qquad y \sharp P \\ \dfrac{b = \mathit{InputS}\ a \qquad P' = (x\ y) \cdot P}{\mathit{Prop}\ (\mathit{InputS}\ a)\ y\ ((x\ y) \cdot P)} \end{array}\right]}{\mathit{Prop}\ b\ y\ P'}\ \textsc{Input}$$

$$\frac{\left[\overline{a}b.P \xrightarrow{\alpha} P' \qquad \dfrac{\alpha = \overline{a}b \qquad P = P'}{\mathit{Prop}\ \overline{a}b\ P}\right]}{\mathit{Prop}\ \alpha\ P'}\ \textsc{Output}$$

$$\frac{\left[\tau.P \xrightarrow{\alpha} P' \qquad \dfrac{\alpha = \tau \qquad P = P'}{\mathit{Prop}\ (\tau)\ P}\right]}{\mathit{Prop}\ \alpha\ P'}\ \textsc{Tau}$$

$$\frac{\left[(\nu x)P \xrightarrow{\alpha} P' \qquad \bigwedge P'.\ \dfrac{P \xrightarrow{\alpha} P' \qquad x \sharp \alpha}{\mathit{Prop}\ ((\nu x)P')}\right]}{\mathit{Prop}\ P'}\ \textsc{ScopeF}$$

$$\frac{\left[\begin{array}{l} (\nu y)P \xrightarrow{a\langle\langle x\rangle\rangle} Q \qquad x \neq y \qquad x \sharp P \\ \bigwedge b\ P'.\ \dfrac{P \xrightarrow{\overline{b}y} P' \qquad b \neq y \qquad a = \mathit{BoundOutputS}\ b}{\mathit{Prop}\ (\mathit{BoundOutputS}\ b)\ ((x\ y) \cdot P')} \\ \bigwedge P'.\ \dfrac{P \xrightarrow{a\langle\langle x\rangle\rangle} P' \qquad y \sharp a}{\mathit{Prop}\ a\ ((\nu y)P')} \end{array}\right]}{\mathit{Prop}\ a\ Q}\ \textsc{ScopeB}$$

$$\frac{\left[!P \longmapsto V \qquad \dfrac{P \mid !P \longmapsto V}{\mathit{Prop}}\right]}{\mathit{Prop}}\ \textsc{Repl}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{a\langle\langle x\rangle\rangle} R \qquad x \sharp P \qquad x \sharp Q \\ \bigwedge P'.\ \dfrac{P \xrightarrow{a\langle\langle x\rangle\rangle} P'}{\mathit{Prop}\,(P' \mid Q)} \qquad \bigwedge Q'.\ \dfrac{Q \xrightarrow{a\langle\langle x\rangle\rangle} Q'}{\mathit{Prop}\,(P \mid Q')} \end{array}\right]}{\mathit{Prop}\,R}\ \textsc{ParB}$$

$$\frac{\left[\begin{array}{l} P \mid Q \xrightarrow{\alpha} R \\ \bigwedge P'.\ \dfrac{P \xrightarrow{\alpha} P'}{F\,\alpha\,(P' \mid Q)} \qquad \bigwedge Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{F\,\alpha\,(P \mid Q')} \\ \bigwedge P'\,Q'\,a\,b\,x.\ \dfrac{\left(\begin{array}{llll} P \xrightarrow{a(x)} P' & Q \xrightarrow{\overline{a}b} Q' & x \sharp P & x \sharp Q \\ x \neq a & x \neq b & x \sharp Q' & x \sharp \mathscr{C} \qquad \alpha = \tau \end{array}\right)}{F\,(\tau)\,(P'\{^b/_x\} \mid Q')} \\ \bigwedge P'\,Q'\,a\,b\,x.\ \dfrac{\left(\begin{array}{llll} P \xrightarrow{\overline{a}b} P' & Q \xrightarrow{a(x)} Q' & x \sharp P & x \sharp Q \\ x \neq a & x \neq b & x \sharp P' & x \sharp \mathscr{C} \qquad \alpha = \tau \end{array}\right)}{F\,(\tau)\,(P' \mid Q'\{^b/_x\})} \\ \bigwedge P'\,Q'\,a\,x\,y.\ \dfrac{\left(\begin{array}{llll} P \xrightarrow{a(x)} P' & Q \xrightarrow{\overline{a}(\nu y)} Q' & x \sharp P & x \sharp Q \\ x \neq a & x \neq y & x \sharp Q' & y \sharp P \qquad y \sharp Q \\ y \neq a & y \sharp P' & x \sharp \mathscr{C} & y \sharp \mathscr{C} \qquad \alpha = \tau \end{array}\right)}{F\,(\tau)\,((\nu y)(P'\{^y/_x\} \mid Q'))} \\ \bigwedge P'\,Q'\,a\,x\,y.\ \dfrac{\left(\begin{array}{llll} P \xrightarrow{\overline{a}(\nu y)} P' & Q \xrightarrow{a(x)} Q' & x \sharp P & x \sharp Q \\ x \neq a & x \neq y & x \sharp P' & y \sharp P \qquad y \sharp Q \\ y \neq a & y \sharp Q' & x \sharp \mathscr{C} & y \sharp \mathscr{C} \qquad \alpha = \tau \end{array}\right)}{F\,(\tau)\,((\nu y)(P' \mid Q'\{^y/_x\}))} \end{array}\right]}{F\,\alpha\,R}\ \textsc{ParF}$$

Figure 17.2: Custom inversion rules for the late operational semantics

$$(\forall\, \alpha\ Q'.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}))$$

The *derivative* predicate does a case analysis on a subject. If the subject is a bound output, it ensures that the derivatives are in the candidate relation. If the subject is an input, all derivatives obtainable by substituting the bound name of the input action for any other name must also be in the candidate relation.

Before defining bisimilarity we must prove that simulation is monotonic.

Lemma 17.5. *If $P \hookrightarrow_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \hookrightarrow_{\mathscr{R}'} Q$.*

Proof. Follows directly from Definition 17.4. □

Late bisimilarity is defined in the standard way.

Definition 17.6. *Late bisimimulation is defined coinductively as the greatest fixed point satisfying:*

$$\begin{array}{rll} P \dot{\sim} Q \Longrightarrow P \hookrightarrow_{\dot{\sim}} Q & & \textsc{Simulation} \\ \wedge\ Q \dot{\sim} P & & \textsc{Symmetry} \end{array}$$

17.2.1 Introduction and elimination rules

The introduction rule for simulation is defined similarly to the one for the early semantics and allows the bound names in the transitions to avoid any other context of names under consideration.

Lemma 17.7. *Introduction rule for late simulation.*

$$\frac{\begin{array}{l} \textit{eqvt}\ \mathscr{R} \\ \bigwedge a\ x\ Q'.\ \dfrac{Q \xrightarrow{a\langle\langle x\rangle\rangle} Q' \qquad x \sharp P \qquad x \sharp Q \qquad x \sharp a \qquad x \sharp \mathscr{C}}{\exists P'.\ P \xrightarrow{a\langle\langle x\rangle\rangle} P' \wedge \textit{derivative}\ P'\ Q'\ a\ x\ \mathscr{R}} \\ \bigwedge \alpha\ Q'.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}} \end{array}}{P \hookrightarrow_{\mathscr{R}} Q}\ \hookrightarrow\text{-I}$$

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{a\langle\langle x\rangle\rangle} Q' \qquad x \sharp P}{\exists P'.\ P \xrightarrow{a\langle\langle x\rangle\rangle} P' \wedge \textit{derivative}\ P'\ Q'\ a\ x\ \mathscr{R}}\ \hookrightarrow\text{-E1}$$

$$\frac{P \hookrightarrow_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathscr{R}}\ \hookrightarrow\text{-E2}$$

Proof. Follows from Definition 17.4 where the bound names of the transitions are alpha-converted to not clash with $\mathscr{C}$. □

This introduction rule requires the candidate relation to be equivariant, otherwise alpha-converting the derivatives would make them fall outside the relation.

Lemma 17.8. *Introduction and elimination rules for strong late bisimilarity.*

$$\frac{P \leadsto_{\dot{\sim}} Q \qquad Q \dot{\sim} P}{P \dot{\sim} Q}\ \dot{\sim}\text{-I} \qquad \frac{P \dot{\sim} Q}{P \leadsto_{\dot{\sim}} Q}\ \dot{\sim}\text{-E1} \qquad \frac{P \dot{\sim} Q}{Q \dot{\sim} P}\ \dot{\sim}\text{-E2}$$

Proof. Follows from Definition 17.6. □

The coinduction rule follows the standard pattern.

Lemma 17.9.

$$\frac{(P, Q) \in \mathscr{X} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{R \leadsto_{\mathscr{X} \cup \dot{\sim}} S} \qquad \bigwedge R\, S.\ \dfrac{(R, S) \in \mathscr{X}}{(S, R) \in \mathscr{X}}}{P \dot{\sim} Q}$$

17.3 Preservation properties

Late bismiulation is preserved by the same operators as early bisimilarity – everything except Input Even though the simulation definitions differ, the requisites for the candidate relation for all simulation lemmas are the same.

As the lemmas are identical, they are omitted, but the proof that late simulation is preserved by Parallel is presented in Figure 17.3.

Lemma 17.10. *Late bisimilarity is an equivalence relation*

Proof. Similar to Lemma 14.12. □

Theorem 17.1. *Late bisimilarity is preserved by all operators except Input.*

Proof. Similar to Theorem 14.1. □

17.4 Strong equivalence

A strong equivalence is obtained in the standard way by closing strong bisimilarity under all substitutions.

Figure 17.3: Isabelle proof that simulation is preserved by the Parallel

lemma simParCompose:
fixes P :: pi **and** Q :: pi **and** R :: pi **and** T :: pi
and $\mathscr{R}$ :: (pi × pi) set **and** $\mathscr{R}'$ :: (pi × pi) set **and** $\mathscr{R}''$ :: (pi × pi) set

assumes $P \hookrightarrow_{\mathscr{R}} Q$ **and** $(P, Q) \in \mathscr{R}$ **and** $R \hookrightarrow_{\mathscr{R}'} T$ **and** $(R, T) \in \mathscr{R}'$
and C1: $\bigwedge P' Q' R' T'.\ [\![(P', Q') \in \mathscr{R}; (R', T') \in \mathscr{R}']\!] \Longrightarrow (P' \mid R', Q' \mid T') \in \mathscr{R}''$
and C2: $\bigwedge P' Q' x.\ (P', Q') \in \mathscr{R}'' \Longrightarrow (<\nu x>P', <\nu x>Q') \in \mathscr{R}''$
and eqvt $\mathscr{R}''$

shows $P \mid R \hookrightarrow_{\mathscr{R}''} Q \mid T$

using ⟨eqvt $\mathscr{R}''$⟩
proof(induct rule: simI[**where** C={}]) — *Apply introduction rule* $\hookrightarrow$-I
case(*Bound a x V*)
from ⟨$x \sharp P \mid R$⟩ ⟨$x \sharp Q \mid T$⟩ **have** $x \sharp P$ **and** $x \sharp Q$ **and** $x \sharp R$ **and** $x \sharp T$ **by** *simp+*
from ⟨$Q \mid T \xrightarrow{a«x»} V$⟩ ⟨$x \sharp Q$⟩ ⟨$x \sharp T$⟩
show $\exists S.\ P \mid R \xrightarrow{a«x»} S \wedge$ *derivative S V a x* $\mathscr{R}''$
proof(*induct rule*: *parCasesB*) — *Apply* PARB *inversion rule from Figure 17.2*
PAR1 *case*
Given that $Q \xrightarrow{a«x»} Q'$ *prove that there exists an S*
such that $P \mid R \xrightarrow{a«x»} S$ *and derivative* $S\ (Q' \mid T)\ a\ x\ \mathscr{R}''$.
case(*cPar1 Q'*)
from ⟨$P \hookrightarrow_{\mathscr{R}} Q$⟩ ⟨$Q \xrightarrow{a«x»} Q'$⟩ ⟨$x \sharp P$⟩
obtain P' **where** $P \xrightarrow{a«x»} P'$ **and** *A*: *derivative* $P'\ Q'\ a\ x\ \mathscr{R}$
by(*blast dest*: *simE*)
from ⟨$P \xrightarrow{a«x»} P'$⟩ ⟨$x \sharp R$⟩ **have** $P \mid R \xrightarrow{a«x»} P' \mid R$ **by**(*rule Par1B*)
moreover from *A* ⟨$x \sharp R$⟩ ⟨$x \sharp T$⟩ ⟨$(R, T) \in \mathscr{R}'$⟩
have *derivative* $(P' \mid R)\ (Q' \mid T)\ a\ x\ \mathscr{R}''$
by(*cases a, auto intro*: *C1 simp add*: *derivative-def forget*)
ultimately show $\exists S.\ P \mid R \xrightarrow{a«x»} S \wedge$ *derivative* $S\ (Q' \mid T)\ a\ x\ \mathscr{R}''$ **by** *blast*

next PAR2 *case*
Given that $T \xrightarrow{a«x»} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{a«x»} S$ *and derivative* $S\ (Q \mid T')\ a\ x\ \mathscr{R}''$.
case(cPar2 T′)
from ⟨R ↪$_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{a«x»}$ T′⟩ ⟨x ♯ R⟩
obtain R′ **where** R $\xrightarrow{a«x»}$ R′ **and** A: derivative R′ T′ a x $\mathscr{R}'$
by(blast dest: simE)
from ⟨R ⟼ a«x» ≺ R′⟩ ⟨x ♯ P⟩ **have** P | R $\xrightarrow{a«x»}$ P | R′ **by**(rule Par2B)
moreover from A ⟨x ♯ P⟩ ⟨x ♯ Q⟩ ⟨(P, Q) ∈ $\mathscr{R}$⟩
have derivative (P | R′) (Q | T′) a x $\mathscr{R}''$
by(cases a, auto intro: C1 simp add: derivative-def forget)
ultimately show ∃S. P | R $\xrightarrow{a«x»}$ S ∧ derivative S (Q | T′) a x $\mathscr{R}''$ **by** blast
qed
next
case(Free α V)
from ⟨Q | T $\xrightarrow{\alpha}$ V⟩ **show** ∃S. P | R $\xrightarrow{\alpha}$ S ∧ (S, V) ∈ $\mathscr{R}''$
proof(induct rule: parCasesF[**where** C=(P, R)]) — *Apply* PARF *inversion rule*
PAR1 *case*
Given that $Q \xrightarrow{\alpha} Q'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\alpha} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.
case(cPar1 Q′)
from ⟨P ↪$_{\mathscr{R}}$ Q⟩ ⟨Q $\xrightarrow{\alpha}$ Q′⟩ **obtain** P′ **where** P $\xrightarrow{\alpha}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(blast dest: simE)
from ⟨P $\xrightarrow{\alpha}$ P′⟩ **have** Trans: P | R $\xrightarrow{\alpha}$ P′ | R **by**(rule Par1F)
moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R, T) ∈ $\mathscr{R}'$⟩ **have** (P′ | R, Q′ | T) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\alpha}$ S ∧ (S, Q′ | T) ∈ $\mathscr{R}''$ **by** blast
next
PAR2 *case*
Given that $T \xrightarrow{\alpha} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\alpha} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.
case(cPar2 T′)
from ⟨R ↪$_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{\alpha}$ T′⟩ **obtain** R′ **where** R $\xrightarrow{\alpha}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(blast dest: simE)
from ⟨R $\xrightarrow{\alpha}$ R′⟩ **have** Trans: P | R $\xrightarrow{\alpha}$ P | R′ **by**(rule Par2F)
moreover from ⟨(P, Q) ∈ $\mathscr{R}$⟩ ⟨(R′, T′) ∈ $\mathscr{R}'$⟩ **have** (P | R′, Q | T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\alpha}$ S ∧ (S, Q | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM1 *case*
Given that $Q \xrightarrow{a(b)} Q'$ *and* $T \xrightarrow{\overline{a}b} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm1 Q′ T′ a b x)
from ‹x ♯ (P, R)› **have** x ♯ P **by** simp
from ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{a(x)}$ Q'› ‹x ♯ P›
obtain P′ **where** P $\xrightarrow{a(x)}$ P′ **and** A: derivative P′ Q′ (InputS a) x $\mathscr{R}$
by(blast dest: simE)
from A **have** (P′{$^b/_x$}, Q′{$^b/_x$}) ∈ $\mathscr{R}$ **by**(simp add: derivative-def)
from ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}b}$ T'›
obtain R′ **where** R $\xrightarrow{\overline{a}b}$ R′ **and** (R′, T') ∈ $\mathscr{R}'$
by(blast dest: simE)
from ‹P $\xrightarrow{a(x)}$ P'› ‹R $\xrightarrow{\overline{a}b}$ R'› **have** P | R $\xrightarrow{\tau}$ P′{$^b/_x$} | R′
by(rule Comm1)
moreover from ‹(P′{$^b/_x$}, Q′{$^b/_x$}) ∈ $\mathscr{R}$› ‹(R′, T') ∈ $\mathscr{R}'$›
have (P′{$^b/_x$} | R′, Q′{$^b/_x$} | T') ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\tau}$ S ∧ (S, Q′{$^b/_x$} | T') ∈ $\mathscr{R}''$ **by** blast

next

COMM2 *case*
Given that $Q \xrightarrow{\overline{a}b} Q'$ *and* $T \xrightarrow{a(b)} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, Q' \mid T') \in \mathscr{R}''$.

case(cComm2 Q′ T′ a b x)
from ‹x ♯ (P, R)› **have** x ♯ R **by** simp
from ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}b}$ Q'›
obtain P′ **where** P $\xrightarrow{\overline{a}b}$ P′ **and** (P′, Q') ∈ $\mathscr{R}$ **by**(blast dest: simE)
from ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{a(x)}$ T'› ‹x ♯ R›
obtain R′ **where** R $\xrightarrow{a(x)}$ R′ **and** A: derivative R′ T′ (InputS a) x $\mathscr{R}'$
by(blast dest: simE)
from A **have** (R′{$^b/_x$}, T′{$^b/_x$}) ∈ $\mathscr{R}'$ **by**(simp add: derivative-def)
from ‹P $\xrightarrow{\overline{a}b}$ P'› ‹R $\xrightarrow{a(x)}$ R'› **have** P | R $\xrightarrow{\tau}$ P′ | R′{$^b/_x$}
by(rule Comm2)
moreover from ‹(P′, Q') ∈ $\mathscr{R}$› ‹(R′{$^b/_x$}, T′{$^b/_x$}) ∈ $\mathscr{R}'$›
have (P′ | R′{$^b/_x$}, Q′ | T′{$^b/_x$}) ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P | R $\xrightarrow{\tau}$ S ∧ (S, Q′ | T′{$^b/_x$}) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE1 *case*
Given that $Q \xrightarrow{a(x)} Q'$ *and* $T \xrightarrow{\overline{a}(\nu x)} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, (\nu x)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose1 Q′ T′ a x y)
from ‹x ♯ (P, R)› **have** x ♯ P **by** simp
with ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{a(x)}$ Q′›
obtain P′ **where** P $\xrightarrow{a(x)}$ P′ **and** A: derivative P′ Q′ (InputS a) x $\mathscr{R}$
by(blast dest: simE)
from A **have** (P′{$^y/_x$}, Q′{$^y/_x$}) ∈ $\mathscr{R}$ **by**(simp add: derivative-def)
from ‹y ♯ (P, R)› **have** y ♯ R **and** y ♯ P **by** simp+
from ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}(\nu y)}$ T′› ‹y ♯ R›
obtain R′ **where** R $\xrightarrow{\overline{a}(\nu y)}$ R′
and B: derivative R′ T′ (BoundOutputS a) y $\mathscr{R}'$
by(blast dest: simE)
from B **have** (R′, T′) ∈ $\mathscr{R}'$ **by**(simp add: derivative-def)
from ‹P $\xrightarrow{a(x)}$ P′› ‹R $\xrightarrow{\overline{a}(\nu y)}$ R′› ‹y ♯ P› **have** P | R $\xrightarrow{\tau}$ <νy>(P′{$^y/_x$} | (R′))
by(rule Close1)
moreover from ‹(P′{$^y/_x$}, Q′{$^y/_x$}) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$›
have ((ν y)(P′{$^y/_x$} | R′), (ν y)(Q′{$^y/_x$} | T′)) ∈ $\mathscr{R}''$
by(blast intro: C1 C2)
ultimately show ∃S. P | R $\xrightarrow{\tau}$ S ∧ (S, <νy>(Q′{$^y/_x$} | T′)) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE2 *case*
Given that $Q \xrightarrow{\overline{a}(\nu y)} Q'$ *and* $T \xrightarrow{a(x)} T'$ *prove that there exists an S such that* $P \mid R \xrightarrow{\tau} S$ *and* $(S, (\nu x)(Q' \mid T')) \in \mathscr{R}''$.

case(cClose2 Q′ T′ a x y)
from ‹y ♯ (P, R)› **have** y ♯ R **and** y ♯ P **by** simp+
from ‹P $\hookrightarrow_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}(\nu y)}$ Q′› ‹y ♯ P›
obtain P′ **where** P $\xrightarrow{\overline{a}(\nu y)}$ P′ **and** A: derivative P′ Q′ (BoundOutputS a) y $\mathscr{R}$
by(blast dest: simE)
from A **have** (P′, Q′) ∈ $\mathscr{R}$ **by**(simp add: derivative-def)
from ‹x ♯ (P, R)› **have** x ♯ R **by** simp
with ‹R $\hookrightarrow_{\mathscr{R}'}$ T› ‹T $\xrightarrow{a(x)}$ T′›
obtain R′ **where** R $\xrightarrow{a(x)}$ R′ **and** B: derivative R′ T′ (InputS a) x $\mathscr{R}'$
by(blast dest: simE)
from B **have** (R′{$^y/_x$}, T′{$^y/_x$}) ∈ $\mathscr{R}'$ **by**(simp add: derivative-def)
from ‹P $\xrightarrow{\overline{a}(\nu y)}$ P′› ‹R $\xrightarrow{a(x)}$ R′› ‹y ♯ R› **have** P | R $\xrightarrow{\tau}$ <νy>(P′ | R′{$^y/_x$})
by(rule Close2)
moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R′{$^y/_x$}, T′{$^y/_x$}) ∈ $\mathscr{R}'$›
have ((νy)(P′ | R′{$^y/_x$}), (νy)(Q′ | T′{$^y/_x$})) ∈ $\mathscr{R}''$
by(blast intro: C1 C2)
ultimately show ∃S. P | R $\xrightarrow{\tau}$ S ∧ (S, <νy>(Q′ | T′{$^y/_x$})) ∈ $\mathscr{R}''$ **by** blast
qed
qed

Definition 17.11. $P \sim Q \stackrel{\text{def}}{=} P \dot{\sim}^{s} Q$

The congruence theorem is established in the same way as for early equivalence.

Theorem 17.2. *Strong late equivalence is a congruence*

Proof. Similar to Theorem 14.2. □

17.5 Weak equivalences

The weak equivalences that have been covered so far have followed the standard pattern that an action is mimicked by the same action, preceded and succeeded by an arbitrary number of silent τ-actions. This turns out not to work for weak late bisimilarity, and the reason stems from how the late semantics handle input-actions. The following example is taken from [67]. Consider the following three agents.

$$Q \stackrel{\text{def}}{=} a(x).[x{=}v]P + a(x).\tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P))$$

$$R \stackrel{\text{def}}{=} a(x).\tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P)) \qquad S \stackrel{\text{def}}{=} a(x).\tau.\mathbf{0} + \tau.P$$

First observe that R and S even are strongly bisimilar, since

$$\tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P)) \sim \tau.\mathbf{0} + \tau.P$$

the last summand in R can be mimicked by one of the summands in S, depending on whether x is substituted by v or not.

Next observe that since the agents Q and R only differ in that Q has one extra summand, the relevant case is where

$$a(x).[x{=}v]P \xrightarrow{a(x)} [x{=}v]P.$$

The agent R can mimic this action with a strong one

$$a(x).\tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P)) \xrightarrow{a(x)} \tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P)),$$

and then a τ-action to

$$\tau.\mathbf{0} + (\tau.P + \tau.([x{=}v]P)) \xrightarrow{\tau} [x{=}v]P.$$

Since the derivatives are equal, any substitution for x applied to both of them will still equate them, as $([x{=}v]P)\{^{u}/_{x}\} \sim ([x{=}v]P)\{^{u}/_{x}\}$ for all u.

Assume that we define a weak bisimilarity, denoted $\dot{\approx}'$, for the late semantics in the same way as for the early one – every action is mimicked by the same action with a preceding and succeeding τ-chain.

We then have that $Q \dot{\approx}' R$ and $R \dot{\approx}' S$, and we would expect that $Q \dot{\approx}' S$, since we require our bisimulation relations to be transitive. For this candidate weak bisimilarity, this turns out not to be the case. Again, since $R \dot{\approx}' S$, the relevant action is when the first summand of Q does the action

$$a(x).[x{=}v]P \xrightarrow{a(x)} [x{=}v]P.$$

In this case there is no way that S can mimic with a weak action such that the derivatives are bisimilar for all substitutions. If the name x is being replaced by v, then the second summand should be chosen; if it is not equal to v, then the first one should be chosen – the τ-chain after the visible action depends on the new name.

The solution is hinted above – the trailing τ-chain depends on which new name is introduced; the solution is to require that for every possible name u substituted for x in the derivative of the strong input action, there exists a τ-chain leading to a bisimilar agent. If p is weakly bisimilar to Q, a strong transition $Q \xrightarrow{a(x)} Q'$ is mimicked if

$$\exists P''' P''.\, P \Longrightarrow_\tau P''' \wedge P''' \xrightarrow{a(x)} P'' \wedge (\forall u.\, \exists P'.\, P''\{^u/_x\} \Longrightarrow_\tau P' \wedge P' \dot{\approx} Q')$$

Here the input substitution is done immediately after the mimicking input action, and it is sufficient that a τ-chain exists for every possible input name. With this definition, we have that Q and S are weakly bisimilar.

17.5.1 Weak semantics

Apart from the input-action, weak late semantics corresponds to the early semantics completely. A weak input action needs to carry additional information on the labels in order to determine the name that is substituted after the strong input action.

Definition 17.12 (Weak input transitions)**.**

$$P \xRightarrow{u:a(x)@P''} P' \stackrel{\text{def}}{=} \exists P'''.\, P \Longrightarrow_\tau P''' \wedge P''' \xrightarrow{a(x)} P'' \wedge P''\{^u/_x\} \Longrightarrow_\tau P'$$

The transition $P \xRightarrow{u:a(x)@P''} P'$ means that P can do a τ-chain and then the input action $a(x)$ to an agent P'' where x is substituted for u and another τ-chain is done to P'. The agent P'' represents the exact state where the substitution is made. This will be important when we define weak simulation.

$$\frac{}{a(x).P \xRightarrow{u:a(x)@P} P\{^u/_x\}}\ \textsc{Input} \qquad \frac{P \xRightarrow{u:a(x)@P''} P' \qquad x \sharp Q}{P \mid Q \xRightarrow{u:a(x)@P'' \mid Q} P' \mid Q}\ \textsc{Par1}$$

$$\frac{Q \xRightarrow{u:a(x)@Q''} Q' \qquad x \sharp P}{P \mid Q \xRightarrow{u:a(x)@P \mid Q''} P \mid Q'}\ \textsc{Par2} \qquad \frac{P \xRightarrow{b:a(x)@P''} P' \qquad Q \xRightarrow{\widehat{\overline{a}b}} Q'}{P \mid Q \xRightarrow{\hat{\tau}} P' \mid Q'}\ \textsc{Comm1}$$

$$\frac{P \xRightarrow{\widehat{\overline{a}b}} P' \qquad Q \xRightarrow{b:a(x)@Q''} Q'}{P \mid Q \xRightarrow{\hat{\tau}} P' \mid Q'}\ \textsc{Comm2}$$

$$\frac{P \xRightarrow{y:a(x)@P''} P' \qquad Q \xRightarrow{\overline{a}(\nu y)} Q' \qquad y \sharp P \qquad y \sharp Q}{P \mid Q \xRightarrow{\hat{\tau}} (\nu y)(P' \mid Q')}\ \textsc{Close1}$$

$$\frac{P \xRightarrow{\overline{a}(\nu y)} P' \qquad Q \xRightarrow{y:a(x)@Q''} Q' \qquad y \sharp P \qquad y \sharp Q}{P \mid Q \xRightarrow{\hat{\tau}} (\nu y)(P' \mid Q')}\ \textsc{Close2}$$

$$\frac{P \xRightarrow{u:a(x)@P''} P' \qquad y \neq a \qquad y \neq x \qquad y \neq u}{(\nu y)P \xRightarrow{u:a(x)@(\nu y)P''} (\nu y)P'}\ \textsc{Res}$$

Figure 17.4: The weak late lifted semantic rules for input transitions.

The name x is not bound in P' as it is substituted for u before the τ-chain. We can still do alpha-conversions through the following lemma:

Lemma 17.13. *If* $P \xRightarrow{u:a(x)@P''} P'$ *and* $y \sharp P$ *then* $P \xRightarrow{u:a(y)@(x\ y)\cdot P''} P'$.

Proof. Similar to Lemma 15.7. □

The alpha-converting swapping shows up on the label, as it represents the derivative of the strong action.

The weak late semantics is lifted in the same way as the weak early semantics. The rules that differ are those containing input actions. Their lifted rules can be found in Figure 17.4.

17.5.2 Weak bisimilarity

The definition of weak late simulation has three cases – one for input actions, one for bound output actions, and one for actions with no binders.

Definition 17.14 (Weak simulation)**.** *An agent P weakly simulating an agent Q preserving $\mathscr{R}$ is denoted $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$.*

$P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \stackrel{\text{def}}{=}$

$(\forall Q'\, a\, x.\ Q \xrightarrow{\overline{a}(vx)} Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P \xRightarrow{\overline{a}(vx)} P' \wedge (P', Q') \in \mathscr{R})) \wedge$

$(\forall Q'\, a\, x.\ Q \xrightarrow{a(x)} Q' \wedge x \sharp P \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \xRightarrow{u:a(x)@P''} P' \wedge (P', Q'\{^u/_x\}) \in \mathscr{R})) \wedge$

$(\forall Q'\, \alpha.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}))$

The important aspect of weak late simulation is the fact mentioned above – that an input action $a(x)$ must be matched by a weak transition with the same input derivative P'' for all possible instantiations u of the bound name.

Before defining weak bisimilarity, we must prove that weak simulation is monotonic.

Lemma 17.15. *If $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} P'$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} P'$.*

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$. □

Definition 17.16. *Weak late bisimilarity is defined as the greatest fixed point satisfying:*

$$P \dot{\approx} Q \Longrightarrow P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q \qquad \textsc{Simulation}$$
$$\wedge\ Q \dot{\approx} P \qquad \textsc{Symmetry}$$

From our definition, we can derive introduction and elimination rules similar to the one done for strong simulation, and weak early simulation.

Lemma 17.17. *Introduction and elimination rules for weak simulation*

$$\textit{eqvt}\ \mathscr{R} \qquad \bigwedge Q'\, a\, x.\ \frac{x \sharp \mathscr{C} \qquad Q \xrightarrow{\overline{a}(\nu x)} Q'}{\exists P'.\ P \xRightarrow{\overline{a}(\nu x)} P' \wedge (P', Q') \in \mathscr{R}}$$

$$\bigwedge Q'\, a\, x.\ \frac{x \sharp \mathscr{C} \qquad Q \xrightarrow{a(x)} Q'}{\exists P''.\ \forall u.\ \exists P'.\ P \xRightarrow{u:a(x)@P''} P' \wedge (P', Q'\{^u/_x\}) \in \mathscr{R}}$$

$$\frac{\bigwedge Q'\, \alpha.\ \dfrac{Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}}}{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\ \widehat{\rightsquigarrow}\text{-I}$$

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{\overline{a}(\nu x)} Q' \qquad x \sharp P}{\exists P'.\ P \xRightarrow{\overline{a}(\nu x)} P' \wedge (P', Q') \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E1}$$

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{a(x)} Q' \qquad x \sharp P}{\exists P''.\ \forall u.\ \exists P'.\ P \xRightarrow{u:a(x)@P''} P' \wedge (P', Q'\{^u/_x\}) \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E2}$$

$$\frac{P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad Q \xrightarrow{\alpha} Q'}{\exists P'.\ P \xRightarrow{\widehat{\alpha}} P' \wedge (P', Q') \in \mathscr{R}}\ \widehat{\rightsquigarrow}\text{-E3}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$. For the introduction rule, the bound names are alpha-converted to not clash with the freshness context $\mathscr{C}$. □

The preservation proof for weak late simulation are exactly those for weak early simulation. However, their proofs are different due to the different quantifier ordering for the weak input action. Typically, those proofs amount to juggling quantifier eliminations and introductions between the assumptions and the conclusions of the goal. The proof that weak late simulation is preserved by Parallel is presented in Figure 17.5.

Theorem 17.3. *Weak bisimilarity is preserved by all operators except Sum, and Input.*

Proof. Similar to Theorem 15.1. □

17.5.3 τ-bisimilarity

We define τ-bisimilarity in a similar way to the early semantics – an agent may not mimic a τ-action by doing nothing. Otherwise, the definition is the same as the definition for $\widehat{\rightsquigarrow}$.

Figure 17.5: Isabelle proof that weak simulation is preserved by Parallel

lemma weakSimParCompose:
fixes P :: pi **and** Q :: pi **and** R :: pi **and** T :: pi
and Rel :: (pi × pi) set **and** $\mathscr{R}'$ :: (pi × pi) set

assumes P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q **and** (P, Q) ∈ $\mathscr{R}$
and R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T **and** (R, T) ∈ $\mathscr{R}'$
and C1: ⋀P Q R T. ⟦(P, Q) ∈ $\mathscr{R}$; (R, T) ∈ $\mathscr{R}'$⟧ ⟹ (P | R, Q | T) ∈ $\mathscr{R}''$
and C2: ⋀P Q a. (P, Q) ∈ $\mathscr{R}''$ ⟹ (<νa>P, <νa>Q) ∈ $\mathscr{R}''$
and eqvt $\mathscr{R}''$

shows P | R $\widehat{\rightsquigarrow}_{\mathscr{R}''}$ Q | T
using ‹eqvt $\mathscr{R}''$›
proof(induct rule: weakSimI[**where** C=()]) — *Apply introduction rule* $\widehat{\rightsquigarrow}$-I
case(*Bound V a x*)
from ‹$x \sharp P \mid R$› ‹$x \sharp Q \mid T$› **have** $x \sharp P$ **and** $x \sharp R$ **and** $x \sharp Q$ **and** $x \sharp T$ **by** *simp+*
from ‹$Q \mid T \xrightarrow{\overline{a}(vx)} V$› ‹$x \sharp Q$› ‹$x \sharp T$›
show ∃*S*. $P \mid R \xRightarrow{\overline{a}(vx)} S \wedge (S, V) \in \mathscr{R}''$
proof(*induct rule*: *parCasesB*) — *Apply* ParB *inversion rule from Figure 13.4.2*

ParB1 *case*
Given that $Q \xrightarrow{\overline{a}(vx)} Q'$ *prove that there exists an S such that* $P \mid R \xRightarrow{\overline{a}(vx)} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(*cPar1 Q'*)
from ‹$P \widehat{\rightsquigarrow}_{\mathscr{R}} Q$› ‹$Q \xrightarrow{\overline{a}(vx)} Q'$› ‹$x \sharp P$›
obtain P' **where** $P \xRightarrow{\overline{a}(vx)} P'$ **and** $(P', Q') \in \mathscr{R}$ **by**(*blast dest*: *weakSimE*)
from ‹$P \xRightarrow{\overline{a}(vx)} P'$› ‹$x \sharp R$› **have** $P \mid R \xRightarrow{\overline{a}(vx)} P' \mid R$ **by**(*rule weakPar1B*)
moreover from ‹$(P', Q') \in \mathscr{R}$› ‹$(R, T) \in \mathscr{R}'$› **have** $(P' \mid R, Q' \mid T) \in \mathscr{R}''$
by(*rule C1*)
ultimately show ∃*S*. $P \mid R \xRightarrow{\overline{a}(vx)} S \wedge (S, Q' \mid T) \in \mathscr{R}''$ **by** *blast*

next

PAR2 *case*
Given that $T \xrightarrow{\overline{a}(vx)} T'$ *prove that there exists an S such that* $P \mid R \xRightarrow{\overline{a}(vx)} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T′)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}(vx)}$ T′› ‹x ♯ R›
obtain R′ **where** R $\xRightarrow{\overline{a}(vx)}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: weakSimE)
from ‹R $\xRightarrow{\overline{a}(vx)}$ R′› ‹x ♯ P› **have** ParTrans: P ∣ R $\xRightarrow{\overline{a}(vx)}$ P ∣ R′
by(rule weakPar2B)
moreover from ‹(P, Q) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$› **have** (P ∣ R′, Q ∣ T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P ∣ R $\xRightarrow{\overline{a}(vx)}$ S ∧ (S, Q ∣ T′) ∈ $\mathscr{R}''$ **by** blast
qed

next
case(Input V a x)
from ‹x ♯ P ∣ R› ‹x ♯ Q ∣ T› **have** x ♯ P **and** x ♯ R **and** x ♯ Q **and** x ♯ T **by** simp+
from ‹Q ∣ T $\xrightarrow{a(x)}$ V› ‹x ♯ Q› ‹x ♯ T›
show ∃S′. ∀u. ∃S. P ∣ R $\xRightarrow{u:a(x)@S'}$ S ∧ (S, V{$^u/_x$}) ∈ $\mathscr{R}''$
proof(induct rule: parCasesB) — *Apply* PARB *inversion rule*

PAR1 *case*
Given that $Q \xrightarrow{a(x)} Q'$ *prove that there exists an* S' *such that for all u there exists an S such that* $P \mid R \xRightarrow{u:a(x)@S'} S$ *and* $(S, (Q' \mid T)\{^u/_x\}) \in \mathscr{R}''$.

case(*cPar1 Q′*)
from ‹$P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q$› ‹$Q \xrightarrow{a(x)} Q'$› ‹$x \sharp P$› **obtain** P''
where *L1*: ∀ *u*. ∃P'. $P \xRightarrow{u:a(x)@P''} P' \wedge (P', Q'\{^u/_x\}) \in \mathscr{R}$
by(*blast dest*: *weakSimE*)
have ∀ *u*. ∃*S*. $P \mid R \xRightarrow{u:a(x)@P'' \mid R} S \wedge (S, Q'\{^u/_x\} \mid T\{^u/_x\}) \in \mathscr{R}''$
proof(*rule allI*)
fix *u*
from *L1* **obtain** P' **where** $P \xRightarrow{u:a(x)@P''} P'$ **and** $(P', Q'\{^u/_x\}) \in \mathscr{R}$ **by** *blast*
from ‹$P \xRightarrow{u:a(x)@P''} P'$› ‹$x \sharp R$› **have** $P \mid R \xRightarrow{u:a(x)@P'' \mid R} P' \mid R$
by(*rule weakPar1Input*)
moreover from ‹$(P', Q'\{^u/_x\}) \in \mathscr{R}$› ‹$(R, T) \in \mathscr{R}'$›
have $(P' \mid R, Q'\{^u/_x\} \mid T) \in \mathscr{R}''$ **by**(*rule C1*)
ultimately show ∃*S*. $P \mid R \xRightarrow{u:a(x)@P'' \mid R} S \wedge (S, Q'\{^u/_x\} \mid (T\{^u/_x\})) \in \mathscr{R}''$
using ‹$x \sharp T$› **by**(*force simp add*: *forget*)
qed
thus ∃S'. ∀ *u*. ∃*S*. $P \mid R \xRightarrow{u:a(x)@S'} S \wedge (S, (Q' \mid T)\{^u/_x\}) \in \mathscr{R}''$ **by** *force*

next

PAR2 *case*

Given that $T \xrightarrow{a(x)} T'$ *prove that there exists an* S' *such that for all* u *there exists an* S *such that* $P \mid R \xRightarrow{u:a(x)@S'} S$ *and* $(S, (Q \mid T')\{^u/_x\}) \in \mathscr{R}''$.

case(cPar2 T′)

from ⟨R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T⟩ ⟨T $\xrightarrow{a(x)}$ T′⟩ ⟨x ♯ R⟩ **obtain** R″

where L1: ∀u. ∃R′. R $\xRightarrow{u:a(x)@R''}$ R′ ∧ (R′, T′$\{^u/_x\}$) ∈ $\mathscr{R}'$

by(blast dest: weakSimE)

have ∀u. ∃S. P ∣ R $\xRightarrow{u:a(x)@P \mid R''}$ S ∧ (S, Q$\{^u/_x\}$ ∣ T′$\{^u/_x\}$) ∈ $\mathscr{R}''$

proof(rule allI)

fix u

from L1 **obtain** R′ **where** R $\xRightarrow{u:a(x)@R''}$ R′ **and** (R′, T′$\{^u/_x\}$) ∈ $\mathscr{R}'$ **by** blast

from ⟨R $\xRightarrow{u:a(x)@R''}$ R′⟩ ⟨x ♯ P⟩ **have** ParTrans: P ∣ R $\xRightarrow{u:a(x)@P \parallel R''}$ P ∣ R′

by(rule weakPar2Input)

moreover from ⟨(P, Q) ∈ $\mathscr{R}$⟩ ⟨(R′, T′$\{^u/_x\}$) ∈ $\mathscr{R}'$⟩

have (P ∣ R′, Q ∣ T′$\{^u/_x\}$) ∈ $\mathscr{R}''$ **by**(rule C1)

ultimately show ∃S. P ∣ R $\xRightarrow{u:a(x)@P \mid R''}$ S ∧ (S, Q$\{^u/_x\}$ ∣ T′$\{^u/_x\}$) ∈ $\mathscr{R}''$

using ⟨x ♯ Q⟩ **by**(force simp add: forget)

qed

thus ∃S′. ∀u. ∃S. P ∣ R $\xRightarrow{u:a(x)@S'}$ S ∧ (S, (Q ∣ T′)$\{^u/_x\}$) ∈ $\mathscr{R}''$ **by** force

qed

next

case(Free V α)

from ⟨Q ∣ T $\xrightarrow{\alpha}$ V⟩ **show** ∃S. P ∣ R $\xRightarrow{\widehat{\alpha}}$ S ∧ (S, V) ∈ $\mathscr{R}''$

proof(induct rule: parCasesF[**where** C=(P, R)]) — *Apply* PARF *inversion rule*

PAR1 *case*

Given that $Q \xrightarrow{\alpha} Q'$ *prove that there exists an* S *such that* $P \mid R \xRightarrow{\widehat{\alpha}} S$ *and* $(S, Q' \mid T) \in \mathscr{R}''$.

case(cPar1 Q′)

from ⟨P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q⟩ ⟨Q $\xrightarrow{\alpha}$ Q′⟩ **obtain** P′ **where** P $\xRightarrow{\widehat{\alpha}}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$

by(blast dest: weakSimE)

from ⟨P $\xRightarrow{\widehat{\alpha}}$ P′⟩ **have** Trans: P ∣ R $\xRightarrow{\widehat{\alpha}}$ P′ ∣ R **by**(rule weakPar1F)

moreover from ⟨(P′, Q′) ∈ $\mathscr{R}$⟩ ⟨(R, T) ∈ $\mathscr{R}'$⟩ **have** (P′ ∣ R, Q′ ∣ T) ∈ $\mathscr{R}''$

by(rule C1)

ultimately show ∃S. P ∣ R $\xRightarrow{\widehat{\alpha}}$ S ∧ (S, Q′ ∣ T) ∈ $\mathscr{R}''$ **by** blast

next

PAR2 *case*
Given that $T \xrightarrow{\alpha} T'$ *prove that there exists an S*
such that $P \mid R \stackrel{\widehat{\alpha}}{\Longrightarrow} S$ *and* $(S, Q \mid T') \in \mathscr{R}''$.

case(cPar2 T′)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\alpha}$ T′› **obtain** R′ **where** R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(blast dest: weakSimE)
from ‹R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ R′› **have** P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ P | R′ **by**(rule weakPar2F)
moreover from ‹(P, Q) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$› **have** (P | R′, Q | T′) ∈ $\mathscr{R}''$
by(rule C1)
ultimately show ∃S. P | R $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ S ∧ (S, Q | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM1 *case*
Given that $Q \xrightarrow{a(x)} Q'$ *and* $T \xrightarrow{\overline{a}b} T'$ *prove that there exists an S*
such that $P \mid R \stackrel{\widehat{\tau}}{\Longrightarrow} S$ *and* $(S, Q'\{^b/_x\} \mid T') \in \mathscr{R}''$.

case(cComm1 Q′ T′ a b x)
from ‹x ♯ (P, R)› **have** x ♯ P **by** simp
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{a(x)}$ Q′› ‹x ♯ P›
obtain P′ P″ **where** P $\xrightarrow{b:a(x)@P''}$ P′ **and** (P′, Q′{$^b/_x$}) ∈ $\mathscr{R}$
by(blast dest: weakSimE)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}b}$ T′› **obtain** R′ **where** R $\stackrel{\widehat{\overline{a}b}}{\Longrightarrow}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$
by(blast dest: weakSimE)
from ‹P $\xrightarrow{b:a(x)@P''}$ P′› ‹R $\stackrel{\widehat{\overline{a}b}}{\Longrightarrow}$ R′› **have** P | R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ P′ | R′ **by**(rule weakComm1)
moreover from ‹(P′, Q′{$^b/_x$}) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$›
have (P′ | R′, Q′{$^b/_x$} | T′) ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P | R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ S ∧ (S, Q′{$^b/_x$} | T′) ∈ $\mathscr{R}''$ **by** blast

next

COMM2 *case*
Given that $Q \xrightarrow{\overline{a}b} Q'$ *and* $T \xrightarrow{a(x)} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\tau}}{\Longrightarrow} S$ *and* $(S, Q' \mid T'\{^b/_x\}) \in \mathscr{R}''$.

case(cComm2 Q′ T′ a b x)
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}b}$ Q′› **obtain** P′ **where** P $\stackrel{\widehat{\overline{a}b}}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$
by(blast dest: weakSimE)
from ‹x ♯ (P, R)› **have** x ♯ R **by** simp
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{a(x)}$ T′› ‹x ♯ R›
obtain R′ R″ **where** R $\xRightarrow{b:a(x)@R''}$ R′ **and** (R′, T′$\{^b/_x\}$) ∈ $\mathscr{R}'$
by(blast dest: weakSimE)
from ‹P $\stackrel{\widehat{\overline{a}b}}{\Longrightarrow}$ P′› ‹R $\xRightarrow{b:a(x)@R''}$ R′› **have** P ∣ R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ P′ ∣ R′ **by**(rule weakComm2)
moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R′, T′$\{^b/_x\}$) ∈ $\mathscr{R}'$›
have (P′ ∣ R′, Q′ ∣ T′$\{^b/_x\}$) ∈ $\mathscr{R}''$ **by**(rule C1)
ultimately show ∃S. P ∣ R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ S ∧ (S, Q′ ∣ T′$\{^b/_x\}$) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE1 *case*
Given that $Q \xrightarrow{a(x)} Q'$ *and* $T \xrightarrow{\overline{a}(\nu x)} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\tau}}{\Longrightarrow} S$ *and* $(S, (\nu x)(Q'\{^y/_x\} \mid T')) \in \mathscr{R}''$.

case(cClose1 Q′ T′ a x y)
from ‹x ♯ (P, R)› ‹y ♯ (P, R)› **have** x ♯ P **and** y ♯ P **and** y ♯ R **by** auto
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{a(x)}$ Q′› ‹x ♯ P›
obtain P′ P″ **where** P $\xRightarrow{y:a(x)@P''}$ P′ **and** (P′, Q′$\{^y/_x\}$) ∈ $\mathscr{R}$
by(blast dest: weakSimE)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{\overline{a}(\nu y)}$ T′› ‹y ♯ R›
obtain R′ **where** R $\xRightarrow{\overline{a}(\nu y)}$ R′ **and** (R′, T′) ∈ $\mathscr{R}'$ **by**(blast dest: weakSimE)
from ‹P $\xRightarrow{y:a(x)@P''}$ P′› ‹R $\xRightarrow{\overline{a}(\nu y)}$ R′› ‹y ♯ P› ‹y ♯ R› **have** P ∣ R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ (ν y)(P′ ∣ R′)
by(rule weakClose1)
moreover from ‹(P′, Q′$\{^y/_x\}$) ∈ $\mathscr{R}$› ‹(R′, T′) ∈ $\mathscr{R}'$›
have ((ν y)(P′ ∣ R′), (ν y)(Q′$\{^y/_x\}$ ∣ T′)) ∈ $\mathscr{R}''$ **by**(blast intro: C1 C2)
ultimately show ∃S. P ∣ R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ S ∧ (S, (ν y)(Q′$\{^y/_x\}$ ∣ T′)) ∈ $\mathscr{R}''$ **by** blast

next

CLOSE2 *case*
Given that $Q \xrightarrow{\overline{a}(vx)} Q'$ *and* $T \xrightarrow{a(x)} T'$ *prove that there exists an S such that* $P \mid R \stackrel{\widehat{\tau}}{\Longrightarrow} S$ *and* $(S, (vx)(Q' \mid T'\{^y/_x\})) \in \mathscr{R}''$.

case(cClose2 Q′ T′ a x y)
from ‹x ♯ (P, R)› ‹y ♯ (P, R)› **have** x ♯ R **and** y ♯ P **and** y ♯ R **by** auto
from ‹P $\widehat{\rightsquigarrow}_{\mathscr{R}}$ Q› ‹Q $\xrightarrow{\overline{a}(vy)}$ Q′› ‹y ♯ P›
obtain P′ **where** P $\stackrel{\overline{a}(vy)}{\Longrightarrow}$ P′ **and** (P′, Q′) ∈ $\mathscr{R}$ **by**(blast dest: weakSimE)
from ‹R $\widehat{\rightsquigarrow}_{\mathscr{R}'}$ T› ‹T $\xrightarrow{a(x)}$ T′› ‹x ♯ R›
obtain R′ R″ **where** R $\stackrel{y:a(x)@R''}{\Longrightarrow}$ R′ **and** (R′, T′$\{^y/_x\}$) ∈ $\mathscr{R}'$
by(blast dest: weakSimE)
from ‹P $\stackrel{\overline{a}(vy)}{\Longrightarrow}$ P′› ‹R $\stackrel{y:a(x)@R''}{\Longrightarrow}$ R′› ‹y ♯ P› ‹y ♯ R› **have** P | R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ (vy)(P′ | R′)
by(rule weakClose2)
moreover from ‹(P′, Q′) ∈ $\mathscr{R}$› ‹(R′, T′$\{^y/_x\}$) ∈ $\mathscr{R}'$›
have ((vy)(P′ | R′), (vy)(Q′ | T′$\{^y/_x\}$)) ∈ $\mathscr{R}''$ **by**(blast intro: C1 C2)
ultimately show ∃S. P | R $\stackrel{\widehat{\tau}}{\Longrightarrow}$ S ∧ (S, (vy)(Q′ | T′$\{^y/_x\}$)) ∈ $\mathscr{R}''$ **by** blast
qed
qed

Definition 17.18 (τ-simulation)**.** *An agent P* τ*-simulating an agent Q preserving* $\mathscr{R}$ *is denoted* $P \rightsquigarrow_{\mathscr{R}} Q$. $P \rightsquigarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=}$

$(\forall Q'\, a\, x.\ Q \xrightarrow{\overline{a}(vx)} Q' \wedge x \sharp P \longrightarrow (\exists P'.\ P \stackrel{\overline{a}(vx)}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R})) \wedge$

$(\forall Q'\, a\, x.\ Q \xrightarrow{a(x)} Q' \wedge x \sharp P \longrightarrow (\exists P''.\ \forall u.\ \exists P'.\ P \stackrel{u:a(x)@P''}{\Longrightarrow} P' \wedge (P', Q'\{^u/_x\}) \in \mathscr{R})) \wedge$

$(\forall Q'\, \alpha.\ Q \xrightarrow{\alpha} Q' \longrightarrow (\exists P'.\ P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (P', Q') \in \mathscr{R}))$

We can now define weak late equivalence in the standard way – the derivatives of the simulations need not be weakly equivalent, but only weakly bisimilar.

Definition 17.19 (τ-bisimilarity)**.** *An agent P which is* τ*-bisimilar to an agent Q is denoted* $P \cong Q$.

$$P \cong Q \stackrel{\text{def}}{=} P \dot{\rightsquigarrow}_{\dot{\approx}} Q \wedge Q \dot{\rightsquigarrow}_{\dot{\approx}} P$$

Weak equivalence is preserved by all operators except Input. The lemmas are exactly the same as for early equivalence, but as for late bisimilarity, the proofs differ as input actions are mimicked differently.

Theorem 17.4. *Weak equivalence is preserved by all operators except Input.*

Proof. Similar to Theorem 16.1. □

17.5.4 Weak congruence

Finally, we obtain a congruence by closing weak equivalence under all possible substitutions.

Definition 17.20. *Two agents P and Q are weakly congruent, denoted $P \approx Q$ if they are τ-bisimilar for all possible substitutions.*

$$P \approx Q \stackrel{\text{def}}{=} \forall \sigma . P\sigma \cong Q\sigma$$

The lemmas required to prove that weak late congruence is preserved by Input are the same as for weak early congruence, but their proofs are different as the weak late semantics handles input actions differently.

Theorem 17.5. *Weak late congruence is a congruence.*

Proof. Similar to Theorem 16.2 □

18. Structural congruence

The structural congruence laws for the pi-calculus resembles the ones for CCS. The rules can be found in Figure 18.1 and include the standard abelian monoid laws for PAR and SUM as well as scope extension rules and an unfolding law for replication.

Even though the laws are nearly identical to the ones for CCS their proofs are different. The reason for this is the way the pi-calculus handles scoping. The OPEN and CLOSE rules change the scope of binders and hence the candidate relations for the bisimulation proofs must preserve scope migration. These proofs require infrastructure to reason about how binders behave as their scope is changed.

In this chapter we will prove that all structurally congruent terms are also bisimilar. The proofs will be presented in order of complexity, with the simpler rules involving the Parallel and Sum first, followed by the scope extension laws and finally the abelian monoid laws for Parallel.

18.1 Abelian monoid laws for Sum

These structural congruence laws for the Sum are relatively straightforward. Any path taken by the simulated process will discharge the other agents. The simulating process can then follow the same path.

18.1.1 Sum is commutative

The simulation lemma is symmetric, and only one simulation rule is required.

Lemma 18.1. *If* $Id \subseteq \mathscr{R}$ *then* $P + Q \hookrightarrow_{\mathscr{R}} Q + P$.

Proof. By the definition of $\hookrightarrow$ and case analysis of the possible transitions using the SUM inversion rules from Figure 17.2. The individual cases are discharged by the SUM1 and SUM2 rules. □

Using this lemma, we can prove that Sum is commutative.

Lemma 18.2. $P + Q \mathrel{\dot{\sim}} Q + P$

The structural congruence $\equiv$ is defined as the smallest congruence satisfying the following laws:

1. The abelian monoid laws for Parallel: commutativity $P \mid Q \equiv Q \mid P$, associativity $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and Nil as unit $P \mid \mathbf{0} \equiv P$.
2. The abelian monoid laws for Sum commutativity $P + Q \equiv Q + P$, associativity $(P + Q) + R \equiv P + (Q + R)$, and Nil as unit $P + \mathbf{0} \equiv P$.
3. The unfolding law $!P \equiv P \mid !P$.
4. The scope extension laws:

$$
\begin{array}{rcll}
(\nu x)\mathbf{0} & \equiv & \mathbf{0} & \\
(\nu x)(P + Q) & \equiv & (\nu x)P + (\nu x)Q & \\
(\nu x)(P \mid Q) & \equiv & P \mid (\nu x)Q & \text{if } x \sharp P \\
(\nu x)a(y).P & \equiv & a(y).(\nu x)P & \text{if } x \neq a \text{ and } x \neq y \\
(\nu x)\overline{a}b.P & \equiv & \overline{a}b.(\nu x)P & \text{if } x \neq a \text{ and } x \neq b \\
(\nu x)\tau.P & \equiv & \tau.((\nu x)P) & \\
(\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P &
\end{array}
$$

Figure 18.1: The definition of structural congruence.

Proof. By coinduction setting $\mathscr{X}$ to $\{(P + Q, Q + P), (Q + P, P + Q)\}$. The simulation is then proved using Lemma 18.1, and the fact that bisimulation is reflexive. The symmetry case follows directly from the fact that $\mathscr{X}$ is symmetric. □

18.1.2 Sum is associative

Lemma 18.3.

$$\frac{Id \subseteq \mathscr{R}}{(P + Q) + R \hookrightarrow_{\mathscr{R}} P + (Q + R)} \qquad \frac{Id \subseteq \mathscr{R}}{P + (Q + R) \hookrightarrow_{\mathscr{R}} (P + Q) + R}$$

Proof. By the definition of $\hookrightarrow$ and case analysis of the possible transitions using the SUM inversion rules from Figure 17.2. The individul cases are then discharged by the SUM1 and SUM2 rules. □

Lemma 18.4. $(P + Q) + R \mathrel{\dot{\sim}} P + (Q + R)$

Proof. By coinduction setting $\mathscr{X}$ to $\{((P + Q) + R, P + (Q + R)), (P + (Q + R), (P + Q) + R)\}$. The simulation is then proved using Lemma 18.3, and the fact that bisimulation is reflexive. The symmetry case follows directly from the fact that $\mathscr{X}$ is symmetric. □

18.1.3 Sum has Nil as unit

Lemma 18.5.

$$\frac{Id \subseteq \mathscr{R}}{P + \mathbf{0} \hookrightarrow_{\mathscr{R}} P} \qquad \frac{Id \subseteq \mathscr{R}}{P \hookrightarrow_{\mathscr{R}} P + \mathbf{0}}$$

Proof. By the definition of $\hookrightarrow$ and case analysis of the possible transitions using the SUM inversion rules from Figure 17.2. The individul cases are then discharged by the SUM1 rule. □

Lemma 18.6. $P + \mathbf{0} \dot{\sim} P$

Proof. By coinduction setting $\mathscr{X}$ to $\{(P, P + \mathbf{0}), (P + \mathbf{0}, P)\}$. The simulation is then proved using Lemma 18.5, and the fact that bisimulation is reflexive. The symmetry case follows directly from the fact that $\mathscr{X}$ is symmetric. □

18.2 Scope extension laws

Since agents in the pi-calculus can migrate the scope of their binders by communicating them to other processes, any rule involving Parallel must handle these scope changes, which requires a solid attention to detail.

18.2.1 Scope extension for Sum

The scope extension law for Sum allows a binder to distribute over the operator.

Lemma 18.7.

$$\frac{Id \subseteq \mathscr{R} \qquad eqvt\ \mathscr{R}}{(\nu x)P + (\nu x)Q \hookrightarrow_{\mathscr{R}} (\nu x)(P + Q)} \qquad \frac{Id \subseteq \mathscr{R} \qquad eqvt\ \mathscr{R}}{(\nu x)(P + Q) \hookrightarrow_{\mathscr{R}} (\nu x)P + (\nu x)Q}$$

Proof. By the definition of $\hookrightarrow$ and case analysis of the possible transitions using the SUM and SCOPE inversion rules from Figure 17.2. This proof requires more work than the previous ones for Sum due to the binder. The cases are discharged using the SUM1, SUM2, RES, and OPEN rules from the operational semantics. The candidate relation must be equivariant so that the binders in the agents can be freely alpha-converted. □

Lemma 18.8. $(\nu x)(P + Q) \dot{\sim} (\nu x)P + (\nu x)Q$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu x)(P + Q), (\nu x)P + (\nu x)Q) : True\} \cup$$
$$\{((\nu x)P + (\nu x)Q, (\nu x)(P + Q)) : True\}.$$

The reason that a simple tuple candidate relation, like the ones used in previous lemmas for Sum, cannot be used for this proof is the requirement from Lemma 18.7 that the candidate relation be equivariant. If $\mathscr{X}$ only contained the pair of agents in the conclusion of the lemma, alpha-converting them would make them fall out of $\mathscr{X}$.

The simulation part of the proof uses Lemma 18.7, the fact that $\mathscr{X}$ and bisimulation are equivariant, and the fact that bisimulation is reflexive. The symmetry case follows since $\mathscr{X}$ is symmetric. □

18.2.2 Discharging impossible transitions

Many of the structural congruence laws deal with agents of a very speciffic form. These agents are often restricted in which actions they can do. When proving that two agents are bisimilar it must be possible to determine which actions and agent can not do, in which case the other agent should not try to mimic it.

A necessary first step is to derive elimination rules for all impossible transitions and allow Isabelle to use those in its automatic heuristics. If such a transition appears in a case, then the whole case can be trivially removed from the proof.

Lemma 18.9. *The following transitions can never occur.*

$$\textit{If } \mathbf{0} \longmapsto Rs \textit{ then False.} \qquad \textit{If } \tau.P \xrightarrow{a\langle\!\langle x \rangle\!\rangle} P' \textit{ then False.}$$

$$\textit{If } \tau.P \xrightarrow{\overline{a}b} P' \textit{ then False.} \qquad \textit{If } a(x).P \xrightarrow{\alpha} P' \textit{ then False.}$$

$$\textit{If } a(x).P \xrightarrow{\alpha} P' \textit{ then False.} \qquad \textit{If } a(x).P \xrightarrow{\overline{b}(\nu y)} P' \textit{ then False.}$$

$$\textit{If } \overline{a}b.P \xrightarrow{c\langle\!\langle x \rangle\!\rangle} P' \textit{ then False.} \qquad \textit{If } \overline{a}b.P \xrightarrow{\tau} P' \textit{ then False.}$$

$$\textit{If } \overline{a}b.P \xrightarrow{\overline{c}d} P' \textit{ and } a \neq c \vee b \neq d \textit{ then False.}$$

$$\textit{If } \overline{a}b.P \xrightarrow{\alpha} P' \textit{ and } a \sharp \alpha \vee b \sharp \alpha \textit{ then False.}$$

$$\textit{If } a(x).P \xrightarrow{b\langle\!\langle y \rangle\!\rangle} P' \textit{ and } a \sharp b \textit{ then False.}$$

$$\textit{If } [a{=}b]P \longmapsto Rs \textit{ and } a \neq b \textit{ then False.}$$

$$\textit{If } [a{\neq}a]P \longmapsto Rs \textit{ then False.} \qquad \textit{If } (\nu x)\tau.P \xrightarrow{a\langle\!\langle y \rangle\!\rangle} P' \textit{ then False.}$$

$$\textit{If } (\nu x)\tau.P \xrightarrow{\overline{a}b} P' \textit{ then False.} \qquad \textit{If } (\nu x)a(y).P \xrightarrow{\alpha} P' \textit{ then False.}$$

$$\textit{If } (\nu x)a(y).P \xrightarrow{\overline{b}(\nu z)} P' \textit{ then False.} \qquad \textit{If } (\nu x)\overline{a}b.P \xrightarrow{\tau} P' \textit{ then False.}$$

$$\textit{If } (\nu x)\overline{a}b.P \xrightarrow{c(y)} P' \textit{ then False.} \qquad \textit{If } (\nu x)\overline{a}x.P \xrightarrow{\overline{b}y} P' \textit{ then False.}$$

$$\textit{If } (\nu x)\overline{x}b.P \longmapsto Rs \textit{ then False.} \qquad \textit{If } (\nu x)x(y).P \longmapsto Rs \textit{ then False.}$$

Proof. All of these sublemmas are proven by case analysis on their possible transitions. When proven in the order listed, Isabelle can use the already proven cases for the later ones. □

18.2.3 Restricting deadlocked agents

Lemma 18.10.

$$(\nu x)\mathbf{0} \hookrightarrow_{\mathscr{R}} \mathbf{0} \qquad\qquad \mathbf{0} \hookrightarrow_{\mathscr{R}} (\nu x)\mathbf{0}$$

Proof. Follows from the definition of $\hookrightarrow$, the SCOPE inversion rule from Figure 13.4.2 and the fact that the agent $\mathbf{0}$ has no transitions (Lemma 18.9). □

Lemma 18.11. $(\nu x)\mathbf{0} \mathrel{\dot{\sim}} \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)\mathbf{0},\ \mathbf{0}),\ (\mathbf{0},\ (\nu x)\mathbf{0})\}$ and Lemma 18.10. □

18.2.4 Scope extension for prefixes

The elimination rules from Lemma 18.9 simplify the proofs in this section considerably, as all actions not derivable from a given prefix will automatically be discharged. Moreover, the introduction rule for $\hookrightarrow$ ensures that any bound name appearing in the transitions is fresh for every other name and agent under consideration.

Lemma 18.12.

$$\frac{x \neq a \qquad x \neq y \qquad eqvt\ \mathscr{R} \qquad Id \subseteq \mathscr{R}}{(\nu x)a(y).P \hookrightarrow_{\mathscr{R}} a(y).(\nu x)P}$$

$$\frac{x \neq a \qquad x \neq y \qquad eqvt\ \mathscr{R} \qquad Id \subseteq \mathscr{R}}{a(y).(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)a(y).P}$$

Proof. The introduction rule for $\hookrightarrow$ ensures that the bound name in the input action is fresh for x, y, a, and P respectively. The possible cases are then derived by the SCOPE and INPUT inversion rules from Figure 17.2, and the impossible ones discharged by Lemma 18.9. The INPUT and RES rules from the operational semantics is then used to finish the proof. □

Lemma 18.13. *If $x \neq a$ and $x \neq y$ then $(\nu x)a(y).P \mathrel{\dot{\sim}} a(y).(\nu x)P$.*

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)a(y).P,\ a(y).(\nu x)P) : x \neq a \wedge x \neq y\} \cup \{(a(y).(\nu x)P,\ (\nu x)a(y).P) : x \neq a \wedge x \neq y\}$. The simulation case is then proven using Lemma 18.12, and the fact that $\mathscr{X}$ is equivariant and that bisimulation is reflexive. The symmetry case follows from that $\mathscr{X}$ is symmetric. □

Lemma 18.14.

$$\frac{x \neq a \qquad x \neq b \qquad Id \subseteq \mathscr{R}}{(\nu x)\overline{a}b.P \hookrightarrow_{\mathscr{R}} \overline{a}b.(\nu x)P} \qquad \frac{x \neq a \qquad x \neq b \qquad Id \subseteq \mathscr{R} \qquad eqvt\,\mathscr{R}}{\overline{a}b.(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)\overline{a}b.P}$$

Proof. The first case follows from the definition of $\hookrightarrow$, the OUTPUT inversion rule from Figure 17.2, and the RES and the OUTPUT rules from the operational semantics.

The second case uses the introduction rule for $\hookrightarrow$ ensuring that any bound name of the transitions is fresh for x. Since $x \neq b$, the OPEN-case, from the SCOPE inversion rule, is not applicable and is discharged automatically by Lemma 18.9. The remaining RES case is discharged using the RESF and the OUTPUT rules of the operational semantics.

□

These simulation lemmas assymetric in the sence that the candidate relation must be equivariant when the simulation is proven one way, but not the other. The case where equivariance is required is when Restriction is the topmost operator. The SCOPE inversion rule requires that the bound names of an action are fresh for the original agent, and the easiest way to ensure this is through the introduction rule for $\hookrightarrow$, which requires an equivariant candidate relation to do alpha-conversions.

Lemma 18.15. *If $x \neq a$ and $x \neq b$ then $(\nu x)\overline{a}b.P \dot{\sim} \overline{a}b.(\nu x)P$.*

Proof. By coinduction setting $\mathscr{X}$ to $\{((\nu x)\overline{a}b.P,\ \overline{a}b.(\nu x)P) : x \neq a \wedge x \neq b\} \cup \{(\overline{a}b.(\nu x)P,\ (\nu x)\overline{a}b.P) : x \neq a \wedge x \neq b\}$. The simulation case is proved by Lemma 18.14 and the facts that $\mathscr{X}$ is equivariant and that bisimulation is reflexive. The symmetry case follows from that $\mathscr{X}$ is symmetric. □

Lemma 18.16.

$$\frac{Id \subseteq \mathscr{R}}{(\nu x)\tau.P \hookrightarrow_{\mathscr{R}} \tau.((\nu x)P)} \qquad \frac{Id \subseteq \mathscr{R}}{\tau.((\nu x)P) \hookrightarrow_{\mathscr{R}} (\nu x)\tau.P}$$

Proof. Follows from the definition of $\hookrightarrow$, the SCOPE and TAU inversion rules from Figure 17.2, and the RESF and TAU rules from the operational semantics. □

Lemma 18.17. $(\nu x)\tau.P \dot{\sim} \tau.((\nu x)P)$

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)\tau.P,\ \tau.((\nu x)P)),\ (\tau.((\nu x)P),\ (\nu x)\tau.P)\}$. The simulation case is proven using Lemma 18.16 and the fact that bisimulation is reflexive. The symmetry case follows from that $\mathscr{X}$ is symmetric. □

Since the simulation lemmas does not require the candidate relation to be equivariant, we prove bisimilarity with a simpler candidate relation.

18.2.5 Restriction is commutative

The structural rule for that Restriction is commutative is symmetric and simulation needs only be proven one way.

Lemma 18.18.

$$\frac{\bigwedge c\ d\ Q.\ ((\nu c)(\nu d)Q, (\nu d)(\nu c)Q) \in \mathscr{R} \qquad Id \subseteq \mathscr{R} \qquad eqvt\ \mathscr{R}}{(\nu a)(\nu b)P \hookrightarrow_{\mathscr{R}} (\nu b)(\nu a)P}$$

Proof. If $a = b$, the proof is straightforward as simulation is reflexive. In the case that $a \neq b$ the introduction rule for $\hookrightarrow$ is used with the bound names of the actions being fresh for a, b, and P. The SCOPE inversion rule is used to strip away the binders.

This lemma is rather complex with the different combinations of the OPEN and RES rules. The requirement that $\mathscr{R}$ is reflexive is used when one or both of a and b are opened, to ensure that the derivatives stay in the candidate relation. The first assumption is used when the RES rule is used twice keeping both binders in the derivative – they must then be commuted in order for the derivative to stay in $\mathscr{R}$. □

Lemma 18.19. $(\nu x)(\nu y)P \dot{\sim} (\nu y)(\nu x)P$

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)(\nu y)P, (\nu y)(\nu x)P) : True\}$. The simulation proof follows from Lemma 18.18 and the fact that bisimulation is reflexive. The symmetry case follows from that $\mathscr{X}$ is symmetric. □

18.3 Bisimulation upto techniques

The scope migrating capabilities of the pi-calculus require more powerful candidate relations than we have encountered so far. Bisimulation upto techniques were covered in Sections 9.6 and 15.5.5. In this section we will work upto bisimulation, i.e. a proof being proven with the candidate relation $\mathscr{X}$ only requires the simulation to hold for the extended relation $\dot{\sim} \circ (\mathscr{X} \cup \dot{\sim}) \circ \dot{\sim}$. Moreover, the candidate relation itself need not be symmetric – it is enough if the extended relation is symmetric. This allows us to instantiate our bisimulation proofs with candidate relations of the same complexity as we have seen so far, and use the laws of structural congruence to rewrite the

derivatives to stay in the extended candidate relation. A new coinductive rule is derived by strengthening the general coinduction lemma, Lemma 14.7.

Lemma 18.20.

$$\frac{\begin{array}{l}(P,\,Q) \in \mathscr{Y} \qquad \mathit{eqvt}\ \mathscr{Y} \\ \bigwedge R\ S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{R \hookrightarrow_{\dot\sim \circ (\mathscr{Y} \cup \dot\sim) \circ \dot\sim} S} \quad \textsc{Simulation} \\ \bigwedge R\ S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{(S,\,R) \in \dot\sim \circ (\mathscr{Y} \cup \dot\sim) \circ \dot\sim} \quad \textsc{Symmetry}\end{array}}{P \dot\sim Q}$$

$$\frac{(P,\,Q) \in \mathscr{Y} \qquad \mathit{eqvt}\ \mathscr{Y} \qquad \bigwedge R\ S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{R \hookrightarrow_{\dot\sim \circ (\mathscr{Y} \cup \dot\sim) \circ \dot\sim} S} \qquad \bigwedge R\ S.\ \dfrac{(R,\,S) \in \mathscr{Y}}{(S,\,R) \in \dot\sim \circ (\mathscr{Y} \cup \dot\sim) \circ \dot\sim}}{P \dot\sim Q}$$

Proof. By coinduction using Lemma 14.7 setting $\mathscr{X}$ to $\dot\sim \circ (\mathscr{Y} \cup \dot\sim) \circ \dot\sim$. The proof is splint into one part considering $\dot\sim$ and another cosiderding $\mathscr{Y}$. In the first case, the proof follows trivially from transitivity of $\dot\sim$. In the second case, the assumptions in the lemma are used for simulation, the fact that simulation is transitive, and symmetry respectively. □

With this new coinduction lemma, we can prove the scope extension law for Parallel.

18.3.1 Scope extension for Parallel

Lemma 18.21.

$$\frac{\begin{array}{l} x \sharp P \qquad Id \subseteq \mathscr{R} \qquad eqvt\,\mathscr{R} \\ \bigwedge R\,S\,y.\ \dfrac{y \sharp R}{((\nu y)(R \mid S),\, R \mid (\nu y)S) \in \mathscr{R}}(1) \\ \bigwedge R\,S\,y\,z.\ \dfrac{y \sharp R}{((\nu y)(\nu z)(R \mid S),\, (\nu z)(R \mid (\nu y)S)) \in \mathscr{R}}(2) \end{array}}{(\nu x)(P \mid Q) \hookrightarrow_{\mathscr{R}} P \mid (\nu x)Q}$$

$$\frac{\begin{array}{l} x \sharp P \qquad Id \subseteq \mathscr{R} \qquad eqvt\,\mathscr{R} \\ \bigwedge R\,S\,y.\ \dfrac{y \sharp R}{(R \mid (\nu y)S,\, (\nu y)(R \mid S)) \in \mathscr{R}}(1) \\ \bigwedge R\,S\,y\,z.\ \dfrac{y \sharp R}{((\nu z)(R \mid (\nu y)S),\, (\nu y)(\nu z)(R \mid S)) \in \mathscr{R}}(2) \end{array}}{P \mid (\nu x)Q \hookrightarrow_{\mathscr{R}} (\nu x)(P \mid Q)}$$

Proof. Follows from $\hookrightarrow$-I, where any new bound name avoids x, P, and Q. The PAR and SCOPE inversion rules from Figure 17.2 are then used to generate all possible cases, and the PAR, COMM, RES, OPEN, and CLOSE rules from the operational semantics are used to discharge them. □

The requisites on the candidate relation deserves special mention. Assumption (1) in both lemmas is straightforward – the relation must preserve scope extension. Assumption (2) does the same, but it requires that the extended name propagates outside the binder generated by the CLOSE rules. It is this requisite which will require the use of bisimulation upto techniques.

Lemma 18.22. *If $x \sharp P$ then $(\nu x)(P \mid Q) \dot{\sim} P \mid (\nu x)Q$.*

Proof. By coinduction using Lemma 18.20 with $\mathscr{Y}$ set to

$$\{((\nu \widetilde{y})(\nu x)(P \mid Q),\, (\nu \widetilde{y})(P \mid (\nu x)Q)) : x \sharp P\} \cup$$
$$\{((\nu \widetilde{y})(P \mid (\nu x)Q),\, (\nu \widetilde{y})(\nu x)(P \mid Q)) : x \sharp P\}.$$

The candidate relation is symmetric, but the simulation case requires a bit of work. When a name x is extruded in the candidate relation it will end up inside the scope of the binders $\widetilde{y}$, but requisite (2) of the candidate relation in Lemma 18.21 require that x is extruded past the binders. The requisite is proven by instantiating $\widetilde{y}$ to y, and then using Lemma 18.19 to commute y with x. □

With these lemmas in place, we can prove the final structural congruence laws for Parallel.

18.4 Abelian monoid laws for Parallel

18.4.1 Parallel has Nil as unit

Lemma 18.23.

$$\frac{\bigwedge Q.\ (Q \mid \mathbf{0}\ ,\ Q) \in \mathscr{R}}{P \mid \mathbf{0} \hookrightarrow_{\mathscr{R}} P} \qquad \frac{\bigwedge Q.\ (Q, Q \mid \mathbf{0}\ ) \in \mathscr{R}}{P \hookrightarrow_{\mathscr{R}} P \mid \mathbf{0}}$$

Proof. Follows from the definition of $\hookrightarrow$, the PAR1 semantic rules for the first sublemma, and the PAR inversion rules for the second one. □

Lemma 18.24. $P \mid \mathbf{0} \stackrel{\cdot}{\sim} P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(P \mid \mathbf{0}\ ,\ P) : \mathit{True}\} \cup \{(P,\ P \mid \mathbf{0}\ ) : \mathit{True}\}.$$

The candidate relation is inherently symmetric, and the simulation cases follow from Lemma 18.23. □

18.4.2 Parallel is commutative

Lemma 18.25.

$$\frac{\bigwedge R\ S.\ (R \mid S\ ,\ S \mid R\ ) \in \mathscr{R} \qquad \bigwedge R\ S\ x.\ \dfrac{(R,\ S) \in \mathscr{R}}{((\nu x) R,\ (\nu x) S) \in \mathscr{R}}}{P \mid Q \hookrightarrow_{\mathscr{R}} Q \mid P}$$

Proof. Follows from the definition of $\hookrightarrow$. The PAR inversion rules are used to derive the relevant cases, and the PAR, COMM, and CLOSE semantic rules are used to discharge them. In all cases the symmetric semantic rule is used from the inversion rule used to derive the case; PAR2 for PAR1 and so on. □

Lemma 18.26. $P \mid Q \stackrel{\cdot}{\sim} Q \mid P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{((\nu \tilde{x})(P \mid Q\ ),\ (\nu \tilde{x})(Q \mid P\ )) : \mathit{True}\}.$$

The candidate relation is inherently symmetric, and the simulation case is proven using Lemma 18.25. □

18.4.3 *Parallel is associative*

The remaining proof is to show that Parallel is associative. Of all the proofs covered so far, this is the one with the most cases, with 18 cases to be proven in each direction of simulation. It turns out, however, that only one direction needs to be proven as bisimulation upto techniques will allow us to infer a symmetric simulation case automatically.

Lemma 18.27.

$$\bigwedge S\ T\ U.\ ((S \mid T) \mid U,\ S \mid (T \mid U)) \in \mathscr{R} \qquad \bigwedge S\ T\ x.\ \frac{(S,\ T) \in \mathscr{R}}{((\nu x)S,\ (\nu x)T) \in \mathscr{R}}$$

$$\bigwedge S\ T\ U\ x.\ \frac{x \sharp S}{((\nu x)((S \mid T) \mid U),\ S \mid (\nu x)(T \mid U)) \in \mathscr{R}}$$

$$\bigwedge S\ T\ U\ x.\ \frac{x \sharp U}{((\nu x)(S \mid T) \mid U,\ (\nu x)(S \mid (T \mid U))) \in \mathscr{R}}$$

$$\frac{}{(P \mid Q) \mid R \hookrightarrow_{\mathscr{R}} P \mid (Q \mid R)}$$

Proof. Follows from the $\hookrightarrow$-I, and the PAR inversion rules. At all steps newly occurring bound names avoid all other terms under consideration. The cases are then discharged using the PAR, COMM, and CLOSE semantic rules, and the requisites on $\mathscr{R}$ ensure that the derivatives remain in the candidate relation. There are a total of 18 cases. □

Lemma 18.28.

$$(P \mid Q) \mid R \mathrel{\dot{\sim}} P \mid (Q \mid R)$$

Proof. By coinduction using Lemma 18.20 with $\mathscr{Y}$ set to

$$\{((\nu \tilde{x})((P \mid Q) \mid R),\ (\nu \tilde{x})(P \mid (Q \mid R))) : \mathit{True}\}.$$

The simulation case is proven by Lemma 18.27. The candidate relation $\mathscr{Y}$ is not symmetric, but Lemmas 28.23 and 27.42 can be used to show the symmetry case. □

18.5 The unfolding law

Lemma 18.29.

$$\frac{Id \subseteq \mathscr{R}}{!P \hookrightarrow_{\mathscr{R}} P \mid !P} \qquad \frac{Id \subseteq \mathscr{R}}{P \mid !P \hookrightarrow_{\mathscr{R}} !P}$$

Proof. Follows from the definition of $\hookrightarrow$, the REPL inversion rule from Figure 13.4.2 and the REPL semantic rule. □

Lemma 18.30. $!P \dot{\sim} P \mid !P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(!P,\, P \mid !P),\, (P \mid !P,\, !P)\},$$

Lemma 18.29 and the fact that bisimulation is reflexive. □

18.6 Bisimilarity subsumes structural congruence

We finally prove our main theorem – that all structurally congruent terms are also bisimilar.

Theorem 18.1. *If* $P \equiv Q$ *then* $P \dot{\sim} Q$.

Proof.

Abelian monoid laws for Parallel: Follows from lemmas 18.24, 18.26, and 18.28.

Abelian monoid laws for Sum: Follows from lemmas 18.2, 18.4, and 18.6.

The unfolding law: Follows from lemma 18.30.

Scope extension laws: Follows from lemmas 18.8, 18.11, 18.13, 18.15, 18.17, 18.19, and 18.22.

□

The same result for strong equivalence follows directly.

Theorem 18.2. *If* $P \equiv Q$ *then* $P \sim Q$.

Proof. By Theorem 18.1 and the defintion of $\sim$ – any bound names of the agents must be alpha-converted to not clash with the substitution. □

19. An axiomatisation of strong late bisimilarity

The equivalences that have been covered thus far are bisimulation equivalences. A different way to characterise an equivalence is through a set of algebraic laws which do not use the operational semantics of the calculus at all.

In this chapter we use the algebraic laws from [58] and formalise the now well known result that they precisely capture late strong bisimilarity of the pi-calculus without Replication. The axioms can be found in Figure 19.1. The proof follows the standard structure and is partitioned into a soundness part – saying that each law is sound, i.e. that in all instances the right hand side is bisimilar to the left hand side, and a completeness part – saying that if two agents are bisimilar then there exists a proof of equivalence from the algebraic laws. As is typical in these situations the soundness proof is straightforward but tedious to write out in detail, while the completeness proof requires a bit more ingenuity.

19.1 Proof outline

The original manual proof of this particular result dates back to the very first presentation of the pi-calculus [58] and its sketch occupies about one page (on pages 67 and 68) in the journal. The result was not controversial since it is one of many similar results on complete axiomatisations in process algebras, the first being by Hennessy and Milner on CCS [44]. Variants of the proof have been used in variants of the calculus, but never written down in full detail.

We shall follow the general structure of that proof. The soundness part is straightforward – any agents equated by the set of axioms must also be bisimilar; most of these proofs have already been done in chapters 17 and 18. The completeness result is more involved. A subset of the agents are defined to be so called head normal forms. An agent is on head normal form if it is a sum of prefixes and is written

$$\alpha_1 . P_1 + \alpha_2 . P_2 + \cdots + \alpha_n . P_n$$

In a head normal form, the first step of the operational semantics is apparent from the syntactic structure of the agent. The subagents $\alpha_i.P_i$ are called the summands of the head normal form.

The completeness proof is done by induction on the depth of the agents, where the depth of an agent is an upper bound for the number of transitions that agent can do. We first show that each agent has a provably equivalent head normal form of no greater depth, and then that two bisimilar head normal forms have the same summands and are thus provably equivalent.

Existing formalisations of this proof contain several sweeping statements which require quite some attention to detail to verify formally. An example is the claim that if two head normal forms have provably equivalent summands then the agents are provably equivalent using laws for commutativity, associativity and idempotence of Sum (formalised as Lemma 19.25 in Section 19.3 below).

19.1.1 Formalisation outline

The formalisation of the completeness proof has the following structure.

1. Prove that every agent has a provably equivalent agent on head normal form of no greater depth.
2. Prove that for any agent P on head normal form, P can do a transition if and only if there exists a summand in P that can do the same transition.
3. Prove that the depth of any agent P is strictly greater than the depth of any of its derivatives.
4. Prove that two agents are provably equivalent if they have provably equivalent summands.
5. Prove that any two bisimilar agents on head normal form are provably equivalent by induction on the sum of their depth. The induction hypothesis can be applied to the derivatives as they have strictly smaller depth.

This sequence of steps will initially be proven for a subcalculus containing only the prefixes, Match, Mismatch and Sum. In Section 19.4 we will add Restriction, and in Section 19.5 we will add Parallel. These sections will focus on how to convert agents with these operators to head normal forms, and how to include them to the existing proof structure.

19.2 Soundness

Most of the lemmas needed to prove that the axiomatisation presented in Figure 19.1.is sound have already been proven in chapters 17 and 18. The exceptions are the CONGR2 rule, the rules for Match and Mismatch, the S2 rule for Sum, and the R1, R5 and R7 rules for Restriction.

Laws of equational reasoning

REFL $P \equiv_p P$

SYM *If* $P \equiv_p Q$ *then* $Q \equiv_p P$.

TRANS *If* $P \equiv_p Q$ *and* $Q \equiv_p R$ *then* $P \equiv_p R$.

Congruence laws

CONGR1 *If* $P \equiv_p Q$ *then*

$$\overline{a}b.P \equiv_p \overline{a}b.Q$$
$$\tau.P \equiv_p \tau.Q$$
$$P + R \equiv_p Q + R$$
$$(\nu x)P \equiv_p (\nu x)Q$$

CONGR2 *If* $\forall y \in supp\,(P, Q, x).\ P\{^y/_x\} \equiv_p Q\{^y/_x\}$ *then* $a(x).P \equiv_p a(x).Q$

Sum

S1 $P + \mathbf{0} \equiv_p P$

S2 $P + P \equiv_p P$

S3 $P + Q \equiv_p Q + P$

S4 $(P + Q) + R \equiv_p P + (Q + R)$

Restriction

R1 $(\nu x)P \equiv_p P$ *if* $x \sharp P$

R2 $(\nu x)(\nu y)P \equiv_p (\nu y)(\nu x)P$

R3 $(\nu x)(P + Q) \equiv_p (\nu x)P + (\nu x)Q$

R4 $(\nu x)a(y).P \equiv_p a(y).(\nu x)P$ *if* $x \neq a$ *and* $x \neq y$

R5 $(\nu x)x(y).P \equiv_p \mathbf{0}$

R6 $(\nu x)\overline{a}b.P \equiv_p \overline{a}b.(\nu x)P$ *if* $x \neq a$ *and* $x \neq b$

R7 $(\nu x)\overline{x}b.P \equiv_p \mathbf{0}$

R8 $(\nu x)\tau.P \equiv_p \tau.((\nu x)P)$

Match

M1 $[a{=}a]P \equiv_p P$

M2 $[a{=}b]P \equiv_p \mathbf{0}$ *if* $a \neq b$

Mismatch

MM1 $[a{\neq}a]P \equiv_p \mathbf{0}$

MM2 $[a{\neq}b]P \equiv_p P$ *if* $a \neq b$

Figure 19.1: Axiomatisation of strong late bisimulation for the pi-calculus, except for Parallel.

19.2.1 Match

To prove M1 sound we first need to prove a corresponding simulation lemma.

Lemma 19.1.

$$\frac{Id \subseteq \mathscr{R}}{[a{=}a]P \leadsto_{\mathscr{R}} P} \qquad \frac{Id \subseteq \mathscr{R}}{P \leadsto_{\mathscr{R}} [a{=}a]P}$$

Proof. In the first case, any action made by P can be mimicked by $[a{=}a]P$, and any derivatives will be in $\mathscr{R}$ as it contains the identity relation.

In the second case, the MATCH inversion rule from Figure 17.2 ensures that the only actions that $[a{=}a]P$ can do are ones which P can do. Again, the derivatives will be in $\mathscr{R}$, as $\mathscr{R}$ contains the identity relation. □

Lemma 19.2 (M1)**.** $[a{=}a]P \mathrel{\dot{\sim}} P$

Proof. By coinduction with $\mathscr{X}$ set to $\{([a = a]P,\ P),\ (P,\ [a = a]P)\}$. The candidate relation is symmetric, and the simulations are discharged by Lemma 19.1. □

To prove M2 sound we do not need a simulation lemma – all cases simplify away automatically.

Lemma 19.3 (M2)**.** *If* $a \neq b$ *then* $[a{=}b]P \mathrel{\dot{\sim}} \mathbf{0}$.

Proof. By coinduction with $\mathscr{X}$ set to $\{([a{=}b]P, \mathbf{0}),\ (\mathbf{0},\ [a{=}b]P)\}$. The candidate relation is symmetric, and all possible simulations can be discharged by Lemma 18.9 as neither $\mathbf{0}$ nor $[a{=}b]P$, when $a \neq b$, have any transitions. □

19.2.2 Mismatch

Lemma 19.4 (MM1)**.** $[a{\neq}a]P \mathrel{\dot{\sim}} \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to $\{([a{\neq}a]P, \mathbf{0}),\ (\mathbf{0},\ [a{\neq}a]P)\}$. The candidate relation is symmetric, and all possible simulations can be discharged by Lemma 18.9 as neither $\mathbf{0}$ nor $[a{\neq}a]P$, have any transitions. □

Lemma 19.5.

$$\frac{Id \subseteq \mathscr{R} \qquad a \neq b}{[a{\neq}b]P \leadsto_{\mathscr{R}} P} \qquad \frac{Id \subseteq \mathscr{R} \qquad a \neq b}{P \leadsto_{\mathscr{R}} [a{\neq}b]P}$$

Proof. In the first case, any action made by P can be mimicked by $[a{\neq}b]P$, since $a \neq b$. Any derivatives will be in $\mathscr{R}$ as it contains the identity relation.

In the second case, the MISMATCH inversion rule from Figure 17.2 ensures that the only actions that $[a \neq b]P$ can do are ones which P can do. Again, the derivatives will be in $\mathscr{R}$, as $\mathscr{R}$ contains the identity relation. □

Lemma 19.6 (MM2)**.** *If $a \neq b$ then $[a \neq b]P \dot{\sim} P$.*

Proof. By coinduction with $\mathscr{X}$ set to $\{([a \neq b]P,\ P),\ (P,\ [a \neq b]P)\}$. The candidate relation is symmetric, and the simulations are discharged by Lemma 19.5. □

19.2.3 Input

Strong bisimulation is not a congruence as it is not closed under the input-prefix, as was demonstrated in Section 17.3. The CONGR1-rule is a special case of this lemma, which only requires that equivalence is preserved by all substitutions of names which occur in the processes.

Lemma 19.7.

$$\frac{\bigwedge y.\ \dfrac{y \in supp\,(P, Q, x)}{(P\{^y/_x\},\ Q\{^y/_x\}) \in \mathscr{R}} \qquad eqvt\ \mathscr{R}}{a(x).P \hookrightarrow_{\mathscr{R}} a(x).Q}$$

Proof. The simulation definition requires all derivatives obtained by substituting y for any name x are in $\mathscr{R}$. The proof is done by case analysis on whether or not y is in the support of (P, Q, x). If it is, the proof follows directly from the assumptions. If not, we know that $y \sharp P$ and $y \sharp Q$ and by Lemma 13.8 that the substitution is equal to a permutation, and hence the derivatives are in $\mathscr{R}$, as $\mathscr{R}$ is equivariant. □

Lemma 19.8 (CONGR1)**.**

$$\frac{\bigwedge y.\ \dfrac{y \in supp\,(P, Q, x)}{P\{^y/_x\} \dot{\sim} Q\{^y/_x\}}}{a(x).P \dot{\sim} a(x).Q}$$

Proof. By coinduction with $\mathscr{X}$ set to $\{(a(x).P,\ a(x).Q) : \forall\, y \in supp\,(P, Q, x).\ P\{^y/_x\} \dot{\sim} Q\{^y/_x\}\}$. The candidate relation is symmetric since bisimulation is symmetric, and the simulation case follows immediately from Lemma 19.7. □

19.2.4 Sum

The only rule for Sum which does not follow from the laws of structural congruence is idempotence (S2).

Lemma 19.9.

$$\frac{Id \subseteq \mathscr{R}}{P \leadsto_{\mathscr{R}} P + P} \qquad \frac{Id \subseteq \mathscr{R}}{P + P \leadsto_{\mathscr{R}} P}$$

Proof. In the first case, any action made by P can be mimicked by $P + P$ using the SUM rule, and any derivatives will be in $\mathscr{R}$ as it contains the identity relation.

In the second case, the SUM inversion rule from Figure 17.2 ensures that the only actions that $P + P$ can do are ones which P can do. Again, the derivatives will be in $\mathscr{R}$, as $\mathscr{R}$ contains the identity relation. □

Lemma 19.10 (S1)**.** $P + P \mathrel{\dot{\sim}} P$

Proof. By coinduction with $\mathscr{X}$ set to $\{(P + P, P), (P, P + P)\}$. The candidate relation is symmetric, and the simulations are discharged by Lemma 19.9. □

19.2.5 Restriction

The laws for Restriction which must be proven sound are R1, R5, and R7.

Lemma 19.11 (R1)**.** *If* $x \sharp P$ *then* $(\nu x)P \mathrel{\dot{\sim}} P$.

Proof. Similar to Lemma 8.17, which is the corresponding lemma for CCS. □

Lemma 19.12 (R5)**.** $(\nu x)x(y).P \mathrel{\dot{\sim}} \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)x(y).P, \mathbf{0}), (\mathbf{0}, (\nu x)x(y).P)\}$. The candidate relation is symmetric, and all possible simulations can be discharged by Lemma 18.9 as neither $\mathbf{0}$ nor $(\nu x)x(y).P$, have any transitions. □

Lemma 19.13 (R7)**.** $(\nu x)\overline{x}b.P \mathrel{\dot{\sim}} \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to $\{((\nu x)\overline{x}b.P, \mathbf{0}), (\mathbf{0}, (\nu x)\overline{x}b.P)\}$. The candidate relation is symmetric, and all possible simulations can be discharged by Lemma 18.9 as neither $\mathbf{0}$ nor $(\nu x)\overline{x}b.P$, have any transitions. □

19.2.6 Soundness

With these lemmas in place, we can prove the soundness result.

Theorem 19.1. *If $P \equiv_p Q$ then $P \dot{\sim} Q$.*

Proof. Proof by induction on the inference rules of provable equality from Figure 19.1. Every case is listed with the lemma which proves that it is sound.

Equivalence laws

- REFL Lemma 17.10
- SYM Lemma 17.10
- TRANS Lemma 17.10

Congruence laws

- CONGR1 All axioms follow from Theorem 14.1.
- CONGR2 Lemma 19.8.

Sum

- S1 Lemma 18.6
- S2 Lemma 19.10
- S3 Lemma 18.2
- S4 Lemma 18.4

Match

- M1 Lemma 19.2
- M2 Lemma 19.3

Mismatch

- MM1 Lemma 19.4
- MM2 Lemma 19.6

Restriction

- R1 Lemma 19.11
- R2 Lemma 18.19
- R3 Lemma 18.8
- R4 Lemma 18.13
- R5 Lemma 19.12
- R6 Lemma 18.15
- R7 Lemma 19.13
- R8 Lemma 18.17

□

19.3 Completeness

We begin by restricting attention to a subcalculus which does not contain Parallel or Restriction, and introduce a predicate *valid P* to mean that P is in the subcalculus. This predicate will be extended as support for more operators are added.

Definition 19.14 (valid)**.**

$$\begin{aligned} valid\ \mathbf{0} &= True \\ valid\ (\tau.P) &= valid\ a(x).P = valid\ \overline{a}b.P = valid\ P \\ valid\ (P+Q) &= valid\ P \wedge valid\ Q \\ valid\ \_ &= False \end{aligned}$$

The first step of the proof is to define what it means for an agent to be on head normal form, or hnf for short. An agent is on hnf if it is a sum of prefixed agents.

Definition 19.15 (hnf)**.**

$$\begin{aligned} hnf\ \mathbf{0} &= hnf\ (\tau.P) = hnf\ a(x).P = hnf\ \overline{a}b.P = True \\ hnf\ (P + Q) &= hnf\ P \wedge hnf\ Q \wedge P \neq \mathbf{0} \wedge Q \neq \mathbf{0} \\ hnf\ \_ &= False \end{aligned}$$

We will reason about hnfs in terms of their summands. The summands of a term is the set of its prefixed subterms that are composed by Sum.

Definition 19.16 (summands)**.**

$$\begin{aligned} summands\ (\tau.P) &= \{\tau.P\} \\ summands\ a(x).P &= \{a(x).P\} \\ summands\ \overline{a}b.P &= \{\overline{a}b.P\} \\ summands\ (P + Q) &= summands\ P \cup summands\ Q \\ summands\ \_ &= \emptyset \end{aligned}$$

For many proofs, it is convenient to reason about summands as sets of equivalence classes, corresponding to provable equivalence by the axioms. In order to mimic this behaviour in Isabelle, we introduce the notion of a unique head normal form. Intuitively, an agent is on unique head normal form, or uhnf, if it is on head normal form and no summand is provably equivalent to any other.

Definition 19.17 (uhnf)**.**

$$\begin{aligned} &uhnf\ P \stackrel{\text{def}}{=} \\ &hnf\ P \wedge (\forall\ R \in summands\ P.\ \forall\ R' \in summands\ P.\ R \neq R' \longrightarrow \neg\ R \equiv_p R') \end{aligned}$$

The main proof is by induction over the depth of agents. Intuitively, the depth of a term is an upper bound of the number of transitions it can make. We define the following function:

Definition 19.18 (depth)**.**

$$\begin{aligned} depth\ \mathbf{0} &= 0 \\ depth\ (\tau.P) &= depth\ a(x).P = depth\ \overline{a}b.P = 1 + depth\ P \\ depth\ ([a{=}b]P) &= depth\ ([a{\neq}b]P) = depth\ ((\nu x)P) = depth\ P \\ depth\ (P + Q) &= max\ (depth\ P)\ (depth\ Q) \\ depth\ (P \mid Q) &= depth\ P + depth\ Q \end{aligned}$$

The following lemma ensures that adding an agent which is provably equivalent to a summand of an agent has no effect.

Lemma 19.19. *If $Q \in$ summands P and $Q \equiv_p Q'$ then $P + Q' \equiv_p P$.*

Proof. By induction on the structure of P. □

The completeness proof follows the pattern described in Section 19.1.1. The first step is to prove that for every agent, there exists a provably equivalent agent on unique head normal form of no greater depth. Prefixed agents are trivially on uhnf, and any occurrences of Match or Mismatch can always be removed by case analysis on the equality, or inequality, of the names in the condition. The remaining operator is Sum.

Lemma 19.20. *Convert Sum to uhnf.*

If uhnf P and uhnf Q and valid P and valid Q then
$\exists R.$ uhnf $R \wedge$ valid $R \wedge P + Q \equiv_p R \wedge$ depth $R \leq$ depth $(P + Q)$.

Proof. By induction on the structure of P. In the prefix cases, Lemma 19.19 is used to filter out the terms which are provably equivalent to some term in the summands of Q. Since P is on uhnf, the only relevant cases are the Nil case, the prefix cases, and the Sum case.

Nil case ($P = \mathbf{0}$): Follows immediately by setting R to Q, the S1 rule, and the fact that the depth of $\mathbf{0}$ is 0.

Output case ($P = \overline{a}b.P'$): If $Q = \mathbf{0}$ then set R to P, otherwise if there is a $Q' \in$ *summands* Q such that $Q' \equiv_p P$, then $Q + P \equiv_p Q$ by Lemma 19.19, so set R to Q, otherwise set R to $P + Q$.

The other prefix cases follow the same pattern.

Sum case ($P = P_1 + P_2$): Since P is on uhnf, we know that P_1 and P_2 are on uhnf.

- Since Q is on uhnf, we obtain from the induction hypothesis an S where *uhnf* S, $P_2 + Q \equiv_p S$, and *depth* $S \leq$ *depth* $(P_2 + Q)$.
- Since S is on uhnf, we obtain from the induction hypothesis an R' where *uhnf* R', $P_1 + S \equiv_p R'$, and *depth* $R' \leq$ *depth* $(P_1 + S)$.
- The proof is concluded by setting R to R'.
  - We already know that R' is valid, and on uhnf.
  - Since $P_2 + Q \equiv_p S$ and $P_1 + S \equiv_p R'$, we have that $(P_1 + P_2) + Q \equiv_p R'$
  - Since *depth* $S \leq$ *depth* $(P_2 + Q)$ and *depth* $R' \leq$ *depth* $(P_1 + S)$ we have that *depth* $R' \leq$ *depth* $((P_1 + P_2) + Q)$.

□

We now prove the main lemma which states that for every valid agent, there exists a provably equivalent agent on uhnf.

Lemma 19.21.

$$\textit{If valid } P \textit{ then } \exists Q.\ \textit{uhnf } Q \wedge \textit{valid } Q \wedge Q \equiv_p P \wedge \textit{depth } Q \leq \textit{depth } P.$$

Proof. By induction on the structure of P. When P is of the form $P_1 + P_2$, Lemma 19.20 is used. □

The next step is to connect provably equivalent agents and the transition system through the summands.

Lemma 19.22.

$$\textit{If hnf } P \textit{ then } P \xrightarrow{\tau} P' = (\tau.P' \in \textit{summands } P).$$
$$\textit{If hnf } P \textit{ then } P \xrightarrow{a(x)} P' = (a(x).P' \in \textit{summands } P).$$
$$\textit{If hnf } P \textit{ then } P \xrightarrow{\overline{a}b} P' = (\overline{a}b.P' \in \textit{summands } P).$$

Proof. By induction on the structure of P. □

We proceed to prove that two agents are provably equivalent if their summands are provably equivalent. The proof is by induction over the summands of the agents. The following lemma takes a summand of an agent, pulls it to the leftmost position of the agent, using the S2 and S3 rules, and removes any provably equivalent summands with Lemma 19.19.

Lemma 19.23.

$$\textit{If } P \in \textit{summands } Q \textit{ and uhnf } Q \textit{ then}$$
$$\exists Q'.\ P + Q' \equiv_p Q \wedge$$
$$\textit{summands } Q' = \textit{summands } Q - \{P' : P' \in \textit{summands } Q \wedge P' \equiv_p P\} \wedge \textit{uhnf } Q'.$$

Proof. By induction on the structure of Q. □

By extracting a summand from an agent, the set of summand shrinks. In order to conduct induction on sets, we must prove that exactly one element is removed from the summand set. This holds for agents on uhnf.

Lemma 19.24.

$$\textit{If } P \in \textit{summands } Q \textit{ and uhnf } Q \textit{ then}$$
$$\textit{summands } Q - \{P' : P' \in \textit{summands } Q \wedge P' \equiv_p P\} = \textit{summands } Q - \{P\}.$$

Proof. Follows directly from the definition of uhnf and the REFL rule. □

With these lemmas in place, we can prove that two agents are provably equivalent if their summands are provably equivalent.

Lemma 19.25.

$$\frac{\begin{array}{l} uhnf\,P \qquad uhnf\,Q \\ \forall\, P' \in summands\,P.\ \exists\, Q' \in summands\,Q.\ P' \equiv_p Q' \\ \forall\, Q' \in summands\,Q.\ \exists\, P' \in summands\,P.\ Q' \equiv_p P' \end{array}}{P \equiv_p Q}$$

Proof. By induction on $summands\,P$

Base case ($summands\,P = \emptyset$)**:**

- From $\forall\, Q' \in summands\ Q.\ \exists\, P' \in summands\ P.\ Q' \equiv_p P'$ and $summands\,P = \emptyset$ we have that $summands\,Q = \emptyset$.
- From $summands\,P = \emptyset$ and $uhnf\,P$ we have that $P = \mathbf{0}$.
- Moreover from $summands\,Q = \emptyset$ and $uhnf\,Q$ we have that $Q = \mathbf{0}$.
- Finally we have that $P \equiv_p Q$ by REFL.

Inductive step ($summands\,P = \{P'\} \cup \mathscr{L}$)**:**

- From Lemmas 19.23 and 19.24 we obtain a P'' such that $P' + P'' \equiv_p P$, $summands\,P = summands\,P'' - \{P'\}$, and $uhnf\,P''$.
- We have that $\mathscr{L} = summands\,P''$.
- From $\forall\, P' \in summands\ P.\ \exists\, Q' \in summands\ Q.\ P' \equiv_p Q'$ and $summands\,P = \{P'\} \cup \mathscr{L}$ we obtain a Q' such that $Q' \in summands\,Q$ and $P' \equiv_p Q'$.
- From Lemmas 19.23 and 19.24 we obtain a Q'' such that $Q' + Q'' \equiv_p Q$, $summands\,Q = summands\,Q'' - \{Q'\}$, and $uhnf\,Q''$.
- From the assumptions, $uhnf\,P'$, $uhnf\,Q'$, and $P' \equiv_p Q'$ we have that

$$\forall\, P'' \in summands\,P'.\ \exists\, Q'' \in summands\,Q'.\ P'' \equiv_p Q''$$

and

$$\forall\, Q'' \in summands\,Q'.\ \exists\, P'' \in summands\,P'.\ Q'' \equiv_p P''.$$

- With $uhnf\,P'$ and $uhnf\,Q'$ we have that $P'' \equiv_p Q''$ by the induction hypothesis.
- With $P' \equiv_p Q'$, $P' + P'' \equiv_p P$, and $Q' + Q'' \equiv_p Q$ we have that $P \equiv_p Q$ by the laws for Sum, and SYM and TRANS.

□

The premises of the lemma display striking similarities with bisimilarity – for all summands in an agent, there must exists a summand in the other agent such that they are provably equivalent, and vice versa. This similarity will be exploited in the lemma which proves the completeness result.

In order to do induction over the depth of agents, we need to know that agents have strictly greater depth than their derivatives.

Lemma 19.26.

$$\textit{If hnf}\, P \textit{ and } P \xrightarrow{\overline{a}b} P' \textit{ then } \textit{depth}\, P' < \textit{depth}\, P.$$
$$\textit{If hnf}\, P \textit{ and } P \xrightarrow{a(x)} P' \textit{ then } \textit{depth}\, P' < \textit{depth}\, P.$$
$$\textit{If hnf}\, P \textit{ and } P \xrightarrow{\tau} P' \textit{ then } \textit{depth}\, P' < \textit{depth}\, P.$$

Proof. By induction on the derivation of the transitions. □

We can now prove that the axiomatisation is complete.

Lemma 19.27. *If valid P and valid Q and* $P \dot{\sim} Q$ *then* $P \equiv_p Q$.

Proof. We begin by using Lemma 19.21 to obtain provably equivalent versions of P and Q on uhnf. We then use Lemma 19.25, and hence we must prove that for all summands in P there exists a provably equivalent summand of Q and vice versa. Since bisimilarity is symmetric, we only need to prove this property one way – the symmetric version will be inferred automatically.

The proof is done by induction on $\textit{depth}\, P + \textit{depth}\, Q$.

Base case ($\textit{depth}\, P + \textit{depth}\, Q = 0$)**:** The only case where this can hold is if $P = \mathbf{0}$ and $Q = \mathbf{0}$, and hence $P \equiv Q$ by REFL.

Inductive step ($\textit{depth}\, P + \textit{depth}\, Q \leq n$)**:** We pick an arbitrary summand $\alpha.P'$ from the summands of P, we must prove that there exists a Q' such that $\alpha.Q'$ is in the summands of Q. We begin with the case where α is not an input prefix.

- Since $\alpha.P' \in \textit{summands}\, P$, we have by Lemma 19.22 that $P \xrightarrow{\alpha} P'$.
- Since $P \dot{\sim} Q$ we obtain a Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \dot{\sim} Q'$.
- From Lemma 19.21 we obtain a P'' and a Q'' where $P' \equiv_p P''$, $Q' \equiv_p Q''$, $\textit{depth}\, P'' \leq \textit{depth}\, P'$, and $\textit{depth}\, Q'' \leq \textit{depth}\, Q'$.
- With Lemma 19.26 we have that $\textit{depth}\, P'' + \textit{depth}\, Q'' < n$
- Moreover with $P' \equiv_p P''$, $Q' \equiv_p Q''$, and $P' \dot{\sim} Q'$ we have that $P'' \dot{\sim} Q''$ using Theorem 19.1, and symmetry and transitivity of bisimilarity.
- Finally we get that $P'' \equiv Q''$ from the induction hypothesis.
- With $P' \equiv_p P''$ and $Q' \equiv_p Q''$ we have that $P' \equiv_p Q'$ by SYM and TRANS.
- Hence $\alpha.P' \equiv_p \alpha.Q'$ by CONGR1.

If α is an input prefix of the form $a(x)$, we prove that $P\{^y/_x\} \equiv_p Q\{^y/_x\}$ for all y such that $y \in \textit{supp}\,(P, Q, x)$. The strategy is the same as for free actions, and we get that $P\{^u/_x\} \equiv_p Q\{^u/_x\}$ for all u, more specifically for $u = y$, hence $P\{^y/_x\} \equiv_p Q\{^y/_x\}$, and hence $a(x).P' \equiv_p a(x).Q'$ by CONGR2.

□

19.4 Adding Restriction

We have already proven that the axioms involving Restriction are sound. It remains to prove that they are complete.

The first step is to augment the *valid* predicate so that it encompasses restricted agents.

$$valid\ ((\nu x)P) = valid\ P$$

For the completeness proof we need to expand the definitions of summands and hnf. With Restriction there is a possibility of bound outputs. A process which generates a bound output is of form $(\nu x)\overline{a}x.P$ where $x \neq a$. We extend the definition of hnf with the following case:

$$hnf\ ((\nu x)P) = (\exists a\ P'.\ a \neq x \wedge P = \overline{a}x.P')$$

We also extend the definition of summands with:

$$summands\ ((\nu x)P) = (\mathit{if}\ \exists a\ P'.\ a \neq x \wedge P = \overline{a}x.P'\ \mathit{then}\ \{(\nu x)P\}\ \mathit{else}\ \emptyset)$$

Finally we expand the proofs of Lemma 19.21, which states that for every agent there exists a provably equivalent agent on uhnf. We need the following auxiliary lemma:

Lemma 19.28.

If uhnf P and valid P then
$\exists P'.\ uhnf\ P' \wedge valid\ P' \wedge (\nu x)P \equiv_p P' \wedge depth\ P' \leq depth\ ((\nu x)P)$.

Proof. By induction on the structure of P. □

The extension to Lemma 19.21 now becomes quite simple.

We will also need an expanded version of Lemma 19.22 to connect summands and transitions.

$$\mathit{If}\ hnf\ P\ \mathit{and}\ a \neq x\ \mathit{then}\ P \xrightarrow{\overline{a}(\nu x)} P' = ((\nu x)\overline{a}x.P' \in summands\ P).$$

The proof for this addition has the same structure as the rest of the lemma.

exp
Let $P = \sum_i \alpha_i . P_i$ and $Q = \sum_j \beta_j . Q_j$ where $\text{bn}(\alpha_i) \cap \text{fn}(Q) = \emptyset$ and $\text{bn}(\beta_j) \cap \text{fn}(P) = \emptyset$ for all i, j. Then

$$P \mid Q \quad = \quad \sum_i \alpha_i . (P_i \mid Q) + \sum_j \beta_j . (P \mid Q_j) + \sum_{\alpha_i \, comp \, \beta_j} \tau . R_{ij}$$

where the relation $\alpha_i comp \beta_j$ and R_{ij} are defined through the follwing four cases:
1. $\alpha_i = a(x)$ and $\beta_j = \overline{a}u$ in which case $R_{ij} = P_i\{^x/_u\} \mid Q_j$,
2. $\alpha_i = a(x)$ and $\beta_j = (\nu u)\overline{a}u$ in which case $R_{ij} = (\nu u)(P_i\{^x/_u\} \mid Q_j)$,
3. The converse of 1,
4. The converse of 2.

Figure 19.2: The traditional expansion law for strong bisimilarity. Here α_i and β_j range over the prefix forms (Input, Output and Silent) and also over the combination of bound output prefix $(\nu u)\overline{a}u$.

19.5 Adding Parallel

We complete our proof by adding Parallel. To do this we must encode the expansion law, which can be found in Figure 19.2 in Isabelle. Unfortunately, the law as presented here and elsewhere is not completely formally correct. The reason is that it makes use of a function Σ which takes a set of agents as a parameter and returns the sum of them. However, such a function cannot be defined in the usual inductive way (as $\Sigma\{P\} \cup S = P + \Sigma S$) since this does not define Σ uniquely; elements can be pulled from a set in different order, resulting in different order of the summands. The reason the expansion law nevertheless is considered valid is that it implicitly operates on equivalence classes of agents up to $\equiv_p$ and here Sum is idempotent, associative and commutative. As there is no easy way to incorporate such functions in Isabelle we need to be a little more creative. We define a relation $\mathscr{S}$ of type *pi* × *pi set* with the intuition that if $\mathscr{F}$ is a set of agents then $\mathscr{S}(P, \mathscr{F})$ if P can be obtained as a sum of the agents in $\mathscr{F}$.

Definition 19.29.

$$\frac{}{(\mathbf{0}, \emptyset) \in \mathscr{S}} \qquad \frac{Q \in \mathscr{F} \qquad (P, \mathscr{F} - \{Q\}) \in \mathscr{S}}{(P + Q, \mathscr{F}) \in \mathscr{S}}$$

We define the set of all agents generated by the expansion law.

Definition 19.30. *The predicate expandSet takes two agents P and Q as arguments, and returns the set of all their possible expansions.*

expandSet P Q $\stackrel{\text{def}}{=}$
$\{\tau.(P' \mid Q) : \tau.P' \in \textit{summands } P\} \cup \{\tau.(P \mid Q') : \tau.Q' \in \textit{summands } Q\} \cup$

$\{\overline{a}b.P' \mid Q : \overline{a}b.P' \in \mathit{summands}\ P\} \cup$
$\{\overline{a}b.P \mid Q' : \overline{a}b.Q' \in \mathit{summands}\ Q\} \cup$
$\{a(x).P' \mid Q : a(x).P' \in \mathit{summands}\ P \wedge x \sharp Q\} \cup$
$\{a(x).P \mid Q' : a(x).Q' \in \mathit{summands}\ Q \wedge x \sharp P\} \cup$
$\{(\nu x)\overline{a}x.P' \mid Q : (\nu x)\overline{a}x.P' \in \mathit{summands}\ P \wedge x \sharp Q\} \cup$
$\{(\nu x)\overline{a}x.P \mid Q' : (\nu x)\overline{a}x.Q' \in \mathit{summands}\ Q \wedge x \sharp P\} \cup$
$\{\tau.(P'\{^b/_x\} \mid Q') : \exists a.\ a(x).P' \in \mathit{summands}\ P \wedge \overline{a}b.Q' \in \mathit{summands}\ Q\} \cup$
$\{\tau.(P' \mid Q'\{^b/_x\}) : \exists a.\ \overline{a}b.P' \in \mathit{summands}\ P \wedge a(x).Q' \in \mathit{summands}\ Q\} \cup$
$\{\tau.((\nu y)(P'\{^y/_x\} \mid Q')) : \exists a.\ a(x).P' \in \mathit{summands}\ P \wedge (\nu y)\overline{a}y.Q' \in \mathit{summands}\ Q \wedge y \sharp P\} \cup$
$\{\tau.((\nu y)(P' \mid Q'\{^y/_x\})) : \exists a.\ (\nu y)\overline{a}y.P' \in \mathit{summands}\ P \wedge a(x).Q' \in \mathit{summands}\ Q \wedge y \sharp Q\}$

To complete the axiomatisation we add the following two axioms:

EXPAND	*If* $(R, \mathit{expandSet}\ P\ Q) \in \mathscr{S}$, *hnf* P, *and hnf* Q *then* $P \mid Q \equiv_p R$
CONGRPAR	*If* $P \equiv_p P'$ *and* $Q \equiv_p Q'$ *then* $P \mid Q \equiv_p P' \mid Q'$.

19.5.1 Soundness

The proof that CONGRPAR is sound follows from Theorems 17.1 and 18.1.

To prove that the EXPAND axiom is sound, we need the following lemmas

Lemma 19.31. *If* $(P, \mathscr{F}) \in \mathscr{S}$ *and* $Q \in \mathscr{F}$ *and* $Q \longmapsto V$ *then* $P \longmapsto V$.

Proof. By induction on the construction of $\mathscr{S}$. □

Lemma 19.32. *If* $(R, \mathscr{F}) \in \mathscr{S}$ *and* $R \longmapsto V$ *then* $\exists P \in \mathscr{F}.\ P \longmapsto V$.

Proof. By induction on the construction of $\mathscr{S}$. The base case is trivially true since $\mathscr{S}$ is empty. In the inductive step a case analysis is made on whether or not the inserted term can do the transition to V. If so, that term is picked; otherwise the term is obtained through the induction hypothesis. □

Lemma 19.33. *If* $(R, \mathit{expandSet}\ P\ Q) \in \mathscr{S}$ *and hnf* P *and hnf* Q *then*

$$\begin{aligned}
& P \mid Q \xrightarrow{\tau} P' = R \xrightarrow{\tau} P' \\
\textit{and}\quad & P \mid Q \xrightarrow{\overline{a}b} P' = R \xrightarrow{\overline{a}b} P' \\
\textit{and}\quad & P \mid Q \xrightarrow{a(x)} P' = R \xrightarrow{a(x)} P' \\
\textit{and}\quad & P \mid Q \xrightarrow{\overline{a}(\nu x)} P' = R \xrightarrow{\overline{a}(\nu x)} P'
\end{aligned}$$

Proof. In this proof, Lemma 19.22 is used to connect summands to transitions and vice versa.

$\Longrightarrow$ By case analysis on the transition done by $P \mid Q$. Each case matches one construction in the expansion law and lemma 19.31 is used to prove that R can make the desired transition.

$\Longleftarrow$ Lemma 19.32 is used to find the summand in R in *expandSet P Q* which can make the transition. A case analysis of R is made and each case is matched to its corresponding rule in the operational semantics.

□

We can now prove soundness.

Lemma 19.34. *If* $(R, \mathit{expandSet}\ P\ Q) \in \mathscr{S}$ *and* $\mathit{hnf}\ P$ *and* $\mathit{hnf}\ Q$ *then* $P \mid Q \dot{\sim} R$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(P \mid Q, R) : (R, \mathit{expandSet}\ P\ Q) \in \mathscr{S} \wedge \mathit{hnf}\ P \wedge \mathit{hnf}\ Q\} \cup \{(R, P \mid Q) : (R, \mathit{expandSet}\ P\ Q) \in \mathscr{S} \wedge \mathit{hnf}\ P \wedge \mathit{hnf}\ Q\}$. The relation is symmetric, and the simulation case follows from Lemma 19.33.

□

19.5.2 Completeness

The first step is to augment the *valid* predicate so that it encompasses parallel agents.

$$\mathit{valid}\ (P \mid Q) = (\mathit{valid}\ P \wedge \mathit{valid}\ Q)$$

No additions need to be made to the actual completeness proof as it operates on agents on uhnf, and as such does not contain Parallel at all. The lemmas that need to be augmented are only the ones which prove that every agent has a provably equivalent uhnf of no greater depth.

We need two auxiliary lemmas.

Lemma 19.35.

If $(P, \mathscr{F}) \in \mathscr{S}$ *and* $\forall P{\in}\mathscr{F}.\ \mathit{uhnf}\ P \wedge \mathit{valid}\ P$ *then*
$\exists P'.\ \mathit{uhnf}\ P' \wedge \mathit{valid}\ P' \wedge P \equiv_p P' \wedge \mathit{depth}\ P' \leq \mathit{depth}\ P$.

Proof. By induction on the construction of $\mathscr{S}$. □

Lemma 19.36. *If* $\mathit{hnf}\ P$ *and* $\mathit{hnf}\ Q$ *then* $\exists R.\ (R, \mathit{expandSet}\ P\ Q) \in \mathscr{S} \wedge \mathit{depth}\ R \leq \mathit{depth}\ (P \mid Q)$.

Proof. By case analysis on the construction of *expandSet P Q*. □

We finally need to modify Lemma 19.21. In the parallel case, we use the parallel congruence laws from CONGRPAR and Lemma 19.36 to find an expanded R of no greater depth than $\mathit{depth}\ (P \mid Q)$. Lemma 19.35 can then be used to convert R to a provably equivalent uhnf.

19.6 Conclusion

In this chapter we proved that the axiomatisation of strong late bisimilarity for pi-calculus is sound and complete. Most of the soundness results were already proved in previous chapters, and most of the work was to formalise the completeness result. For this, we had to make precise the hand sweeping notions which were found in the pen-and-paper proofs – most notably the treatment of head normal forms and summands, where the pen-and-paper variants implicitly use the idempotence rules for Sum in induction over the summands.

20. Early late correspondences

Structural congruence is included by all bisimilarities we have covered so far. However, we have only proven this fact for the late equivalences. Late equivalences include the early ones, and by proving this we get all of the structural congruence results for early ones as well.

In this chapter we will be using material from chapters which cover both early and late equivalences, and we will use a subscript e for transitions and relations using early semantics, and a subscript l for the late semantics, to distinguish between the two. For instance, $P \xrightarrow{\tau}_e P'$ denotes an early τ-transition, whereas $P \xrightarrow{\tau}_l P'$ denotes a late one.

20.1 Transitions

In order to prove that all late bisimilar agents are also early bisimilar we must prove how the early and late transitions systems correspond. All transitions, except input transitions, map directly to each other – any action done in one semantics can be done in the other, and the derivatives are the same.

20.1.1 Output actions

Lemma 20.1. *If* $P \xrightarrow{\overline{a}b}_l P'$ *then* $P \xrightarrow{\overline{a}b}_e P'$.

Proof. By induction on $P \xrightarrow{\overline{a}b}_l P'$. All of the relevant cases are discharged from the corresponding rules from the early operational semantics in Figure 13.1. □

Lemma 20.2. *If* $P \xrightarrow{\overline{a}b}_e P'$ *then* $P \xrightarrow{\overline{a}b}_l P'$.

Proof. By induction on $P \xrightarrow{\overline{a}b}_e P'$. All of the relevant cases are discharged from the corresponding rules from the late operational semantics in Figure 17.1. □

20.1.2 Bound output actions

Lemma 20.3. *If* $P \xrightarrow{\overline{a}(vx)}_l P'$ *then* $P \xrightarrow{\overline{a}(vx)}_e P'$.

Proof. By induction on $P \xrightarrow{\overline{a}(vx)}_l P'$. Before induction, the transitions are alpha-converted such that x is fresh for a and P.

All cases, except the OPEN-case, are discharged from the corresponding rules from the early operational semantics in Figure 13.3. The OPEN-case is discharged by Lemma 20.1. □

Lemma 20.4. *If* $P \xrightarrow{\overline{a}(vx)}_e P'$ *then* $P \xrightarrow{\overline{a}(vx)}_l P'$.

Proof. By induction on $P \xrightarrow{\overline{a}(vx)}_e P'$. Before induction, the transitions are alpha-converted such that x is fresh for a and P.

All cases, except the OPEN-case, are discharged from the corresponding rules from the late operational semantics in Figure 17.1. The OPEN-case is discharged by Lemma 20.2. □

20.1.3 Input actions

The difference between the late and the early operational semantics is how they handle input actions, the early semantics instantiates them as early as possible, whereas the late one as late as possible. The correspondance lemmas for the input actions reflect this.

Lemma 20.5. *If* $P \xrightarrow{a(x)}_l P'$ *then* $P \xrightarrow{au}_e P'\{^u/_x\}$.

Proof. By induction on $P \xrightarrow{a(x)}_l P'$. Before induction, the transitions are alpha-converted such that x is fresh for a, u, and P.

All cases are discharged from the corresponding rules from the early operational semantics in Figure 13.3. □

Lemma 20.6.

$$\textit{If } P \xrightarrow{au}_e P' \textit{ and } x \sharp P \textit{ then } \exists P''.\ P \xrightarrow{a(x)}_l P'' \wedge P' = P''\{^u/_x\}.$$

Proof. By induction on $P \xrightarrow{au}_e P'$. All of the relevant cases are discharged by their corresponding rule from the late operational semantics in Figure 17.1. □

20.1.4 Tau actions

Lemma 20.7. *If* $P \xrightarrow{\tau}_l P'$ *then* $P \xrightarrow{\tau}_e P'$.

Proof. By induction on $P \xrightarrow{\tau}_l P'$. All cases except the COMM and CLOSE cases are discharged by their corresponding rule from the early operational semantics in Figure 13.3. The output, bound output, and input actions from the COMM and CLOSE cases are obtained using lemmas 20.1, 20.3, and 20.5 respectively. □

Lemma 20.8. *If* $P \xrightarrow{\tau}_e P'$ *then* $P \xrightarrow{\tau}_l P'$.

Proof. By induction on $P \xrightarrow{\tau}_e P'$. All cases except the COMM and CLOSE cases are discharged by their corresponding rule from the late operational semantics in Figure 17.1. The output, bound output, and input actions from the COMM and CLOSE cases are obtained using lemmas 20.2, 20.4, and 20.6 respectively. □

20.2 Strong bisimilarity

To prove that late bisimilarity includes early bisimilarity, we start with simulations.

Lemma 20.9. *If* $P \hookrightarrow_{l\mathscr{R}} Q$ *then* $P \hookrightarrow_{e\mathscr{R}} Q$.

Proof. Follows from the definition of $\hookrightarrow_e$.

Lemmas 20.2, 20.4, 20.6 and 20.8 are used to transfer the early actions to late ones. The definition of $\hookrightarrow_l$ is then used to obtain the mimicking actions, and lemmas 20.1, 20.3, 20.5, and 20.7 are used to transfer them back to early actions. □

We can now prove the theorem that all late bisimilar agents are also early bisimilar.

Theorem 20.1. *If* $P \dot{\sim}_l Q$ *then* $P \dot{\sim}_e Q$.

Proof. By coinduction using Lemma 14.7 with $\mathscr{X}$ set to $\dot{\sim}_e$. The candidate relation is inherently symmetric, as early bisimulation is symmetric, and the simulation case is proven by Lemma 20.9. □

The theorem is easily extendible for late and early equivalence.

Theorem 20.2. *If* $P \sim_l Q$ *then* $P \sim_e Q$.

Proof. Follows immediately from the definitions of $\sim_e$, $\sim_l$, and theorems 20.1 and 20.2. □

20.3 Structural congruence

All early bisimulations we have covered so far are preserved by bisimulation. In Chapter 15 we made this assumption and used structural congruence rules when proving preservation properties for weak bisimulation. We must now prove this theorem.

Theorem 20.3.

$$\textit{If } P \equiv Q \textit{ then } P \dot{\sim}_e Q.$$
$$\textit{If } P \equiv Q \textit{ then } P \sim_e Q.$$

Proof. Follows directly from theorems 18.1, 18.2, and 20.1. □

We then also have that early weak bisimilarity and early weak congruence include structural congruence.

Theorem 20.4.

$$\textit{If } P \equiv Q \textit{ then } P \dot{\approx}_e Q.$$
$$\textit{If } P \equiv Q \textit{ then } P \approx_e Q.$$

Proof. Follows directly from Theorem 20.3 and Lemma 15.22. □

21. Conclusions

In this part of the thesis we formalise a substantial part of the meta theory of the pi-calculus. We prove most results for both the early and the late operational semantics. We define the early semantics of the pi-calculus in Chapter 13. In Chapter 14 we define strong bisimilarity and strong equivalence and prove that strong bisimilarity is preserved by all operators except the input prefix, and that strong equivalence is a congruence. In Chapter 15 we define weak bisimilarity and prove that it is preserved by all operators except Sum and the input prefix. In Chapter 16 we define weak equivalence, prove that it is preserved by all operators except the input prefix, and finally weak congruence, also proving that it is a congruence. In Chapter 17 we define the late semantics of the pi-calculus. We also define all of the bisimulation relations which are formalised for the early semantics, and prove the same results. In Chapter 18 we define structural congruence, and prove that it is included by all late bisimulation relations. In Chapter 19 we present the axiomatisation of late bisimilarity for the finite segment of the pi-calculus (without replication), and prove that it is sound and complete. Finally, in Chapter 20 we prove that the late semantics subsumes the early one – all late bisimilar agents are also early bisimilar for all bisimulation relations.

We believe this to be the most extensive formalisation of the pi-calculus ever done inside a theorem prover. Earlier formalisations by e.g. Röckl [70], Hirschkoff [45], or Honsell [46] focussed on the late operational semantics, and only on strong equivalences. To the best of our knowledge, neither weak equivalences nor axiomatisations of bisimilarities have previously been formalised in a theorem prover.

As with CCS, in Part II of this thesis, no bugs were found during the formalisation efforts, although we did encounter several proofs in the literature where a rigourous formulation is not immediately obvious. This was to be expected. The meta theory for the pi-calculus has been around for nearly twenty years, and has been extensively used – the risk that bugs have been missed is slim. The techniques used to formalise the pi-calculus are the same as for CCS. The main extension is the encoding of residuals. If the pi-calculus semantics were encoded using a ternary predicate like the one for CCS the OPEN rule would become inconsistent with the Barendregt variable convention, as demonstrated in Section 3.1.1.

The sizes of the different parts of the formalisation can be found in Figure 21.1 for both the early and the late semantics. As for CCS, the theories

Part	Lines of code	
	Early semantics	Late semantics
Agents	443	
Semantics	1296	1484
Strong bisimilarity	2517	2515
Weak bisimilarity	2491	2758
Weak congruence	3264	3503
Structural congruence	-	2022
Late axiomatisation	-	3620
Early late correspondence	827	
Total	24913	

Figure 21.1: The size of the different parts of the Isabelle formalisation of the pi-calculus meta-theory.

for weak bisimilarity and weak congruence are intertwined. The formalisation of the late weak equivalences are slightly larger than their early counterparts as the actions of weak late bisimilarity are more complex.

This formalisation took me around one and a half years to complete, working on average around 30 hours a week. When I started Nominal Isabelle did not exist, but the ideas were there and I coded all of the infrastructure to reason about binders by hand. As newer versions of Nominal Isabelle were released I could discard thousands of lines of code. A significant part of this time was also spent learning Nominal Isabelle. If I were to write the formalisation again, it would go much faster.

This part of the thesis is mainly based on our article from 2007 [19], which was later extended into a journal version [20]. The only results not present in there is the axiomatisation from Chapter 19; that work was published at a workshop in 2007 [18].

21.1 Future work

There are several important aspects left to formalise. There are axiomatisations for strong equivalence [68], weak bisimilarity, and weak congruence [53], for both early and late operational semantics. These axiomatisations are more complicated than the one presented here.

The operational semantics presented used in this thesis works well for mathematical proofs. However, tools which use this semantics turn out to

be inefficient due to state space explosions in the transition system. Consider the INPUT rule for the early operational semantics.

$$\frac{}{a(x).P \xrightarrow{au} P\{^{u}/_{x}\}} \text{ INPUT}$$

Deriving all possible transitions that this rule can generate causes a state space explosion – there is one transition for every name u. In 1994 Lin proposed a symbolic semantics for the pi-calculus [52] where each transition in addition to its label has a constraint store which keeps track of which names have been received and what their possible values are. Symbolic semantics does not suffer from the state explosion properties. The Mobility Workbench [77] by Victor and Moller use a symbolic semantics. For a symbolic semantics to be useful it needs to be sound and complete with respect to the standard operational semantics – to the best of our knowledge, no such result has ever been formalised using a theorem prover.

An alternative bisimulation relation is barbed bisimilarity. Barbed bisimilarites were originally introduced by Milner and Sangiorgi for CCS [59], and later extended to the pi-calculus by Sangiorgi. They use a reduction semantics, i.e. a semantics without any labels on the transitions, and a STRUCT rule. Two agents are barbed bisimilar if they have the same observable actions, and if they reduce to barbed bisimilar agents. A congruence is then obtained by closing this bisimilarity by arbitrary agent contexts – a barbed congruence is hence a congruence by definition. Its main advantage is that is easier to get an intuitive grasp of a reduction semantics with structural congruence, than of a labeled semantics without it. The disadvantage is that a direct proof of barbed congruence needs a universal quantification over all contexts. By proving that barbed congruence coincides with bisimilarity of a labeled transition system we get the best of two worlds: the intuitively attractive barbed congruence gains the powerful proof methods of bisimilarity. Barbed congruence for the pi-calculus corresponds to strong early congruence, and this result is relevant enough to warrant mechanisation.

Another area of interest is to apply Isabelle to verify properties of actual agents, such as proving that two agents are bisimilar. The Mobility Workbench for the pi-calculus, and the Concurrency Workbench for CCS can automatically compute bisimulation equivalences for simple agents. These programs are not verified, and we lack the techniques to do so effectively, but their output can be verified. Given two agents these tools produce a candidate bisimulation relation if the agents are bisimilar. With a symbolic semantics implemented in Isabelle, we could check whether or not this candidate relation actually is a bisimulation.

The Mobility Workbench does not reason efficiently about agents using Replication – the state space explodes for replicated agents even in a sym-

bolic semantics. Properties of bisimilarity for such agents can be checked in Isabelle, and heuristicts to facilitate such reasoning can be added.

Part IV:
Psi-calculi

22. Parametric calculi

The pi-calculus is expressive, but its only type of data are names, which also function as channels. This limits its use when modeling advanced systems and protocols which often require more expressive data such as inductively defined datatypes. These can be encoded in the pi-calculus, but the resulting agents will be bulky and cumbersome to work with – as the complexity of the agents increase, so does the effort required to reason about them.

To circumvent this problem, augmentations and variants of the pi-calculus have been proposed, where the desired datatypes are taken as primitive – there is the Spi-calculus by Abadi and Gordon [8], and the Applied pi-calculus by Abadi and Fournet [7] just to name two. Whereas the modeling convenience of these calculi significantly exceeds that of the pi-calculus, this convenience comes at a price – the meta theoretical proofs of the calculi become significantly more complex. For example, the semantics of the Applied pi-calculus is defined using two levels of processes (pure and extended), an inductively defined reduction relation, an algebraically defined structural congruence, and (in order to achieve compositionality) a notion of a barb and an explicit quantification over contexts.

22.1 Psi-calculi

Our contribution is to define psi-calculi: a framework where a range of calculi can be formulated with a lean and symmetric semantics, and where proofs can be conducted using straightforward induction without resorting to a structural congruence or explicit quantification of contexts. We claim to be the first to formulate such truly compositional labeled operational semantics for calculi of this caliber. Psi-calculi accommodate not only the examples covered in the thesis so far, but also extensions such as the pi-calculus with polyadic synchronisation [30], and fusion [40]. For a full overview of how these calculi are encoded, see [17].

The main idea is that a psi-calculus is obtained by extending the basic untyped pi-calculus with three parameters. The first is a set of data terms which can function as both communication channels and communicated objects. The second is a set of conditions, for use in conditional constructs such as **if** statements. The third is a set of assertions, used to express e.g.

constraints or aliases, which can resolve the conditions. These sets need not be disjoint, and one of our main results is to identify minimal requirements on them. They turn out to be quite general and natural.

In comparison to the applied pi-calculus We allow arbitrary assertions (and not only declarations of aliases), and arbitrary conditions (and not only equality tests). Also, we base our exposition on nominal datatypes and these accommodate e.g. alpha-equivalence classes of terms with binders. For example, we can use a higher-order logic for assertions and conditions, and higher-order formalisms such as the lambda calculus for data terms and channels. Last but not least the formalisation is lean and symmetric. Thus we get the best of two worlds: expressiveness and therefore modeling convenience significantly exceeds that of the applied pi-calculus, while the purity of the semantics is on par with the original pi-calculus.

The straightforward definitions make our proofs suitable for checking in a theorem prover. As for CCS and the pi-calculus, we have formalised all of our meta theoretical results completely in Isabelle.

For the rest of this chapter we will define psi-calculi, its semantics and a notion of strong bisimilarity. Later chapters will cover the formalisation in Isabelle.

22.2 Definitions

A *nominal data type* is a nominal set together with a set of equivariant functions on it. In particular we require a substitution function, which intuitively substitutes elements for names. If X is an element of a data type, $\tilde{x}$ is a sequence of names without duplicates and $\tilde{T}s$ is an equally long sequence of elements, the *substitution* $X[\tilde{x} := \tilde{T}]$ is an element of the same data type as X. In a traditional data type substitution can be thought of as replacing all occurrences of names $\tilde{x}$ by $\tilde{T}s$. In a calculus with binders it can be thought of as replacing the free names, alpha-converting any binders to avoid capture. Formally, we define substitution as any equivariant function satisfying a set of substitution laws. The full list is given in Section 24.2.1

The main point of using nominal datatypes is that we obtain a general framework, allowing many different instantiations. Our only requirements are on the notions of support, name swapping, and substitution. This corresponds precisely to the essential ingredients for data transmitted between agents. Since names can be statically scoped and data sent into and out of scope boundaries, it must be possible to discern exactly what names are contained in what data items, and this is just the role of the support. In case a data element intrudes a scope, the scoped name needs to be alpha converted to avoid clashes, and name swapping can achieve precisely this. When a term is received in a communication between agents it must

replace all occurrences of the placeholder in the input construct, in other words, the placeholder is substituted by the term.

Since these are the only things we assume about data terms we can handle datatypes that are not inductively defined, such as equivalences classes and sets defined by comprehension or co-induction. Examples include higher-order datatypes such as the lambda calculus, and even agents of a psi-calculus. As long as it satisfies the axioms of a nominal data type it can be used in our framework. Similarly, the notions of conditions, i.e., the tests on data that agents can perform during their execution, and assertions, i.e. the facts that can be used to resolve conditions, are formulated as nominal datatypes. This means that logics with binders and even higher-order logics can be used. Moreover, alpha-variants of terms can be formally equated by taking the quotient of terms under alpha equality, thereby facilitating the formalism and proofs.

22.2.1 Terms, assertions, and conditions

Formally, a psi-calculus is defined by instantiating three nominal datatypes and four operators:

Definition 22.1 (Psi-calculus parameters)**.** *A psi-calculus requires the three (not necessarily disjoint) nominal datatypes:*

$\mathbf{T}$	*the (data) terms, denoted by* M, N
$\mathbf{C}$	*the conditions, denoted by* φ
$\mathbf{A}$	*the assertions, denoted by* Ψ

and the four equivariant operators:

$\leftrightarrow\!\!\!\!\!\cdot : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	*Channel Equivalence*
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	*Composition*
$\mathbf{1} : \mathbf{A}$	*Unit*
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	*Entailment*

The binary functions above will be written in infix. Thus, if M and N are terms then $M \leftrightarrow\!\!\!\!\!\cdot\ N$ is a condition, pronounced "M and N are channel equivalent" and if Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$. Also we write $\Psi \vdash \varphi$, pronounced "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$.

The data terms are used to represent all kinds of data, including communication channels. Intuitively, two agents can communicate if one sends and the other receives along the same channel. This is why we require a condition $M \leftrightarrow\!\!\!\!\!\cdot\ N$ to say that M and N represent the same communication channel. For example, in the pi-calculus $\leftrightarrow\!\!\!\!\!\cdot$ would be just identity of names.

The assertions will be used to declare information necessary to resolve the conditions. Assertions can be contained in agents and represent constraints; they can contain names and thereby be syntactically scoped and represent information known only to the agents within that scope. The operator $\otimes$ on assertions will, intuitively, be used to represent conjunction of the information in the assertions. The assertion $\mathbf{1}$ is the unit for $\otimes$.

Intuitively $\Psi \vdash \varphi$ means that given the information in Ψ, it is possible to infer φ. We say that two assertions are equivalent if they entail the same conditions:

Definition 22.2 (assertion equivalence)**.** *Two assertions are* equivalent, *written* $\Psi \simeq \Psi'$*, if for all* φ *we have that* $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$.

We can now formulate our requisites on valid psi-calculus parameters:

Definition 22.3 (Requisites on valid psi-calculus parameters)**.**

Channel Symmetry:	$\Psi \vdash M \leftrightarrow N \Longrightarrow \Psi \vdash N \leftrightarrow M$
Channel Transitivity:	*If* $\Psi \vdash M \leftrightarrow N$ *and* $\Psi \vdash N \leftrightarrow L$ *then* $\Psi \vdash M \leftrightarrow L$.
Compositionality:	*If* $\Psi \simeq \Psi'$ *then* $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$.
Identity:	$\Psi \otimes \mathbf{1} \simeq \Psi$
Associativity:	$(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$
Commutativity:	$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$

Our requisites on a psi-calculus is that the channel equivalence is a partial equivalence relation, that $\otimes$ is compositional, and that the equivalence classes of assertions form an abelian monoid. These requisites turn out to be strictly minimal for our results in Section 22.3 to hold. Note that channel equivalence is not required to be reflexive. Thus it is possible to have data terms that are not channel equivalent to anything at all, meaning that they cannot be used as channels.

As an example, we can choose data terms inductively generated by some signature, assertions and conditions to be elements of a first-order logic with equality over these terms, entailment to be logical implication, $\otimes$ to be conjunction and $\mathbf{1}$ to be TRUE. To verify that this indeed is a psi-calculus one needs to check the requirements on a nominal data type (meaning the notions of swapping, support and substitution are defined and fulfills the required properties), and that the requisites in Definition 22.3 hold.

22.2.2 Frames

Assertions can contain information about names, and names can be scoped using the familiar pi-calculus operator ν. For example, in a

cryptography application an assertion Ψ could be that the a datum represents the encoding of a message using a key k. This Ψ can occur under the scope of νk, to signify that the key is known only locally. In order to admit this in a general way we use the notion of a frame, first introduced by Abadi and Fournet [7]. Basically, a frame is just an assertion with additional information about which names are scoped. The example above where Ψ occurs under the scope of k will be written $(\nu k)\Psi$, to signify a frame consisting of the assertion Ψ where the name k is local.

In the following $\tilde{a}$ means a finite (possibly empty) sequence of names, $a_1,\ldots,a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1,\ldots,a_n\}$. We also use sequences of terms, conditions, assertions etc. in the same way.

Definition 22.4 (Frame)**.** *A* frame *is of the form* $(\nu\widetilde{b})\Psi$ *where* $\widetilde{b}$ *is a sequence of names that bind into the assertion* Ψ*. We identify alpha variants of frames.*
[1]

We use F, G to range over frames. Since we identify alpha variants we can always choose the bound names freely.

Notational conventions: We write just Ψ for $(\nu\varepsilon)\Psi$ when there is no risk of confusing a frame with an assertion, and $\otimes$ to mean composition on frames defined by $(\nu\widetilde{b_1})\Psi_1\otimes(\nu\widetilde{b_2})\Psi_2 = (\nu\widetilde{b_1}\widetilde{b_2})\Psi_1\otimes\Psi_2$ where $\widetilde{b_1} \sharp \widetilde{b_2}, \Psi_2$ and vice versa. We write $(\nu c)((\nu\widetilde{b})\Psi)$ to mean $(\nu c\widetilde{b})\Psi$.

Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame. Two frames are equivalent if they entail the same conditions:

Definition 22.5 (Equivalence of frames)**.** *We define* $F \vdash \varphi$ *to mean that there exists an alpha variant* $(\nu\widetilde{b})\Psi$ *of* F *such that* $\widetilde{b} \sharp \varphi$ *and* $\Psi \vdash \varphi$*. We also define* $F \simeq G$ *to mean that for all* φ *it holds that* $F \vdash \varphi$ *iff* $G \vdash \varphi$*.*

Continuing the example of first-order logic with equality, assume that the term $\mathsf{enc}(M,k)$ represents the result of encoding message M with key k. Let Ψ be the assertion $C = \mathsf{enc}(M,k)$, stating that the ciphertext C is the result of encoding M by k. If an agent contains this assertion the environment of the agent will be able to use it to resolve tests on the data, in particular to infer that $C = \mathsf{enc}(M,k)$. In other words, if the environment receives C it can test if this is the encryption of M. In order to restrict access to the key k it can be enclosed in a scope νk. The environment of the agent will then have access to the frame $(\nu k)\Psi$ rather than Ψ itself. This frame is much less informative,

[1] In some presentations frames have been written just as pairs $\langle\widetilde{b},\Psi\rangle$. The notation in this paper better conveys the idea that the names bind into the assertion, at the slight risk of confusing frames with agents. Formally, we establish frames and agents as separate types, although a valid intuition is to regard a frame as a special kind of agent, containing only scoping and assertions. This is the view taken in [7].

for example it does *not* hold that $(\nu k)\Psi \vdash C = \mathsf{enc}(M,k)$. Here great care has to be made to formulate the class of allowed conditions. If these only contain equivalence tests of terms, $(\nu k)\Psi$ will entail nothing but tautologies and be equivalent to $\mathbf{1}$. But if quantifiers are allowed in the conditions, then by existential introduction $\Psi \vdash \exists k.C = \mathsf{enc}(M,k)$, and since this condition has no free k we get $(\nu k)\Psi \vdash \exists k.C = \mathsf{enc}(M,k)$. In other words the environment will learn that C is the encryption of M for some key k.

Most of the properties of assertions carry over to frames. Channel symmetry and channel transitivity, identity, associativity and commutativity all hold, but compositionality in general does not. In other words, there are psi-instances with frames F,G,H where $F \simeq G$ but not $F \otimes H \simeq G \otimes H$. An example is if there are assertions Ψ, Ψ' and Ψ_a for all names a, conditions φ' and φ_a for all names a, and where the entailment relation satisfies $\Psi_a \vdash \varphi_a$ and $\Psi' \vdash \varphi'$. Suppose composition is defined such that $\Psi \otimes \Psi = \Psi$ and all other compositions yield Ψ'. By adding a unit element this satisfies all requirements on a psi-calculus. In particular $\otimes$ is trivially compositional because no two different assertions are equivalent. Also $(\nu a)\Psi_a \simeq \Psi$, but $\Psi \otimes (\nu a)\Psi_a \neq \Psi \otimes \Psi$ since $\Psi \otimes \Psi_a = \Psi' \vdash \varphi'$.

22.2.3 Agents

Definition 22.6 (psi-calculus agents)**.** *Given valid psi-calculus parameters as in Definitions 22.1 and 22.3, the psi-calculus* agents*, are denoted by by $P,Q,\ldots$, and are of the following forms.*

$\mathbf{0}$	Nil
$\overline{M}N.P$	Output
$\underline{M}(\lambda\tilde{x})N.P$	Input
$\mathbf{case}\ \varphi_1 : P_1\ [] \cdots [] \ \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\lvert \Psi \rvert\!)$	Assertion

In the Input $\underline{M}(\lambda\tilde{x})N.P$ we require that $\tilde{x} \subseteq \mathrm{n}(N)$ is a sequence without duplicates, and the names $\tilde{x}$ bind occurrences in both N and P. Restriction binds a in P. We identify alpha equivalent agents. An assertion is guarded *if it is a subterm of an Input or Output. In a replication $!P$ there may be no unguarded assertions in P, and in* **case** $\varphi_1 : P_1\ [] \cdots [] \ \varphi_n : P_n$ *there may be no unguarded assertion in any P_i.*

In the Output and Input forms M is called the subject and N the object. Output and Input are similar to those in the pi-calculus, but arbitrary terms

can function as both subjects and objects. In the input $\underline{M}(\lambda\tilde{x})N.P$ the intuition is that there is a pattern matching where the pattern $(\lambda\tilde{x})N$ can match any term obtained by instantiating $\tilde{x}$, e.g., $\underline{M}(\lambda x, y)f(x, y).P$ can only communicate with an output $\overline{M}f(N_1, N_2)$ for some data terms N_1, N_2. This can be thought of as a generalisation of the polyadic pi-calculus where the patterns are just tuples of names. Another significant extension is that we allow arbitrary data terms also as communication channels. Thus it is possible to include functions that create channels.

The **case** construct as expected works by behaving as one of the P_i for which the corresponding φ_i is true. **case** $\varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ is sometimes abbreviated as **case** $\tilde{\varphi} : \tilde{P}$, or if $n = 1$ as **if** φ_1 **then** P_1. In psi-calculi where a condition $\top$ exists such that $\Psi \vdash \top$ for all Ψ we write $P + Q$ to mean **case** $\top : P \,[]\, \top : Q$.

Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline. In the traditional pi-calculus terms are just names and its input construct $a(x).P$ can be represented as $\underline{a}(\lambda x)x.P$. In some of the examples to follow we shall use the simpler notation $a(x).P$ for this input form.

In some examples we omit a trailing $\mathbf{0}$, writing just $\overline{M}N$ for $\overline{M}N.\mathbf{0}$. If the object of an Output is a long term we enclose it in brackets $\langle\,\rangle$ to make it easier to parse.

For a simple example, the pi-calculus [58] can be represented as a psi-calculus where the only data terms are names, the only assertion is $\mathbf{1}$, and the conditions are equality tests on names. Formally:

$$\begin{aligned}
\mathbf{T} &= \mathcal{N} \\
\mathbf{C} &= \{a = b : a, b \in \mathbf{T}\} \cup \{a \leftrightarrow b : a, b \in \mathbf{T}\} \\
\mathbf{A} &= \{\mathbf{1}\} \\
\otimes &= \lambda\Psi_1, \Psi_2.\, \mathbf{1} \\
\vdash &= \{(\mathbf{1}, a = a) : a \in \mathcal{N}\} \cup \{(\mathbf{1}, a \leftrightarrow a) : a \in \mathcal{N}\}
\end{aligned}$$

Here we can let $\top$ be $a = a$ and can thus represent pi-calculus sum through **case**. We obtain the polyadic pi-calculus by adding the tupling symbols t_n for tuples of arity n to $\mathbf{T}$., i.e. $\mathbf{T} = \mathcal{N} \cup \{\mathrm{t}_n(a_1, \ldots, a_n) : \; a_1, \ldots, a_n \in \mathcal{N}\}$. The polyadic output is to simply output the corresponding tuple of object names, and the polyadic input $a(b_1, \ldots, b_n).P$ is represented by a pattern matching $\underline{a}(\lambda b_1, \ldots, b_n)\mathrm{t}_n(b_1, \ldots, b_n).P$. Strictly speaking this allows tuples also in subject position in agents, but as we shall see such prefixes will not give rise to any transition, since in this psi-calculus $M \leftrightarrow M$ is only entailed when M is a name, i.e., only names are channels.

In a psi-calculus the channels can be arbitrary terms. This means that it is possible to introduce functions on channels (e.g., if M is a channel then so is $f(M)$). It also means that a channel can contain more than one name.

An extension of this kind is explored by Carbone and Maffeis [30] in the so called pi-calculus with polyadic synchronisation, ${}^{e}\pi$. Here action subjects are tuples of names, and it is demonstrated that this allows a gradual enabling of communication by opening the scope of names in a subject, results in simple representations of localities and cryptography, and gives a strictly greater expressiveness than standard pi-calculus. We can represent ${}^{e}\pi$ by using tuples of names in subject position. The only modification to the representation of the polyadic pi-calculus is to extend $\vdash$ to $\vdash = \{(\mathbf{1}, M \leftrightarrow M) : M \in \mathbf{T}\}$, and to remove the conditions of type $M = N$ (since they can be encoded in ${}^{e}\pi$).

The data terms can be also be drawn from a higher-order formalisms. It is thus possible to transmit functions between agents. For example, let $\mathbf{T}$ be the lambda calculus, containing abstractions $\boldsymbol{\lambda} x.M$ and applications MN. In the parallel composition $\overline{a}\langle \boldsymbol{\lambda} x.M\rangle . P \mid a(z) . \overline{b}\langle zN\rangle . Q$ the left hand component transmits the function $\boldsymbol{\lambda} x.M$ to the right, where the application of it to N is transmitted along b. Reduction would be represented as a binary predicate over lambda terms and could be tested in psi-calculus conditions (the reduction rules would be part of the definition of entailment). In this sense psi can resemble a higher-order calculus. It is even possible to let the terms be the psi-calculus agents themselves. An agent transmitted as a term cannot directly communicate with the agent that sent or received it, but there is a possibility of indirect interaction through the entailment relation. This area we leave for further study.

22.2.4 Operational semantics

In this section we define an inductive transition relation on agents. In particular it establishes what transitions are possible from a parallel composition $P \mid Q$. In the standard pi-calculus the transitions from a parallel composition can be uniquely determined by the transitions from its components, but in psi-calculi the situation is more complex. Here the assertions contained in P can affect the conditions tested in Q and vice versa. For this reason we introduce the notion of the *frame of an agent* as the combination of its top level assertions, retaining all the binders. It is precisely this that can affect a parallel agent.

Definition 22.7 (Frame of an agent)**.** *The* frame $\mathcal{F}(P)$ of an agent P *is defined inductively as follows:*

$$\mathcal{F}(\underline{M}(\lambda\widetilde{x})N.P) = \mathcal{F}(\overline{M}N.P) = \mathcal{F}(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) = \mathcal{F}(!P) = \mathcal{F}(\mathbf{0}) = \mathbf{1}$$

$$\mathcal{F}(\llparenthesis \Psi \rrparenthesis) = \Psi$$

$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q)$$

$$\mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P)$$

For a simple example, if $a \sharp \Psi_1$:

$$\mathscr{F}(\langle\!|\Psi_1|\!\rangle \mid (\nu a)(\langle\!|\Psi_2|\!\rangle) \mid \overline{M}N.\langle\!|\Psi_3|\!\rangle) \quad = \quad (\nu a)(\Psi_1 \otimes \Psi_2)$$

Here Ψ_3 occurs under a prefix and is therefore not on top level. An agent where all assertions are guarded thus has the frame $\mathbf{1}$. In the following we often write $(\nu \widetilde{b_P})\Psi_P$ for $\mathscr{F}(P)$, but note that this is not a unique representation since frames are identified up to alpha equivalence.

The actions α that agents can perform are of three kinds: output actions, input actions of the early kind, meaning that the input action contains the received object, and the silent action τ. The operational semantics will consist of transitions of the form $\Psi \rhd P \xrightarrow{\alpha} P'$. This transition intuitively means that P can perform an action α leading to P', in an environment that asserts Ψ.

Definition 22.8 (Actions)**.** *The* actions *are denoted by by α, β and are of the following three kinds:*

$\overline{M}(\nu \tilde{a})N$	*Output, where* $\tilde{a} \subseteq \mathrm{n}(N)$
$\underline{M}N$	*Input*
τ	*Silent*

For actions we refer to M as the *subject* and N as the *object*. We define $\mathrm{bn}(\overline{M}(\nu \tilde{a})N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input or τ. We also define $\mathrm{n}(\tau) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(N) \cup \mathrm{n}(M)$ if α is an output or input. As in the pi-calculus, the output $\overline{M}(\nu \tilde{a})N$ represents an action sending N along M and opening the scopes of the names $\tilde{a}$. Note in particular that the support of this action includes $\tilde{a}$. Thus $\overline{M}(\nu a)a$ and $\overline{M}(\nu b)b$ are different actions.

Definition 22.9 (Transitions)**.** *A* transition *is of the kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that when the environment contains the assertion Ψ the agent P can do an α to become P'. The transitions are defined inductively in Table 22.1. We write $P \xrightarrow{\alpha} P'$ to mean $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$.*

Both agents and frames are identified by alpha equivalence. This means that we can choose the bound names fresh in the premise of a rule. In a transition the names in $\mathrm{bn}(\alpha)$ count as binding into both the action object and the derivative, and transitions are identified up to alpha equivalence. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative. This is the reason why $\mathrm{bn}(\alpha)$ is in the support of the output action: otherwise it could be alpha-converted in the action alone. Also, for the side conditions in SCOPE and OPEN it is important that $\mathrm{bn}(\alpha) \subseteq \mathrm{n}(\alpha)$. The freshness conditions on the involved frames will ensure that if a name is bound in one agent its representative in a frame

$$\textsc{In}\ \frac{\Psi \vdash M \doteq K}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N.P \xrightarrow{\underline{K}N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]} \qquad \textsc{Out}\ \frac{\Psi \vdash M \doteq K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}$$

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P' \\ \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q' \qquad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \doteq K\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\widetilde{a})(P' \mid Q')}\ \widetilde{a} \sharp Q$$

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P|Q \xrightarrow{\alpha} P'|Q}\ \mathrm{bn}(\alpha) \sharp Q \qquad \textsc{Scope}\ \frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b \sharp \alpha, \Psi$$

$$\textsc{Open}\ \frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\overline{M}(\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{l} b \sharp \widetilde{a}, \Psi, M \\ b \in \mathit{supp}(N)\end{array} \qquad \textsc{Rep}\ \frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

Figure 22.1: Operational semantics. Symmetric versions of COM and PAR are elided. In the rule COM we assume that $\mathscr{F}(P) = (\nu\widetilde{b_P})\Psi_P$ and $\mathscr{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ where $\widetilde{b_P}$ is fresh for all of $\Psi, \widetilde{b_Q}, Q, M$ and P, and that $\widetilde{b_Q}$ is correspondingly fresh. In the rule PAR we assume that $\mathscr{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ where $\widetilde{b_Q}$ is fresh for Ψ, P and α. In OPEN the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

is distinct from names in parallel agents, and also (in PAR) that it does not occur on the transition label.

The environmental assertions $\Psi \rhd \cdots$ in Table 22.1 express the effect that the environment has on the agent: enabling conditions in CASE, giving rise to action subjects in IN and OUT and enabling interactions in COM. Thus Ψ never changes between hypothesis and conclusion except for the parallel operator, where an agent is part of the environment for another agent. In a derivation tree for a transition, the assertion will therefore increase towards the leafs by application of PAR and COM. If all environmental assertions are erased and channel equivalence replaced by identity we get the standard laws of the pi-calculus enriched with data structures.

In comparison to the applied pi-calculus and the concurrent constraint pi calculus one main novelty is the inclusion of environmental assertions in the rules. They are necessary to make our semantics compositional, i.e., the effect of the environment on an agent is wholly captured by the semantics. In contrast, the labeled transitions of the applied and the concurrent constraint pi-calculi must rely on an auxiliary structural congruence, containing axioms such as scope extension $(\nu a)(P \mid Q) \equiv (\nu a)P \mid Q$ if $a \sharp Q$. With our semantics such laws are derived rather than postulated. The advantage of our approach is that proofs of meta-theoretical results such as compositionality are much simpler since there is only the one inductive definition of transitions.

22.2.5 Illustrative examples

For a simple example of a transition, suppose for an assertion Ψ and condition φ that $\Psi \vdash \varphi$. Assume that

$$\forall \Psi'.\Psi' \rhd Q \xrightarrow{\alpha} Q'$$

i.e., Q has an action α regardless of the environment. Then by the CASE rule we get

$$\Psi \rhd \textbf{if } \varphi \textbf{ then } Q \xrightarrow{\alpha} Q'$$

i.e., **if** φ **then** Q has the same transition if the environment is Ψ. Since $\mathscr{F}(\llparenthesis \Psi \rrparenthesis) = \Psi$ and $\Psi \otimes \mathbf{1} = \Psi$ we get by PAR that

$$\mathbf{1} \rhd \llparenthesis \Psi \rrparenthesis \mid \textbf{if } \varphi \textbf{ then } Q \xrightarrow{\alpha} \llparenthesis \Psi \rrparenthesis \mid Q'$$

Data terms may also represent communication channels and here the channel equivalence comes into play. For example, in a polyadic pi-calculus the terms include tuples and projection functions with the usual equalities, for example $\pi_1(\mathrm{t}_2(a,b)) = a$. If these terms can represent channels then they must represent the *same* channel, consequently we must have

$\Psi \vdash \pi_1(\mathrm{t}_2(a,b)) \leftrightarrows a$ for all Ψ. As an example,

$$\overline{a}N.P \mid \underline{\pi_1(\mathrm{t}_2(a,b))}\,(y).Q \xrightarrow{\tau} P \mid Q[y := N]$$

Agents such as $\underline{\pi_1(\mathrm{t}_2(a,b))}\,(y).Q$ can arise naturally if tuples of channels are transmitted as objects. For example, an agent that receives a pair of channels along c and then inputs along the first of them is written $\underline{c}(x).\underline{\pi_1(x)}(y).Q$. When put in parallel with an agent that sends $\mathrm{t}_2(a,b)$ along c it will have a transition leading to the agent where x is substituted by $\mathrm{t}_2(a,b)$, i.e. $\underline{\pi_1(\mathrm{t}_2(a,b))}\,(y).Q$.

The semantics makes no particular provision for an equality of terms in object position. Thus, the agents $\overline{c}a.P$ and $\overline{c}\pi_1(\mathrm{t}_2(a,b)).P$ have different transitions, and correspond to sending out the unevaluated "texts" a and $\pi_1(\mathrm{t}_2(a,b))$ respectively. To represent agents which send evaluated "values" we can do as in the applied pi-calculus where assertions declare equivalence of terms and agents send freshly generated aliases, e.g.

$$(\nu z)(\overline{c}z.P \mid (\!|z = \pi_1(\mathrm{t}_2(a,b))|\!))$$

This agent has the same transition as $(\nu z)(\overline{c}z.P \mid (\!|z = a|\!))$. Any agent receiving the z will not be able to distinguish if z is a or $\mathrm{t}_2(a,b))$ since these terms are equated by all assertions. Also, if a and b are scoped as in

$$(\nu a,b,z)(\overline{c}z.P \mid (\!|z = \pi_1(\mathrm{t}_2(a,b))|\!))$$

then their scopes will *not* open as a consequence of the output. In the applied pi-calculus this is the only form of communication and it is not possible to directly transmit data structures containing channel names, like the name tuples of the polyadic pi-calculus above. In psi-calculi these communication possibilities can coexist.

The main technical issue in the semantics is the treatment of scoping, as illustrated by the following example where the terms are just names. The intuition is that there is a communication channel available to all agents, and agents can declare any name to represent it through an assertion. The assertions are thus sets of names, and any name occurring in the assertion can be used as the subject of an action. Any two names in the assertion are deemed channel equivalent. Formally,

$$\begin{array}{lcl}
\mathbf{T} & = & \mathcal{N} \\
\mathbf{C} & = & \{a \leftrightarrows b : a,b \in \mathbf{T}\} \\
\mathbf{A} & = & \mathscr{P}_{\mathrm{fin}}(\mathcal{N}) \\
\otimes & = & \cup \\
\mathbf{1} & = & \emptyset \\
\vdash & = & \{(\Psi, a \leftrightarrows b) : a,b \in \Psi\}
\end{array}$$

Omitting the action and prefix objects we get

$$\{a, b\} \rhd \overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}$$

and also

$$\{a, b\} \rhd \overline{a}.\mathbf{0} \xrightarrow{\overline{b}} \mathbf{0}$$

By the PAR rule we have

$$\varnothing \rhd \overline{a}.\mathbf{0} \mid (\!|\{a, b\}|\!) \xrightarrow{\overline{a}} \mathbf{0} \mid (\!|\{a, b\}|\!)$$

and

$$\varnothing \rhd \overline{a}.\mathbf{0} \mid (\!|\{a, b\}|\!) \xrightarrow{\overline{b}} \mathbf{0} \mid (\!|\{a, b\}|\!)$$

Applying a restriction we get

$$\varnothing \rhd (\nu a)(\overline{a}.\mathbf{0} \mid (\!|\{a, b\}|\!)) \xrightarrow{\overline{b}} (\nu a)(\mathbf{0} \mid (\!|\{a, b\}|\!))$$

but no corresponding action with subject a because of the side condition on SCOPE. Thus, a communication through COM can be inferred from

$$(\nu a)(\overline{a}.\mathbf{0} \mid (\!|\{a, b\}|\!)) \mid b.\mathbf{0}$$

but *not* from

$$(\nu a)(\overline{a}.\mathbf{0} \mid (\!|\{a, b\}|\!)) \mid a.\mathbf{0}$$

This instance of a psi-calculus also illustrates two features of the semantics: firstly that channel equivalence is used in all three rules IN, OUT and COM, and secondly that assertions rather than frames represent the environment. Both issues are related to the law of scope extension, i.e., if $a \sharp Q$ then the agents $(\nu a)(P \mid Q)$ and $((\nu a)P) \mid Q$ should have the same transitions. Elaborating the example above and noting that $\{a\} \cup \{b\} \vdash a \leftrightarrow b$, we get that

$$(\nu a, b)((\!|\{a\}|\!) \mid (\!|\{b\}|\!) \mid \overline{a}.\mathbf{0} \mid b.\mathbf{0})$$

has an internal communication. By scope extension this agent should have the same transitions as $P \mid Q$ where

$$P = (\nu a)((\!|\{a\}|\!) \mid \overline{a}.\mathbf{0}) \qquad Q = (\nu b)((\!|\{b\}|\!) \mid b.\mathbf{0})$$

Here $\mathscr{F}(P) = (\nu a)\{a\}$ and $\mathscr{F}(Q) = (\nu b)\{b\}$ are alpha equivalent. Since they will be composed below we choose different representatives for the bound

names. A communication from $P \mid Q$ is inferred by COM and the premises

1. $\{b\} \rhd P \xrightarrow{\overline{b}} (\nu a)(\langle\!|\{a\}|\!\rangle \mid \mathbf{0})$
   (derived using $\{a\} \otimes \{b\} = \{a, b\} \vdash a \leftrightarrow b$ in OUT)
2. $\{a\} \rhd Q \xrightarrow{a} (\nu b)(\langle\!|\{b\}|\!\rangle \mid \mathbf{0})$
   (derived using $\{b\} \otimes \{a\} = \{a, b\} \vdash a \leftrightarrow b$ in IN)
3. $\{a\} \otimes \{b\} = \{a, b\} \vdash a \leftrightarrow b$

Note how the action subjects are derived by the assertions in both cases to not clash with the binders, and that channel equivalence is necessary in all three rules.

The same example demonstrates why transitions in Table 22.1 are defined with assertions and not frames, for whereas $\{a, b\} \vdash a \leftrightarrow b$ the corresponding result cannot be obtained from the frames of the agents. We have that $\mathcal{F}(Q) \otimes \{a\} = (\nu b)\{a, b\} \nvdash a \leftrightarrow b$, so that frame is not useful for deriving a transition from P. Our earlier attempt [48] erroneously used frames rather than assertions, and this means that scope extension does not hold unless a further condition is imposed on the entailment relation to eliminate this kind of example. We shall discuss this issue in more detail in Section 34.1.3.

Finally, a more complicated variant of the example motivates the requisite that $\leftrightarrow$ must be transitive. Consider any psi-calculus where Ψ_1 and Ψ_2 such that $\Psi_1 \otimes \Psi_2 \vdash a \leftrightarrow b$ and $\Psi_1 \otimes \Psi_2 \vdash b \leftrightarrow c$. Then the agent

$$(\nu a, b, c)(\langle\!|\Psi_1|\!\rangle \mid \langle\!|\Psi_2|\!\rangle \mid \overline{a}.\mathbf{0} \mid c.\mathbf{0})$$

has an internal communication using b. If $c \sharp \Psi_1$ and $a, b \sharp \Psi_2$, by scope extension the agent should behave similarly as

$$(\nu a, b)(\langle\!|\Psi_1|\!\rangle \mid \overline{a}.\mathbf{0}) \mid (\nu c)(\langle\!|\Psi_2|\!\rangle \mid c.\mathbf{0})$$

Clearly the left hand component cannot act with a subject a or b, so the only possibility to derive an internal communication is that it can act with subject c given assertion Ψ_2, i.e. that

$$\Psi_2 \rhd (\nu a, b)(\langle\!|\Psi_1|\!\rangle \mid \overline{a}.\mathbf{0}) \xrightarrow{\overline{c}} \cdots$$

This requires that $\Psi_1 \otimes \Psi_2 \vdash a \leftrightarrow c$.

22.3 Bisimilarity

In this section we define a notion of strong bisimilarity on agents.

22.3.1 Definition

In the standard pi-calculus the notion of strong bisimulation is used to formalise the intuition that two agents "behave in the same way"; it is defined as a symmetric binary relation $\mathscr{R}$ satisfying the simulation property: $\mathscr{R}(P,Q)$ implies that for α such that $\mathrm{bn}(\alpha) \sharp Q$,

$$P \xrightarrow{\alpha} P' \Longrightarrow Q \xrightarrow{\alpha} Q' \wedge \mathscr{R}(P',Q')$$

For a psi-calculus we in addition need to take the assertions into consideration. The behaviour of an agent is always taken with respect to an environmental assertion, and therefore a bisimulation must include the assertion to represent this. In other words, we get a ternary relation $\mathscr{R}(\Psi,P,Q)$, saying that P and Q behave in the same way when the environment asserts Ψ. Because of this two additional issues arise. The first is that the agents can affect their environment through their frames (and not only by performing actions), and this must be represented in the definition of bisimulation. The second is that the environment (represented by Ψ in $\mathscr{R}(\Psi,P,Q)$) can change, and for P and Q to be bisimilar they must continue to be related after such changes. This leads to the following definition of strong bisimulation.

Definition 22.10 (Bisimulation)**.** *A* bisimulation $\mathscr{R}$ *is a ternary relation between assertions and pairs of agents such that* $\mathscr{R}(\Psi,P,Q)$ *implies all of*

1. *Static equivalence:* $\Psi \otimes \mathscr{F}(P) \simeq \Psi \otimes \mathscr{F}(Q)$
2. *Symmetry:* $\mathscr{R}(\Psi,Q,P)$
3. *Extension of arbitrary assertion:*
   $\forall \Psi'.\ \mathscr{R}(\Psi \otimes \Psi', P, Q)$
4. *Simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha) \sharp \Psi, Q$ *there exists a* Q' *such that*

$$\Psi \rhd P \xrightarrow{\alpha} P' \Longrightarrow \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge \mathscr{R}(\Psi,P',Q')$$

We define $P \dot{\sim}_\Psi Q$ *to mean that there exists a bisimulation* $\mathscr{R}$ *such that* $\mathscr{R}(\Psi,P,Q)$*, and write* $\dot{\sim}$ *for* $\dot{\sim}_{\mathbf{1}}$*.*

Clauses 2 and 4 are familiar from the pi-calculus. Clause 1 captures the fact that the related agents have exactly the same influence on the environment through their frames, namely that when they add to the existing environment (Ψ) then exactly the same conditions are entailed. Clause 3 means that when the environment changes (by adding a new assertion Ψ') the agents are still related. An example may clarify the role of this clause. Let β be a prefix and let φ be any non-trivial condition, and consider

$$P = \beta.\beta.\mathbf{0} + \beta.\mathbf{0} + \beta.\,\mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{0}$$
$$Q = \beta.\beta.\mathbf{0} + \beta.\mathbf{0}$$

P can nondeterministically choose between three branches and Q between the two first of them. Here P and Q are not bisimilar. If P performs an action corresponding to its third case, reaching the agent $P' = \mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{0}$, there is no way that Q can simulate since neither $Q' = \mathbf{0}$ nor $Q' = \beta.\mathbf{0}$ is equivalent to P' in all environments. In fact, any reasonable variant of bisimulation that equates P and Q will not be preserved by parallel. To see this, let T be $\gamma.(\!|\Psi|\!)$, where γ is any prefix and Ψ an assertion that entails φ. Then the transition $P \mid T \xrightarrow{\beta} P' \mid T$ cannot be simulated by $Q|T$, since $P'|T$ can only do an action γ followed by an action β, whereas $\beta.\mathbf{0}|T$ can do β immediately, and $\mathbf{0}|T$ can do no β at all. This demonstrates why clause 3, extension of arbitrary assertion, is necessary: it says that after each step all possible extensions of the assertion must be considered. If we would merely require this at top level, i.e. remove clause 3 and instead require $\forall\Psi.\mathscr{R}(\Psi, P, Q)$ in the definition of $P \dot{\sim} Q$, the extensions would not recur; as a consequence P and Q in the example would be equivalent, and the equivalence would not be preserved by parallel.

For another example, consider

$$R = \mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{0} \qquad S = \mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\beta.\mathbf{0}$$

In R the condition φ is checked twice. In general R and S are not equivalent. To see this, let Ψ and Ψ' be such that $\Psi \vdash \varphi$ and $\Psi \otimes \Psi' \nvdash \varphi$. We then have that $\Psi \rhd R \xrightarrow{\beta} \mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{0}$ and it cannot be simulated by $\Psi \rhd S \xrightarrow{\beta} \beta.\mathbf{0}$ because of the recurring clause of extension of arbitrary assertion: $\mathbf{if}\ \varphi\ \mathbf{then}\ \beta.\mathbf{0}$ has no transition in the environment $\Psi \otimes \Psi'$. However, if the entailment relation satisfies weakening, i.e. $\Psi \vdash \varphi \Rightarrow \Psi \otimes \Psi' \vdash \varphi$, we get the intuitive result that R and S are bisimilar. Weakening is a quite natural requirement, intuitively it says that no assertion can "undo" any entailments. This also demonstrates why we rejected the smaller and simpler definition of $\dot{\sim}$ as the largest relation satisfying

$$\forall\Psi.\Psi \rhd P \xrightarrow{\beta} P' \Longrightarrow \Psi \rhd Q \xrightarrow{\beta} Q' \wedge P' \dot{\sim} Q'$$

The difference is that here bisimulation recurringly requires to hold for *all* assertions, not only for those that are extensions of the ones passed so far. This would have the unintuitive effect of making R and S in the example above non-bisimilar, even if weakening holds.

22.4 Part outline

This part of the thesis will formalise the meta-theoretical results for psi-calculi, which are more intricate than the ones for CCS or the pi-calculus covered previously in the thesis.

In Chapter 23 we describe how binding sequences are encoded in Isabelle, what properties are required of the sequences, and how alpha-equivalence of terms with binding sequences is computed.

In Chapter 24 we describe how we achieve the parametricity for the terms, assertions, conditions, and operators for psi-calculi; we also define psi-calculi agents, and define parallel substitution on agents.

In Chapter 25 we encode the operational semantics of psi-calculi, and derive nominal induction rules for the transition system.

In Chapter 26 we provide heuristics to automatically derive inversion rules for calculi which use sequences of binders.

In Chapter 27 we define strong bisimilarity for psi-calculi and prove that it is preserved by all operators except Input and Replication. We also define strong equivalence and prove that it is preserved by all operators except Replication. The proofs that both of these equivalences are preserved by Replication are deferred to Chapter 28.

In Chapter 28 we define structural congruence and prove that all structurally congruent agents are also strongly bisimilar and strongly equivalent. We also prove that strong bisimilarity and strong equivalence are preserved by Replication, and hence that strong equivalence is a congruence.

In Chapter 29 we introduce weak bisimilarity for psi-calculi. We define two versions, one where the entailment relation satisfy weakening, i.e. that any condition made true stays true, and one where it does not. The weak bisimilarity with weakening is substantially simpler than the one without.

In Chapter 30 we formalise weak bisimilarity without weakening, and prove that it preserved by all operators except Input and Case.

In Chapter 31 we define weak congruence and prove that it is a congruence.

In Chapter 32 we add weakening to the entailment relation, define a simpler version of weak bisimilarity, and prove that it coincides with the weak bisimilarity defined in Chapter 30.

In Chapter 33 we encode the τ-prefix, and Sum, and prove the standard τ-laws.

Chapter 34 concludes and discusses the experiences made by formalising psi-calculi in parallel with its theoretical development. We also discuss inconsistencies we uncovered in three other process algebras as a result of our formalisation.

23. Binding sequences

The main difficulty when formalising any calculus with binders is to handle alpha-equivalence. The techniques that have been used thus far by theorem provers share the trait that they only reason about single binders. This works well for many calculi, but psi-calculi require binding sequences of arbitrary length. For our psi-calculus datatype (Definition 22.6), a binding sequence is needed in the Input-case where the term $\underline{M}(\lambda\tilde{x})N.P$ has the sequence $\tilde{x}$ binding into N and P. The second place sequences are needed is when defining frames (Definition 22.4). Frames are derived from agents, and as agents can have an arbitrary number of binders, so can the frames. The third occurrence of binding sequences can be found in the operational semantics (Figure 22.1). In the transition $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{a})N} P'$, the sequence $\tilde{a}$ represents the bound names in P which occur in the object N.

In order to formalise these types of calculi efficiently in a theorem prover, libraries with support for sequences of binders have to be added. In this chapter we will describe the techniques used in this thesis. They build on ideas from Urban and Berghofer, and parts of the the theories have already been incorporated into Nominal Isabelle.

23.1 Definitions

To obtain a binding sequence we initially define a nominal datatype which binds a single name to a term of finite support.

Definition 23.1 (bindSeq)**.** *The nominal datatype bindSeq takes one type parameter α of finite support as argument.*

nominal_datatype α *bindSeq* = *Base* α
| *Bind* «*name*» (α *bindSeq*)

We then obtain a binding sequence by recursing over a list of names, binding them one at a time.

Definition 23.2 (bindSequence)**.**

$$\begin{array}{rcl} \mathit{bindSequence} :: \mathit{name\ list} \Rightarrow \alpha \Rightarrow \alpha\ \mathit{bindSeq} & & \\ \mathit{bindSequence}\ \varepsilon\ T & = & \mathit{Base}\ T \\ \mathit{bindSequence}\ (x\widetilde{x})\ T & = & \mathit{Bind}\ x\ \mathit{bindSequence}\ \widetilde{x}\ T \end{array}$$

We will use the syntactic sugar $[\widetilde{x}].T$ to mean $\mathit{bindSequence}\ \widetilde{x}\ T$. Moreover, we overload the freshness operator to include sequences.

Definition 23.3 (Freshness)**.** *A sequence $\widetilde{y}$ fresh for the nominal term X is denoted $\widetilde{y} \sharp X$.*

$$\widetilde{y} \sharp X \stackrel{\text{def}}{=} \forall x \in \mathit{set}\ \widetilde{y}.\ x \sharp X$$

The function *set* converts a list to a set.

23.2 Generating fresh sequences

Alpha-conversion of single binders is achieved by picking a sufficiently fresh name, exchanging it, and swapping all occurrences of the old binder under its scope with the new one. Nominal Isabelle uses the following lemma to obtain sufficiently fresh sequences.

Lemma 23.4. $\exists p.\ (p \cdot \widetilde{x}) \sharp \mathscr{C} \wedge \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x})$

The intuition is that instead of creating a fresh sequence, a permutation is created which when applied to a sequence ensures the needed freshness conditions. The following corollary makes it possible to discard permutations which are sufficiently fresh:

Corollary 23.5.

$$\mathit{If}\ \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ \widetilde{y}\ \mathit{and}\ \mathit{set}\ \widetilde{x} \sharp T\ \mathit{and}\ \mathit{set}\ \widetilde{y} \sharp T\ \mathit{then}\ p \cdot T = T.$$

Proof. By induction on p. □

From this, a corollary to perform alpha-conversions follows.

Corollary 23.6. *If* $\mathtt{set}\ p \subseteq \mathtt{set}\ \tilde{x} \times \mathtt{set}(p \cdot \tilde{x})$ *and* $(p \cdot \tilde{x}) \sharp T$ *then* $[\tilde{x}].T = [p \cdot \tilde{x}].p \cdot T$.

Proof. since $\tilde{x} \sharp [\tilde{x}].T$ and $(p \cdot \tilde{x}) \sharp T$ we have by Cor. 23.5 that $[\tilde{x}].T = p \cdot [\tilde{x}].T$ and hence by equivariance that $[\tilde{x}].T = [p \cdot \tilde{x}]\,.\,p \cdot T$. □

Long proofs tend to introduce alpha-converting permutations and it is therefore important to have a strategy for canceling these. If a term T has

been alpha-converted using the swapping $(a\ b)$, becoming $(a\ b)\cdot T$, it is possible to apply the same swapping to the expression where $(a\ b)\cdot T$ occurs. Using equivariance properties, the swapping can be distributed over the expression, and when it reaches $(a\ b)\cdot T$, it will cancel since $(a\ b)\cdot(a\ b)\cdot T = T$. It can also be canceled from any remaining term U in the expression, as long as $a \sharp U$ and $b \sharp U$. This technique is not directly applicable when dealing with sequences where alpha-converting permutations are used rather than swappings. Even though $(a\ b)\cdot(a\ b)\cdot T = T$, it is not generally the case that $p \cdot p \cdot T = T$. To cancel a permutation on a term, its inverse must be applied, i.e. $p^- \cdot p \cdot T = T$. The permutation will also be canceled from any remaining term U as long as no names of the permutation occur in U.

This method has the problem that the reverse of a permutation does not maintain the logical properties of the original permutation – for instance $(p \cdot \widetilde{x}) \sharp T$ does not imply $(p^- \cdot \widetilde{x}) \sharp T$. For this reason, it is simpler to work with permutations which are their own inverses. The following predicate accomplishes this.

Definition 23.7 (distinctPerm)**.**

$$\mathit{distinctPerm}\ p \stackrel{\text{def}}{=} \mathit{distinct}((\mathit{map}\ \mathit{fst}\ p)@(\mathit{map}\ \mathit{snd}\ p))$$

The *distinct* predicate takes a list as an argument and holds if there are no duplicates in that list. Intuitively, the *distinctPerm* predicate ensures that all names in a permutation are distinct.

Corollary 23.8. *If distinctPerm p then* $p \cdot p \cdot T = T$.

Proof. By induction on p. □

Finally, we extend Lemma 23.4 with the condition *distinctPerm p* to obtain permutations p which can be canceled by applying p again rather than its inverse.

23.3 Alpha-equivalence

For single binders, the nominal approach to alpha-equivalence is quite straightforward. Two terms $[x].T$ and $[y].U$ are equal if and only if either $x = y$ and $T = U$ or $x \neq y$, $x \sharp U$ and $U = (x\ y)\cdot T$. Reasoning about binding sequences is more difficult. Exactly what does it mean for two terms $[\widetilde{x}].T$ and $[\widetilde{y}].U$ to be equal? As long as T and U cannot themselves have binding sequences on a top level we know that $\texttt{length}\ \widetilde{a} = \texttt{length}\ \widetilde{b}$, the problem is what happens when $\widetilde{x}$ and $\widetilde{y}$ partially share names.

The times where we get assumptions such as $[\widetilde{x}].T = [\widetilde{y}].U$ in proofs are when we do induction or inversion over a term with binders. Typically, $[\widetilde{y}].U$ is the term we start with, and $[\widetilde{x}].T$ is the term that appears in the induction

or inversion rule. These rules are designed in such a way that any bound names appearing in the rules can be assumed to be sufficiently fresh. More precisely, we can ensure that $\tilde{x} \sharp \tilde{y}$ and $\tilde{x} \sharp U$.

We know that $[\tilde{x}].T$ is alpha-equivalent to $[\tilde{y}].U$, therefore, there is a permutation which equates them. We first prove the following corollary:

Corollary 23.9.

$$\begin{array}{l} \mathit{If}\ [a].T = [b].U\ \mathit{then}\ (a \in \mathit{supp}\ T) = (b \in \mathit{supp}\ U). \\ \mathit{If}\ [a].T = [b].U\ \mathit{then}\ a \sharp T = b \sharp U. \end{array}$$

Proof. By the definition of alpha-equivalence on terms. □

We can now prove the following lemma:

Lemma 23.10.

$$\frac{[\tilde{x}].T = [\tilde{y}].U \qquad \tilde{x} \sharp \tilde{y}}{\exists p.\ \mathit{set}\ p \subseteq \mathit{set}\ \tilde{x} \times \mathit{set}\ \tilde{y} \wedge \mathit{distinctPerm}\ p \wedge U = p \cdot T}$$

Proof. The intuition here is to construct p by using Cor. 23.9 to filter out the pairs of names from $\tilde{x}$ and $\tilde{y}$ that do not occur in T and U respectively and pairing the rest. Since $\tilde{x} \sharp \tilde{y}$ we can construct p such that p contains no duplicates. The proof is done by induction on the length of $\tilde{x}$ and $\tilde{y}$. □

The problem with this approach is that we do not know how $\tilde{x}$ and $\tilde{y}$ are related – we can equate T and U with this technique, but not $\tilde{x}$ and $\tilde{y}$. To do this, we must also know that $\tilde{x}$ and $\tilde{y}$ are distinct.

Lemma 23.11.

$$\frac{[\tilde{x}].T = [\tilde{y}].U \qquad \tilde{x} \sharp \tilde{y} \qquad \mathit{distinct}\ \tilde{x} \qquad \mathit{distinct}\ \tilde{y}}{\exists p.\ \mathit{set}\ p \subseteq \mathit{set}\ \tilde{x} \times \mathit{set}\ (p \cdot \tilde{x}) \wedge \mathit{distinctPerm}\ p \wedge \tilde{y} = p \cdot \tilde{x} \wedge U = p \cdot T}$$

Proof. Similar to Lemma 23.11, but as we know that $\tilde{x}$ and $\tilde{y}$ are distinct, and that they share no names, the permutation p is created by pairwise combining the names from the sequences. □

23.4 Distinct binding sequences

For calculi with binding sequences we require that all binding sequences are distinct. As we are working up to alpha-equivalence this is a property which must be maintained. Consider the term

$$[xy].\mathit{Base}\ y$$

where the distinct sequence xy binds into the name y. Since we have that

$$[xy].Base\ y = [yy].Base\ y$$

we cannot a priori assume that distinctness is maintained when working up to alpha-equivalence. The reason this equivalence holds is that a name can only bind into a term once – any further occurrence of the binder in the sequence will by definition be fresh for everything under its scope, and hence can be freely switched to any other fresh name.

This example motivates the following corollary.

Corollary 23.12. *If* $\widetilde{x} \sharp \mathscr{C}$ *then* $\exists \widetilde{y}.\ [\widetilde{x}].T = [\widetilde{y}].T \wedge distinct\ \widetilde{y} \wedge \widetilde{y} \sharp \mathscr{C}$.

Proof. Since each name in $\tilde{x}$ can only bind once in T we can construct $\tilde{y}$ by replacing any duplicate name in $\tilde{x}$ with a sufficiently fresh name. □

This lemma states that any binding sequence can be replaced with a distinct one which is at least as fresh as the original.

If we know that all members of a distinct binding sequence are in the support of the term it is binding into, then alpha-converting the sequence maintains that it is distinct.

Lemma 23.13.

$$\frac{[\widetilde{x}].T = [\widetilde{y}].U \qquad distinct\ \widetilde{x} \qquad set\ \widetilde{x} \subseteq supp\ T}{distinct\ \widetilde{y}}$$

Proof. By induction on the length of $\widetilde{x}$ and $\widetilde{y}$. □

Corollary 23.12 and Lemma 23.13 allow us to ensure that binding sequences are kept distinct in the proof contexts.

24. Definitions

The framework for psi-calculi is parametric. When reasoning about the meta-theory, it is not known what terms, assertions, and conditions are being used, how the entailment relation or assertion composition is defined, or what the unit element is; all we know is that they exist, and that they satisfy a set of constraints. This poses certain requirements on the theorem prover. Isabelle has excellent support for this style of reasoning through the use of locales [49] .

Locales allow the user to localise variables, assumptions, and definitions to a restricted part of the theories. Within a locale, these assumptions can be used as if they were proved theorems; definitions can build on them, and lemmas can be proved using them. It can then be instantiated by providing concrete values for the variables, and prove the assumptions of the locale. This is often referred to as interpreting the locale. Once this is done, all lemmas, definitions, and theorems inside the locale can be used for the interpretation.

As all of the assumptions of locales are local to the locale only, there is no risk of introducing inconsistencies in the framework – a locale itself may be inconsistent if falsity can be derived from the assumptions, but the theories outside of the locale will still be sound.

For psi-calculi we will use locales to create a framework which have the axioms of substitution and static equivalence as assumptions. All of the meta-theory of psi-calculi is proven inside this locale. Specific psi-calculus instances can then be derived by providing the necessary parameters and proving that they satisfy the assumptions of the locales.

24.1 Defining psi-calculus agents

Agents for psi-calculi are defined as nominal datatypes, in a similar manner as was done for CCS, and the pi-calculus. However, nominal datatypes in Isabelle have one restriction – neither nested datatypes, nor nested binders are permitted. When defining psi-calculi agents this is problematic in two cases: Firstly, the input prefix requires that an arbitrary number of names can be bound and secondly, the **case** operator requires a list of pairs, with one condition and one agent for each conditional branch.

A nominal datatype for agents is therefore defined using mutual recursion; the datatype is parametrised with three type variables α, β, and γ, for terms, assertions, and conditions respectively.

Isabelle code	Syntax
nominal_datatype (α, β, γ) *psi* =	
Output α α $((\alpha, \beta, \gamma)$ *psi*$)$	$\overline{M}N.Q$
| *Input* α $((\alpha, \beta, \gamma)$ *input*$)$	
| *Case* $((\alpha, \beta, \gamma)$ *psiCase*$)$	
| *Par* $((\alpha, \beta, \gamma)$ *psi*$)$ $((\alpha, \beta, \gamma)$ *psi*$)$	$P \mid Q$
| *Res* «*name*» $((\alpha, \beta, \gamma)$ *psi*$)$	$(\nu x)P$
| *Assert* β	$(\!\mid \Psi \mid\!)$
| *Bang* (α, β, γ) *psi*	$!P$
and (α, β, γ) *psiInput* =	
Trm α $((\alpha, \beta, \gamma)$ *psi*$)$	
| *Bind* «*name*» $((\alpha, \beta, \gamma)$ *input*$)$	
and (α, β, γ) *psiCase* =	
EmptyCase	
| *Cond* γ $((\alpha, \beta, \gamma)$ *psi*$)$ $((\alpha, \beta, \gamma)$ *psiCase*$)$	

Although this datatype formalises agents in psi-calculi, it is not convenient to work with. It is better to reason about input prefixes and cases in such a way that the mutually recursive structure of the *psi*-datatype is transparent. To accomplish this, we create the following wrapper functions.

Definition 24.1 (*inputChain*)**.**

$$\begin{aligned} &\mathit{inputChain} :: \mathit{name\ list} \Rightarrow \alpha \Rightarrow (\alpha, \beta, \gamma)\ \mathit{pi} \Rightarrow (\alpha, \beta, \gamma)\ \mathit{psiInput} \\ &\quad \mathit{inputChain}\ \varepsilon\ N\ P = \mathit{Trm}\ N\ P \\ &\mathit{inputChain}\ (x\widetilde{y})\ N\ P = \mathit{Bind}\ x\ \mathit{inputChain}\ \widetilde{y}\ N\ P \end{aligned}$$

Definition 24.2 (*psiCases*)**.**

$$\begin{aligned} &\mathit{psiCases} :: (\gamma, (\alpha, \beta, \gamma)\ \mathit{pi})\ \mathit{list} \Rightarrow (\alpha, \beta, \gamma)\ \mathit{psiCase} \\ &\quad \mathit{psiCases}\ \varepsilon = \mathit{EmptyCase} \\ &\mathit{psiCases}\ ((\varphi, P)\widetilde{C}) = \mathit{Cond}\ \varphi\ P\ \mathit{psiCases}\ \widetilde{C} \end{aligned}$$

We will use the syntactic sugar $\underline{M}(\lambda\widetilde{x})N.P$ to mean *Input* M (*inputChain* $\widetilde{x}$ N P), and *Cases* $\widetilde{C}$ to mean and *Case* (*psiCases* $\widetilde{C}$)

24.2 Substitution

Substitution on agents in psi-calculi is defined in a similar way as in the pi-calculus – substitutions propagate over the structure of the agents, avoiding capture by the binders. It will be formally defined in Section 24.2.2. What cannot be defined by the framework is how substitution operates on the terms, assertions and conditions as these are parameters, and their exact structure is unknown. Each of them must be equipped with a substitution function which substitutes a sequence of names for a sequence of terms. We will write $X[\tilde{x} := \tilde{T}]$ to denote a term, assertion, or condition X which has its names $\tilde{x}$ substituted for the terms $\tilde{T}$. The intuition is that this substitution should be parallel, and we will refer to them as parallel substitutions – once a name has been replaced by a term, no further substitution is done on the term replaced. In practice, substitution can be any function which satisfies a minimal set of constraints. To represent this, we introduce the notion of a substitution type.

24.2.1 Substitution types

A locale is created which takes a substitution function of type $\alpha \Rightarrow \mathit{name\ list} \Rightarrow \beta\ \mathit{list} \Rightarrow \alpha$ as an argument. The intuition is that the function will substitute names for terms of type β, in the term, assertion or condition of type α. Hence the types α and β can be equal, but this is not required. A locale is created which imposes the following three constraints on substitution.

Definition 24.3. *Locale for substitution types. Given a substitution function of the type $\alpha \Rightarrow \mathit{name\ list} \Rightarrow \beta\ \mathit{list} \Rightarrow \alpha$, the following three requisites must hold:*

$$p \cdot X[\tilde{x} := \tilde{T}] = (p \cdot X)[p \cdot \tilde{x} := p \cdot \tilde{T}] \qquad \textsc{SubstEqvt}$$

$$\frac{|\tilde{x}| = |\tilde{T}| \qquad \mathit{distinct}\ \tilde{x} \qquad \mathit{set}\ \tilde{x} \subseteq \mathit{supp}\ X \qquad y \sharp X[\tilde{x} := \tilde{T}]}{y \sharp \tilde{T}} \qquad \textsc{SubstFresh}$$

$$\frac{|\tilde{x}| = |\tilde{T}| \qquad \mathit{distinctPerm}\ p \qquad \mathit{set}\ p \subseteq \mathit{set}\ \tilde{x} \times \mathit{set}\ (p \cdot \tilde{x}) \qquad (p \cdot \tilde{x}) \sharp X}{X[\tilde{x} := \tilde{T}] = (p \cdot X)[p \cdot \tilde{x} := \tilde{T}]} \qquad \textsc{SubstAlpha}$$

Note that for all requisites, except for SubstEqvt, we require the length of the vectors that are being substituted and substituted for to be of equal length. Moreover, there must be no duplicates in the name vector. As the function is supposed to model parallel substitution this does not impose any serious constraints – if a duplicate name would occur, it would never be

substituted as the parallel substitution terminates whenever a substitution is done. We will now address the three requisites in turn.

The requisite SUBSTEQVT requires that the substitution function is equivariant, as we must be able to propagate permutations over substitutions. Moreover, equivariance gives us an upper bound on the support of substitutions – the result of any equivariant function can not have greater support than the support of its parameters.

Lemma 24.4. *supp* $(X[\tilde{x} := \tilde{T}]) \subseteq \mathit{supp}\, X \cup \mathit{supp}\, \tilde{x} \cup \mathit{supp}\, \tilde{T}$

Proof. Follows from the definition of *supp* (Definition 5.3). □

From this lemma we derive a corresponding freshness lemma.

Lemma 24.5. *If* $y \sharp X$ *and* $y \sharp \tilde{x}$ *and* $y \sharp \tilde{T}$ *then* $y \sharp X[\tilde{x} := \tilde{T}]$.

Proof. Follows directly from Lemma 24.4. □

The requisite SUBSTFRESH intuitively states that the substitution function may not discard the terms that are being substituted into a substitution type – if a term is to be substituted for a name, then the result of the substitution must not have smaller support than that term. The requisite only applies when all names being substituted are in the support of the substitution type. We will come back to exactly why this requisite is necessary in Section 34.3.3.

The final requisite SUBSTALPHA is required to mimic alpha-conversions. If the bound names of an input prefix are alpha-converted, then the corresponding names of the substitution must be similarly converted. This requisite achieves this.

We also derive freshness lemmas to reason about sequences of names.

Lemma 24.6.

- *If* $|\tilde{x}| = |\tilde{T}|$ *and distinct* $\tilde{x}$ *and set* $\tilde{x} \subseteq \mathit{supp}\, X$ *and* $\tilde{y} \sharp X[\tilde{x} := \tilde{T}]$ *then* $\tilde{y} \sharp \tilde{T}$.
- *If* $\tilde{y} \sharp M$ *and* $\tilde{y} \sharp \tilde{x}$ *and* $\tilde{y} \sharp \tilde{T}$ *then* $\tilde{y} \sharp M[\tilde{x} := \tilde{T}]$.

Proof. By induction on $\tilde{y}$. □

With this locale in place, we can proceed to define substitution on psi-calculi agents.

24.2.2 Agent substitution

In order to define substitution for psi-calculi, a locale is created with three instances of the locale for substitution types – one instance for terms, assertions and conditions respectively. Since psi-calculi agents are defined by a mutually inductive definition, the substitution function must be mutually recursive.

Definition 24.7 (Name capture avoiding parallel substitution for psi-calculi agents)**.**

$$\mathbf{0}[\tilde{x} := \tilde{T}] = \mathbf{0}$$
$$(\overline{M}N.P)[\tilde{x} := \tilde{T}] = \overline{M[\tilde{x} := \tilde{T}]}N[\tilde{x} := \tilde{T}].P[\tilde{x} := \tilde{T}]$$
$$(\mathit{Input}\ M\ I)[\tilde{x} := \tilde{T}] = \mathit{Input}\ M[\tilde{x} := \tilde{T}]\ I[\tilde{x} := \tilde{T}]$$
$$(\mathit{Case}\ C)[\tilde{x} := \tilde{T}] = \mathit{Case}\ C[\tilde{x} := \tilde{T}]$$
$$(P \mid Q)[\tilde{x} := \tilde{T}] = P[\tilde{x} := \tilde{T}] \mid Q[\tilde{x} := \tilde{T}]$$
$$\mathit{If}\ y \sharp \tilde{x}\ \mathit{and}\ y \sharp \tilde{T}\ \mathit{then}\ ((\nu y)P)[\tilde{x} := \tilde{T}] = (\nu y)P[\tilde{x} := \tilde{T}].$$
$$(\!|\Psi|\!)[\tilde{x} := \tilde{T}] = (\!|\Psi[\tilde{x} := \tilde{T}]|\!)$$
$$(!P)[\tilde{x} := \tilde{T}] = !P[\tilde{x} := \tilde{T}]$$

$$(\mathit{Trm}\ M\ P)[\tilde{x} := \tilde{T}] = \mathit{Trm}\ (M[\tilde{x} := \tilde{T}])\ (P[\tilde{x} := \tilde{T}])$$
$$\mathit{If}\ y \sharp \tilde{x}\ \mathit{and}\ y \sharp \tilde{T}\ \mathit{then}\ (\mathit{Bind}\ y\ I)[\tilde{x} := \tilde{T}] = \mathit{Bind}\ y\ I[\tilde{x} := \tilde{T}].$$

$$\mathit{EmptyCase}[\tilde{x} := \tilde{T}] = \mathit{EmptyCase}$$
$$(\mathit{Cond}\ \varphi\ P\ C)[\tilde{x} := \tilde{T}] = \mathit{Cond}\ \varphi[\tilde{x} := \tilde{T}]\ P[\tilde{x} := \tilde{T}]\ C[\tilde{x} := \tilde{T}]$$

As in the pi-calculus, the substitutions propagate through the structure of the agents, avoiding capture by the binders. When they reach the terms, assertions or conditions, the appropriate substitution function is applied. Hence, even though substitution is not explicitly defined for the parameters, it is explicitly defined for agents. Therefore, the substitution axioms for the substitution type locale can then be used to derive the same lemmas for agent substitution.

24.3 Nominal morphisms

The four nominal morphisms required for psi-calculi are modeled using locales. A locale is created which requires the four nominal datatypes as parameters, and that they are equivariant.

Definition 24.8 (Nominal morphisms)**.**

$$\begin{array}{rcll} p \cdot \Psi \vdash \varphi & = & (p \cdot \Psi) \vdash (p \cdot \varphi) & \textsc{ImpEqvt} \\ p \cdot \Psi \otimes \Psi' & = & (p \cdot \Psi) \otimes (p \cdot \Psi') & \textsc{CompEqvt} \\ p \cdot M \leftrightarrow N & = & (p \cdot M) \leftrightarrow (p \cdot N) & \textsc{ChanEqEqvt} \\ p \cdot \mathbf{1} & = & \mathbf{1} & \textsc{UnitEqvt} \end{array}$$

24.3.1 Freshness and support

The support of a nominal datatype is derived using permutations, as was described in Section 4.2. As for substitution, the equivariance properties provided in the locale are enough to derive the following results.

Lemma 24.9. *Support properties for the nominal morphisms.*

$$\begin{aligned} supp\,(\Psi \otimes \Psi') &\subseteq supp\,\Psi \cup supp\,\Psi' \\ supp\,(M \leftrightarrow N) &\subseteq supp\,M \cup supp\,N \\ supp\,\mathbf{1} &= \emptyset \end{aligned}$$

Proof. Follows from the definition of *supp*. □

In the same way as for substitution, support inclusion is all that is required and we do not need to strengthen the requisites of the locale.

The following freshness properties can be derived.

Lemma 24.10.

If $x \sharp \Psi$ *and* $x \sharp \Psi'$ *then* $x \sharp \Psi \otimes \Psi'$.
If $x \sharp M$ *and* $x \sharp N$ *then* $x \sharp M \leftrightarrow N$.
$x \sharp \mathbf{1}$

If $\widetilde{x} \sharp \Psi$ *and* $\widetilde{x} \sharp \Psi'$ *then* $\widetilde{x} \sharp \Psi \otimes \Psi'$.
If $\widetilde{x} \sharp M$ *and* $\widetilde{x} \sharp N$ *then* $\widetilde{x} \sharp M \leftrightarrow N$.
$\widetilde{x} \sharp \mathbf{1}$

Proof. By the definition of *fresh*, and Lemma 24.9. The lemmas with sequences of names are discharged by induction on $\widetilde{x}$. □

24.3.2 Static equivalence

In Definition 22.2 two assertions are considered statically equivalent if they entail the same conditions. We will make this definition in two steps. First we will define static implication. An assertion statically implies another one if it entails *at least* the same conditions at the other one.

Definition 24.11 (Static implication)**.**

$$\Psi \leq \Psi' \stackrel{\text{def}}{=} \forall \Phi.\, \Psi \vdash \Phi \longrightarrow \Psi' \vdash \Phi$$

Two assertions can then be defined to be statically equivalent if the statically imply each other.

Definition 24.12 (Static equivalence)**.**

$$\Psi \simeq \Psi' \stackrel{\text{def}}{=} \Psi \leq \Psi' \wedge \Psi' \leq \Psi$$

We also require that static equivalence is equivariant.

Lemma 24.13.

$$\textit{If } \Psi \leq \Psi' \textit{ then } (p \cdot \Psi) \leq (p \cdot \Psi').$$
$$\textit{If } \Psi \simeq \Psi' \textit{ then } (p \cdot \Psi) \simeq (p \cdot \Psi').$$

Proof. Follows from Definitions 24.11, 24.12, and the IMPEQVT axiom. □

24.4 Frames

A frame consist of an assertion and a binding sequence.

Definition 24.14 (Frames)**.**

Isabelle code	*Syntax*
nominal_datatype β *frame* =	
FAssert β	$(\nu\varepsilon)\Psi$
| *FRes* «*name*» (β *frame*)	$(\nu x)F$

As for input prefixes, we bind sequences of names to a frame by recursing over the sequence.

Definition 24.15 (*frameResChain*)**.**

$$\begin{aligned} &\textit{frameResChain} :: \textit{name list} \Rightarrow \beta \Rightarrow \beta\ \textit{frame} \\ &\quad \textit{frameResChain}\ \varepsilon\ F = F \\ &\textit{frameResChain}\ (x\tilde{x})\ F = (\nu x)\textit{frameResChain}\ \tilde{x}\ F \end{aligned}$$

We will use the syntactic sugar $(\nu\tilde{x})F$ to mean *frameResChain* $\tilde{x}$ *F*, and $(\nu\widetilde{b_F})\Psi_F$ to mean $(\nu\widetilde{b_F})((\nu\epsilon)\Psi_F)$.

24.4.1 Frame composition

The frame of an agent is defined by its unguarded assertions, i.e. those not behind a prefix, and its unguarded binders. We define a function which creates a frame by recursively parsing an agent, retaining any assertion and binder it finds, stopping when a prefix is reached. If two agents run in parallel, their frames are composed, and we first need to make this notion precise.

We overload the $\otimes$-operator using nominal functions which compose frames with assertions and frames with frames.

Definition 24.16 (Composing frames with assertions)**.** *A frame F composed with an assertion* Ψ *is written* $F \otimes \Psi$.

$$
\begin{aligned}
((\nu\varepsilon)\Psi) \otimes \Psi' &= (\nu\varepsilon)\Psi' \otimes \Psi \\
\textit{If } x \sharp \Psi' \textit{ then } ((\nu x)F) \otimes \Psi' &= (\nu x)F \otimes \Psi'
\end{aligned}
$$

Definition 24.17 (Composing frames with frames)**.** *A frame F composed with a frame G is written $F \otimes G$.*

$$
\begin{aligned}
((\nu\varepsilon)\Psi) \otimes G &= G \otimes \Psi \\
\textit{If } x \sharp G \textit{ then } ((\nu x)F) \otimes G &= (\nu x)F \otimes G
\end{aligned}
$$

With these functions in place, the following lemma can be created to reason about frame composition.

Lemma 24.18.

$$
\frac{\widetilde{b_F} \sharp \Psi}{((\nu\widetilde{b_F})\Psi_F) \otimes \Psi = (\nu\widetilde{b_F})\Psi \otimes \Psi_F}
$$

$$
\frac{\widetilde{b_F} \sharp \widetilde{b_G} \qquad \widetilde{b_F} \sharp \Psi_G \qquad \widetilde{b_G} \sharp \Psi_F}{((\nu\widetilde{b_F})\Psi_F) \otimes ((\nu\widetilde{b_G})\Psi_G) = (\nu\widetilde{b_F}\widetilde{b_G})\Psi_F \otimes \Psi_G}
$$

Proof. By induction on $\widetilde{b_F}$ for the first case, and $\widetilde{b_G}$ for the second one. □

In order for frame composition to be defined, the bound names of the frames must not clash. This matches precisely the intuition presented in Section 22.2.2.

24.4.2 Frame extraction

Frame extraction is now defined as a nominal function over the mutually recursive *psi* datatype. The function does not recurse past the prefix of an agent.

Definition 24.19 ($\mathscr{F}$)**.** *The function $\mathscr{F}$ returns the frame of an agent and has the type: (α, β, γ) psi $\Rightarrow \beta$ frame.*

$$\begin{aligned}
\mathscr{F}\ \mathbf{0} &= \mathbf{1} \\
\mathscr{F}\ (\mathit{Input}\ M\ I) &= \mathbf{1} \\
\mathscr{F}\ (\overline{M}N.P) &= \mathbf{1} \\
\mathscr{F}\ (\mathit{Case}\ C) &= \mathbf{1} \\
\mathscr{F}\ (P \mid Q) &= (\mathscr{F}\ P) \otimes (\mathscr{F}\ Q) \\
\mathscr{F}\ (\!|\Psi|\!) &= (\nu\varepsilon)\Psi \\
\mathscr{F}\ ((\nu x)P) &= (\nu x)\mathscr{F}\ P \\
\mathscr{F}\ (!P) &= \mathbf{1}
\end{aligned}$$

24.5 Guarded agents

Agents which are either replicated, or have the **case** operator as their topmost operator must be guarded, i.e. no assertion may syntactically occur if it is not under a prefix. More formally, we have the following definition.

Definition 24.20. *The guarded function is a mutually recursive function defined over psi-calculi agents.*

$$\begin{aligned}
&\mathit{guarded}\ (\mathbf{0}) = \mathit{guarded}\ (\overline{M}N.P) = \mathit{guarded}\ (\mathit{Input}\ M\ I) = \mathit{True} \\
&\mathit{guarded}\ (\mathit{Case}\ C) = \mathit{guarded}''\ C \\
&\mathit{guarded}\ (P \mid Q) = (\mathit{guarded}\ P \wedge \mathit{guarded}\ Q) \\
&\mathit{guarded}\ ((\nu x)P) = \mathit{guarded}\ (!P) = \mathit{guarded}\ P \\
\\
&\mathit{guarded}'\ (\mathit{Trm}\ M\ P) = \mathit{guarded}'\ (\mathit{Bind}\ y\ I) = \mathit{False} \\
\\
&\mathit{guarded}''\ \mathit{EmptyCase} = \mathit{True} \\
&\mathit{guarded}''\ (\mathit{Cond}\ \varphi\ P\ C) = (\mathit{guarded}\ P \wedge \mathit{guarded}''\ C)
\end{aligned}$$

In the original version of psi-calculi we defined $\mathbf{0}$ to be $(\!|\mathbf{1}|\!)$ – i.e. the deadlocked process was defined to be the unit frame. All results hold with this definition, but it does not match the desired intuition, as the agent $\mathbf{0}$ would by definition be unguarded.

24.6 Requisites of static equivalence

With the definition of static equivalence in place, we can create a locale with the requisites described in Section 22.2.1.

Definition 24.21 (Requirements of static equivalence)**.**

If $\Psi \vdash (M \leftrightarrow N)$ *then* $\Psi \vdash (N \leftrightarrow M)$.	CESYM
If $\Psi \vdash (M \leftrightarrow N)$ *and* $\Psi \vdash (N \leftrightarrow L)$ *then* $\Psi \vdash (M \leftrightarrow L)$.	CETRANS
If $\Psi \simeq \Psi'$ *then* $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$.	ACOMP
$\Psi \otimes \mathbf{1} \simeq \Psi$	AID
$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$	ACOMM
$(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$	AASSOC

These laws can then be used to form derive a set of inferred static equivalence rules. One such rule states that the assertion component of the frame of any guarded agent is statically equivalent to $\mathbf{1}$.

Lemma 24.22.

If guarded P *and* $\mathcal{F}\, P = (\nu \widetilde{b_P})\Psi_P$ *then* $\Psi_P \simeq \mathbf{1} \wedge \mathit{supp}\, \Psi_P = \emptyset$.

Proof. By induction over P where any bound names avoid $\widetilde{b_P}$ and Ψ_P. □

25. Operational semantics

One of the main contributions of psi-calculi is that the operational semantics, although significantly more expressive than the one for the pi-calculus, has a complexity which remains on par with the original pi-calculus. The main addition is that that an agent must take parallel frames into consideration. A consequence is that transitions are defined with respect to environmental contexts. While the operational semantics for the pi-calculus is an inductively defined binary predicate of agents and residuals, the semantics for psi-calculi will be modeled using a ternary predicate of assertions, agents and residuals.

In this chapter we will cover the basic tactics for defining the psi-calculi operational semantics in Isabelle, comparing it to the ones already formalised for the pi-calculus and CCS. While the pi-calculus splits the operational rules which reason about binders on the label, we will present general tactics to avoid these splits, making the proofs more like their pen-and-paper counterparts.

25.1 Residuals

As with the early semantics of the pi-calculus, the labels of the operational semantics of psi-calculi contain binders in the OPEN-case. Whereas the pi-calculus only requires single binders on the label, psi-calculi require binding sequences; since a term can contain an arbitrary number of names, any output term must open all restricted names it contains. To model this, a nominal datatype is created containing an object, a derivative and a sequence of opened names binding into both.

Definition 25.1 (*boundOutput*)**.**

nominal_datatype (α, β, γ) *boundOutput* =
    BOut α $((\alpha, \beta, \gamma)$ *psi*)
  | *BStep* «*name*» $((\alpha, \beta, \gamma)$ *boundOutput)*

Binding sequences are added by recursively binding names to the nominal datatype in the standard way.

Definition 25.2 (BOresChain)**.**

$$\begin{aligned} &\mathit{BOresChain} :: \mathit{name\ list} \Rightarrow (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} \Rightarrow (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} \\ &\mathit{BOresChain}\ \varepsilon\ B \quad = \quad B \\ &\mathit{BOresChain}\ x\tilde{x}\ B \quad = \quad \mathit{BStep}\ x\ (\mathit{BOresChain}\ \tilde{x}\ B) \end{aligned}$$

We can now define residuals

Definition 25.3. *A residual is parametrised with the same parameters as psi-calculi agents.*

$$\begin{aligned} &\textbf{nominal\_datatype}\ (\alpha,\ \beta,\ \gamma)\ \mathit{residual} = \\ &\quad \mathit{Rin}\ \alpha\ \alpha\ ((\alpha,\ \beta,\ \gamma)\ \mathit{psi}) \\ &\mid\ \mathit{ROut}\ \alpha\ ((\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput}) \\ &\mid\ \mathit{RTau}\ ((\alpha,\ \beta,\ \gamma)\ \mathit{psi}) \end{aligned}$$

Residuals will be denoted using V and W.

When modeling the semantics for the pi-calculus, the PAR and SCOPE rules are split into two cases – one where the label contains bound names, and another where it does not. The reason for this is that the rules need to state conditions on the bound names on the label, namely that they are sufficiently fresh. Also in the psi-calculi semantics, the rules state such conditions.. When doing proofs on paper, it is common to define a function which extracts bound names of the label. However, nominal functions cannot access bound names and hence, a function which extracts the bound names of a residual does not exist.

In order to avoid duplication of the inference rules, and to be able to conduct our proofs as we would on paper, we create a nominal datatype $\alpha\ action$, containing the three types of actions, including the bound names, and a function of type $\alpha\ action \Rightarrow (\alpha,\ \beta,\ \gamma)\ psi \Rightarrow (\alpha,\ \beta,\ \gamma)\ residual$. Given an action, and a derivative this function creates a residual, binding any sequence of names in a bound output action.

Definition 25.4 (action)**.**

Isabelle code	*Syntax*
nominal_datatype α *action* =	
In α α	$\underline{M}N$
| *Out* α "*name list*" α	$\overline{M}\,(\nu\tilde{x})N$
| *Tau*	τ

We now define an infix function $\prec$, which creates a residual from an action and an agent.

Definition 25.5 ($\prec$)**.**

$$\begin{aligned}
&\prec :: \alpha\ action \Rightarrow (\alpha,\ \beta,\ \gamma)\ psi \Rightarrow (\alpha,\ \beta,\ \gamma)\ residual\\
&\underline{M}N \prec P \quad = \quad RIn\ M\ N\ P\\
&\overline{M}\,(\nu\,\tilde{x})N \prec P \quad = \quad ROut\ M\ (BOresChain\ \tilde{x}\ (BOut\ N\ P))\\
&\tau \prec P \quad = \quad RTau\ P
\end{aligned}$$

In this manner we retain the necessary requisite that the bound names of the actions are actually bound in the derivative. We also define functions to extract the subject, the object and the bound names from an action.

Definition 25.6 (*subject*, *object*, *bn*)**.**

$$\begin{aligned}
&subject :: \alpha\ action \Rightarrow \alpha\ option\\
&subject\ (\underline{M}N) \quad = \quad Some\ M\\
&subject\ (\overline{M}\,(\nu\,\tilde{x})N) \quad = \quad Some\ M\\
&subject\ (\tau) \quad = \quad None
\end{aligned}$$

$$\begin{aligned}
&object :: \alpha\ action \Rightarrow \alpha\ option\\
&object\ (\underline{M}N) \quad = \quad Some\ N\\
&object\ (\overline{M}\,(\nu\,\tilde{x})N) \quad = \quad Some\ N\\
&object\ (\tau) \quad = \quad None
\end{aligned}$$

$$\begin{aligned}
&bn :: \alpha\ action \Rightarrow name\ list\\
&bn\ (\underline{M}N) \quad = \quad \varepsilon\\
&bn\ (\overline{M}\,(\nu\,\tilde{x})N) \quad = \quad \tilde{x}\\
&bn\ (\tau) \quad = \quad \varepsilon
\end{aligned}$$

Since actions do not have any binders on their own, functions such as *bn* and *object* can be defined. By modeling residuals in this manner we get the best of two worlds – the bound names of actions do bind into the derivatives, but we can still uniquely determine which binders are used, and the terms and agents under their scope. This means that we will be able to define the operational semantics without the duplicated inference rules, in contrast to the pi-calculus formalisation.

25.1.1 Alpha-equivalence

Dealing with alpha-equivalences of residuals is not entirely straightforward. What does it mean for two residuals $\alpha \prec P$ and $\beta \prec Q$ to be equivalent? The subjects are not under the scope of the binders, so clearly they must be equal, but the object and the derivatives are under the

scope of the bound names of α and β, and hence they are not necessarily syntactically equal, but alpha-equivalent. Moreover, we do not want to resort to case analysis of the structure of the residuals whenever an equality such as this appears in a proof state – the whole point of the residual construction is to avoid separate cases for actions with bound names and for those without. The following lemma obtains an alpha-converting permutation which equates two residuals.

Lemma 25.7.

$$\frac{\begin{array}{lll} \alpha \prec P = \beta \prec Q & bn\ \alpha \sharp bn\ \beta & distinct\ (bn\ \alpha) \\ distinct\ (bn\ \beta) & bn\ \alpha \sharp \alpha \prec P & bn\ \beta \sharp \beta \prec Q \end{array}}{\begin{array}{l} \exists p.\ set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha)) \wedge \beta = p \cdot \alpha \wedge Q = p \cdot P \wedge \\ \quad bn\ \alpha \sharp \beta \wedge bn\ \alpha \sharp Q \wedge bn\ (p \cdot \alpha) \sharp \alpha \wedge bn\ (p \cdot \alpha) \sharp P \end{array}}$$

This lemma is similar to Lemma 23.10. Given two residuals with disjoint bound names, an alpha-converting permutation is constructed by pairing the bound names together. The requirements that $bn\ \alpha \sharp \alpha \prec P$ and $bn\ \beta \sharp \beta \prec Q$ deserve special mention. Bound names of residuals can appear in the subject position. For the equivalence to hold, the bound names must not occur outside their scope; if they did, the alpha-converting permutation would switch them as well.

We also prove an alpha-conversion lemma for residuals.

Lemma 25.8.

$$\frac{\begin{array}{lll} bn\ (p \cdot \alpha) \sharp object\ \alpha & bn\ (p \cdot \alpha) \sharp P & bn\ \alpha \sharp subject\ \alpha \\ bn\ (p \cdot \alpha) \sharp subject\ \alpha & set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha)) & \end{array}}{\alpha \prec P = (p \cdot \alpha) \prec (p \cdot P)}$$

As when alpha-converting binding sequences, the permutation applied to the sequence must be fresh for everything under the scope of the binder. Moreover, for the same reason as the previous lemma, neither the original bound names, nor the new ones, may occur in the subject of the action.

With the residuals in place and a means to alpha-convert them we define the operational semantics as an inductively defined predicate in the standard way. As for the pi-calculus we require that the bound names are fresh for everything outside their scope, and that all binding sequences are distinct.

Definition 25.9 (Operational semantics)**.**

A transition is written $\Psi \rhd P \longmapsto V$ *and intuitively means that* Ψ *is an assertion environment which enables P to enact the residual V. We will use the notation* $\Psi \rhd P \xrightarrow{\alpha} P'$ *to mean* $\Psi \rhd P \longmapsto \alpha \prec P'$

The operational semantics can be found in Figure 25.1.

25.2 Induction rules

The nominal techniques used to derive induction and inversion rules for the pi-calculus and for CCS is not applicable to psi-calculi, as they only work for calculi with single binders.

Recent additions to Nominal Isabelle, by Urban and Berghofer, allow the user to specify sets of names for every inductive case which avoid a specified freshness context when applying the inductive rules. This set does not necessarily have to be the set of binders, although for psi-calculi it is. In [75], Urban and Nipkow prove soundness and completeness of the Milner-Damas typing algorithm W where this is not the case.

The context avoiding sets for the different cases from Figure 25.1 are the following:

$set\ \tilde{x}$	for the INPUT case
$set\ \widetilde{b_Q} \cup set\ (bn\ \alpha)$	for the PAR1 case
$set\ \widetilde{b_P} \cup set\ (bn\ \alpha)$	for the PAR2 case
$set\ \widetilde{b_P} \cup set\ \widetilde{b_Q} \cup set\ \tilde{x}$	for the COMM cases
$\{x\} \cup set\ \tilde{x} \cup set\ \tilde{y}$	for the OPEN case
$\{x\} \cup set\ (bn\ \alpha)$	for the SCOPE case

Isabelle will generate proof obligations to ensure that the sets provided can be alpha-converted to avoid any freshness context provided. These are discharged using the alpha-converting lemmas for binding sequences. As for the single binding case, all inductive rules are saturated with freshness conditions for the binding sequences to be as fresh as possible. Moreover, all binding sequences are required to be distinct. The latter case is not strictly necessary for rules other than the INPUT and OPEN rules, but it makes the inductive rule more powerful and the proofs simpler.

Isabelle also proves the following equivariance lemma automatically.

Lemma 25.10. *If* $\Psi \rhd P \longmapsto V$ *then* $p \cdot \Psi \rhd p \cdot P \longmapsto p \cdot V$.

We will later derive the semantic rules found in Figure 22.1 and discharge the superfluous freshness and distinctness conditions.

25.2.1 Switching assertions

The actions of an agent depend on its environment. The assertion Ψ in $\Psi \rhd P \longmapsto V$ determines which subjects are channel equivalent, as well as which conditions hold in the Case operator.

One main lemma is that a transition can exchange its environmental assertion for a statically equivalent one.

$$\frac{\begin{pmatrix}\Psi \vdash M \doteq K & \mathit{distinct}\,\tilde{x} & |\tilde{x}| = |\tilde{T}| & \mathit{set}\,\tilde{x} \subseteq \mathit{supp}\,N\\ \tilde{x} \sharp \tilde{T} & \tilde{x} \sharp \Psi & \tilde{x} \sharp M & \end{pmatrix}}{\Psi \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{\underline{K}N[\tilde{x}:=\tilde{T}]} P[\tilde{x} := \tilde{T}]}\ \textsc{Input}$$

$$\frac{\Psi \vdash M \doteq K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}\ \textsc{Output}$$

$$\frac{\Psi \rhd P \longmapsto V \qquad (\varphi, P)\ \mathit{mem}\ \tilde{C} \qquad \Psi \vdash \varphi \qquad \mathit{guarded}\,P}{\Psi \rhd \mathit{Cases}\,\tilde{C} \longmapsto V}\ \textsc{Case}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' & \mathscr{F}\,Q = (\nu\widetilde{b_Q})\Psi_Q & \mathit{distinct}\,\widetilde{b_Q}\\ \widetilde{b_Q} \sharp P & \widetilde{b_Q} \sharp Q & \widetilde{b_Q} \sharp \Psi & \widetilde{b_Q} \sharp \alpha & \widetilde{b_Q} \sharp P'\\ \mathit{distinct}\,(\mathit{bn}\,\alpha) & \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha & & & \\ \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp \Psi_Q & \mathit{bn}\,\alpha \sharp Q & \mathit{bn}\,\alpha \sharp P & \end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' & \mathscr{F}\,P = (\nu\widetilde{b_P})\Psi_P & \mathit{distinct}\,\widetilde{b_P}\\ \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp Q & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp \alpha & \widetilde{b_P} \sharp Q'\\ \mathit{distinct}\,(\mathit{bn}\,\alpha) & \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha & & & \\ \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp \Psi_P & \mathit{bn}\,\alpha \sharp P & \mathit{bn}\,\alpha \sharp Q & \end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par2}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{y}\tilde{z})N} P' & x \in \mathit{supp}\,N & & & \\ x \sharp \tilde{y} & x \sharp \tilde{z} & x \sharp M & x \sharp \Psi & \\ \mathit{distinct}\,\tilde{y} & \tilde{y} \sharp \Psi & \tilde{y} \sharp P & \tilde{y} \sharp M & \tilde{y} \sharp \tilde{z}\\ \mathit{distinct}\,\tilde{z} & \tilde{z} \sharp \Psi & \tilde{z} \sharp P & \tilde{z} \sharp M & \end{pmatrix}}{\Psi \rhd (\nu x)P \xrightarrow{\overline{M}(\nu\tilde{y}x\tilde{z})N} P'}\ \textsc{Open}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' & \mathit{distinct}\,(\mathit{bn}\,\alpha) & & & \\ x \sharp \Psi & x \sharp \alpha & x \sharp \mathscr{C} & \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp P\end{pmatrix}}{\Psi \rhd (\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ \textsc{Scope}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{MN} P' \quad \mathcal{F}\, P = (\nu \widetilde{b_P})\Psi_P \quad \textit{distinct}\, \widetilde{b_P} \\ \Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu \widetilde{x})N} Q' \quad \mathcal{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \quad \textit{distinct}\, \widetilde{b_Q} \\ \Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\ \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\ \widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\ \widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\ \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\ \textit{distinct}\, \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \quad \widetilde{x} \sharp \Psi_Q \\ \widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu \widetilde{x})(P' \mid Q')}\ \textsc{Comm1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P' \quad \mathcal{F}\, P = (\nu \widetilde{b_P})\Psi_P \quad \textit{distinct}\, \widetilde{b_P} \\ \Psi \otimes \Psi_P \rhd Q \xrightarrow{KN} Q' \quad \mathcal{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \quad \textit{distinct}\, \widetilde{b_Q} \\ \Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\ \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\ \widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\ \widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\ \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\ \textit{distinct}\, \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \quad \widetilde{x} \sharp \Psi_Q \\ \widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \end{pmatrix}}{\textit{Prop}\, \mathcal{C}\, \Psi\, (P \mid Q)\, (\tau \prec (\nu \widetilde{x})(P' \mid Q'))}\ \textsc{Comm2}$$

$$\frac{\Psi \rhd P \mid\, !P \longmapsto V \qquad \textit{guarded}\, P}{\Psi \rhd\, !P \longmapsto V}\ \textsc{Repl}$$

Figure 25.1: The operational semantics for psi-calculi. All bound names are assumed to be fresh for everything outside of their scope, and all binding sequences are assumed to be distinct. The exceptions are the COMM-rules, where the binding sequences $\widetilde{b_P}$ and $\widetilde{b_Q}$ cannot be assumed to be fresh for K and M respectively.

$$\Psi \rhd R \longmapsto W$$

$$\frac{\begin{pmatrix}\Psi \vdash M \leftrightarrows K & \mathit{distinct}\,\tilde{x} & \mathit{set}\,\tilde{x} \subseteq \mathit{supp}\,N \\ \tilde{x} \sharp \tilde{T} & |\tilde{x}| = |\tilde{T}| & \tilde{x} \sharp \Psi & \tilde{x} \sharp M & \tilde{x} \sharp K & \tilde{x} \sharp \mathscr{C}\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,\underline{M}(\lambda\tilde{x})N.P\,(\underline{K}N[\tilde{x} := \tilde{T}] \prec P[\tilde{x} := \tilde{T}])}\ \textsc{Input}$$

$$\frac{\Psi \vdash M \leftrightarrows K}{\mathit{Prop}\,\mathscr{C}\,\Psi\,(\overline{M}N.P)\,(\overline{K}N \prec P)}\ \textsc{Output}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \longmapsto V & \bigwedge\mathscr{C}.\,\mathit{Prop}\,\mathscr{C}\,\Psi\,P\,V \\ (\varphi, P)\ \mathit{mem}\,\tilde{C} & \Psi \vdash \varphi & \mathit{guarded}\,P\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,(\mathit{Cases}\,\tilde{C})\,V}\ \textsc{Case}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' & \bigwedge\mathscr{C}.\,\mathit{Prop}\,\mathscr{C}\,(\Psi \otimes \Psi_Q)\,P\,(\alpha \prec P') \\ \mathscr{F}\,Q = (\nu\widetilde{b_Q})\Psi_Q & \mathit{distinct}\,\widetilde{b_Q} & \widetilde{b_Q} \sharp \mathscr{C} \\ \widetilde{b_Q} \sharp P & \widetilde{b_Q} \sharp Q & \widetilde{b_Q} \sharp \Psi & \widetilde{b_Q} \sharp \alpha & \widetilde{b_Q} \sharp P' \\ \mathit{distinct}\,(\mathit{bn}\,\alpha) & \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha \\ \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp \Psi_Q & \mathit{bn}\,\alpha \sharp Q & \mathit{bn}\,\alpha \sharp P\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,(P \mid Q)\,(\alpha \prec P' \mid Q)}\ \textsc{Par1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' & \bigwedge\mathscr{C}.\,\mathit{Prop}\,\mathscr{C}\,(\Psi \otimes \Psi_P)\,Q\,(\alpha \prec Q') \\ \mathscr{F}\,P = (\nu\widetilde{b_P})\Psi_P & \mathit{distinct}\,\widetilde{b_P} & \widetilde{b_P} \sharp \mathscr{C} \\ \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp Q & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp \alpha & \widetilde{b_P} \sharp Q' \\ \mathit{distinct}\,(\mathit{bn}\,\alpha) & \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha \\ \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp \Psi_P & \mathit{bn}\,\alpha \sharp P & \mathit{bn}\,\alpha \sharp Q\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,(P \mid Q)\,(\alpha \prec P \mid Q')}\ \textsc{Par2}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' & \bigwedge\mathscr{C}.\,\mathit{Prop}\,\mathscr{C}\,\Psi\,P\,(\alpha \prec P') \\ \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha & \mathit{distinct}\,(\mathit{bn}\,\alpha) \\ x \sharp \Psi & x \sharp \alpha & x \sharp \mathscr{C} \\ \mathit{bn}\,\alpha \sharp \Psi & \mathit{bn}\,\alpha \sharp P & \mathit{bn}\,\alpha \sharp \mathscr{C}\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,((\nu x)P)\,(\alpha \prec (\nu x)P')}\ \textsc{Scope}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{y})\tilde{z}N} P' & \bigwedge\mathscr{C}.\,\mathit{Prop}\,\mathscr{C}\,\Psi\,P\,(\overline{M}(\nu\tilde{x}\tilde{y})N \prec P') \\ x \in \mathit{supp}\,N & x \sharp \tilde{y} & x \sharp \tilde{z} & x \sharp M & x \sharp \Psi & x \sharp \mathscr{C} \\ \mathit{distinct}\,\tilde{y} & \tilde{y} \sharp \Psi & \tilde{y} \sharp P & \tilde{y} \sharp M & \tilde{y} \sharp \tilde{z} & \tilde{y} \sharp \mathscr{C} \\ \mathit{distinct}\,\tilde{z} & \tilde{z} \sharp \Psi & \tilde{z} \sharp P & \tilde{z} \sharp M & \tilde{z} \sharp \mathscr{C}\end{pmatrix}}{\mathit{Prop}\,\mathscr{C}\,\Psi\,((\nu x)P)\,(\overline{M}(\nu\tilde{y}x\tilde{z})N \prec P')}\ \textsc{Open}$$

$$\frac{\left(\begin{array}{l}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\underline{M}N} P' \quad \mathcal{F}\, P = (\nu \widetilde{b_P})\Psi_P \quad \mathit{distinct}\, \widetilde{b_P} \\
\bigwedge \mathscr{C}.\, \mathit{Prop}\, \mathscr{C}\, (\Psi \otimes \Psi_Q)\, P\, (\underline{M}N \prec P') \quad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu \tilde{x})N} Q' \quad \mathcal{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \quad \mathit{distinct}\, \widetilde{b_Q} \\
\bigwedge \mathscr{C}.\, \mathit{Prop}\, \mathscr{C}\, (\Psi \otimes \Psi_P)\, Q\, (\overline{K}(\nu \tilde{x})N \prec Q') \quad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \tilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\
\widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \tilde{x} \\
\mathit{distinct}\, \tilde{x} \quad \tilde{x} \sharp \Psi \quad \tilde{x} \sharp \Psi_P \quad \tilde{x} \sharp \Psi_Q \\
\tilde{x} \sharp P \quad \tilde{x} \sharp M \quad \tilde{x} \sharp Q \quad \tilde{x} \sharp K \quad \tilde{x} \sharp \mathscr{C}
\end{array}\right)}{\mathit{Prop}\, \mathscr{C}\, \Psi\, (P \mid Q)\, (\tau \prec (\nu \tilde{x})(P' \mid Q'))}\ \textsc{Comm1}$$

$$\frac{\left(\begin{array}{l}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu \tilde{x})N} P' \quad \mathcal{F}\, P = (\nu \widetilde{b_P})\Psi_P \quad \mathit{distinct}\, \widetilde{b_P} \\
\bigwedge \mathscr{C}.\, \mathit{Prop}\, \mathscr{C}\, (\Psi \otimes \Psi_Q)\, P\, (\overline{M}(\nu \tilde{x})N \prec P') \quad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\underline{K}N} Q' \quad \mathcal{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \quad \mathit{distinct}\, \widetilde{b_Q} \\
\bigwedge \mathscr{C}.\, \mathit{Prop}\, \mathscr{C}\, (\Psi \otimes \Psi_P)\, Q\, (\underline{K}N \prec Q') \quad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \tilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\
\widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \tilde{x} \\
\mathit{distinct}\, \tilde{x} \quad \tilde{x} \sharp \Psi \quad \tilde{x} \sharp \Psi_P \quad \tilde{x} \sharp \Psi_Q \\
\tilde{x} \sharp P \quad \tilde{x} \sharp M \quad \tilde{x} \sharp Q \quad \tilde{x} \sharp K \quad \tilde{x} \sharp \mathscr{C}
\end{array}\right)}{\mathit{Prop}\, \mathscr{C}\, \Psi\, (P \mid Q)\, (\tau \prec (\nu \tilde{x})(P' \mid Q'))}\ \textsc{Comm2}$$

$$\frac{\Psi \rhd P \mid !P \longmapsto V \quad \bigwedge \mathscr{C}.\, \mathit{Prop}\, \mathscr{C}\, \Psi\, (P \mid !P)\, V \quad \mathit{guarded}\, P}{\mathit{Prop}\, \mathscr{C}\, \Psi\, (!P)\, V}\ \textsc{Repl}$$

$$\overline{\mathit{Prop}\, \mathscr{C}\, \Psi\, R\, W}$$

Figure 25.2: Derived induction rule for the operational semantics of psi-calculi. The inductive cases share the name of the semantic rule which they are derived from. For space reasons, all meta quantifiers for each inductive step has been removed – every term of every case is locally universally quantified.

Lemma 25.11. *If* $\Psi \rhd P \longmapsto V$ *and* $\Psi \simeq \Psi'$ *then* $\Psi' \rhd P \longmapsto V$.

Proof. By induction on $\Psi \rhd P \longmapsto V$. The only non-trivial cases are the PAR and COMM cases where ACOMP is used in order to apply the induction hypothesis. □

25.2.2 Deriving freshness conditions

In order to derive the semantic rules from Figure 22.1, we must remove the superfluous freshness conditions from the semantics defined in Isabelle. The standard way is to provide lemmas which derive freshness conditions from the semantics. Moreover, as psi-calculi uses an early operational semantics, lemmas which allow the input actions to mimic alpha-conversions must be provided. Finally, distinctness of all binding sequences must be derivable. We will start proving the relevant properties for input transitions, then output transitions and finally for τ-transitions.

25.2.2.1 Input actions

We first prove which freshness conditions can be derived from input transitions.

Lemma 25.12. *If* $\Psi \rhd P \xrightarrow{MN} P'$ *and* $x \sharp P$ *and* $x \sharp N$ *then* $x \sharp P'$.

Proof. By induction on $\Psi \rhd P \xrightarrow{MN} P'$. □

In order to guarantee that a name is fresh for the derivative of an agent, it may not occur in the agent itself, nor in the object of the action, as new names can be introduced by substitution. Proving the same for sequences follows naturally.

Lemma 25.13. *If* $\Psi \rhd P \xrightarrow{MN} P'$ *and* $\tilde{x} \sharp P$ *and* $\tilde{x} \sharp N$ *then* $\tilde{x} \sharp P'$.

Proof. By induction on $\tilde{x}$ and Lemma 25.12. □

A subject on a label does not necessarily have to be in the agent which generates the transition; the subject on the label needs only be channel equivalent to the subject in the agent. This is in contrast to the pi-calculus or CCS where the subject on the label syntactically have to occur in the agent. In these calculi we can always know that a name fresh for an agent is always fresh for the subjects of its actions. As the psi-calculi semantics has freshness conditions on the subject of actions, we must admit alpha-converting permutations to be applied to a subject. The following lemma does this.

Lemma 25.14.

$$\frac{\Psi \rhd P \xrightarrow{\underline{M}N} P' \qquad x \sharp P \qquad y \sharp P}{(x\,y)\cdot\Psi \rhd P \xrightarrow{(x\,y)\cdot\underline{M}N} P'}$$

Proof. By induction on $\Psi \rhd P \xrightarrow{\underline{M}N} P'$. □

The intuition of this lemma is that if two names do not occur in an agent, they can still be swapped for the subject of its action as long as the corresponding permutation is applied to the environment. As the environment determines which terms are channel equivalent.

A corresponding lemma is also needed for permutations.

Lemma 25.15.

$$\frac{\Psi \rhd P \xrightarrow{\underline{M}N} P' \qquad \mathit{set}\, p \subseteq X \times Y \qquad X \sharp P \qquad Y \sharp P}{p\cdot\Psi \rhd P \xrightarrow{(p\cdot M)N} P'}$$

Proof. By induction on p and Lemma 25.14. □

Since the COMM rules contain both input and output actions, and the output actions can be alpha-converted, it must be possible for the input transition to match these alpha-conversions. The following lemma does this.

Lemma 25.16.

$$\frac{\begin{array}{l}\Psi \rhd P \xrightarrow{\underline{M}N} P' \\ \mathit{set}\, p \subseteq \mathit{set}\, \tilde{x} \times \mathit{set}\,(p\cdot\tilde{x}) \qquad \mathit{distinctPerm}\, p \qquad \tilde{x} \sharp P \qquad (p\cdot\tilde{x}) \sharp P\end{array}}{\Psi \rhd P \xrightarrow{\underline{M}(p\cdot N)} p\cdot P'}$$

Proof.

- From the assumptions we have that $p \cdot \Psi \rhd P \xrightarrow{(p\cdot M)N} P'$ by Lemma 25.15.
- Hence $p\cdot p\cdot\Psi \rhd p\cdot P \longmapsto p\cdot\underline{(p\cdot M)}N \prec P'$ by Lemma 25.10.
- With $\mathit{set}\, p \subseteq \mathit{set}\, \tilde{x} \times \mathit{set}\,(p\cdot\tilde{x})$, $\mathit{distinctPerm}\, p$, $\tilde{x} \sharp P$, and $(p\cdot\tilde{x}) \sharp P$ the permutations simplify away, and we get that $\Psi \rhd P \xrightarrow{\underline{M}(p\cdot N)} p\cdot P'$.

□

Finally, we must prove that whenever an agent of the form $\underline{M}(\lambda\tilde{x})N.P$ does an action, $\tilde{x}$ is distinct. Since this is a requisite of the semantics, what we in reality must prove is that there is no way to alpha-convert $\tilde{x}$ such that it is not distinct.

Lemma 25.17. *If* $\Psi \rhd \underline{M}(\lambda\tilde{x})N.P \longmapsto V$ *then distinct* $\tilde{x}$.

Proof. By case analysis on $\Psi \rhd \underline{M}(\lambda \tilde{x})N.P \longmapsto V$. The INPUT case gives us an agent $\underline{K}(\lambda \tilde{y})L.Q$ such that $\underline{M}(\lambda \tilde{x})N.P = \underline{K}(\lambda \tilde{y})L.Q$, *distinct* $\tilde{y}$, and *set* $\tilde{y} \subseteq$ *supp* L. Lemma 23.13 then proves that *distinct* $\tilde{x}$. □

25.2.2.2 Output actions

The induction rule from Figure 25.2 works well only as long as the predicate to be proven does not depend on anything under the scope of a binder. Trying to prove the following lemma illustrates the problem.

$$\textit{If } \Psi \rhd P \xrightarrow{\overline{M}(\nu \tilde{x})N} P' \textit{ and } x \sharp P \textit{ and } x \sharp \tilde{x} \textit{ then } x \sharp N \textit{ and } x \sharp P'$$

This lemma is provable by induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu \tilde{x})N} P'$. The problem is that the induction rule will not prove this lemma in a satisfactory way. Every applicable case in the induction rule will introduce its own bound output term $\overline{K}(\nu \tilde{y})L \prec P''$ where we know that $\overline{K}(\nu \tilde{y})L \prec P'' = \overline{M}(\nu \tilde{x})N \prec P'$. What we need to prove relates to the term P', what the inductive hypotheses will give us is something related to the term P'', where all we know is that they are part of alpha-equivalent terms.

Proving this lemma on its own is not too difficult but in every step of every proof of this type, manual alpha-conversions and equivariance properties are needed. An induction rule which solves this problem can be found in Figure 25.3

The difference between this rule and the induction rule in Figure 25.2 is that the predicate in Figure 25.2 takes a residual V as one argument and the predicate in this rule takes α and P' as two separate ones. By disassociating the action from the derivative in this manner we have lost the ability to alpha-convert the residual, but we have gained the ability to reason about terms under the scope of its binders. The extra ALPHA case in the induction rule is designed to allow the predicate to mimic the alpha-conversion abilities we have lost. Moreover, when proving this induction rule, Lemma 25.8 is used in each step to generate the alpha-converting permutation. This lemma requires that the bound names of the transition are distinct, and do not occur free in the residual, in effect the subject of the action. These requirements can be found as two extra requirements $bn\ \alpha \sharp subject\ \alpha$, and *distinct* $(bn\ \alpha)$ to the induction rule in Figure 25.2. In every case, *Prop* is proven in the standard way and then alpha-converted using the new inductive case ALPHA.

With this lemma, we must prove that the predicate we are trying to prove respects alpha-conversions. The advantage is that it only has to be done once for each proof. Moreover, the case is very general and does not require the agents or actions to be of a specific form.

We can now prove the freshness lemma.

Lemma 25.18. *If* $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ *and* $\tilde{x} \sharp M$ *and distinct* $\tilde{x}$ *and* $x \sharp P$ *and* $x \sharp \tilde{x}$ *then* $x \sharp N$. *and* $x \sharp P'$

Proof. By induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$, using the induction rule from Figure 25.3. □

This lemma illustrates a problem with the method used to derive general induction rules – it requires the extra freshness and distinctness assumptions $\tilde{x} \sharp M$ and *distinct* $\tilde{x}$. These assumptions are not strictly necessary to solve this lemma, but they are needed in order to invoke the induction rule in Figure 25.2. If required, they can later be removed by manual alpha-conversions. This is a tedious process, and the user has to decide for each proof if its worth the effort. Nominal Isabelle is very good at ensuring that binding sequences are sufficiently fresh, and infrastructure to prove the distinctness property will be provided. In practice, these extra constraints are unproblematic; if the standard induction rule were used, manual alpha-conversions would have to be done for every inductive step.

The corresponding proof for binding sequences follows immediately.

Lemma 25.19. *If* $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ *and* $\tilde{x} \sharp M$ *and distinct* $\tilde{x}$ *and* $\tilde{y} \sharp P$ *then* $\tilde{y} \sharp N$ *and* $\tilde{y} \sharp P'$

Proof. By induction on $\tilde{x}$ and Lemma 25.18. □

We must also prove that for every transition $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$, $\tilde{x}$ is distinct.

Lemma 25.20. *If* $\Psi \rhd P \xrightarrow{\alpha} P'$ *then distinct* $(bn\,\alpha)$.

Proof. Similar to Lemma 25.17 □

We also require lemmas which allow us to apply permutations to the subject position of output actions.

Lemma 25.21. *If* $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ *and* $\tilde{x} \sharp M$ *and* $x \sharp P$ *and* $y \sharp P$ *then* $(x\,y) \cdot \Psi \rhd P \xrightarrow{\overline{(x\,y) \cdot M}(\nu\tilde{x})N} P'$.

Proof. By induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ using the induction rule from Figure 25.2. The condition that $\tilde{x}$ is distinct follows from Lemma 25.20. □

The corresponding lemma for permutations is not entirely straightforward.

Lemma 25.22. *If* $\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ *and* $set\,p \subseteq set\,\tilde{y} \times set\,\tilde{z}$ *and* $\tilde{y} \sharp P$ *and* $\tilde{z} \sharp P$ *then* $p \cdot \Psi \rhd P \xrightarrow{\overline{(p \cdot M)}(\nu\tilde{x})N} P'$.

Figure 25.3: Derived induction rule for transitions of the form $\Psi \rhd R \xrightarrow{\alpha} R'$. The extra ALPHA case ensures that the bound names of α can be freely alpha-converted in *Prop*. The inductive cases share the name of the semantic rule which they are derived from. For space reasons, all meta quantifiers for each inductive step have been removed – every term of every case is locally quantified. To apply the rule, the requisites $bn\ \alpha \sharp subject\ \alpha$ and $distinct\ (bn\ \alpha)$ must be met.

$$\left[\Psi \rhd R \xrightarrow{\alpha} R' \qquad bn\ \alpha \sharp subject\ \alpha \qquad distinct\ (bn\ \alpha)\right]$$

$$\frac{\begin{pmatrix} bn\ \alpha \sharp \Psi \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp subject\ \alpha \qquad bn\ \alpha \sharp \mathscr{C} \\ bn\ \alpha \sharp bn\ (p \cdot \alpha) \qquad set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha)) \\ distinctPerm\ p \qquad bn\ (p \cdot \alpha) \sharp \alpha \qquad bn\ (p \cdot \alpha) \sharp P' \qquad Prop\ \mathscr{C}\ P\ \alpha\ P' \end{pmatrix}}{Prop\ \mathscr{C}\ \Psi\ P\ (p \cdot \alpha)\ (p \cdot P')}\ \text{ALPHA}$$

$$\frac{\begin{pmatrix} \Psi \vdash M \leftrightarrow K \qquad distinct\ \widetilde{x} \qquad set\ \widetilde{x} \subseteq supp\ N \\ \widetilde{x} \sharp \widetilde{T} \qquad |\widetilde{x}| = |\widetilde{T}| \qquad \widetilde{x} \sharp \Psi \qquad \widetilde{x} \sharp M \qquad \widetilde{x} \sharp K \qquad \widetilde{x} \sharp \mathscr{C} \end{pmatrix}}{Prop\ \mathscr{C}\ \Psi\ \underline{M}(\lambda\widetilde{x})N.P\ (\underline{K}N[\widetilde{x} := \widetilde{T}])\ (P[\widetilde{x} := \widetilde{T}])}\ \text{INPUT}$$

$$\frac{\Psi \vdash M \leftrightarrow K}{Prop\ \mathscr{C}\ \Psi\ (\overline{M}N.P)\ (\overline{K}N)\ P}\ \text{OUTPUT}$$

$$\frac{\begin{pmatrix} \Psi \rhd P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ \Psi\ P\ \alpha\ P' \\ (\varphi, P)\ mem\ \widetilde{C} \qquad \Psi \vdash \varphi \qquad guarded\ P \end{pmatrix}}{Prop\ \mathscr{C}\ \Psi\ (Cases\ \widetilde{C})\ \alpha\ P'}\ \text{CASE}$$

$$\frac{\begin{pmatrix} \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ \alpha\ P' \\ \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad distinct\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp \mathscr{C} \\ \widetilde{b_Q} \sharp P \qquad \widetilde{b_Q} \sharp Q \qquad \widetilde{b_Q} \sharp \Psi \qquad \widetilde{b_Q} \sharp \alpha \qquad \widetilde{b_Q} \sharp P' \\ distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp subject\ \alpha \\ bn\ \alpha \sharp \Psi \qquad bn\ \alpha \sharp \Psi_Q \qquad bn\ \alpha \sharp Q \qquad bn\ \alpha \sharp P \end{pmatrix}}{Prop\ \mathscr{C}\ \Psi\ (P \mid Q)\ \alpha\ (P' \mid Q)}\ \text{PAR1}$$

$$\frac{\begin{pmatrix} \Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' \qquad \bigwedge \mathscr{C}.\ Prop\ \mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ \alpha\ Q' \\ \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \mathscr{C} \\ \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp Q \qquad \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp \alpha \qquad \widetilde{b_P} \sharp Q' \\ distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp subject\ \alpha \\ bn\ \alpha \sharp \Psi \qquad bn\ \alpha \sharp \Psi_P \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp Q \end{pmatrix}}{Prop\ \mathscr{C}\ \Psi\ (P \mid Q)\ \alpha\ (P \mid Q')}\ \text{PAR2}$$

$$\frac{\left(\begin{array}{l}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\underline{M}N} P' \qquad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ (\underline{M}N)\ P' \\
\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu \widetilde{x})N} Q' \qquad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ (\overline{K}(\nu \widetilde{x})N)\ Q' \\
\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad distinct\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrows K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\
\widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\
distinct\ \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \quad \widetilde{x} \sharp \Psi_Q \\
\widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \quad \widetilde{x} \sharp \mathscr{C}
\end{array}\right)}{Prop\,\mathscr{C}\ \Psi\ (P \mid Q)\ (\tau)\ ((\nu \widetilde{x})(P' \mid Q'))}\ \textsc{Comm1}$$

$$\frac{\left(\begin{array}{l}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P' \qquad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ (\overline{M}(\nu \widetilde{x})N)\ P' \\
\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\underline{K}N} Q' \qquad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ (\underline{K}N)\ Q' \\
\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad distinct\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrows K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\
\widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\
distinct\ \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \quad \widetilde{x} \sharp \Psi_Q \\
\widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \quad \widetilde{x} \sharp \mathscr{C}
\end{array}\right)}{Prop\,\mathscr{C}\ \Psi\ (P \mid Q)\ (\tau)\ ((\nu \widetilde{x})(P' \mid Q'))}\ \textsc{Comm2}$$

$$\frac{\left(\begin{array}{l}
\Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{y}\widetilde{z})N} P' \qquad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ P\ (\overline{M}(\nu \widetilde{x}\widetilde{y})N)\ P' \\
x \in supp\,N \qquad x \sharp \widetilde{y} \qquad x \sharp \widetilde{z} \qquad x \sharp M \qquad x \sharp \Psi \qquad x \sharp \mathscr{C} \\
distinct\ \widetilde{y} \qquad \widetilde{y} \sharp \Psi \qquad \widetilde{y} \sharp P \qquad \widetilde{y} \sharp M \qquad \widetilde{y} \sharp \widetilde{z} \qquad \widetilde{y} \sharp \mathscr{C} \\
distinct\ \widetilde{z} \qquad \widetilde{z} \sharp \Psi \qquad \widetilde{z} \sharp P \qquad \widetilde{z} \sharp M \qquad \widetilde{z} \sharp \mathscr{C}
\end{array}\right)}{Prop\,\mathscr{C}\ \Psi\ ((\nu x)P)\ (\overline{M}(\nu \widetilde{y}x\widetilde{z})N)\ P'}\ \textsc{Open}$$

$$\left[\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' \qquad \bigwedge\mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ P\ \alpha\ P' \\ bn\,\alpha \sharp subject\,\alpha \qquad distinct\,(bn\,\alpha) \\ bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp P \qquad bn\,\alpha \sharp \mathscr{C} \\ x \sharp \Psi \qquad x \sharp \alpha \qquad x \sharp \mathscr{C}\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ ((\nu x)P)\ \alpha\ ((\nu x)P')}\ \textsc{Scope}\right.$$

$$\left.\frac{\left(\Psi \rhd P \mid !P \xrightarrow{\alpha} P' \qquad \bigwedge\mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ (\alpha \prec P') \qquad guarded\,P\right)}{Prop\,\mathscr{C}\ \Psi\ (!P)\ (\alpha \prec P')}\ \textsc{Repl}\right]$$

$$\overline{Prop\,\mathscr{C}\ \Psi\ R\ \alpha\ R'}$$

Proof. First, the special case of this lemma is proven with the extra requirements that $\tilde{x} \sharp \tilde{y}$ and $\tilde{x} \sharp \tilde{z}$. The proof is done by induction on p. The extra requirements are needed for the inductive steps where Lemma 25.21 is used to add a swapping of names obtained from $\tilde{x}$ and $\tilde{y}$ to M. If these names where not fresh for $\tilde{x}$, the requisite that $\tilde{x} \sharp M$ from Lemma 25.21 would not be fulfilled, and the proof not possible to complete.

Once the special case is proven, the main lemma is proven by manually alpha-converting $\tilde{x}$ to not clash with $\tilde{y}$ or $\tilde{z}$. □

25.2.2.3 τ-actions

Once the freshness proofs have been done for input and output actions, the ones for τ-actions follow directly.

Lemma 25.23. *If $\Psi \rhd P \xrightarrow{\tau} P'$ and $x \sharp P$ then $x \sharp P'$.*

Proof. By induction on $\Psi \rhd P \xrightarrow{\tau} P'$. In the COMM case, lemmas 25.12 and 25.18 are used do ensure that x is fresh for the derivatives. □

Lemma 25.24. *If $\Psi \rhd P \xrightarrow{\tau} P'$ and $\tilde{x} \sharp P$ then $\tilde{x} \sharp P'$.*

Proof. By induction on $\tilde{x}$ and Lemma 25.23 □

We can now prove the following general freshness lemmas.

Lemma 25.25.

$$\frac{\Psi \rhd P \xrightarrow{\alpha} P' \qquad bn\,\alpha \sharp subject\,\alpha \qquad distinct\,(bn\,\alpha) \qquad x \sharp \alpha \qquad x \sharp P}{x \sharp P'}$$

Proof. By case analysis of α and Lemmas 25.12, 25.18, and 25.23. □

Lemma 25.26.

$$\frac{\Psi \rhd P \xrightarrow{\alpha} P' \qquad bn\,\alpha \sharp subject\,\alpha \qquad distinct\,(bn\,\alpha) \qquad \widetilde{x} \sharp P \qquad \widetilde{x} \sharp \alpha}{\widetilde{x} \sharp P'}$$

Proof. By induction on $\widetilde{x}$ and Lemma 25.25. □

With the lemmas to derive freshness and distinctness conditions in place, it is possible to derive the operational semantics from Figure 22.1 from the ones in Figure 25.1. The resulting semantics can be found in Figure 25.4

25.3 Frame induction rules

The induction rule in Figure 25.2 demonstrates the need to create custom induction rules allowing for proofs which reason about terms under the scope of binders. Another common type of proof for psi-calculi is to do induction over a transition where the agent has a specific frame. Trying to prove the following lemma illustrates this.

$$\frac{\Psi \rhd P \xrightarrow{\underline{\underline{M}}N} P' \qquad \mathscr{F}\,P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\,\widetilde{b_P} \qquad \Psi \otimes \Psi_P \vdash M \leftrightarrow K \qquad \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp M \qquad \widetilde{b_P} \sharp K}{\Psi \rhd P \xrightarrow{\underline{\underline{K}}N} P'}$$

This lemma states that an action subject can be replaced by a channel equivalent one, where the frame of P is allowed to help. The proof is done by induction on $\Psi \rhd P \xrightarrow{\underline{\underline{M}}N} P'$ and suffers from the same problem as Lemma 25.18 – every inductive step will generate a frame alpha-equivalent to $(\nu \widetilde{b_P})\Psi_P$ and many tedious alpha-conversions have to be done to prove the lemma. A similar induction rule as for output transitions can be derived to solve the problem. The induction rule for doing induction on transitions where the agent has a specific frame can be found in Figure 25.5.

There are times where a subject has to be fresh for any name which is fresh for the originating agent. Any subject of a transition has to be channel equivalent to the subject of a prefix which syntactically occurs in the originating agent. Therefore, any name which is fresh for the agent, and which does not occur in the bound names of its frame, must also be fresh for its subjects. Intuitively, the following lemma retrieves the prefix subject of an agent which is channel equivalent to the subject of its transition.

$$\frac{\left(\Psi \vdash M \leftrightarrow K \qquad \mathit{distinct}\ \tilde{x} \qquad \mathit{set}\ \tilde{x} \subseteq \mathit{supp}\ N \qquad |\tilde{x}| = |\tilde{T}|\right)}{\Psi \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{K N[\tilde{x}:=\tilde{T}]} P[\tilde{x} := \tilde{T}]}\ \textsc{Input}$$

$$\frac{\left(\Psi \vdash M \leftrightarrow K\right)}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}\ \textsc{Output} \qquad \frac{\begin{pmatrix}\Psi \rhd P \longmapsto V \\ (\varphi, P)\ \mathit{mem}\ \mathit{Cs} \\ \Psi \vdash \varphi \qquad \mathit{guarded}\ P\end{pmatrix}}{\Psi \rhd \mathit{Cases}\ \mathit{Cs} \longmapsto V}\ \textsc{Case}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \qquad \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \\ bn\ \alpha \sharp Q \qquad \widetilde{b_Q} \sharp \Psi \qquad \widetilde{b_Q} \sharp P \qquad \widetilde{b_Q} \sharp \alpha\end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' \qquad \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \\ bn\ \alpha \sharp P \qquad \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp Q \qquad \widetilde{b_P} \sharp \alpha\end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{MN} P' \\ \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad \Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu\tilde{x})N} Q' \\ \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad \Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\ \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp Q \qquad \widetilde{b_P} \sharp M \qquad \widetilde{b_P} \sharp \widetilde{b_Q} \\ \widetilde{b_Q} \sharp \Psi \qquad \widetilde{b_Q} \sharp P \qquad \widetilde{b_Q} \sharp Q \qquad \widetilde{b_Q} \sharp K \qquad \tilde{x} \sharp P\end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\tilde{x})(P' \mid Q')}\ \textsc{Comm1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P' \\ \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad \Psi \otimes \Psi_P \rhd Q \xrightarrow{KN} Q' \\ \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad \Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\ \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp Q \qquad \widetilde{b_P} \sharp M \qquad \widetilde{b_P} \sharp \widetilde{b_Q} \\ \widetilde{b_Q} \sharp \Psi \qquad \widetilde{b_Q} \sharp P \qquad \widetilde{b_Q} \sharp Q \qquad \widetilde{b_Q} \sharp K \qquad \tilde{x} \sharp Q\end{pmatrix}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\tilde{x})(P' \mid Q')}\ \textsc{Comm2}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x}\tilde{y})N} P' \\ x \in \mathit{supp}\ N \qquad x \sharp \Psi \\ x \sharp M \qquad x \sharp \tilde{x} \qquad x \sharp \tilde{y}\end{pmatrix}}{\Psi \rhd (\nu x)P \xrightarrow{\overline{M}(\nu\tilde{x}x\tilde{y})N} P'}\ \textsc{Open} \qquad \frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' \\ x \sharp \Psi \qquad x \sharp \alpha\end{pmatrix}}{\Psi \rhd (\nu x)P \xrightarrow{\alpha} (\nu x)P'}\ \textsc{Scope}$$

$$\frac{\left(\Psi \rhd P \mid !P \longmapsto V \qquad \mathit{guarded}\ P\right)}{\Psi \rhd !P \longmapsto V}\ \textsc{Repl}$$

Figure 25.4: Derived operational semantics for psi-calculi.

Figure 25.5: Derived induction rule for the psi-calculi operational semantics for transitions of the form $\Psi \rhd R \longmapsto W$, where R has the frame $(\nu \widetilde{b_R})\Psi_R$. The extra ALPHA case ensures that the frame of R can be alpha-converted in the predicate *Prop*. The inductive cases share the name of the semantic rule which they are derived from. For space reasons, all meta quantifiers for each inductive step have been removed – every term of every case is locally quantified.

$$\left[\begin{array}{l} \Psi \rhd R \longmapsto W \qquad \mathcal{F}\ R = (\nu \widetilde{b_R})\Psi_R \qquad \mathit{distinct}\ \widetilde{b_R} \\ \\ \dfrac{\begin{pmatrix} \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp p \cdot \widetilde{b_P} \qquad \widetilde{b_P} \sharp V \qquad \widetilde{b_P} \sharp \mathscr{C} \\ \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{b_P} \times \mathit{set}\ (p \cdot \widetilde{b_P}) \qquad \mathit{distinctPerm}\ p \\ \mathit{Prop}\ \mathscr{C}\ \Psi\ P\ V\ \widetilde{b_P}\ \Psi_P \end{pmatrix}}{\mathit{Prop}\ \mathscr{C}\ \Psi\ P\ V\ (p \cdot \widetilde{b_P})\ (p \cdot \Psi_P)}\ \textsc{Alpha} \\ \\ \dfrac{\begin{pmatrix} \Psi \vdash M \leftrightarrows K \qquad \mathit{distinct}\ \widetilde{x} \qquad \mathit{set}\ \widetilde{x} \subseteq \mathit{supp}\ N \\ \widetilde{x} \sharp \widetilde{T} \qquad |\widetilde{x}| = |\widetilde{T}| \qquad \widetilde{x} \sharp \Psi \qquad \widetilde{x} \sharp M \qquad \widetilde{x} \sharp K \qquad \widetilde{x} \sharp \mathscr{C} \end{pmatrix}}{\mathit{Prop}\ \mathscr{C}\ \Psi\ \underline{M}(\lambda \widetilde{x})N.P\ (\underline{K}N[\widetilde{x} := \widetilde{T}] \prec P[\widetilde{x} := \widetilde{T}])\ \varepsilon\ (\mathbf{1})}\ \textsc{Input} \\ \\ \dfrac{\Psi \vdash M \leftrightarrows K}{\mathit{Prop}\ \mathscr{C}\ \Psi\ (\overline{M}N.P)\ (\overline{K}N \prec P)\ \varepsilon\ (\mathbf{1})}\ \textsc{Output} \\ \\ \dfrac{\begin{pmatrix} \Psi \rhd P \longmapsto V \qquad \bigwedge \mathscr{C}.\ \mathit{Prop}\ \mathscr{C}\ \Psi\ P\ V\ \widetilde{b_P}\ \Psi_P \\ (\varphi, P)\ \mathit{mem}\ \widetilde{C} \qquad \Psi \vdash \varphi \qquad \mathit{guarded}\ P \\ \mathcal{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad \mathit{distinct}\ \widetilde{b_P} \qquad \Psi_P \simeq \mathbf{1} \qquad \mathit{supp}\ \Psi_P = \emptyset \\ \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp V \qquad \widetilde{b_P} \sharp \mathscr{C} \end{pmatrix}}{\mathit{Prop}\ \mathscr{C}\ \Psi\ (\mathit{Cases}\ \widetilde{C})\ V\ \varepsilon\ (\mathbf{1})}\ \textsc{Case} \\ \\ \dfrac{\begin{pmatrix} \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \qquad \bigwedge \mathscr{C}.\ \mathit{Prop}\ \mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ (\alpha \prec P')\ \widetilde{b_P}\ \Psi_P \\ \\ \mathcal{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad \mathit{distinct}\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \widetilde{b_Q} \qquad \widetilde{b_P} \sharp \Psi_Q \\ \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp Q \qquad \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp \alpha \qquad \widetilde{b_P} \sharp P' \qquad \widetilde{b_P} \sharp \mathscr{C} \\ \\ \mathcal{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad \mathit{distinct}\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp P \qquad \widetilde{b_Q} \sharp Q \\ \widetilde{b_Q} \sharp \Psi \qquad \widetilde{b_Q} \sharp \alpha \qquad \widetilde{b_Q} \sharp P' \qquad \widetilde{b_Q} \sharp \Psi_P \qquad \widetilde{b_Q} \sharp \mathscr{C} \\ \\ \mathit{distinct}\ (\mathit{bn}\ \alpha) \qquad \mathit{bn}\ \alpha \sharp \mathit{subject}\ \alpha \qquad \mathit{bn}\ \alpha \sharp \Psi \\ \mathit{bn}\ \alpha \sharp \Psi_P \qquad \mathit{bn}\ \alpha \sharp \Psi_Q \qquad \mathit{bn}\ \alpha \sharp P \qquad \mathit{bn}\ \alpha \sharp Q \end{pmatrix}}{\mathit{Prop}\ \mathscr{C}\ \Psi\ (P \mid Q)\ (\alpha \prec P' \mid Q)\ (\widetilde{b_P}\widetilde{b_Q})\ (\Psi_P \otimes \Psi_Q)}\ \textsc{Par1} \end{array}\right]$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' \\ \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ (\alpha \prec P')\ \widetilde{b_Q}\ \Psi_Q \\ \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \quad distinct\ \widetilde{b_P} \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \Psi_Q \\ \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \alpha \quad \widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp \mathscr{C} \\ \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \quad distinct\ \widetilde{b_Q} \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp Q \\ \widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \alpha \quad \widetilde{b_Q} \sharp P' \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp \mathscr{C} \\ distinct\ (bn\ \alpha) \quad bn\ \alpha \sharp subject\ \alpha \quad bn\ \alpha \sharp \Psi \\ bn\ \alpha \sharp \Psi_P \quad bn\ \alpha \sharp \Psi_Q \quad bn\ \alpha \sharp P \quad bn\ \alpha \sharp Q \end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid Q)\ (\alpha \prec P \mid Q')\ (\widetilde{b_P}\widetilde{b_Q})\ (\Psi_P \otimes \Psi_Q)}\ \textsc{Par2}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' \quad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ P\ (\alpha \prec P') \\ bn\ \alpha \sharp subject\ \alpha \quad distinct\ (bn\ \alpha) \\ x \sharp \Psi \quad x \sharp \alpha \quad x \sharp \mathscr{C} \\ bn\ \alpha \sharp \Psi \quad bn\ \alpha \sharp P \quad bn\ \alpha \sharp \mathscr{C} \end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ ((\nu x)P)\ (\alpha \prec (\nu x)P')}\ \textsc{Scope}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\overline{M}\,(\nu \tilde{y}\tilde{z})N} P' \quad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ P\ (\overline{M}\,(\nu \tilde{x}\tilde{y})N \prec P') \\ x \in supp\,N \quad x \sharp \tilde{y} \quad x \sharp \tilde{z} \quad x \sharp M \quad x \sharp \Psi \quad x \sharp \mathscr{C} \\ distinct\ \tilde{y} \quad \tilde{y} \sharp \Psi \quad \tilde{y} \sharp P \quad \tilde{y} \sharp M \quad \tilde{y} \sharp \tilde{z} \quad \tilde{y} \sharp \mathscr{C} \\ distinct\ \tilde{z} \quad \tilde{z} \sharp \Psi \quad \tilde{z} \sharp P \quad \tilde{z} \sharp M \quad \tilde{z} \sharp \mathscr{C} \end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ ((\nu x)P)\ (\overline{M}\,(\nu \tilde{y}x\tilde{z})N \prec P')}\ \textsc{Open}$$

$$\frac{\begin{pmatrix}\Psi \rhd P \mid !P \longmapsto V \quad \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ V\ \widetilde{b_P}\ \Psi_P \\ guarded\ P \quad \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \quad distinct\ \widetilde{b_P} \quad \Psi_P \simeq \mathbf{1} \\ supp\,\Psi_P = \emptyset \quad \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp V \quad \widetilde{b_P} \sharp \mathscr{C} \end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (!P)\ V\ \varepsilon\ (\mathbf{1})}\ \textsc{Repl}$$

$$\left[\frac{\begin{pmatrix}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\underline{MN}} P' \\
\bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ (\underline{M}N \prec P')\ \widetilde{b_P}\ \Psi_P \\
\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu \widetilde{x})N} Q' \\
\bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ (\overline{K}(\nu \widetilde{x})N \prec Q')\ \widetilde{b_Q}\ \Psi_Q \\
\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad distinct\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \\
\widetilde{b_Q} \sharp P' \quad \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\
distinct\ \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \\
\widetilde{x} \sharp \Psi_Q \quad \widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \quad \widetilde{x} \sharp \mathscr{C}
\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid Q)\ (\tau \prec (\nu \widetilde{x})(P' \mid Q'))\ (\widetilde{b_P}\widetilde{b_Q})\ (\Psi_P \otimes \Psi_Q)}\ \textsc{Comm1}$$

$$\left[\frac{\begin{pmatrix}
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P' \\
\bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_Q)\ P\ (\overline{M}(\nu \widetilde{x})N \prec P')\ \widetilde{b_P}\ \Psi_P \\
\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \qquad \widetilde{b_P} \sharp \mathscr{C} \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{\underline{K}N} Q' \\
\bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ (\Psi \otimes \Psi_P)\ Q\ (\underline{K}N \prec Q')\ \widetilde{b_P}\ \Psi_P \\
\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad distinct\ \widetilde{b_Q} \qquad \widetilde{b_Q} \sharp \mathscr{C} \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \\
\widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\
\widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \\
\widetilde{b_Q} \sharp P' \quad \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \\
distinct\ \widetilde{x} \quad \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \\
\widetilde{x} \sharp \Psi_Q \quad \widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \quad \widetilde{x} \sharp \mathscr{C}
\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid Q)\ (\tau \prec (\nu \widetilde{x})(P' \mid Q'))\ (\widetilde{b_P}\widetilde{b_Q})\ (\Psi_P \otimes \Psi_Q)}\ \textsc{Comm2}\right]$$

$$\overline{Prop\,\mathscr{C}\ \Psi\ R\ W\ \widetilde{b_R}\ \Psi_R}$$

Lemma 25.27.

$$\frac{\begin{array}{llllll}\Psi \rhd P \xrightarrow{\alpha} P' & & & & & \\ \mathscr{F}\, P = (\nu \widetilde{b_P})\Psi_P & \mathit{distinct}\, \widetilde{b_P} & \mathit{bn}\, \alpha \sharp \mathit{subject}\, \alpha & \mathit{distinct}\, (\mathit{bn}\, \alpha) & & \\ \alpha \neq \tau & \widetilde{x} \sharp P & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp \widetilde{x} & \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp \mathit{subject}\, \alpha \end{array}}{\exists M.\, \Psi \otimes \Psi_P \vdash \mathit{the}\, (\mathit{subject}\, \alpha) \leftrightarrow M \wedge \widetilde{x} \sharp M}$$

Proof. By induction on $\Psi \rhd P \xrightarrow{\alpha} P'$. Intuitively, the lemma extracts the subject of P. The *the*-function retrieves the value of an option type, which is always possible since $\alpha \neq \tau$ □

This lemma obtains the subject from the prefix of P. Moreover, it ensures that any sequence of names $\widetilde{x}$ that are fresh for P and the bound names $\widetilde{b_P}$ from the frame of Ps are also fresh for the obtained subject.

The following lemmas can then be used to replace the subject of an action.

Lemma 25.28. *Replacing the subject of an input action.*

$$\frac{\begin{array}{llll}\Psi \rhd P \xrightarrow{MN} P' & \mathscr{F}\, P = (\nu \widetilde{b_P})\Psi_P & \mathit{distinct}\, \widetilde{b_P} & \\ \Psi \otimes \Psi_P \vdash M \leftrightarrow K & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp M \qquad \widetilde{b_P} \sharp K \end{array}}{\Psi \rhd P \xrightarrow{KN} P'}$$

Proof. By induction on $\Psi \rhd P \xrightarrow{MN} P'$ □

Lemma 25.29. *Replacing the subject of an output action.*

$$\frac{\begin{array}{llll}\Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P' & \mathscr{F}\, P = (\nu \widetilde{b_P})\Psi_P & \mathit{distinct}\, \widetilde{b_P} & \\ \Psi \otimes \Psi_P \vdash M \leftrightarrow K & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp M \qquad \widetilde{b_P} \sharp K \end{array}}{\Psi \rhd P \xrightarrow{\overline{K}(\nu \widetilde{x})N} P'}$$

Proof. By induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P'$ □

25.4 Replication

Similarly to CCS and the pi-calculus we require an induction rule to reason about the transitions of replicated agents. This is required as instances of Replication occur in the assumptions of the REPL rule. The induction rule can be simplified considerably compared to the general induction rule from Figure 25.2 as all replicated agents have guarded frames; the frames of the

rule can thus be removed altogether as they are empty by definition. The induction rule for replicated agents can be found in Figure 25.6.

$$\left[\begin{array}{l}
\Psi \rhd !P \longmapsto W \\[2ex]
\bigwedge \alpha\ \mathrm{P'}\ \mathscr{C}.\ \dfrac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' & distinct\,(bn\,\alpha) & bn\,\alpha \sharp \Psi \\ bn\,\alpha \sharp subject\,\alpha & bn\,\alpha \sharp P & bn\,\alpha \sharp \mathscr{C}\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ (\alpha \prec P' \mid !P)}\ \textsc{Par1} \\[2ex]
\bigwedge \alpha\ \mathrm{P'}\ \mathscr{C}.\ \dfrac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\alpha} P' & \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (!P)\ (\alpha \prec P') \\ distinct\,(bn\,\alpha) & bn\,\alpha \sharp subject\,\alpha \\ bn\,\alpha \sharp \Psi & bn\,\alpha \sharp P \quad bn\,\alpha \sharp \mathscr{C}\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ (\alpha \prec P \mid P')}\ \textsc{Par2} \\[2ex]
\bigwedge \mathrm{M}\ \tilde{x}\ \mathrm{N\ P'\ K\ P''}\ \mathscr{C}.\ \dfrac{\begin{pmatrix}\Psi \rhd P \xrightarrow{MN} P' & \Psi \rhd !P \xrightarrow{\overline{K}(\nu\tilde{x})N} P'' \\ \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (!P)\ (\overline{K}(\nu\tilde{x})N \prec P') \\ \Psi \vdash M \leftrightarrow K & \tilde{x} \sharp \Psi \quad \tilde{x} \sharp P \\ \tilde{x} \sharp M \quad \tilde{x} \sharp K & \tilde{x} \sharp \mathscr{C} \quad distinct\,\tilde{x}\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ (\tau \prec (\nu\tilde{x})(P \mid P'))}\ \textsc{Comm1} \\[2ex]
\bigwedge \mathrm{M}\ \tilde{x}\ \mathrm{N\ P'\ K\ P''}\ \mathscr{C}.\ \dfrac{\begin{pmatrix}\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P' & \Psi \rhd !P \xrightarrow{KN} P'' \\ \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (!P)\ (\underline{K}N \prec P') \\ \Psi \vdash M \leftrightarrow K & \tilde{x} \sharp \Psi \quad \tilde{x} \sharp P \\ \tilde{x} \sharp M \quad \tilde{x} \sharp K & \tilde{x} \sharp \mathscr{C} \quad distinct\,\tilde{x}\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ (\tau \prec (\nu\tilde{x})(P \mid P'))}\ \textsc{Comm2} \\[2ex]
\bigwedge \mathrm{V}\ \mathscr{C}.\ \dfrac{\begin{pmatrix}\Psi \rhd P \mid !P \longmapsto V & \bigwedge \mathscr{C}.\ Prop\,\mathscr{C}\ \Psi\ (P \mid !P)\ V \\ guarded\,P\end{pmatrix}}{Prop\,\mathscr{C}\ \Psi\ (!P)\ V}\ \textsc{Repl}
\end{array}\right]$$
$$\overline{Prop\,\mathscr{C}\ \Psi\ (!P)\ W}$$

Figure 25.6: Induction rule for transitions of the form $\Psi \rhd !P \longmapsto V$. Since all replicated agents are guarded, their frames are empty and can be removed from the cases.

26. Inversion rules

For calculi with single binders, like the pi-calculus and CCS, Isabelle automatically creates induction and inversion rules from the operational semantics. For calculi with sequences of binders, like psi-calculi, Isabelle only manages to create induction rules, and it has been an open problem how to create inversion rules for calculi with sequences of binders. In this chapter we propose a candidate technique to generate inversion rules for these types of formalisations. The proofs are currently manual, but the techniques are general enough to allow for automatisation, and have been used successfully to generate inversion rules for psi-calculi, and also by Berghofer in a formalisation of the simply type lambda-calculus extended with let patterns for tuples [22].

26.1 Rule generation

Examples of inversion rules for calculi with single binders can be found in Figure 6.4 for CCS, and in Figure 13.4 for the pi-calculus. The main requisite of these rules is that no bound names are locally quantified for each semantic case – they are all globally quantified and must hence be initialised before the inversion rule is instantiated.

When inversion is done on a transition, it is necessary that the bound names of this transition are not swapped – if we do inversion over the transition

$$\Psi \rhd (\nu x)P \xrightarrow{\alpha} P'$$

then the name x is already fixed in the proof environment and may not be freely alpha-converted by the inversion rule. The way this is achieved is that the standard inversion rule generated by Isabelle is used, which locally quantify the binders, and that binder is then alpha-converted back to the original one, which is possible as long as the original binder is sufficiently fresh. For an overview, see [23].

The general inversion rule that we are aiming for does inversion on transitions of the form

$$\Psi \rhd P \longmapsto V$$

where binding sequences can occur either in input prefixes, or in bound outputs. We will use the same technique as is done for single binders – Isabelle's non nominal inversion rule is used, and the binding sequences are then alpha-converted back to the ones provided by the user. Whereas the only requisite of the bound names for the single binder case is that they are sufficiently fresh, binding sequences are also required to be distinct, and have the same length as the binding sequences in the transition being inverted.

The last requisite deserves special mention. Consider doing inversion on the transition above, and requiring that the bound names of the action of V are equal to $\tilde{x}$. This is only possible if $\tilde{x}$ is sufficiently fresh and has the same length as the binding sequence in V.

The result of a nominal function must not depend on the values of the binders of its arguments, but it is possible to compute the length of binding sequences using nominal functions.

Definition 26.1. *The function inputLength counts the number of binders in an input prefix. It has the type* (α,β,γ) *psi* $\Rightarrow \mathcal{N}$

$$\begin{aligned}
&inputLength\ (\mathbf{0}) = inputLength\ (\overline{M}N\,.\,P) = inputLength\ (Case\ C)\\
&\qquad\qquad = inputLength\ (P \mid Q) \quad = inputLength\ ((\nu x)P)\\
&\qquad\qquad = inputLength\ (\llparenthesis \Psi \rrparenthesis) \quad = inputLength\ (!P) = 0\\
&inputLength\ (Input\ M\ I) = inputLength'\ I
\end{aligned}$$

$$\begin{aligned}
&inputLength'\ (Trm\ M\ P) = 0\\
&inputLength'\ (Bind\ y\ I) \ = 1 + inputLength'\ I
\end{aligned}$$

Definition 26.2. *The function residualLength counts the number of binders in the action of a residual. It has the type* (α,β,γ) *residual* $\Rightarrow \mathcal{N}$ *and uses the auxiliary function boundOutputLength which has the type* (α,β,γ) *boundOutput* $\Rightarrow \mathcal{N}$

$$\begin{aligned}
&boundOutputLength\ (BOut\ M\ P) = 0\\
&boundOutputLength\ (BStep\ x\ B) \ = boundOutputLength\ B + 1
\end{aligned}$$

$$\begin{aligned}
&residualLength\ (RIn\ M\ N\ P) = 0\\
&residualLength\ (ROut\ M\ B) \ = boundOutputLength\ B\\
&residualLength\ (RTau\ P) \quad = 0
\end{aligned}$$

These definitions allow us to check the length of the binding sequences of agents and residuals even though they have not yet been fixed in the proof environment. We will use the notation $|P|_b$ and $|V|_b$ for *inputLength P* and

residualLength V respectively. The following lemma can then be used to generate the alpha-converting permutations.

Lemma 26.3.

$$\frac{|\tilde{x}| = |\tilde{y}| \qquad \tilde{x} \sharp\sharp \tilde{y} \qquad \mathit{distinct}\,\tilde{x} \qquad \mathit{distinct}\,\tilde{y}}{\exists\, p.\ \mathit{set}\, p \subseteq \mathit{set}\,\tilde{x} \times \mathit{set}\,(p \cdot \tilde{x}) \wedge \mathit{distinctPerm}\, p \wedge \tilde{y} = p \cdot \tilde{x}}$$

Proof. By induction on the length of $\tilde{x}$ and $\tilde{y}$. The perumation p is constructed by pairwise composing the elements from the sequences $\tilde{x}$ and $\tilde{y}$. □

The permutations can then be used to alpha-convert the agents and residuals provided by Isabelle such that their binding sequences are the ones provided by the user. For input prefixes and bound output actions, Corollary 23.6 is used, and for residuals of the form $\alpha \prec P'$, Lemma 25.8.

The generated inversion rule for psi-calculi can be found in Figure 26.1. As for its counterparts from CCS and the pi-calculus, the bound names are globally quantified and supplied by the user, and not locally quantified for each case. The exception to this are the PAR and the SCOPE rules; these rules have residuals of the form $\alpha \prec P'$, and even though α contains bound names, it is locally quantified by the corresponding rules. However, the user still provides the bound names of the actions, as seen by the constraints $bn\ \alpha = \tilde{x}$ in these cases.

To summarise the requirements we have for each nominal datatype in order to derive an inversion rule are:

- Any sequence of bound names in a binding structure available for inversion must be obtainable by a *bn* function.
- There must be a length function to determine the length of the binding sequence of the binding structures.
- There must be an alpha-conversion lemma like Lemma 25.8 for every binding structure.

None of these require that what we are doing inversion over have to be of a particular form. We must be able to obtain an alpha-converting permutation to equate the nominal datatypes, but that is all. We believe neither of these requirements to be difficult to automate.

Similarly to the previous inversion rules, the rule in Figure 26.1 is not well suited for inversion in most proofs – every bound sequence must be instantiated, their lengths must be verified and they must be sufficiently fresh. This process is tedious and should preferably not be required every time inversion is used. Inversion rules specifically tailored for use with the different operators can be found in Figure 26.2. The proofs for all of these inversion rules are very short, and follow directly from the general inversion rule. For the rest of the thesis, whenever inversion is used, the rules in Figure 26.2 are the ones used.

$$\boxed{\Psi' \rhd \mathbf{R} \longmapsto \mathbf{V}}$$

$$\frac{\left(\begin{array}{l}\widetilde{\mathbf{x}_1} \sharp \Psi' \quad \widetilde{\mathbf{x}_1} \sharp \mathbf{R} \quad \widetilde{\mathbf{x}_1} \sharp \mathbf{V} \quad |\widetilde{\mathbf{x}_1}| = |\mathbf{R}|_b \quad \mathit{distinct}\,\widetilde{\mathbf{x}_1} \\ \hline \Psi' = \Psi \wedge \mathbf{R} = \underline{M}(\lambda\widetilde{\mathbf{x}_1})N.P \wedge \mathbf{V} = \underline{K}N[\widetilde{\mathbf{x}_1} := \widetilde{T}] \prec P[\widetilde{\mathbf{x}_1} := \widetilde{T}] \wedge \\ \Psi \vdash M \leftrightarrow K \wedge \mathit{set}\,\widetilde{\mathbf{x}_1} \subseteq \mathit{supp}\,N \wedge |\widetilde{\mathbf{x}_1}| = |\widetilde{T}| \wedge \\ \widetilde{\mathbf{x}_1} \sharp \widetilde{T} \wedge \widetilde{\mathbf{x}_1} \sharp \Psi \wedge \widetilde{\mathbf{x}_1} \sharp M \wedge \widetilde{\mathbf{x}_1} \sharp K\end{array}\right)}{\mathbf{Prop}}\ \textsc{Input}$$

$$\frac{\Psi' = \Psi \quad \mathbf{R} = \overline{M}N.P \quad \mathbf{V} = \overline{K}N \prec P \quad \Psi \vdash M \leftrightarrow K}{\mathbf{Prop}}\ \textsc{Output}$$

$$\frac{\left(\begin{array}{l}\Psi' = \Psi \quad \mathbf{R} = \mathit{Cases}\,Cs \quad \mathbf{V} = U \\ \Psi \rhd P \longmapsto U \quad (\varphi, P)\ \mathit{mem}\ Cs \quad \Psi \vdash \varphi \quad \mathit{guarded}\,P\end{array}\right)}{\mathbf{Prop}}\ \textsc{Case}$$

$$\frac{\left(\begin{array}{l}\widetilde{\mathbf{x}_2} \sharp \Psi' \quad \widetilde{\mathbf{x}_2} \sharp \mathbf{R} \quad \widetilde{\mathbf{x}_2} \sharp \mathbf{V} \quad |\widetilde{\mathbf{x}_2}| = |\mathbf{V}|_b \quad \mathit{distinct}\,\widetilde{\mathbf{x}_2} \\ \hline \Psi' = \Psi \wedge \mathbf{R} = P \mid Q \wedge \mathbf{V} = \alpha \prec P' \mid Q \wedge \widetilde{\mathbf{x}_2} = \mathit{bn}\,\alpha \wedge \\ \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \wedge \mathscr{F}\,Q = (\nu\widetilde{b_Q})\Psi_Q \wedge \mathit{distinct}\,\widetilde{b_Q} \wedge \\ \widetilde{b_Q} \sharp P \wedge \widetilde{b_Q} \sharp Q \wedge \widetilde{b_Q} \sharp \Psi \wedge \widetilde{b_Q} \sharp \alpha \wedge \widetilde{b_Q} \sharp P' \wedge \widetilde{b_Q} \sharp \mathscr{C}\end{array}\right)}{\mathbf{Prop}}\ \textsc{Par1}$$

$$\frac{\left(\begin{array}{l}\widetilde{\mathbf{x}_3} \sharp \Psi' \quad \widetilde{\mathbf{x}_3} \sharp \mathbf{R} \quad \widetilde{\mathbf{x}_3} \sharp \mathbf{V} \quad |\widetilde{\mathbf{x}_3}| = |\mathbf{V}|_b \quad \mathit{distinct}\,\widetilde{\mathbf{x}_3} \\ \hline \Psi' = \Psi \wedge \mathbf{R} = P \mid Q \wedge \mathbf{V} = \alpha \prec P \mid Q' \wedge \widetilde{\mathbf{x}_3} = \mathit{bn}\,\alpha \wedge \\ \Psi \otimes \Psi_P \rhd Q \xrightarrow{\alpha} Q' \wedge \mathscr{F}\,P = (\nu\widetilde{b_P})\Psi_P \wedge \mathit{distinct}\,\widetilde{b_P} \wedge \\ \widetilde{b_P} \sharp P \wedge \widetilde{b_P} \sharp Q \wedge \widetilde{b_P} \sharp \Psi \wedge \widetilde{b_P} \sharp \alpha \wedge \widetilde{b_P} \sharp P' \wedge \widetilde{b_P} \sharp \mathscr{C}\end{array}\right)}{\mathbf{Prop}}\ \textsc{Par2}$$

$$\frac{\left(\begin{array}{l}\Psi' = \Psi \quad \mathbf{R} = P \mid Q \quad \mathbf{V} = \tau \prec (\nu\widetilde{x})(P' \mid Q') \\ \Psi \otimes \Psi_Q \rhd P \xrightarrow{\underline{M}N} P' \quad \mathscr{F}\,P = (\nu\widetilde{b_P})\Psi_P \quad \mathit{distinct}\,\widetilde{b_P} \\ \Psi \otimes \Psi_P \rhd Q \xrightarrow{\overline{K}(\nu\widetilde{x})N} Q' \quad \mathscr{F}\,Q = (\nu\widetilde{b_Q})\Psi_Q \quad \mathit{distinct}\,\widetilde{b_Q} \\ \Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K \quad \widetilde{b_P} \sharp \mathscr{C} \quad \widetilde{b_Q} \sharp \mathscr{C} \\ \\ \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp \Psi_Q \quad \widetilde{b_P} \sharp P \quad \widetilde{b_P} \sharp M \quad \widetilde{b_P} \sharp N \\ \widetilde{b_P} \sharp P' \quad \widetilde{b_P} \sharp Q \quad \widetilde{b_P} \sharp Q' \quad \widetilde{b_P} \sharp \widetilde{b_Q} \quad \widetilde{b_P} \sharp \widetilde{x} \\ \\ \widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp \Psi_P \quad \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp N \quad \widetilde{b_Q} \sharp P' \\ \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp K \quad \widetilde{b_Q} \sharp Q' \quad \widetilde{b_Q} \sharp \widetilde{x} \quad \mathit{distinct}\,\widetilde{x} \\ \\ \widetilde{x} \sharp \Psi \quad \widetilde{x} \sharp \Psi_P \\ \widetilde{x} \sharp \Psi_Q \quad \widetilde{x} \sharp P \quad \widetilde{x} \sharp M \quad \widetilde{x} \sharp Q \quad \widetilde{x} \sharp K \quad \widetilde{x} \sharp \mathscr{C}\end{array}\right)}{\mathbf{Prop}}\ \textsc{Comm1}$$

$$\left[\begin{array}{l}
\left(\begin{array}{lllll}
\mathbf{\Psi}' = \Psi & \mathbf{R} = P \mid Q & \mathbf{V} = \tau \prec (\nu\tilde{x})(P' \mid Q') & & \\
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\overline{M}(\nu\tilde{x})N} P' & \mathscr{F}\ P = (\nu\widetilde{b_P})\Psi_P & \mathit{distinct}\ \widetilde{b_P} & & \\
\Psi \otimes \Psi_P \rhd Q \xrightarrow{KN} Q' & \mathscr{F}\ Q = (\nu\widetilde{b_Q})\Psi_Q & \mathit{distinct}\ \widetilde{b_Q} & & \\
\Psi \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K & \widetilde{b_P} \sharp \mathscr{C} & \widetilde{b_Q} \sharp \mathscr{C} & & \\
\widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp \Psi_Q & \widetilde{b_P} \sharp P & \widetilde{b_P} \sharp M & \widetilde{b_P} \sharp N \\
\widetilde{b_P} \sharp P' & \widetilde{b_P} \sharp Q & \widetilde{b_P} \sharp Q' & \widetilde{b_P} \sharp \widetilde{b_Q} & \widetilde{b_P} \sharp \tilde{x} \\
\widetilde{b_Q} \sharp \Psi & \widetilde{b_Q} \sharp \Psi_P & \widetilde{b_Q} \sharp P & \widetilde{b_Q} \sharp N & \\
\widetilde{b_Q} \sharp P' & \widetilde{b_Q} \sharp Q & \widetilde{b_Q} \sharp K & \widetilde{b_Q} \sharp Q' & \widetilde{b_Q} \sharp \tilde{x} \\
\mathit{distinct}\ \tilde{x} & \tilde{x} \sharp \Psi & \tilde{x} \sharp \Psi_P & & \\
\tilde{x} \sharp \Psi_Q & \tilde{x} \sharp P & \tilde{x} \sharp M & \tilde{x} \sharp Q & \tilde{x} \sharp K \quad \tilde{x} \sharp \mathscr{C}
\end{array}\right) \\
\hline
\mathbf{Prop}\ \mathscr{C}\ \Psi\ (P \mid Q)\ (\tau \prec (\nu\tilde{x})(P' \mid Q'))
\end{array}\right] \text{COMM2}$$

$$\left[\begin{array}{l}
\left(\begin{array}{llll}
\widetilde{\mathbf{x}_4} \sharp \mathbf{\Psi}' & \widetilde{\mathbf{x}_4} \sharp \mathbf{R} & \widetilde{\mathbf{x}_4} \sharp \mathbf{V} & |\widetilde{\mathbf{x}_4}| = |\mathbf{V}|_b \quad \mathit{distinct}\ \widetilde{\mathbf{x}_4} \\
\mathbf{x}_1 \sharp \mathbf{\Psi}' & \mathbf{x}_1 \sharp \mathbf{R} & \mathbf{x}_1 \sharp \mathbf{V} & \mathbf{x}_1 \sharp \widetilde{\mathbf{x}_4}
\end{array}\right) \\
\hline
\begin{array}{l}
\mathbf{\Psi}' = \Psi \wedge \mathbf{R} = (\nu x_1)P \wedge \mathbf{V} = \overline{M}\,(\nu\tilde{x}\mathbf{x}_1\tilde{y})N \prec P' \wedge \widetilde{\mathbf{x}_4} = \tilde{x}y\tilde{y} \wedge \\
\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x}\tilde{y})N} P' \wedge \mathbf{x}_1 \in \mathit{supp}\ N \wedge \mathbf{x}_1 \sharp \tilde{x} \wedge \mathbf{x}_1 \sharp \tilde{y} \wedge \\
\mathit{distinct}\ \tilde{x} \wedge \mathit{distinct}\ \tilde{y} \wedge \tilde{x} \sharp \Psi \wedge \tilde{x} \sharp P \wedge \tilde{x} \sharp M \wedge \tilde{x} \sharp \tilde{y} \wedge \\
\tilde{y} \sharp \Psi \wedge \tilde{y} \sharp P \wedge \tilde{y} \sharp M
\end{array} \\
\hline
\mathbf{Prop}
\end{array}\right] \text{OPEN}$$

$$\left[\begin{array}{l}
\left(\begin{array}{llll}
\widetilde{\mathbf{x}_4} \sharp \mathbf{\Psi}' & \widetilde{\mathbf{x}_5} \sharp \mathbf{R} & \widetilde{\mathbf{x}_5} \sharp \mathbf{V} & |\widetilde{\mathbf{x}_5}| = |\mathbf{V}|_b \quad \mathit{distinct}\ \widetilde{\mathbf{x}_5} \\
\mathbf{x}_2 \sharp \mathbf{\Psi}' & \mathbf{x}_2 \sharp \mathbf{R} & \mathbf{x}_2 \sharp \mathbf{V} & \mathbf{x}_2 \sharp \widetilde{\mathbf{x}_5}
\end{array}\right) \\
\hline
\begin{array}{l}
\mathbf{\Psi}' = \Psi \wedge \mathbf{R} = (\nu x_2)P \wedge \mathbf{V} = \alpha \prec (\nu x_2)P' \wedge \widetilde{\mathbf{x}_5} = bn\,\alpha \wedge \\
\Psi \rhd P \xrightarrow{\alpha} P' \wedge \mathbf{x}_2 \sharp \Psi \wedge \mathbf{x}_2 \sharp \alpha \wedge bn\,\alpha \sharp subject\,\alpha \wedge \\
\mathit{distinct}\ (bn\,\alpha)
\end{array} \\
\hline
\mathbf{Prop}
\end{array}\right] \text{SCOPE}$$

$$\left[\begin{array}{l}
\left(\begin{array}{lll}
\mathbf{\Psi}' = \Psi & \mathbf{R} = !P & \mathbf{V} = W \\
\Psi \rhd P \mid !P \longmapsto W & \mathit{guarded}\ P &
\end{array}\right) \\
\hline
\mathbf{Prop}
\end{array}\right] \text{REPL}$$

Prop

Figure 26.1: The nominal inversion rule for the operational semantics of psi-calculi. The predicate **Prop**, the assertion $\mathbf{\Psi}'$, the agent $\mathbf{R}$, the residual $\mathbf{V}$, the binding sequences $\widetilde{\mathbf{x}_1}$-$\widetilde{\mathbf{x}_5}$, the names $\mathbf{x}_1$ and $\mathbf{x}_2$, and the freshness context $\mathscr{C}$ are quantified globally, and presented in bold font for clarity. All other terms are locally quantified in their respective inversion case.

$$\frac{\left[\begin{array}{l}\Psi \rhd \underline{\mathbf{M}}(\lambda\tilde{\mathbf{x}})\mathbf{N}.\mathbf{P} \xrightarrow{\alpha} \mathbf{Q} \\ \dfrac{\left(\Psi \vdash \mathbf{M} \leftrightarrow K \qquad distinct\,\tilde{\mathbf{x}} \qquad set\,\tilde{\mathbf{x}} \subseteq supp\,\mathbf{N} \qquad |\tilde{\mathbf{x}}| = |\tilde{T}|\right)}{\mathbf{Prop}\ (\underline{K}\mathbf{N}[\tilde{\mathbf{x}} := \tilde{T}])\ (\mathbf{P}[\tilde{\mathbf{x}} := \tilde{T}])}\end{array}\right]}{\mathbf{Prop}\ \alpha\ \mathbf{Q}}\ \textsc{Input}$$

$$\frac{\left[\Psi \rhd \overline{\mathbf{M}}\mathbf{N}.\mathbf{P} \xrightarrow{\alpha} \mathbf{Q} \qquad \dfrac{\Psi \vdash \mathbf{M} \leftrightarrow K}{\mathbf{Prop}\ (\overline{K}\mathbf{N})\ \mathbf{P}}\right]}{\mathbf{Prop}\ \alpha\ \mathbf{Q}}\ \textsc{Output}$$

$$\frac{\left[\begin{array}{l}\Psi \rhd Cases\,\tilde{\mathbf{C}} \longmapsto \mathbf{V} \\ \dfrac{\left(\Psi \rhd P \longmapsto \mathbf{V} \qquad (\varphi, P)\ mem\ Cs \qquad \Psi \vdash \varphi \qquad guarded\,P\right)}{\mathbf{Prop}}\end{array}\right]}{\mathbf{Prop}}\ \textsc{Case}$$

$$\frac{\left[\begin{array}{l}\Psi \rhd (\nu\mathbf{a})\mathbf{P} \xrightarrow{\alpha} \mathbf{Q} \qquad \mathbf{a} \sharp \Psi \qquad \mathbf{a} \sharp \alpha \qquad \mathbf{a} \sharp \mathbf{Q} \\ bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp \mathbf{P} \qquad bn\,\alpha \sharp subject\,\alpha \\ \dfrac{\left(\begin{array}{lllllll}\Psi \rhd \mathbf{P} \xrightarrow{\overline{M}(\nu\tilde{x}\tilde{y})(\mathbf{a}\ b)\cdot N} (\mathbf{a}\ b)\cdot P' & & & & b \in supp\,N & & \\ \mathbf{a} \sharp N & \mathbf{a} \sharp P' & \mathbf{a} \neq b & y \sharp \tilde{x} & y \sharp \tilde{y} & & \\ y \sharp M & distinct\,\tilde{x} & distinct\,\tilde{y} & \tilde{x} \sharp \Psi & b \sharp \Psi & \tilde{y} \sharp \Psi & \\ \tilde{x} \sharp \mathbf{P} & b \sharp \mathbf{P} & \tilde{y} \sharp \mathbf{P} & \tilde{x} \sharp M & b \sharp M & \tilde{y} \sharp M & \tilde{x} \sharp \tilde{y}\end{array}\right)}{\mathbf{Prop}\ (\overline{M}(\nu\tilde{x}y\tilde{y})N)\ P'} \\ \dfrac{\Psi \rhd \mathbf{P} \xrightarrow{\alpha} P'}{\mathbf{Prop}\ \alpha\ ((\nu\mathbf{a})P')}\end{array}\right]}{\mathbf{Prop}\ \alpha\ \mathbf{Q}}\ \textsc{Scope}$$

$$\frac{\left[\Psi \rhd !\mathbf{P} \longmapsto \mathbf{V} \qquad \dfrac{\Psi \rhd \mathbf{P} \mid !\mathbf{P} \longmapsto \mathbf{V} \qquad guarded\,\mathbf{P}}{\mathbf{Prop}}\right]}{\mathbf{Prop}}\ \textsc{Repl}$$

Figure 26.2: Derived inversion rules for the operational semantics of psi-calculi. To save space, the meta quantifiers for each rule have been removed. The arguments marked in bold font are globally quantified for the entire rule, and the italic ones are quantified for their respective cases. The PAR and SCOPE rules require freshness conditions of the bound names of the transitions in order for the inversion rule to be applicable.

$$\left[\begin{array}{l}
\boldsymbol{\Psi} \rhd \mathbf{P} \mid \mathbf{Q} \xrightarrow{\boldsymbol{\alpha}} \mathbf{R} \\
bn\,\boldsymbol{\alpha} \sharp \Psi \quad bn\,\boldsymbol{\alpha} \sharp \mathbf{P} \quad bn\,\boldsymbol{\alpha} \sharp \mathbf{Q} \quad bn\,\boldsymbol{\alpha} \sharp subject\,\boldsymbol{\alpha} \\
\dfrac{\left(\begin{array}{llllll}
\boldsymbol{\Psi} \otimes \Psi_Q \rhd \mathbf{P} \xrightarrow{\boldsymbol{\alpha}} P' & \mathscr{F}\,\mathbf{Q} = (\nu \widetilde{b_Q})\Psi_Q & distinct\,\widetilde{b_Q} & & & \\
\widetilde{b_Q} \sharp \mathbf{P} & \widetilde{b_Q} \sharp \mathbf{Q} & \widetilde{b_Q} \sharp \boldsymbol{\Psi} & \widetilde{b_Q} \sharp \boldsymbol{\alpha} & \widetilde{b_Q} \sharp P' & \widetilde{b_Q} \sharp \mathscr{C}
\end{array}\right)}{\mathbf{Prop}\,\boldsymbol{\alpha}\,(P' \mid \mathbf{Q})} \\
\dfrac{\left(\begin{array}{llllll}
\boldsymbol{\Psi} \otimes \Psi_P \rhd \mathbf{Q} \xrightarrow{\boldsymbol{\alpha}} Q' & \mathscr{F}\,\mathbf{P} = (\nu \widetilde{b_P})\Psi_P & distinct\,\widetilde{b_P} & & & \\
\widetilde{b_P} \sharp \mathbf{P} & \widetilde{b_P} \sharp \mathbf{Q} & \widetilde{b_P} \sharp \boldsymbol{\Psi} & \widetilde{b_P} \sharp \boldsymbol{\alpha} & \widetilde{b_P} \sharp Q' & \widetilde{b_P} \sharp \mathscr{C}
\end{array}\right)}{\mathbf{Prop}\,\boldsymbol{\alpha}\,(\mathbf{P} \mid Q')} \\
\dfrac{\left(\begin{array}{llllll}
\boldsymbol{\Psi} \otimes \Psi_Q \rhd \mathbf{P} \xrightarrow{MN} P' & \mathscr{F}\,\mathbf{P} = (\nu \widetilde{b_P})\Psi_P & distinct\,\widetilde{b_P} & & & \\
\boldsymbol{\Psi} \otimes \Psi_P \rhd \mathbf{Q} \xrightarrow{\overline{K}(\nu \tilde{x})N} Q' & \mathscr{F}\,\mathbf{Q} = (\nu \widetilde{b_Q})\Psi_Q & distinct\,\widetilde{b_Q} & & & \\
\boldsymbol{\Psi} \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K & \widetilde{b_P} \sharp \mathscr{C} & \widetilde{b_Q} \sharp \mathscr{C} & & & \\
\widetilde{b_P} \sharp \boldsymbol{\Psi} & \widetilde{b_P} \sharp \Psi_Q & \widetilde{b_P} \sharp \mathbf{P} & \widetilde{b_P} \sharp M & \widetilde{b_P} \sharp N & \\
\widetilde{b_P} \sharp P' & \widetilde{b_P} \sharp \mathbf{Q} & \widetilde{b_P} \sharp Q' & \widetilde{b_P} \sharp \widetilde{b_Q} & \widetilde{b_P} \sharp \tilde{x} & \\
\widetilde{b_Q} \sharp \boldsymbol{\Psi} & \widetilde{b_Q} \sharp \Psi_P & \widetilde{b_Q} \sharp \mathbf{P} & \widetilde{b_Q} \sharp N & & \\
\widetilde{b_Q} \sharp P' & \widetilde{b_Q} \sharp \mathbf{Q} & \widetilde{b_Q} \sharp K & \widetilde{b_Q} \sharp Q' & \widetilde{b_Q} \sharp \tilde{x} & \\
distinct\,\tilde{x} & \tilde{x} \sharp \boldsymbol{\Psi} & \tilde{x} \sharp \Psi_P & & & \\
\tilde{x} \sharp \Psi_Q & \tilde{x} \sharp \mathbf{P} & \tilde{x} \sharp M & \tilde{x} \sharp \mathbf{Q} & \tilde{x} \sharp K & \tilde{x} \sharp \mathscr{C}
\end{array}\right)}{\mathbf{Prop}\,(\tau)\,((\nu \tilde{x})(P' \mid Q'))} \\
\dfrac{\left(\begin{array}{llllll}
\boldsymbol{\Psi} \otimes \Psi_Q \rhd \mathbf{P} \xrightarrow{\overline{M}(\nu \tilde{x})N} P' & \mathscr{F}\,\mathbf{P} = (\nu \widetilde{b_P})\Psi_P & distinct\,\widetilde{b_P} & & & \\
\boldsymbol{\Psi} \otimes \Psi_P \rhd \mathbf{Q} \xrightarrow{KN} Q' & \mathscr{F}\,\mathbf{Q} = (\nu \widetilde{b_Q})\Psi_Q & distinct\,\widetilde{b_Q} & & & \\
\boldsymbol{\Psi} \otimes (\Psi_P \otimes \Psi_Q) \vdash M \leftrightarrow K & \widetilde{b_P} \sharp \mathscr{C} & \widetilde{b_Q} \sharp \mathscr{C} & & & \\
\widetilde{b_P} \sharp \boldsymbol{\Psi} & \widetilde{b_P} \sharp \Psi_Q & \widetilde{b_P} \sharp \mathbf{P} & \widetilde{b_P} \sharp M & \widetilde{b_P} \sharp N & \\
\widetilde{b_P} \sharp P' & \widetilde{b_P} \sharp \mathbf{Q} & \widetilde{b_P} \sharp Q' & \widetilde{b_P} \sharp \widetilde{b_Q} & \widetilde{b_P} \sharp \tilde{x} & \\
\widetilde{b_Q} \sharp \boldsymbol{\Psi} & \widetilde{b_Q} \sharp \Psi_P & \widetilde{b_Q} \sharp \mathbf{P} & \widetilde{b_Q} \sharp N & & \\
\widetilde{b_Q} \sharp P' & \widetilde{b_Q} \sharp \mathbf{Q} & \widetilde{b_Q} \sharp K & \widetilde{b_Q} \sharp Q' & \widetilde{b_Q} \sharp \tilde{x} & \\
distinct\,\tilde{x} & \tilde{x} \sharp \boldsymbol{\Psi} & \tilde{x} \sharp \Psi_P & & & \\
\tilde{x} \sharp \Psi_Q & \tilde{x} \sharp \mathbf{P} & \tilde{x} \sharp M & \tilde{x} \sharp \mathbf{Q} & \tilde{x} \sharp K & \tilde{x} \sharp \mathscr{C}
\end{array}\right)}{\mathbf{Prop}\,(\tau)\,((\nu \tilde{x})(P' \mid Q'))}
\end{array}\right]$$

PAR

$$\mathbf{Prop}\,\boldsymbol{\alpha}\,\mathbf{R}$$

27. Strong bisimilarity

Strong bisimilarity for psi-calculi differs from the pi-calculus and CCS in three main regards. Firstly, bisimilarity in parametrised by an environment in which the agents operate. Secondly, as in the Applied pi-calculus, the frames of bisimilar agents must be statically equivalent. Finally, if two agents are bisimilar in an environment, they must be bisimilar for all possible extensions of that environment.

Whereas these additions add to the complexity of the calculus, the formalisation techniques used for the pi-calculus and CCS scale remarkably well. Simulations are still defined in the standard way, with the exception that the relations are ternary rather than binary, and bisimilarity is defined coinductively.

The proof techniques deserve special mention. In psi-calculi an agent in a parallel composition may use the frame of the other agent to derive its transitions. For the bisimilarity proofs, this requires that any transition derived using the frame of an agent must also be derivable using the frame of any bisimilar agent. One of our main contributions is how this is formalised in a smooth and transparent way.

It must be stressed that even though the actual proofs are substantially more involved than for the ones covered previously in this thesis, their general structure remains the same – case analysis is done on what actions an agent can do, and the simulating agents uses the rules from the operational semantics to mimic these actions.

27.1 Frame equivalences

We will now formalise the frame equivalences defined in Definition 22.5.

Definition 27.1. *A frame F entails a condition φ, written $F \vdash \varphi$.*

$$F \vdash \varphi = (\exists \widetilde{b_F}\ \Psi_F.\ F = (\nu \widetilde{b_F})\Psi_F \wedge \widetilde{b_F} \sharp \varphi \wedge \Psi_F \vdash \varphi)$$

A frame entails a condition φ if it has an alpha-variant such that its assertion component entails φ, and none of its bound names clash with φ.

Introduction and elimination rules for $\vdash$ can then be derived.

Lemma 27.2. *Introduction and elimination rules for frame entailment.*

$$\frac{F = (\nu\widetilde{b_F})\Psi_F \qquad \widetilde{b_F} \sharp \varphi \qquad \Psi_F \vdash \varphi}{F \vdash \varphi} \vdash\text{-I}$$

$$\frac{F \vdash \varphi \qquad F = (\nu\widetilde{b_F})\Psi_F \qquad \widetilde{b_F} \sharp \varphi}{\Psi_F \vdash \varphi} \vdash\text{-E}$$

Proofs involving frame entailment generally require that a candidate with sufficiently fresh bound names is chosen for each frame present in the context.

Static equivalence for frames is then defined in the same way as for assertions.

Definition 27.3. *A frame F which statically implies G is written $F \leq G$.*

$$F \leq G \stackrel{\text{def}}{=} \forall \varphi.\, F \vdash \varphi \longrightarrow G \vdash \varphi$$

Definition 27.4. *Two statically equivalent frames F and G is written $F \simeq G$.*

$$F \simeq G \stackrel{\text{def}}{=} F \leq G \wedge G \leq F$$

Lemma 27.5. *Static equivalence is an equivalence relation*

Proof. Follows from definitions 27.3 and 27.4. □

Lemma 27.6. *Static equivalence is equivariant*

$$\begin{aligned}
&\textit{If } F \vdash \varphi \textit{ then } (p \cdot F) \vdash (p \cdot \varphi).\\
&\textit{If } F \leq G \textit{ then } (p \cdot F) \leq (p \cdot G).\\
&\textit{If } F \simeq G \textit{ then } (p \cdot F) \simeq (p \cdot G).
\end{aligned}$$

Proof. Follows directly from Definitions 27.1, 27.3, 27.4 and Lemma 24.13. □

Static equivalence for frames is not compositional, i.e. we do not have that if $F \simeq G$ then $F \otimes H \simeq G \otimes H$. The reason for this is that the we do not necessarily have that $\Psi_F \simeq \Psi_G$ as the binders of F and G can mask the behaviour which separates the assertions, and hence the ACOMP rule cannot be used. For a counter-example, see Section 22.2.2.

27.2 Definitions

The definition for simulation for psi-calculi is significantly simpler than the one for the pi-calculus, and more on par with the one for CCS. Since resid-

uals for psi-calculi are defined to include both bound and free actions, the case distinction which is made for the pi-calculus is not necessary. The freshness conditions for the bound names additionally require that they must be fresh for the environment.

Definition 27.7 (simulation)**.** *An agent term P simulating an agent Q preserving $\mathscr{R}$ in the environment Ψ is denoted $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$.*

$$\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall \alpha\, Q'.\, \Psi \rhd Q \xrightarrow{\alpha} Q' \;\wedge\; bn\,\alpha \sharp \Psi \;\wedge\; bn\,\alpha \sharp P \longrightarrow \exists P'.\, \Psi \rhd P \xrightarrow{\alpha} P' \;\wedge\; (\Psi, P', Q') \in \mathscr{R}$$

Simulation is monotonic with respect to the candidate relation.

Lemma 27.8. *If $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $\Psi \rhd P \hookrightarrow_{\mathscr{R}'} Q$.*

Proof. By the definition of $\hookrightarrow$. The requisite that $\mathscr{R} \subseteq \mathscr{R}'$ ensures that any derivatives of P and Q are in $\mathscr{R}'$. □

Bisimilarity can now be defined coinductively in the standard way.

Definition 27.9 (Strong bisimilarity)**.** *Bisimilarity, denoted $\dot{\sim}$, is defined coinductively as the greatest fixpoint satisfying:*

$$\begin{aligned} \Psi \rhd P \dot{\sim} Q \Longrightarrow\ & (\mathscr{F}\, P) \otimes \Psi \simeq (\mathscr{F}\, Q) \otimes \Psi && \textsc{StatEq} \\ \wedge\ & \Psi \rhd P \hookrightarrow_{\dot{\sim}} Q && \textsc{Simulation} \\ \wedge\ & \forall \Psi'.\, \Psi \otimes \Psi' \rhd P \dot{\sim} Q && \textsc{Extension} \\ \wedge\ & \Psi \rhd Q \dot{\sim} P && \textsc{Symmetry} \end{aligned}$$

27.2.1 Primitive inference rules

As for the pi-calculus, simulation for psi-calculi requires freshness conditions of the bound names in the actions of the transitions. An introduction rule is created such that these bound names are guaranteed to be sufficiently fresh.

Lemma 27.10. *Introduction rule for simulation.*

$$\frac{\textit{eqvt}\ \mathscr{R} \qquad \bigwedge \alpha\, Q'.\ \dfrac{\begin{pmatrix} \Psi \rhd Q \xrightarrow{\alpha} Q' \\ bn\,\alpha \sharp P \qquad bn\,\alpha \sharp Q \qquad bn\,\alpha \sharp \Psi \\ \textit{distinct}\,(bn\,\alpha) \qquad bn\,\alpha \sharp \textit{subject}\,\alpha \qquad bn\,\alpha \sharp \mathscr{C} \end{pmatrix}}{\exists P'.\, \Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}}}{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q}\ \hookrightarrow\text{-I}$$

Proof. Follows from the definition of $\leadsto$. The bound names in the actions of the transitions are alpha-converted to avoid Ψ, P, Q, *subject* α and $\mathscr{C}$ and the fact that $\mathscr{R}$ is equivariant allows the alpha-converting permutations to be applied to the derivatives in $\mathscr{R}$. □

The freshness lemmas for psi-calculi transitions differ from the pi-calculus in that even though a name is fresh for an agent, it may very well appear in the subject of a transition. This phenomenon was covered in detail in Section 25.2.2.

Lemma 27.11. *Introduction rule for simulation ensuring freshness conditions on the subject.*

$$\frac{\begin{array}{l}\mathit{eqvt}\ \mathscr{R} \qquad \tilde{x} \sharp \Psi \qquad \tilde{x} \sharp P \qquad \tilde{x} \sharp Q \\ \bigwedge \alpha\ Q'.\ \dfrac{\begin{pmatrix}\Psi \rhd Q \xrightarrow{\alpha} Q' \\ bn\,\alpha \sharp P \qquad bn\,\alpha \sharp Q \qquad bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp \mathit{subject}\,\alpha \\ \mathit{distinct}\,(bn\,\alpha) \qquad bn\,\alpha \sharp \mathscr{C} \qquad \tilde{x} \sharp \alpha \qquad \tilde{x} \sharp Q'\end{pmatrix}}{\exists P'.\ \Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}}\end{array}}{\Psi \rhd P \leadsto_{\mathscr{R}} Q}$$

Proof. The introduction rule $\leadsto$-I is used such that the bound names of the transition avoid $\mathscr{C}$ and $\tilde{x}$. A fresh subject is retrieved using Lemma 25.27 and exchanged in the transition by using Lemma 25.28 if it is an input transition, and Lemma 25.29 if it is an output transition. □

Given that a sequence $\tilde{x}$ is fresh for the originating agents in a simulation and their environment, the subject of any transition can be translated in such a way that the sequence is fresh for that subject too. This introduction rule will be useful for some of the congruence proofs in this and future chapters.

Lemma 27.12. *Elimination rule for simulation*

$$\frac{\Psi \rhd P \leadsto_{\mathscr{R}} Q \qquad \Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp P}{\exists P'.\ \Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}}\ \leadsto\text{-}E$$

Proof. Follows from the definition of $\leadsto$. □

The introduction and elimination rule for bisimilarity follow immediately from its definition.

Lemma 27.13. *Introduction and elimination rules for bisimilarity.*

$$\frac{(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi \quad \Psi \rhd P \rightarrowtail_{\dot\sim} Q \qquad \forall \Psi'.\ \Psi \otimes \Psi' \rhd P \mathrel{\dot\sim} Q \qquad \Psi \rhd Q \mathrel{\dot\sim} P}{\Psi \rhd P \mathrel{\dot\sim} Q}\ \dot\sim\text{-I}$$

$$\frac{\Psi \rhd P \mathrel{\dot\sim} Q}{(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi}\ \dot\sim\text{-E1} \qquad \frac{\Psi \rhd P \mathrel{\dot\sim} Q}{\Psi \rhd P \rightarrowtail_{\dot\sim} Q}\ \dot\sim\text{-E2}$$

$$\frac{\Psi \rhd P \mathrel{\dot\sim} Q}{\Psi \otimes \Psi' \rhd P \mathrel{\dot\sim} Q}\ \dot\sim\text{-E3} \qquad \frac{\Psi \rhd P \mathrel{\dot\sim} Q}{\Psi \rhd Q \mathrel{\dot\sim} P}\ \dot\sim\text{-E4}$$

Proof. Follows from the definition of $\dot\sim$. □

In order to prove that two processes are bisimilar, a symmetric candidate relation $\mathscr{X}$ is chosen such that all agent pairs in $\mathscr{X}$ simulate each other in an environment and that their derivatives are either in $\mathscr{X}$ or bisimilar in that environment. Moreover, for any environment and pair of agents in $\mathscr{X}$, the pair of agents and the environment extended with an arbitrary assertion must also be in $\mathscr{X}$, or bisimilar.

Lemma 27.14. *Coinduction rule for bisimilarity.*

$$\frac{\begin{array}{ll} (\Psi, P, Q) \in \mathscr{X} & \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\mathscr{F}\ R) \otimes \Psi' \simeq (\mathscr{F}\ S) \otimes \Psi'} & \textsc{StatEq} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \rightarrowtail_{\mathscr{X} \cup \dot\sim} S} & \textsc{Simulation} \\ \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{X} \vee \Psi' \otimes \Psi'' \rhd R \mathrel{\dot\sim} S} & \textsc{Extension} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi', S, R) \in \mathscr{X} \vee \Psi' \rhd S \mathrel{\dot\sim} R} & \textsc{Symmetry} \end{array}}{\Psi \rhd P \mathrel{\dot\sim} Q}$$

Proof. Derived from the coinductive rule generated by Isabelle. □

27.2.2 Equivariance

The equivariance proof follows the standard pattern. The first step is to prove that simulation is preserved by permutations.

Lemma 27.15. *Simulation is equivariant*

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad \Psi \rhd P \hookrightarrow_{\mathscr{R}} Q}{p \cdot \Psi \rhd p \cdot P \hookrightarrow_{\mathscr{R}} p \cdot Q}$$

Proof. Follows from the definition of $\hookrightarrow$. The fact that the transition system is equivariant (Lemma 25.10) makes it possible to cancel the permutation p by applying its inverse and perform the simulation. The proof can then be finished by applying the permutation p to the simulating transition, canceling the inverse, and the assumption $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$ ensures that the derivatives are still in $\mathscr{R}$. □

The proof that bisimilarity is equivariant is done in two steps, the first is to prove that bisimilarity is closed by permutations.

Lemma 27.16. *If* $\Psi \rhd P \dot{\sim} Q$ *then* $p \cdot \Psi \rhd p \cdot P \dot{\sim} p \cdot Q$.

Proof. By coinduction setting $\mathscr{X}$ to $\{(p \cdot \Psi, p \cdot P, p \cdot Q) : \Psi \rhd P \dot{\sim} Q\}$. The four cases are then proven separately.

Static equivalence: From $(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi$ we have from Lemma 27.6 that $(\mathscr{F}\ (p \cdot P)) \otimes (p \cdot \Psi) \simeq (\mathscr{F}\ (p \cdot Q)) \otimes (p \cdot \Psi)$

Simulation: The candidate relation $\mathscr{X}$ is equivariant. From $\Psi \rhd P \dot{\sim} Q$ we have by $\dot{\sim}$-E2 that $\Psi \rhd P \hookrightarrow_{\dot{\sim}} Q$, and hence by Lemma 27.15 that $p \cdot \Psi \rhd p \cdot P \hookrightarrow_{\mathscr{X}} p \cdot Q$.

Extension: From $\Psi \rhd P \dot{\sim} Q$ we have by $\dot{\sim}$-E3 that $\Psi \otimes (p^{-} \cdot \Psi') \rhd P \dot{\sim} Q$ for any possible Ψ'. Hence $(p \cdot \Psi \otimes (p^{-} \cdot \Psi'), p \cdot P, p \cdot Q) \in \mathscr{X}$, by the definition of $\mathscr{X}$, and hence $((p \cdot \Psi) \otimes \Psi', p \cdot P, p \cdot Q) \in \mathscr{X}$ for all Ψ'.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric since bisimilarity is symmetric.

□

From this lemma, the proof that bisimilarity is equivariant follows directly.

Lemma 27.17. *eqvt* $\dot{\sim}$

Proof. Follows immediately from the definition of *eqvt* and Lemma 27.16 □

27.2.3 Preserved by static equivalence

The remaining proof for the basic infrastructure of bisimilarity is to prove that if two processes are bisimilar in an environment, they are also bisimilar in a statically equivalent one. We first do the proof for simulation.

Lemma 27.18. *Simulation is preserved by static equivalence*

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad \mathit{eqvt}\ \mathscr{R}' \qquad \Psi \simeq \Psi' \qquad \bigwedge \Psi'' R S \Psi'''.\ \frac{(\Psi'', R, S) \in \mathscr{R} \qquad \Psi'' \simeq \Psi'''}{(\Psi''', R, S) \in \mathscr{R}'}}{\Psi' \rhd P \hookrightarrow_{\mathscr{R}'} Q}$$

Proof. The introduction rule $\hookrightarrow$-I is used such that any bound names avoid Ψ. We must then prove that given $\Psi' \rhd Q \xrightarrow{\alpha} Q'$, $bn\ \alpha \sharp \Psi'$, and $bn\ \alpha \sharp P$ there exists a P' such that $\Psi' \rhd P \xrightarrow{\alpha} P'$ and $(\Psi', P', Q') \in \mathscr{R}'$. Moreover we know from the avoiding context that $bn\ \alpha \sharp \Psi$.

- From $\Psi' \rhd Q \xrightarrow{\alpha} Q'$ and $\Psi \simeq \Psi'$, we have by Lemma 25.11 that $\Psi \rhd Q \xrightarrow{\alpha} Q'$.
- With $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp P$ we obtain a P' such that $\Psi \rhd P \xrightarrow{\alpha} P'$ and $(\Psi, P', Q') \in \mathscr{R}$, by $\hookrightarrow$-E.
- From $\Psi \rhd P \xrightarrow{\alpha} P'$ and $\Psi \simeq \Psi'$, we have by Lemma 25.11 that $\Psi' \rhd P \xrightarrow{\alpha} P'$ proving the transition part of the simulation.
- Finally, from $(\Psi, P', Q') \in \mathscr{R}$ and $\Psi \simeq \Psi'$ we have that $(\Psi', P', Q') \in \mathscr{R}'$ by the environment switching rule in the assumptions.

□

Before proving this property for bisimilarity, an auxiliary lemma is needed. As we discussed in Section 27.1, static equivalence on frames is not preserved by composition. A weaker result is however available – static equivalence of assertions is preserved by frame composition. More precisely, we have the following lemma.

Lemma 27.19. *If* $\Psi \simeq \Psi'$ *then* $F \otimes \Psi \simeq F \otimes \Psi'$.

Proof. An instance $(\nu \widetilde{b_F})\Psi_F$ of $\mathscr{F}$ is chosen such that $\widetilde{b_F} \sharp \Psi$ and $\widetilde{b_F} \sharp \Psi'$. The assertions can then propagate past $\widetilde{b_F}$. The proof is then done by induction on $\widetilde{b_F}$.

Base case ($\widetilde{b_P} = \varepsilon$): Since $\Psi \simeq \Psi'$, we have by ACOMP that $\Psi \otimes \Psi_F \simeq \Psi' \otimes \Psi_F$.

Inductive step ($\widetilde{b_P} = x\widetilde{b_{P'}}$): From the induction hypothesis we get that $((\nu \widetilde{b_F}')\Psi \otimes \Psi_F) \simeq ((\nu \widetilde{b_F}')\Psi' \otimes \Psi_F)$ and by Lemma 27.30 that $((\nu x\widetilde{b_F}')\Psi \otimes \Psi_F) \simeq ((\nu x\widetilde{b_F}')\Psi' \otimes \Psi_F)$.

□

Lemma 27.20. *If* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *and* $\Psi \simeq \Psi'$ *then* $\Psi' \rhd P \mathrel{\dot{\sim}} Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi', P, Q) : \exists \Psi.\ \Psi \rhd P \mathrel{\dot{\sim}} Q \wedge \Psi \simeq \Psi'\}$

Static equivalence: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have by $\mathrel{\dot{\sim}}$-E1 that $(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi$. Since $\Psi \simeq \Psi'$ we have by Lemma 27.19 that $(\mathscr{F}\ P) \otimes \Psi' \simeq (\mathscr{F}\ Q) \otimes \Psi'$.

Simulation: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have by $\mathrel{\dot{\sim}}$-E2 that $\Psi \rhd P \hookrightarrow_{\dot{\sim}} Q$. Moreover, $\mathscr{X}$ is equivariant as both bisimilarity and static equivalence are equivariant, and with $\Psi \simeq \Psi'$ we can prove that $\Psi \rhd P \hookrightarrow_{\mathscr{X}} Q$ using Lemma 27.18.

Extension: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have by $\mathrel{\dot{\sim}}$-E3 that $\Psi \otimes \Psi'' \rhd P \mathrel{\dot{\sim}} Q$ for all possible Ψ''. That $(\Psi' \otimes \Psi'', P, Q) \in \mathscr{X}$ follows directly from $\Psi \simeq \Psi'$ and the ACOMP rule.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as bisimilarity is symmetric.

□

27.3 Bisimulation is an equivalence relation

That bisimulation is an equivalence relation is not more difficult to prove than for CCS or the pi-calculus. Bisimilarity is symmetric, so the remaining proofs are the ones for reflexivity and transitivity. For CCS and the pi-calculus, the reflexivity proof requires that the candidate relation is reflexive but for for psi-calculi, the candidate relation is ternary and not binary, and the requirement is that it is reflexive for all possible environments.

Lemma 27.21. *If* $\{(\Psi, P, P) : \mathit{True}\} \subseteq \mathscr{R}$ *then* $\Psi \rhd P \hookrightarrow_{\mathscr{R}} P$.

Proof. Follows from the definition of $\hookrightarrow$. The requisite that $\{(\Psi, P, P) : \mathit{True}\} \subseteq \mathscr{R}$ ensures that for all environments, any derivatives of P are in the candidate relation. □

The proof that simulation is transitive follows the one from the pi-calculus closely.

Lemma 27.22.

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad \Psi \rhd Q \hookrightarrow_{\mathscr{R}'} R \qquad \mathit{eqvt}\ \mathscr{R}'' \qquad \{(\Psi, P, R) : \exists Q.\ (\Psi, P, Q) \in \mathscr{R} \wedge (\Psi, Q, R) \in \mathscr{R}'\} \subseteq \mathscr{R}''}{\Psi \rhd P \hookrightarrow_{\mathscr{R}''} R}$$

Proof. The introduction rule $\hookrightarrow$-I is used such that any bound names avoid Q. We must then prove that given $\Psi \rhd R \xrightarrow{\alpha} R'$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp P$ there exists a P' such that $\Psi' \rhd P \xrightarrow{\alpha} P'$ and $(\Psi, P', R') \in \mathscr{R}''$. Moreover we know from the avoiding context that $bn\ \alpha \sharp Q$.

- From $\Psi \rhd Q \hookrightarrow_{\mathscr{R}'} R$, $\Psi \rhd R \xrightarrow{\alpha} R'$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp Q$ we obtain a Q' such that $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $(\Psi, Q', R') \in \mathscr{R}'$ from $\hookrightarrow$-E.
- From $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$, $\Psi \rhd Q \xrightarrow{\alpha} Q'$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp P$ we obtain a P' such that $\Psi \rhd P \xrightarrow{\alpha} P'$ and $(\Psi, P', Q') \in \mathscr{R}$ from $\hookrightarrow$-E, solving the transition part of the simulation.
- From $(\Psi, P', Q') \in \mathscr{R}$ and $(\Psi, Q', R') \in \mathscr{R}'$ we have from the assumptions that $(\Psi, P', R') \in \mathscr{R}''$.

□

We can now prove that bisimulation is an equivalence relation.

Lemma 27.23. *Bisimulation is an equivalence relation*

Proof.

Reflexivity: $\Psi \rhd P \dot{\sim} P$

Proof by coinduction and setting $\mathscr{X}$ to $\{(\Psi, P, P) : True\}$.

Static equivalence: Follows directly since static equivalence is reflexive.

Simulation: Follows from Lemma 27.21.

Extension: Follows directly since static equivalence is reflexive.

Symmetry: The candidate relation $\mathscr{X}$ is trivially symmetric.

Symmetry: *If* $\Psi \rhd P \dot{\sim} Q$ *then* $\Psi \rhd Q \dot{\sim} P$.

Follows immediately from the definition of $\dot{\sim}$

Transitivity: *If* $\Psi \rhd P \dot{\sim} Q$ *and* $\Psi \rhd Q \dot{\sim} R$ *then* $\Psi \rhd P \dot{\sim} R$.

Proof by coinduction and setting $\mathscr{X}$ to $\{(\Psi, P, R) : \exists Q.\ \Psi \rhd P \dot{\sim} Q \wedge \Psi \rhd Q \dot{\sim} R\}$.

Static equivalence: Follows directly since static equivalence is transitive.

Simulation: Follows from Lemma 27.22 and the fact that bisimilarity is equivariant.

Extension: Follows directly since static equivalence is transitive.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric since bisimilarity is symmetric.

□

27.4 Preservation properties

The proofs will follow the standard pattern and be divided into simulation and bisimilarity lemmas. The requisites of the candidate relations for simulations follow the corresponding ones from the pi-calculus almost completely, but some of the proofs require more work, especially the ones where Parallel is involved.

27.4.1 Output

The requirements of the candidate relation for Output for psi-calculi are the same as for the pi-calculus – the prefixed agents and their environment must be in the relation.

Lemma 27.24. *Simulation is preserved by Output.*

$$\frac{(\Psi, P, Q) \in \mathscr{R}}{\Psi \rhd \overline{M}N.P \hookrightarrow_{\mathscr{R}} \overline{M}N.Q}$$

Proof. Follows from the definition of $\hookrightarrow$. The OUTPUT inversion rule is used to derive the only possible type of transition from the prefixed agent, and the OUTPUT rule from the operational semantics is used to discharge it. The derivatives can only be P and Q, which are in $\mathscr{R}$. □

Lemma 27.25. *If* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *then* $\Psi \rhd \overline{M}N.P \mathrel{\dot{\sim}} \overline{M}N.Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, \overline{M}N.P, \overline{M}N.Q) : \Psi \rhd P \mathrel{\dot{\sim}} Q\}$.

Static equivalence: Since both frames of $\overline{M}N.P$ and $\overline{M}N.Q$ are empty, we have that $(\mathscr{F}\ (\overline{M}N.P)) \otimes \Psi \simeq (\mathscr{F}\ (\overline{M}N.Q)) \otimes \Psi$ since static equivalence is reflexive.

Simulation: Follows immediately from Lemma 27.24.

Extension: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have by $\mathrel{\dot{\sim}}$-E3 that $\Psi \otimes \Psi' \rhd P \mathrel{\dot{\sim}} Q$ for all Ψ', and hence that $(\Psi \otimes \Psi', \overline{M}N.P, \overline{M}N.Q) \in \mathscr{X}$.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric since bisimilarity is symmetric.

□

27.4.2 Case

The preservation property for case differs from the one presented in our original paper [17]. In section 4.2 of that paper, the preservation property for case is defined as

$$\forall i.\Psi \rhd P_i \mathrel{\dot\sim} Q_i \Longrightarrow \Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot\sim} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$$

where the φ, $\widetilde{P}$, and $\widetilde{Q}$ are equally long sequences indexed by an index i. In this formalisation we prove a slightly stronger result – two agents *Cases* $\widetilde{C_P}$ and *Cases* $\widetilde{C_Q}$ are bisimilar, if for every condition φ and P in $\widetilde{C_P}$, there exists a Q such that φ and Q are in $\widetilde{C_Q}$, P is bisimilar to Q, and vice versa. This preservation property does not require the number of **case**-constructs for each agent to be the same, nor does it require that the bisimilar agents are present at the same index of the corresponding sequences.

Lemma 27.26. *Simulation is preserved by Case.*

$$\frac{\left(\begin{array}{l} \mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge \Psi'\ R\ S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{\Psi' \rhd R \hookrightarrow_{\mathscr{R}} S} \\[2ex] \bigwedge \varphi\ Q.\ \dfrac{(\varphi, Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi, P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge (\Psi, P, Q) \in \mathscr{R}} \end{array}\right)}{\Psi \rhd Cases\ \widetilde{C_P} \hookrightarrow_{\mathscr{R}'} Cases\ \widetilde{C_Q}}$$

Proof. From the definition of $\hookrightarrow$ we have that for all transitions $\Psi \rhd Cases\ \widetilde{C_Q} \xrightarrow{\alpha} Q'$ such that $bn\ \alpha \sharp \Psi$ and $bn\ \alpha \sharp \widetilde{C_P}$.

- From $\Psi \rhd Cases\ \widetilde{C_Q} \xrightarrow{\alpha} Q'$ and the CASE inversion rule we obtain a φ and a Q such that $(\varphi, Q)\ mem\ \widetilde{C_Q}$, $guarded\ Q$, $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\Psi \vdash \varphi$.
- From $(\varphi, Q)\ mem\ \widetilde{C_Q}$ and the assumptions we obtain a P such that $(\varphi, P)\ mem\ \widetilde{C_P}$, $guarded\ P$ and $(\Psi, P, Q) \in \mathscr{R}$.
- From $(\Psi, P, Q) \in \mathscr{R}$ and the assumptions we get that $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$.
- From $bn\ \alpha \sharp \widetilde{C_P}$ and $(\varphi, P)\ mem\ \widetilde{C_P}$ we have that $bn\ \alpha \sharp P$
- From $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$, $\Psi \rhd Q \xrightarrow{\alpha} Q'$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp P$ we obtain a P' such that $\Psi \rhd P \xrightarrow{\alpha} P'$ and $(\Psi, P', Q') \in \mathscr{R}$ by $\hookrightarrow$-E.
- From $\Psi \rhd P \xrightarrow{\alpha} P'$, $(\varphi, P)\ mem\ \widetilde{C_P}$, $guarded\ P$, and $\Psi \vdash \varphi$ we have by the CASE rule that $\Psi \rhd \widetilde{C_P} \xrightarrow{\alpha} P'$, solving the transition part of the simulation.
- Finally from $(\Psi, P', Q') \in \mathscr{R}$ and $\mathscr{R} \subseteq \mathscr{R}'$ we have that $(\Psi, P', Q') \in \mathscr{R}'$.

□

To prove that bisimilarity is preserved by Case, the membership requisite of the previous lemma must be encoded in the candidate relation, making it somewhat intimidating.

Lemma 27.27. *Bisimilarity is preserved by Case.*

$$\frac{\left(\begin{array}{c}\bigwedge \varphi\ P.\ \dfrac{(\varphi, P)\ mem\ \widetilde{C_P}}{\exists Q.\ (\varphi, Q)\ mem\ \widetilde{C_Q} \wedge guarded\ Q \wedge \Psi \rhd P \mathrel{\dot{\sim}} Q} \\ \\ \bigwedge \varphi\ Q.\ \dfrac{(\varphi, Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi, P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge \Psi \rhd P \mathrel{\dot{\sim}} Q}\end{array}\right)}{\Psi \rhd Cases\ \widetilde{C_P} \mathrel{\dot{\sim}} Cases\ \widetilde{C_Q}}$$

Proof. By coinduction with $\mathscr{X}$ set to

$$\begin{array}{l}\{(\Psi, Cases\ \widetilde{C_P}, \\ \quad Cases\ \widetilde{C_Q}) : (\forall \varphi\ P.\ (\varphi, P)\ mem\ \widetilde{C_P} \longrightarrow \\ \qquad\qquad (\exists Q.\ (\varphi, Q)\ mem\ \widetilde{C_Q} \wedge guarded\ Q \wedge \Psi \rhd P \mathrel{\dot{\sim}} Q)) \wedge \\ \qquad (\forall \varphi\ Q.\ (\varphi, Q)\ mem\ \widetilde{C_Q} \longrightarrow \\ \qquad\qquad (\exists P.\ (\varphi, P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge \Psi \rhd P \mathrel{\dot{\sim}} Q))\}\end{array}$$

Static equivalence: Since both frames of $Cases\ \widetilde{C_P}$ and $Cases\ \widetilde{C_Q}$ are empty, we have that $(\mathscr{F}\ (Cases\ \widetilde{C_P})) \otimes \Psi \simeq (\mathscr{F}\ (Cases\ \widetilde{C_Q})) \otimes \Psi$ since static equivalence is reflexive.

Simulation: This case is discharged by Lemma 27.24, where the first assumption is derived from the definition of $\mathscr{X}$, and the second follows directly from $\dot{\sim}$-E2

Extension: Follows directly from the definition of $\mathscr{X}$ and $\dot{\sim}$-E3.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric since bisimilarity is symmetric.

□

27.4.3 Restriction

The lemma that simulation is preserved by restriction looks almost exactly like its pi-calculus counterpart, apart from the extra assumption that the bound name is required to be fresh for the environment. The proof also follows the same general structure, but with one observation – as the SCOPE and OPEN rules require that the bound name is fresh for the subject of transitions. This property is not ensured by the definition of $\hookrightarrow$, but must be inferred through the custom introduction rule derived in Lemma 27.11.

Lemma 27.28. *Simulation is preserved by restriction.*

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad \mathit{eqvt}\ \mathscr{R}' \qquad x \sharp \Psi \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge \Psi' R S y.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad y \sharp \Psi'}{(\Psi', (\nu y)R, (\nu y)S) \in \mathscr{R}'}}{\Psi \rhd (\nu x)P \hookrightarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Since $x \sharp \Psi$, $x \sharp (\nu x)P$, and $x \sharp (\nu x)Q$, Lemma 27.11 can be used to ensure that x is fresh for any action or derivative of $(\nu x)Q$.

The SCOPE inversion rule is then used to generate two possible transitions, which are discharged by the SCOPE and OPEN rules respectively. □

The static equivalence case of the bisimilarity requires that static equivalence is preserved by restriction. We start by proving the corresponding property for static implication.

Lemma 27.29. *If $F \leq G$ then $((\nu x)F) \leq ((\nu x)G)$.*

Proof. We must prove that for all φ if $((\nu x)F) \vdash \varphi$ then $((\nu x)G) \vdash \varphi$. The complication of the proof lies in that x is not guaranteed to be fresh for φ

- We obtain a fresh name y such that y is fresh for everything in the proof context.
- From $((\nu x)F) \vdash \varphi$ we have by alpha-conversion that $((\nu y)(x\,y) \cdot F) \vdash \varphi$.
- Since $y \sharp \varphi$ we have that $(x\,y) \cdot F \vdash \varphi$ by $\vdash$-E.
- From $F \leq G$ we have that $(x\,y) \cdot F \leq (x\,y) \cdot G$
- With $(x\,y) \cdot F \vdash \varphi$ we have that $(x\,y) \cdot G \vdash \varphi$.
- Since $y \sharp \varphi$ we have that $((\nu y)(x\,y) \cdot G) \vdash \varphi$ by $\vdash$-I.
- Finally, we have by alpha-conversion that $((\nu x)G) \vdash \varphi$

□

We can now prove that static equivalence for frames is preserved by Restriction.

Lemma 27.30. *If $F \simeq G$ then $((\nu x)F) \simeq ((\nu x)G)$.*

Proof. Follows immediately form the definition of $\simeq$ and Lemma 27.29. □

Lemma 27.31. *If $F \simeq G$ then $((\nu \tilde{x})F) \simeq ((\nu \tilde{x})G)$.*

Proof. By induction on $\tilde{x}$ and Lemma 27.30 □

In order for bisimilarity to be preserved by restriction the restricted name must not occur in the environment.

Lemma 27.32. *If $\Psi \rhd P \mathrel{\dot{\sim}} Q$ and $x \sharp \Psi$ then $\Psi \rhd (\nu x)P \mathrel{\dot{\sim}} (\nu x)Q$.*

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, (\nu x)P, (\nu x)Q) : \Psi \rhd P \mathrel{\dot{\sim}} Q \wedge x \sharp \Psi\}$

Static equivalence: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have that $(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi$ by $\mathrel{\dot{\sim}}$-E1, and hence $(\mathscr{F}\ ((\nu x)P)) \otimes \Psi \simeq (\mathscr{F}\ ((\nu x)Q)) \otimes \Psi$, since $x \sharp \Psi$ and Lemma 27.30.

Simulation: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have that $\Psi \rhd P \hookrightarrow_{\dot{\sim}} Q$ by $\mathrel{\dot{\sim}}$-E2, and hence using $x \sharp \Psi$ and that bisimilarity is equivariant that $\Psi \rhd (\nu x)P \hookrightarrow_{\mathscr{X}} (\nu x)Q$, by Lemma 27.28.

Extension: Given $\Psi \rhd P \mathrel{\dot{\sim}} Q$, we must prove that $(\Psi \otimes \Psi', (\nu x)P, (\nu x)Q) \in \mathscr{X}$ for all possible Ψ', including those containing names that clash with x.

- A fresh name y is chosen such that $y \sharp \Psi$, $y \sharp \Psi'$, $y \sharp P$, and $y \sharp Q$.
- From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have that $\Psi \otimes (x\,y) \cdot \Psi' \rhd P \mathrel{\dot{\sim}} Q$ by $\mathrel{\dot{\sim}}$-E3.
- Since bisimilarity is equivariant we have that $(x\,y) \cdot \Psi \otimes (x\,y) \cdot \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\sim}} (x\,y) \cdot Q$.
- Since $x \sharp \Psi$ and $y \sharp \Psi$ we have that $\Psi \otimes \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\sim}} (x\,y) \cdot Q$.
- Finally, since $y \sharp \Psi$ and $y \sharp \Psi'$, we can derive that $(\Psi \otimes \Psi', (\nu y)(x\,y) \cdot P, (\nu y)(x\,y) \cdot Q) \in \mathscr{X}$, and since $y \sharp P$ and $y \sharp Q$ we have by alpha-conversion that $(\Psi \otimes \Psi', (\nu x)P, (\nu x)Q) \in \mathscr{X}$.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as bisimilarity is symmetric.

□

The corresponding lemmas for binding sequences can then be created.

Lemma 27.33. *Simulation is preserved by binding sequences.*

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad \mathit{eqvt}\ \mathscr{R} \qquad \tilde{x} \sharp \Psi \qquad \bigwedge \Psi' R S y.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad y \sharp \Psi'}{(\Psi', (\nu y)R, (\nu y)S) \in \mathscr{R}}}{\Psi \rhd (\nu \tilde{x})P \hookrightarrow_{\mathscr{R}} (\nu \tilde{x})Q}$$

Proof. By induction on $\tilde{x}$. In the inductive step, Lemma 27.28 is used on the result from the induction hypothesis to discharge the proof. □

Lemma 27.34. *If $\Psi \rhd P \mathrel{\dot{\sim}} Q$ and $\tilde{x} \sharp \Psi$ then $\Psi \rhd (\nu \tilde{x})P \mathrel{\dot{\sim}} (\nu \tilde{x})Q$.*

Proof. By induction on $\tilde{x}$ and Lemma 27.32 □

27.4.4 Parallel

The proof that bisimilarity is preserved by Parallel is the most awkward of the preservation proofs, and historically this is the proof that most often

fails in calculi of this complexity; the intricate correspondences between parallel processes and their assertions are hard to get completely right. We will give a brief outline of the proof, pointing at the complicated cases, beginning with the case where only one of the processes does an action, and then proceeding to the case where they communicate. All freshness conditions and explicit notion of frames will be omitted for the time being, they will be added at the end of the section where we provide a detailed description of the proof.

We pick the candidate relation

$$\mathscr{X} = \{(\Psi, (\nu\tilde{x})(P \mid R), (\nu\tilde{x})(Q \mid R)) : \Psi \otimes \Psi_R \rhd P \stackrel{\cdot}{\sim} Q\}.$$

The proof is done by induction on $\tilde{x}$, where Lemma 27.34 is used to prove the inductive steps. The problematic case is the base case, where $\tilde{x} = \varepsilon$.

Assuming that $\Psi \otimes \Psi_R \rhd P \stackrel{\cdot}{\sim} Q$ we want to prove that $\Psi \rhd P \mid R \stackrel{\cdot}{\sim} Q \mid R$. This means that for any action that $Q \mid R$ does, $P \mid R$ must mimic that action, and their derivatives must be in the candidate relation $\mathscr{X}$. The other direction is proven implicitly since $\mathscr{X}$ is symmetric. The PAR inversion rule will give us the following four cases:

1. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, where we need to find a P' such that $\Psi \rhd P \mid R \xrightarrow{\alpha} P'$ and $(\Psi, P', Q' \mid R) \in \mathscr{X}$.

2. $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$, where we need to find a P' such that $\Psi \rhd P \mid R \xrightarrow{\alpha} P'$ and $(\Psi, P', Q \mid R') \in \mathscr{X}$.

3. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{MN}} Q'$, $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\tilde{x})N} R'$, and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$, where we need to find a P' such that $\Psi \rhd P \mid R \xrightarrow{\tau} P'$ and $(\Psi, P', (\nu\tilde{x})(Q' \mid R')) \in \mathscr{X}$.

4. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{M}(\nu\tilde{x})N} Q'$, $\Psi \otimes \Psi_Q \rhd R \xrightarrow{KN} R'$, and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$, where we need to find a P' such that $\Psi \rhd P \mid R \xrightarrow{\tau} P'$ and $(\Psi, P', (\nu\tilde{x})(Q' \mid R')) \in \mathscr{X}$.

Case number 1 is unproblematic, and can be solved in much the same way as the corresponding proof for the pi-calculus and CCS.

Case number 2 is an easy case for both the pi-calculus and CCS, but there are two complications. First of all, we can only derive the transition $\Psi \rhd P \mid R \xrightarrow{\alpha} P \mid R'$ if we know that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$, but the inversion rule provides $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ – the transition is derived from the frame of Q, but we need the frame of P. Since $\Psi \rhd P \stackrel{\cdot}{\sim} Q$, we know that their frames are statically equivalent, but we need a lemma which switches the frames that enables the transition of R. Secondly, we need to prove that the derivatives are in the candidate relation, i.e. $(\Psi, P \mid R', Q \mid R') \in \mathscr{X}$, but this requires $\Psi \otimes \Psi_{R'} \rhd P \stackrel{\cdot}{\sim} Q$, and we only know that $\Psi \otimes \Psi_R \rhd P \stackrel{\cdot}{\sim} Q$. We need a

lemma which shows that if two processes are bisimilar in the frame of an agent, they are also bisimilar in the frame of any of its derivatives.

Cases 3 and 4 are symmetric, and we focus on case number 3. In order to derive a communication $\Psi \rhd P \mid R \xrightarrow{\tau} (\nu\tilde{x})(P' \mid R')$ we have to know that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{M}(\nu\tilde{x})N} R'$ and $\Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \dot{\leftrightarrow} K$, but we only know that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{M}(\nu\tilde{x})N} R'$ and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$. The technique used to switch the environment for the transition of R in case 2 is not enough here – we have to simultaneously switch the environment in the transition $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{M}(\nu\tilde{x})N} R'$ to $\Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{M}(\nu\tilde{x})N} R'$, and the channel equality $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \dot{\leftrightarrow} K$ to $\Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrow K$. Finally, in order to prove that the derivatives are in the candidate relation, the same technique as was used in case 2 must be employed.

To summarize, the lemmas required are ones which switch the environment of a transition to a statically equivalent one, both for single transitions and for transitions and channel equivalence when agents communicate. Moreover, we need a lemma which switches the environment of a bisimilarity to any possible derivative environment.

27.4.4.1 Switching environments

The problem from case 2 was that given a transition $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$, we wanted to show that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$ as long as $\Psi \otimes \Psi_R \rhd P \dot{\sim} Q$. We have from $\dot{\sim}$-E1 that $(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$, and it is this observation which motivates the following lemma.

Lemma 27.35.

$$\frac{\begin{pmatrix} \Psi_F \rhd P \xrightarrow{\alpha} P' \qquad \mathcal{F}\ P = (\nu\widetilde{b_P})\Psi_P \qquad \mathit{distinct}\ \widetilde{b_P} \\ ((\nu\widetilde{b_F})\Psi_F \otimes \Psi_P) \leq ((\nu\widetilde{b_G})\Psi_G \otimes \Psi_P) \\ \widetilde{b_F} \sharp P \quad \widetilde{b_G} \sharp P \quad \widetilde{b_F} \sharp \mathit{subject}\ \alpha \quad \widetilde{b_G} \sharp \mathit{subject}\ \alpha \\ \widetilde{b_P} \sharp \widetilde{b_F} \quad \widetilde{b_P} \sharp \widetilde{b_G} \quad \widetilde{b_P} \sharp \Psi_G \end{pmatrix}}{\Psi_G \rhd P \xrightarrow{\alpha} P'}$$

Proof. By frame induction using the lemma from Figure 25.5 on the transition $\Psi_F \rhd P \xrightarrow{\alpha} P'$ with the frame $\mathcal{F}\ P = (\nu\widetilde{b_P})\Psi_P$. The reason that the assumption $((\nu\widetilde{b_F})\Psi_F \otimes \Psi_P) \leq ((\nu\widetilde{b_G})\Psi_G \otimes \Psi_P)$ includes the frame of P is that any assertion which comprises that frame will be available to the environment when the action α is derived from a prefix in P. □

With this lemma, the environment can be switched. We have that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$, and from $(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$ we have that

$(\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R) \leq (\mathscr{F}\ P) \otimes (\Psi \otimes \Psi_R)$. By choosing candidate frames for P and Q such that $\widetilde{b_P}$ and $\widetilde{b_Q}$ are sufficiently fresh, and by commutativity and associativity of Composition, we have that $((\nu \widetilde{b_P})(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu \widetilde{b_Q})(\Psi \otimes \Psi_P) \otimes \Psi_R)$, and hence by Lemma 27.35 that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$.

The other case where an environment has to be switched is when two agents communicate. When the agents $\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$ and $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ communicate with the channel equivalence $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$, all occurrences of Ψ_Q in the transition and the entailment must be switched for Ψ_P. A first attempt could be to use Lemma 27.35 to switch the environment of the transition, but to do the corresponding switch in the entailment turns out to be impossible in the general case. Even though $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$ and $(\mathscr{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R)$, it is not possible to deduce that $\Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrows K$. The only way an environment can be switched for any entailment is if no bound names of either frame occur in the condition of the entailment.

The following lemma does a simultaneous switch of the environment of both the transition, and the channel equivalence entailment.

Lemma 27.36.

$$\frac{\left(\begin{array}{l} \Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R' \qquad \Psi \otimes \Psi_R \rhd P \xrightarrow{ML} P' \\ \\ \Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K \\ \\ ((\nu \widetilde{b_Q})(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu \widetilde{b_P})(\Psi \otimes \Psi_P) \otimes \Psi_R) \\ \mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P \qquad \mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad \mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R \\ \\ \textit{distinct}\ \widetilde{b_P} \qquad \textit{distinct}\ \widetilde{b_R} \qquad \widetilde{b_R} \sharp \widetilde{b_P} \\ \widetilde{b_R} \sharp \widetilde{b_Q} \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q \qquad \widetilde{b_R} \sharp R \qquad \widetilde{b_R} \sharp K \\ \widetilde{b_P} \sharp \Psi \qquad \widetilde{b_P} \sharp R \qquad \widetilde{b_P} \sharp P \qquad \widetilde{b_P} \sharp M \qquad \widetilde{b_Q} \sharp R \qquad \widetilde{b_Q} \sharp M \end{array}\right)}{\exists K'.\ \Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{K'}(\nu \tilde{x})N} R' \wedge \Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrows K' \wedge \widetilde{b_R} \sharp K'}$$

Proof. By frame induction using the lemma from Figure 25.5 on the transition $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ with the frame $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$. □

This lemma obtains an alternative channel equivalent term K' which the agents can use to communicate, and which is derivable from the desired environment.

We have that $\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$, and since $\Psi \otimes \Psi_P \rhd P \mathrel{\dot{\sim}} Q$ that $\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$. We also know that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$. From $(\mathscr{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R)$ we can derive that $((\nu \widetilde{b_P})(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu \widetilde{b_Q})(\Psi \otimes \Psi_P) \otimes \Psi_R)$, as we did

for the previous case. Finally, using Lemma 27.36 we can obtain a K' such that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{K'}(\nu\tilde{x})N} R'$ and $\Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrow K'$, allowing the agents P and R to communicate.

A similar lemma to Lemma 27.36 is required for the symmetric case where P does an output action, and R an input action.

27.4.4.2 Generate derivative frame

The final case which requires special attention is how to ensure that the derivatives of the parallel agents are in the candidate relation $\mathscr{X} = \{(\Psi, (\nu\tilde{x})(P \mid R), (\nu\tilde{x})(Q \mid R)) : \Psi \otimes \Psi_R \rhd P \dot{\sim} Q\}$. Consider the case where only R does a transition and we have derived that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$. We can then derive $\Psi \rhd P \mid R \xrightarrow{\alpha} P \mid R'$, and we must then prove that $(\Psi, P \mid R', Q \mid R') \in \mathscr{X}$, but this is only true if $\Psi \otimes \Psi_R{}' \rhd P \dot{\sim} Q$, and we only know that $\Psi \otimes \Psi_R \rhd P \dot{\sim} Q$.

However, two bisimilar terms must also be bisimilar for all possible extensions of their environment. If we can find a Ψ' such that $\Psi_R \otimes \Psi' \simeq \Psi_R{}'$, we could add Ψ' to the environment of the bisimilarity obtaining $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P \dot{\sim} Q$, and hence by Lemma 27.20, that $\Psi \otimes \Psi_R{}' \rhd P \dot{\sim} Q$.

Since the frame of an agent is comprised of all of its top level assertions, the frame of the derivative should be these assertions composed with the ones under the prefix used to derive the action of the transition. An expected property is:

> *If* $\Psi \rhd P \xrightarrow{\alpha} P'$ *and* $\mathscr{F}\, P = (\nu\widetilde{b_P})\Psi_P$ *then there exists a* Ψ' *such that* $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$.

However, this turns out to be false. Consider the following psi-calculus instance.

$$\begin{aligned}
\mathbf{T} &= \mathscr{N} \\
\mathbf{C} &= \{a : a \in \mathbf{T}\} \cup \{a \dot{\leftrightarrow} b : a, b \in \mathbf{T}\} \\
\mathbf{1} &= \emptyset \\
\mathbf{A} &= \mathscr{N} \\
\otimes &= \cup \\
\vdash &= \{(\Psi, a) : a \in \Psi\} \cup \{(\Psi, a \dot{\leftrightarrow} a) : a \in \mathscr{N}\}
\end{aligned}$$

This instance is a variant of the pi-calculus instance defined in Section 22.2.3, but with assertions being sets of names, and a condition consisting of a single name which is entailed by an environment if that name exists in the environment.

Now consider the transition

$$(\nu x)(\llparenthesis\{x\}\rrparenthesis \mid \overline{a}x.\llparenthesis\{b\}\rrparenthesis) \xrightarrow{\overline{a}(\nu x)x} \llparenthesis\{x\}\rrparenthesis \mid \llparenthesis\{b\}\rrparenthesis$$

where an agent contains the bound name x in an assertion, which is then opened. Initially it might appear that the property above would hold. The agent has the frame $(\nu x)\{x\}$, the derivative has the frame $\{x,\ b\}$ and $\{x\}\otimes\{b\} \simeq \{x,\ b\}$.

However, by alpha-conversion the agent also has the frame $(\nu y)\{y\}$, and there exists no Ψ' such that $\{y\}\otimes\Psi' \simeq \{x,\ b\}$. The opened name is fixed in the derivative, but the frame of the original agent can still be alpha-converted freely.

There does, however, exists an alpha-converting permutation which if applied to the assertion component of the frame would make the two assertions statically equivalent. In the example above, there exists a permutation $(x\ y)$, and $((x\ y)\cdot\{y\})\otimes\{b\} \simeq \{x,\ b\}$. More formally, we have the following lemma.

Lemma 27.37.

$$\frac{\begin{array}{l}\Psi \rhd P \xrightarrow{\alpha} P' \qquad \mathcal{F}\ P = (\nu\widetilde{b_P})\Psi_P \qquad distinct\ \widetilde{b_P} \\ bn\ \alpha \sharp subject\ \alpha \qquad distinct\ (bn\ \alpha) \qquad \widetilde{b_P} \sharp \alpha \qquad \widetilde{b_P} \sharp P \\ \widetilde{b_P} \sharp \mathscr{C} \qquad \widetilde{b_P} \sharp \mathscr{C}' \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp \mathscr{C}'\end{array}}{\begin{array}{l}\exists p\ \Psi'\ \widetilde{b_{P'}}\ \Psi_{P'}.\ set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p\cdot\alpha)) \ \wedge\ distinctPerm\ p\ \wedge \\ \qquad (p\cdot\Psi_P)\otimes\Psi' \simeq \Psi_{P'} \ \wedge\ \mathcal{F}\ P' = (\nu\widetilde{b_{P'}})\Psi_{P'}\ \wedge \\ \qquad distinct\ \widetilde{b_{P'}} \ \wedge\ \widetilde{b_{P'}} \sharp P' \ \wedge\ \widetilde{b_{P'}} \sharp \alpha \ \wedge\ \widetilde{b_{P'}} \sharp p\cdot\alpha\ \wedge \\ \qquad \widetilde{b_{P'}} \sharp \mathscr{C} \ \wedge\ bn\ (p\cdot\alpha) \sharp \mathscr{C}'\ \wedge \\ \qquad bn\ (p\cdot\alpha) \sharp \alpha \ \wedge\ bn\ (p\cdot\alpha) \sharp P'\end{array}}$$

This rather intimidating lemma is one of the most complex in the formalisation. It states that for every transition, where the originating agent has a certain frame, there exists a frame for the derivative, an assertion, and an alpha-converting permutation such that the assertion component of the derivative frame is statically equivalent to the originating one with the permutation applied, composed with the new assertion.

Moreover, many lemmas will requires that the bound names of the derivative frames and the alpha-converted bound names are sufficiently fresh, but the freshness contexts $\mathscr{C}$ and $\mathscr{C}'$ for the two cannot be chosen completely arbitrarily. As the bound names of the frame of the originating process may occur in both the opened names and the names of the derivative frame, these must be fresh for both freshness contexts.

27.4.4.3 Bisimilarity is preserved by Parallel

The candidate relation for the proof is $\mathscr{X} = \{(\Psi, (\nu\widetilde{x})(P \mid R), (\nu\widetilde{x})(Q \mid R)) : \Psi \otimes \Psi_R \rhd P \dot{\sim} Q\}$. Therefore, the simulation lemma requires that the agent P

simulates the agent Q in all possible environments of R. Moreover, in order to prove the simulation, the candidate relation must be a bisimulation. This demonstrates why the definition of bisimilarity looks the way it does – it is when proving that simulation is preserved by the Parallel that the clauses for static equivalence and environment extensions are required.

Lemma 27.38. *Simulation is preserved by Parallel.*

$$\frac{\begin{array}{l} eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}' \\ 1: \bigwedge \widetilde{b_R}\ \Psi_R.\ \dfrac{\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q}{(\Psi \otimes \Psi_R\ ,\ P,\ Q) \in \mathscr{R}} \\ 2: \bigwedge \Psi'\ S\ T\ \widetilde{b_U}\ \Psi_U\ U.\ \dfrac{\begin{pmatrix} (\Psi' \otimes \Psi_U\ ,\ S,\ T) \in \mathscr{R} \\ \mathscr{F}\ U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T \end{pmatrix}}{(\Psi',\ S \mid U,\ T \mid U) \in \mathscr{R}'} \\ 3: \bigwedge \Psi'\ S\ T.\ \dfrac{(\Psi',\ S,\ T) \in \mathscr{R}}{(\mathscr{F}\ T) \otimes \Psi' \leq (\mathscr{F}\ S) \otimes \Psi'} \\ 4: \bigwedge \Psi'\ S\ T.\ \dfrac{(\Psi',\ S,\ T) \in \mathscr{R}}{\Psi' \rhd S \hookrightarrow_{\mathscr{R}} T} \\ 5: \bigwedge \Psi'\ S\ T\ \Psi''.\ \dfrac{(\Psi',\ S,\ T) \in \mathscr{R}}{(\Psi' \otimes \Psi'',\ S,\ T) \in \mathscr{R}} \\ 6: \bigwedge \Psi'\ S\ T\ \widetilde{x}.\ \dfrac{(\Psi',\ S,\ T) \in \mathscr{R}' \qquad \widetilde{x} \sharp \Psi'}{(\Psi',\ (\nu \widetilde{x})S,\ (\nu \widetilde{x})T) \in \mathscr{R}'} \\ 7: \bigwedge \Psi'\ S\ T\ \Psi''.\ \dfrac{(\Psi',\ S,\ T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'',\ S,\ T) \in \mathscr{R}} \end{array}}{\Psi \rhd P \mid R \hookrightarrow_{\mathscr{R}'} Q \mid R}$$

Premise 1 states that $(\Psi \otimes \Psi_R\ ,\ P,\ Q) \in \mathscr{R}$ *for all sufficiently fresh frames* $(\nu \widetilde{b_R})\Psi_R$ *of* R*. If this were not the case, it would not be possible to alpha-convert the frame of* R *without falling outside the relation* $\mathscr{R}$*. Premise 2 states the preservation property of* $\mathscr{R}'$*, again, the frame of the parallel agent must be sufficiently fresh. Premises 3-5 are properties of bisimilarity. Premise 6 states that the relation* $\mathscr{R}'$ *must be preserved by Restriction. Premise 7 states that it must be possible to switch the assertion component of the elements of* $\mathscr{R}$ *for statically equivalent ones.*

Proof. From $\hookrightarrow$-I we have that for all α and T' where $\Psi \rhd Q \mid R \xrightarrow{\alpha} T'$ there must exist a S' such that $\Psi \rhd P \mid R \xrightarrow{\alpha} S'$ and $(\Psi, S', T') \in \mathscr{R}'$. We then apply the PAR-inversion rule on the transition $\Psi \rhd Q \mid R \xrightarrow{\alpha} T'$ and ensure that any bound names avoid Ψ, P, Q, and R. We get the following four cases.

PAR1 ($\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$ **and** $T' = Q' \mid R$)**:**

Moreover we know that $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$ and $\widetilde{b_R} \sharp \alpha$.

- From $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$ we have from the assumptions that $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$, and hence that $\Psi \otimes \Psi_R \rhd P \hookrightarrow_{\mathscr{R}} Q$.
- With $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, $bn\ \alpha \sharp \Psi$, and $bn\ \alpha \sharp P$ we obtain a P' such that $\Psi \otimes \Psi_R \rhd P \xrightarrow{\alpha} P'$ and $(\Psi \otimes \Psi_R, P', Q') \in \mathscr{R}$.
- From $\Psi \otimes \Psi_R \rhd P \xrightarrow{\alpha} P'$, $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$, and $\widetilde{b_R} \sharp \alpha$ we have that $\widetilde{b_R} \sharp P'$ and $\widetilde{b_R} \sharp Q'$ by Lemma 25.26.
- From $\Psi \otimes \Psi_R \rhd P \xrightarrow{\alpha} P'$, $bn\ \alpha \sharp R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, and $\widetilde{b_R} \sharp \alpha$ we have that $\Psi \rhd P \mid R \xrightarrow{\alpha} P' \mid R$ by PAR1.
- Moreover from $(\Psi \otimes \Psi_R, P', Q') \in \mathscr{R}$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P'$, and $\widetilde{b_R} \sharp Q'$ we have by the assumptions that $(\Psi, P' \mid R, Q' \mid R) \in \mathscr{R}'$.
- Finally we prove the goal by instantiating S' to $P' \mid R$.

PAR2 **($\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ and $T' = Q \mid R'$):**

Moreover we know that $\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q$, $\widetilde{b_Q} \sharp \Psi$, $\widetilde{b_Q} \sharp P$, and $\widetilde{b_Q} \sharp Q$.

- We pick frames for P and R such that $\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P$, $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$ such that $\widetilde{b_P}$ and $\widetilde{b_R}$ are fresh for everything in the proof context.
- Since $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$ we have from the assumptions that $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$, and hence that $(\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R) \leq (\mathscr{F}\ P) \otimes (\Psi \otimes \Psi_R)$.
- With the definitions of the frames of P and Q, and the laws of static equivalence we have that $((\nu \widetilde{b_Q})(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu \widetilde{b_P})(\Psi \otimes \Psi_P) \otimes \Psi_R)$.
- With $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ we have that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$ using Lemma 27.35.
- With $bn\ \alpha \sharp P$, $\widetilde{b_P} \sharp \Psi$, $\widetilde{b_P} \sharp R$, and $\widetilde{b_P} \sharp \alpha$ we have that $\Psi \rhd P \mid R \xrightarrow{\alpha} P \mid R'$ by PAR2.
- From $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ and derived freshness conditions we obtain a p, a Ψ', a $\widetilde{b_{R'}}$, and a ${\Psi_R}'$ such that $set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha))$, $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$, $\mathscr{F}\ R' = (\nu \widetilde{b_{R'}}){\Psi_R}'$, and that $bn\ (p \cdot \alpha)$ and $\widetilde{b_{R'}}$ are sufficiently fresh, using Lemma 27.37.
- From $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$, and equivariance of $\mathscr{R}$, we have that $(p \cdot \Psi \otimes \Psi_R, p \cdot P, p \cdot Q) \in \mathscr{R}$, and hence by derived freshness conditions that $(\Psi \otimes (p \cdot \Psi_R), P, Q) \in \mathscr{R}$.
- With the assumptions we have that $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P, Q) \in \mathscr{R}$, and hence that $(\Psi \otimes {\Psi_R}', P, Q) \in \mathscr{R}$ since $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$, and the static equivalence laws.

- With $\widetilde{b_{R'}} \sharp \Psi$, $\widetilde{b_{R'}} \sharp P$, and $\widetilde{b_{R'}} \sharp Q$ we have that $(\Psi, P \mid R', Q \mid R') \in \mathscr{R}'$ using the assumptions.
- Finally we prove the goal by instantiating S' to $P \mid R'$.

COMM1 **($\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$, $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} Q'$, and $T' = (\nu \tilde{x})(Q' \mid R')$):**

Moreover we know that $\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q$, $\widetilde{b_Q} \sharp \Psi$, $\widetilde{b_Q} \sharp P$, $\widetilde{b_Q} \sharp Q$, $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$, and that $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$.

One crucial observation is that we cannot guarantee that $\widetilde{b_R} \sharp M$, and this complicates the proof considerably.

- We pick a frame for P such that $\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P$ and that $\widetilde{b_P}$ is fresh for everything in the proof context.
- From $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ and derived freshness conditions we obtain a p, a Ψ', a $\widetilde{b_{R'}}$, and a ${\Psi_R}'$ such that $set\ p \subseteq set\ \tilde{x} \times set\ (\widetilde{p \cdot \tilde{x}})$, $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$, $\mathscr{F}\ R' = (\nu \widetilde{b_{R'}}){\Psi_R}'$, and that $p \cdot \tilde{x}$ and $\widetilde{b_{R'}}$ are sufficiently fresh, using Lemma 27.37. Note that since we do not have that $\widetilde{b_R} \sharp M$ we also do not have that $(p \cdot \tilde{x}) \sharp M$.
- From $\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$, $\tilde{x} \sharp Q$, and $(p \cdot \tilde{x}) \sharp Q$ we have that $p \cdot \Psi \otimes \Psi_R \rhd Q \xrightarrow{(p \cdot M)N} Q'$ using Lemma 25.15, and hence $\Psi \otimes (p \cdot \Psi_R) \rhd Q \xrightarrow{(p \cdot M)N} Q'$ by equivariance and that $\tilde{x} \sharp \Psi$ and $(p \cdot \tilde{x}) \sharp \Psi$.
- From $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$, $\tilde{x} \sharp R$, and $(p \cdot \tilde{x}) \sharp R$ we have that $\mathscr{F}\ R = (\nu (p \cdot \widetilde{b_R}))(p \cdot \Psi_R)$
- The freshness conditions are extended such that for all contexts $\mathscr{C}$ such that $\widetilde{b_R} \sharp \mathscr{C}$, $\tilde{x} \sharp \mathscr{C}$, and $(p \cdot \tilde{x}) \sharp \mathscr{C}$ we have that $(p \cdot \widetilde{b_R}) \sharp \mathscr{C}$.
- Since $(p \cdot \widetilde{b_R}) \sharp \Psi$, $(p \cdot \widetilde{b_R}) \sharp P$, $(p \cdot \widetilde{b_R}) \sharp Q$ we have from the assumptions that $(\Psi \otimes (p \cdot \Psi_R), P, Q) \in \mathscr{R}$, and hence that $\Psi \otimes (p \cdot \Psi_R) \rhd P \hookrightarrow_{\mathscr{R}} Q$.
- With $\Psi \otimes (p \cdot \Psi_R) \rhd Q \xrightarrow{(p \cdot M)N} Q'$ we obtain a P' such that $\Psi \otimes (p \cdot \Psi_R) \rhd P \xrightarrow{(p \cdot M)N} P'$ and $(\Psi \otimes (p \cdot \Psi_R), P', Q') \in \mathscr{R}$.
- Moreover from $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$ we have that $(p \cdot \Psi \otimes (\Psi_Q \otimes \Psi_R)) \vdash (p \cdot M) \leftrightarrows (p \cdot K)$, and hence by the freshness conditions that $\Psi \otimes (\Psi_Q \otimes (p \cdot \Psi_R)) \vdash (p \cdot M) \leftrightarrows K$.
- Moreover Since $(p \cdot \widetilde{b_R}) \sharp \Psi$, $(p \cdot \widetilde{b_R}) \sharp P$, $(p \cdot \widetilde{b_R}) \sharp Q$ we have from the assumptions that $(\Psi \otimes (p \cdot \Psi_R), P, Q) \in \mathscr{R}$, and hence that $(\mathscr{F}\ Q) \otimes (\Psi \otimes (p \cdot \Psi_R)) \leq (\mathscr{F}\ P) \otimes (\Psi \otimes (p \cdot \Psi_R))$.
- With the definitions of the frames of P and Q, and the laws of static equivalence we have that $((\nu \widetilde{b_Q})(\Psi \otimes \Psi_Q) \otimes (p \cdot \Psi_R)) \leq ((\nu \widetilde{b_P})(\Psi \otimes \Psi_P) \otimes (p \cdot \Psi_R))$.

- Finally with $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\widetilde{x})N} Q'$ we obtain a K' such that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{K'}(\nu\widetilde{x})N} Q'$, $\Psi \otimes (\Psi_Q \otimes (p \cdot \Psi_R)) \vdash (p \cdot M) \leftrightarrow K'$, and $\widetilde{b_R} \sharp K'$ using Lemma 27.36.
- Hence $\Psi \rhd P \mid R \xrightarrow{\tau} (\nu\widetilde{x})(P' \mid R')$ by COMM1.
- Moreover, from $(\Psi \otimes (p \cdot \Psi_R), P', Q') \in \mathscr{R}$ and the assumptions we have that $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in \mathscr{R}$, and hence that $(\Psi \otimes {\Psi_R}', P', Q') \in \mathscr{R}$ since $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$, and the static equivalence laws.
- With $\widetilde{b_{R'}} \sharp \Psi$, $\widetilde{b_{R'}} \sharp P'$, and $\widetilde{b_{R'}} \sharp Q'$ we have by the assumptions that $(\Psi, P' \mid R', Q' \mid R') \in \mathscr{R}'$ and hence that $(\Psi, (\nu\widetilde{x})(P' \mid R'), (\nu\widetilde{x})(Q' \mid R')) \in \mathscr{R}'$.
- Finally we prove the goal by instantiating S' to $(\nu\widetilde{x})(P' \mid R')$.

COMM2**:** Symmetric version of COMM1, but considerably simpler as the permutation p simplifies away completely since input actions have no bound names.

□

Before proving that bisimilarity is preserved by Parallel we must prove that static implication of frames is preserved if their binders are commuted. This is done in two steps. First we require a lemma which propagates a single binder over a sequence of binders.

Lemma 27.39. $((\nu x)(\nu\widetilde{x})F) \simeq ((\nu\widetilde{x})(\nu x)F)$

Proof. By induction on $\widetilde{x}$.

Base case $(\widetilde{x} = \varepsilon)$**:** Follows immediately since $\simeq$ is reflexive.

Inductive step $\widetilde{x} = y\widetilde{y}$**:**

- From the induction hypothesis we have that $((\nu x)(\nu\widetilde{y})F) \simeq ((\nu\widetilde{y})(\nu x)F)$
- With Lemma 27.30 we have that $((\nu y)(\nu x)(\nu\widetilde{y})F) \simeq ((\nu y)(\nu\widetilde{y})(\nu x)F)$
- Hence $((\nu x)(\nu y)(\nu\widetilde{y})F) \simeq ((\nu y)(\nu\widetilde{y})(\nu x)F)$ by Lemma 28.12 and transitivity of static equivalence.

□

We can now prove that binding sequences commute.

Lemma 27.40. $((\nu\widetilde{x})(\nu\widetilde{y})F) \simeq ((\nu\widetilde{y})(\nu\widetilde{x})F)$

Proof. By induction on $\widetilde{x}$.

Base case $(\widetilde{x} = \varepsilon)$**:** Follows immediately since $\simeq$ is reflexive.

Inductive step $\tilde{x} = x\tilde{z}$:

- From the induction hypothesis we have that $((\nu\tilde{z})(\nu\tilde{y})F) \simeq ((\nu\tilde{y})(\nu\tilde{z})F)$
- With Lemma 27.30 we have that $((\nu x)(\nu\tilde{z})(\nu\tilde{y})F) \simeq ((\nu x)(\nu\tilde{y})(\nu\tilde{z})F)$
- Hence $((\nu x)(\nu\tilde{z})(\nu\tilde{y})F) \simeq ((\nu\tilde{y})(\nu x)(\nu\tilde{z})F)$ by Lemma 28.12 and transitivity of static equivalence.

□

We will do the proof for bisimilarity in two stages. First we will prove that bisimilarity is preserved by Parallel if P and Q are bisimilar in the frame of R. We will then prove the general case.

Lemma 27.41.

$$\frac{\Psi \otimes \Psi_R \rhd P \dot{\sim} Q \qquad \mathcal{F}\ R = (\nu\widetilde{b_R})\Psi_R \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q}{\Psi \rhd P \mid R \dot{\sim} Q \mid R}$$

Proof. By coinduction with $\mathcal{X}$ set to

$$\{(\Psi, (\nu\tilde{x})(P \mid R), (\nu\tilde{x})(Q \mid R)) : \tilde{x} \sharp \Psi \wedge (\forall \widetilde{b_R}\ \Psi_R.\ \mathcal{F}\ R = (\nu\widetilde{b_R})\Psi_R \wedge \widetilde{b_R} \sharp \Psi \wedge \widetilde{b_R} \sharp P \wedge \widetilde{b_R} \sharp Q \longrightarrow \Psi \otimes \Psi_R \rhd P \dot{\sim} Q)\}$$

Static equivalence: Frames $(\nu\widetilde{b_P})\Psi_P$, $(\nu\widetilde{b_Q})\Psi_Q$, and $(\nu\widetilde{b_R})\Psi_R$ are chosen for agents P, Q, and R respectively, where $\widetilde{b_P}$, $\widetilde{b_Q}$, and $\widetilde{b_R}$ are fresh for everything in the proof context.

Assuming

$$(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$$

we have to prove that

$$(\mathcal{F}\ ((\nu\tilde{x})(P \mid R))) \otimes \Psi \simeq (\mathcal{F}\ ((\nu\tilde{x})(Q \mid R))) \otimes \Psi.$$

- From
$$(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$$
we have that
$$((\nu\widetilde{b_P})(\Psi \otimes \Psi_R) \otimes \Psi_P) \simeq ((\nu\widetilde{b_Q})(\Psi \otimes \Psi_R) \otimes \Psi_Q).$$
- By associativity and commutativity of $\otimes$, Lemma 27.31, and transitivity of $\simeq$ we have that
$$((\nu\widetilde{b_P})\Psi \otimes (\Psi_P \otimes \Psi_R)) \simeq ((\nu\widetilde{b_Q})\Psi \otimes (\Psi_Q \otimes \Psi_R)).$$

- Hence
$$((\nu\widetilde{b_R}\widetilde{b_P})\Psi \otimes (\Psi_P \otimes \Psi_R)) \simeq ((\nu\widetilde{b_R}\widetilde{b_Q})\Psi \otimes (\Psi_Q \otimes \Psi_R))$$
by Lemma 27.31.
- Hence
$$((\nu\widetilde{b_P}\widetilde{b_R})\Psi \otimes (\Psi_P \otimes \Psi_R)) \simeq ((\nu\widetilde{b_Q}\widetilde{b_R})\Psi \otimes (\Psi_Q \otimes \Psi_R))$$
by Lemma 27.40 and transitivity of $\simeq$.
- Finally we have by Lemma 27.31 that
$$((\nu\tilde{x}\widetilde{b_P}\widetilde{b_R})\Psi \otimes (\Psi_P \otimes \Psi_R)) \simeq ((\nu\tilde{x}\widetilde{b_Q}\widetilde{b_R})\Psi \otimes (\Psi_Q \otimes \Psi_R))$$
which is equal to
$$(\mathscr{F}((\nu\tilde{x})(P \mid R))) \otimes \Psi \simeq (\mathscr{F}((\nu\tilde{x})(Q \mid R))) \otimes \Psi.$$

Simulation: Given that $\Psi \otimes \Psi_R \rhd P \hookrightarrow_{\mathscr{X}} Q$ for all possible frames of R such that $\widetilde{b_R}$ is fresh for Ψ, P, and Q, we must prove that $\Psi \rhd (\nu\tilde{x})(P \mid R) \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu\tilde{x})(Q \mid R)$, where $\tilde{x} \sharp \Psi$
- Using Lemma 27.38 we prove that $\Psi \rhd P \mid R \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} Q \mid R$.
- Moreover we have that for all Ψ, P, Q, and x such that $x \sharp \Psi$, $(\Psi, P, Q) \in \mathscr{X} \cup \dot{\sim}$ implies $(\Psi, (\nu x)P, (\nu x)Q) \in \mathscr{X} \cup \dot{\sim}$, by the definition of $\mathscr{X}$ and Lemma 27.32.
- Finally we have that $\Psi \rhd (\nu\tilde{x})(P \mid R) \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu\tilde{x})(Q \mid R)$ using Lemma 27.33.

Extension: Follows from the definition of $\mathscr{X}$. The binding sequence $\tilde{x}$ is alpha-converted to avoid the extended assertion Ψ'.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as bisimilarity is symmetric.

□

From this lemma, the main result follows directly.

Lemma 27.42. *If* $\Psi \rhd P \dot{\sim} Q$ *then* $\Psi \rhd P \mid R \dot{\sim} Q \mid R$.

Proof. Follows directly by chosing a sufficiently fresh frame for R and Lemma 27.41. □

27.5 Strong equivalence

In a similar way to the pi-calculus, we obtain a congruence by closing bisimilarity under substitutions.

27.5.1 Sequential substitution

For the pi-calculus, we used sequences of single substitutions to define strong equivalence, here we use sequences of parallel substitutions. A parallel substitution is a list of names and list of terms, and a sequential substitution can hence be modeled as a list of such pairs.

Definition 27.43. *The sequential substitution σ applied to X is denoted $X\sigma$.*

$$X\sigma \stackrel{\text{def}}{=} foldl\ (\lambda Q\ (\widetilde{x},\ \widetilde{T}).\ Q[\widetilde{x} := \widetilde{T}])\ X\ \sigma$$

This function iterates over a list of substitutions and applies each substitution to X. It is defined for terms, assertions, conditions, and agents.

Lemma 27.44. *Sequential substitution distributes over agents.*

$$\begin{array}{rcl}
\mathbf{0}\sigma & = & \mathbf{0} \\
(\overline{M}N.P)\sigma & = & \overline{M\sigma}N\sigma.P\sigma \\
(Input\ M\ I)\sigma & = & Input\ M\sigma\ I\sigma \\
(Case\ C)\sigma & = & Case\ C\sigma \\
(P\ \mid\ Q)\sigma & = & P\sigma\ \mid\ Q\sigma \\
If\ y \sharp \sigma\ then\ ((\nu y)P)\sigma & = & (\nu y)P\sigma \\
(\!|\Psi|\!)\sigma & = & \!|\Psi\sigma|\! \\
(!P)\sigma & = & !P\sigma \\
 & & \\
(Trm\ M\ P)\sigma & = & Trm\ (M\sigma)\ (P\sigma) \\
If\ y \sharp \sigma\ then\ (Bind\ y\ I)\sigma & = & Bind\ y\ I\sigma \\
 & & \\
EmptyCase\sigma & = & EmptyCase \\
(Cond\ \varphi\ P\ C)\sigma & = & Cond\ \varphi\sigma\ P\sigma\ C\sigma
\end{array}$$

Proof. By induction on σ. □

27.5.2 Closure under substitution

The constraints on substitution types, defined in Section 24.2.1 require that the length of the sequencens of names and terms are equal, and that the names being substituted are distinct. We define the following predicate to define well formed substitutions.

Definition 27.45 (*wellFormedSubst*)**.**

$$\begin{array}{l}
wellFormedSubst\ \sigma \stackrel{\text{def}}{=} \\
(filter\ (\lambda(\widetilde{x},\ \widetilde{T}).\ \neg(length\ \widetilde{x} = length\ \widetilde{T} \wedge distinct\ \widetilde{x}))\ \sigma) = \epsilon
\end{array}$$

Intuitively, the predicate filters out all of the elements $(\widetilde{x}, \widetilde{T})$ of σ such that either $\widetilde{x}$ is not distinct, or the lengths of $\widetilde{x}$ and $\widetilde{T}$ differ – if the resulting list is empty, the sequential substitution is well formed.

We can now define closure under well formed substitution in a similar way as for the pi-calculus.

Definition 27.46 (Closure under substitution)**.** *A relation $\mathscr{R}$ closed under substitution is denoted $\mathscr{R}^s$.*

$$\mathscr{R}^s \stackrel{\text{def}}{=} \{(\Psi, P, Q) : \forall\, \sigma.\ \mathit{wellFormedSubst}\ \sigma \longrightarrow (\Psi, P\sigma, Q\sigma) \in \mathscr{R}\}$$

We derive the standard primitive inference rules.

Lemma 27.47. *Introduction and elimination rule for substitution closed relations.*

$$\frac{\bigwedge\sigma.\ \dfrac{\mathit{wellFormedSubst}\ \sigma}{(\Psi, P\sigma, Q\sigma) \in \mathscr{R}}}{(\Psi, P, Q) \in \mathscr{R}^s} \qquad \frac{(\Psi, P, Q) \in \mathscr{R}^s \qquad \mathit{wellFormedSubst}\ \sigma}{(\Psi, P\sigma, Q\sigma) \in \mathscr{R}}$$

27.5.3 Strong equivalence

Strong equivalence is defined by closing bisimilarity under well formed sequential substitutions.

Definition 27.48 (Strong equivalence)**.**

$$\Psi \rhd P \sim Q \stackrel{\text{def}}{=} (\Psi, P, Q) \in \dot{\sim}^s$$

As for the pi-calculus, in order to prove that strong equivalence is preserved by Input we must first prove when bisimilarity is preserved by Input.

Lemma 27.49. *Simulation is preserved by Input.*

$$\frac{\bigwedge\widetilde{T}.\ \dfrac{|\widetilde{x}| = |\widetilde{T}|}{(\Psi, P[\widetilde{x} := \widetilde{T}], Q[\widetilde{x} := \widetilde{T}]) \in \mathscr{R}}}{\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P \hookrightarrow_{\mathscr{R}} \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. Follows immediately from the definition of $\hookrightarrow$, the INPUT inversion rule, and the INPUT semantic rule. □

Lemma 27.50.

$$\frac{\bigwedge\widetilde{T}.\ \dfrac{|\widetilde{x}| = |\widetilde{T}|}{\Psi \rhd P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} Q[\widetilde{x} := \widetilde{T}]}}{\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P \mathrel{\dot{\sim}} \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(\Psi, \underline{M}(\lambda\widetilde{x})N.P, \underline{M}(\lambda\widetilde{x})N.Q) : \forall\, \widetilde{T}.\ |\widetilde{x}| = |\widetilde{T}| \longrightarrow \Psi \rhd P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} Q[\widetilde{x} := \widetilde{T}]\}.$$

The simulation case is discharged using Lemma 27.49, and all other cases follow immediately from the bisimilarity elimination rules. □

We can now prove that strong equivalence is preserved by Input.

Lemma 27.51.

$$\frac{\Psi \rhd P \sim Q \qquad \widetilde{x} \sharp \Psi \qquad \mathit{distinct}\, \widetilde{x}}{\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P \sim \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. By the introduction rule in Lemma 27.47 we need to prove that for all σ such that *wellFormedSubst* σ, it holds that $\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P\sigma \mathrel{\dot{\sim}} \underline{M}(\lambda\widetilde{x})N.Q\sigma$.

- We obtain a p such that *set* $p \subseteq$ *set* $\widetilde{x} \times$ *set* $(p \cdot \widetilde{x})$ and $p \cdot \widetilde{x}$ is fresh for everything in the proof context.
- By alpha-converting the proof goal, and pushing the substitutions over the binders we get that we have to prove that $\Psi \rhd \underline{M\sigma}(\lambda\, p \cdot \widetilde{x})(p \cdot N)\sigma.(p \cdot P)\sigma \mathrel{\dot{\sim}} \underline{M\sigma}(\lambda\, p \cdot \widetilde{x})(p \cdot N)\sigma.(p \cdot Q)\sigma$.
- Lemma 27.50 proves this goal if we know that for all $\widetilde{T}$ such that $|\widetilde{x}| = |\widetilde{T}|$ we have that $\Psi \rhd (p \cdot P)\sigma[p \cdot \widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} (p \cdot Q)\sigma[p \cdot \widetilde{x} := \widetilde{T}]$.
- From $\Psi \rhd P \sim Q$ we have that $p \cdot \Psi \rhd p \cdot P \sim p \cdot Q$, and hence that $\Psi \rhd p \cdot P \sim p \cdot Q$ since $\widetilde{x} \sharp \Psi$ and $(p \cdot \widetilde{x}) \sharp \Psi$.
- Since $|\widetilde{x}| = |\widetilde{T}|$, *distinct* $\widetilde{x}$, and *wellFormedSubst* σ we have that *wellFormedSubst* $(\sigma[(p \cdot \widetilde{x}, \widetilde{T})])$, and hence $\Psi \rhd (p \cdot P)\sigma[(p \cdot \widetilde{x}, \widetilde{T})] \mathrel{\dot{\sim}} (p \cdot Q)\sigma[(p \cdot \widetilde{x}, \widetilde{T})]$ by the definition of $\sim$.
- Hence we have that $\Psi \rhd (p \cdot P)\sigma[p \cdot \widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} (p \cdot Q)\sigma[p \cdot \widetilde{x} := \widetilde{T}]$.

□

28. Structural congruence

The structural congruence law for psi-calculi are generally the same as the ones for the pi-calculus. The main difference comes from Case. Whereas the pi-calculus has conditional operators such as Match and Mismatch, and nondeterministic choice using Sum, psi-calculi uses Case. Therefore, the only relevant structural congruence law for Case is scope extension. A complete list of the structural congruence laws can be found in Figure 28.1.

28.1 Scope extension laws

When proving the scope extension laws for the psi-calculi extra care has to be taken when reasoning about the binders. The simulation cases for the OPEN and SCOPE rules require that any bound name is fresh for the environment, and hence this is an added requirement for the simulation. When bisimilarity is proven, this has to be taken into account. Bisimilarity is proven with that extra freshness requirement as well, and before proving the main goal, the binders can be alpha-converted to be sufficiently fresh.

Lemma 28.1. *If* $x \sharp F$ *then* $((\nu x)F) \simeq F$.

Proof. From the definitions of $\simeq$ and $\leq$ we have that we must prove for all φ that $((\nu x)F) \vdash \varphi = F \vdash \varphi$

- We pick a y such that $y \sharp \varphi$, $y \sharp P$, and alpha-convert the goal so that we have to prove that $((\nu y)(x\,y) \cdot F) \vdash \varphi = F \vdash \varphi$.
- Since $x \sharp P$ and $y \sharp P$ we must prove that $((\nu y)F) \vdash \varphi = F \vdash \varphi$.
- Since $y \sharp \varphi$ the goal follows directly from the introduction and elimination rules for $\vdash$ found in Lemma 27.2.

□

28.1.1 Scope extension for Case

The informal version of the scope extension law says that

$$\mathbf{case}\ \widetilde{\phi} : \widetilde{(\nu x)P} \dot{\sim} (\nu x)\mathbf{case}\ \widetilde{\phi} : \widetilde{P} \text{ if } x \sharp \widetilde{\phi}$$

The structural congruence $\equiv$ is defined as the smallest congruence satisfying the following laws:

1. The abelian monoid laws for Parallel: commutativity $P \mid Q \equiv Q \mid P$, associativity $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and $\mathbf{0}$ as unit $P \mid \mathbf{0} \equiv P$.
2. The unfolding law: $!P \equiv P \mid !P$, if *guarded P*.
3. The scope extension laws:

$$(\nu x)\mathbf{0} \equiv \mathbf{0}$$
$$(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q \qquad \text{if } x \sharp P$$
$$(\nu x)(\mathit{Cases}\ \mathit{Cs}) \equiv \mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \mathit{Cs} \quad \text{if } x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C}$$
$$(\nu x)\underline{M}(\lambda\widetilde{x})N.P \equiv \underline{M}(\lambda\widetilde{x})N.(\nu x)P \quad \text{if } x \sharp M \text{ and} x \sharp \widetilde{x} \text{ and } x \sharp N$$
$$(\nu x)(\overline{M}N.P) \equiv \overline{M}N.(\nu x)P \qquad \text{if } x \sharp M \text{ and } x \sharp N$$
$$(\nu x)((\nu y)P) \equiv (\nu y)((\nu x)P)$$

Figure 28.1: The structural congruence laws for psi-calculi

and we need to encode this using the *Cases* function, which takes a list of pairs of conditions and agents to construct an agent. We encode this lemma in the following way

If $x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C}$ *then*
$(\nu x)(\mathit{Cases}\ \widetilde{C}) \equiv \mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}$.

The freshness condition $x \sharp \widetilde{\phi}$ is encoded with the condition $x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C}$ – since the conditions are the first element of every tuple in the list $\widetilde{C}$. Secondly, the agent $\mathbf{case}\ \widetilde{\phi} : \widetilde{(\nu x)P}$ is encoded by the function $\mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}$ – the tuple list $\widetilde{C}$ is traversed, and every agent has the name x restricted to it. The simulation lemmas can then be proven.

Lemma 28.2.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C} \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd (\nu x)(\mathit{Cases}\ \widetilde{C}) \leadsto_{\mathscr{R}} \mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}}$$

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C} \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd \mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C} \leadsto_{\mathscr{R}} (\nu x)(\mathit{Cases}\ \widetilde{C})}$$

Proof. Since $x \sharp \mathit{map}\ \mathit{fst}\ \widetilde{C}$, we have by induction on $\widetilde{C}$ that $x \sharp \mathit{Cases}\ \mathit{map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}$. Moreover since $x \sharp \Psi$ and $x \sharp (\nu x)(\mathit{Cases}\ \widetilde{C})$ we can use the simulation introduction rule from Lemma 27.11 to ensure that x is fresh for the subject of any transition under consideration. The CASE and SCOPE inversion rules can then be used to derive the possible transitions, and the the SCOPE, OPEN, and CASE semantic rules to discharge them. □

Lemma 28.3.

If $x \sharp \mathit{map\ fst}\ \widetilde{C}$ *then* $\Psi \rhd (\nu x)(\mathit{Cases}\ \widetilde{C}) \dot{\sim} \mathit{Cases\ map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}$.

Proof. By coinduction with $\mathscr{X}$ set to

$\{(\Psi, (\nu x)(\mathit{Cases}\ \widetilde{C}),$
$\mathit{Cases\ map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}) : x \sharp \Psi \wedge x \sharp \mathit{map\ fst}\ \widetilde{C}\} \cup$
$\{(\Psi, \mathit{Cases\ map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C},$
$(\nu x)(\mathit{Cases}\ \widetilde{C})) : x \sharp \Psi \wedge x \sharp \mathit{map\ fst}\ \widetilde{C}\}$

In order to prove that the agents $(\nu x)(\mathit{Cases}\ \widetilde{C})$ and $\mathit{Cases\ map}\ (\lambda(\varphi, P).\ (\varphi, (\nu x)P))\ \widetilde{C}$ are in the relation $\mathscr{X}$, x must first be alpha-converted to avoid Ψ.

Static equivalence: We must prove that $((\nu x)\mathbf{1}) \otimes \Psi \simeq \mathbf{1} \otimes \Psi$. Since $x \sharp \Psi$ we have that $((\nu x)(\nu\varepsilon)\mathbf{1} \otimes \Psi) \simeq ((\nu\varepsilon)\mathbf{1} \otimes \Psi)$, which is proved by Lemma 28.1.

Simulation: Follows from Lemma 28.2.

Extension: Follows from the definition of $\mathscr{X}$, where x is alpha-converted to avoid the new assertion.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric.

□

28.1.2 Discharging impossible transitions

As for the pi-calculus, we create a set of elimination rules which help discharge cases that contain transitions that can never occur in the assumptions. The rules are considerably fewer than the ones for the pi-calculus, described in Lemma 18.9 for two reasons. Firstly, the formalisation for psi-calculi does not have the case distinction of actions with or without bound names, and thus there is no need for lemmas which state e.g. that an input-action cannot generate an action with bound names. Secondly there are no lemmas which check equality of names with subjects or objects, as in psi-calculi, these are terms.

Lemma 28.4. *The following transitions can never occur.*

$$\mathit{If}\ \Psi \rhd \mathbf{0} \longmapsto \mathit{Rs}\ \mathit{then}\ \mathit{False}.$$
$$\mathit{If}\ \Psi \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{\overline{K}(\nu\tilde{y})N'} P'\ \mathit{then}\ \mathit{False}.$$
$$\mathit{If}\ \Psi \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{\tau} P'\ \mathit{then}\ \mathit{False}.$$
$$\mathit{If}\ \Psi \rhd \overline{M}N.P \xrightarrow{\underline{K}N'} P'\ \mathit{then}\ \mathit{False}.$$
$$\mathit{If}\ \Psi \rhd \overline{M}N.P \xrightarrow{\tau} P'\ \mathit{then}\ \mathit{False}.$$
$$\mathit{If}\ \Psi \rhd (\!|\Psi'|\!) \longmapsto \mathit{Rs}\ \mathit{then}\ \mathit{False}.$$

Proof. By induction on the inference of the transitions. □

28.1.3 Restricting deadlocked agents

Lemma 28.5.

$$\Psi \rhd (\nu x)\mathbf{0} \hookrightarrow_{\mathscr{R}} \mathbf{0} \qquad\qquad \Psi \rhd \mathbf{0} \hookrightarrow_{\mathscr{R}} (\nu x)\mathbf{0}$$

Follows from $\hookrightarrow$-I, and the SCOPE inversion rules to derive the possible cases – neither agent has any actions.

Lemma 28.6. $\Psi \rhd (\nu x)\mathbf{0} \mathrel{\dot{\sim}} \mathbf{0}$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(\Psi, (\nu x)\mathbf{0}, \mathbf{0}) : x \sharp \Psi\} \cup \{(\Psi, \mathbf{0}, (\nu x)\mathbf{0}) : x \sharp \Psi\}.$$

The name x is alpha-converted to be fresh for Ψ and Lemma 28.5 discharges the simulation. The other cases follow from the definition of $\mathscr{X}$. □

28.1.4 Scope extension for prefixes

Lemma 28.7.

$$\frac{eqvt\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp M \qquad x \sharp N \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd (\nu x)(\overline{M}N.P) \hookrightarrow_{\mathscr{R}} \overline{M}N.(\nu x)P}$$

$$\frac{eqvt\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp M \qquad x \sharp N \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd \overline{M}N.(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)(\overline{M}N.P)}$$

Proof. Follows from $\hookrightarrow$-I, the SCOPE and OUTPUT inversion rules to derive the possible cases, and the SCOPE and OUTPUT semantic laws to discharge them. Note that since $x \sharp N$ the OPEN rule is never used. □

Lemma 28.8. *If $x \sharp M$ and $x \sharp N$ then* $\Psi \rhd (\nu x)(\overline{M}N.P) \mathrel{\dot{\sim}} \overline{M}N.(\nu x)P$.

Proof. By coinduction with $\mathscr{X}$ set to

$\{(\Psi, (\nu x)(\overline{M}N.P), \overline{M}N.(\nu x)P) : x \sharp \Psi \wedge x \sharp M \wedge x \sharp N\} \cup$
$\{(\Psi, \overline{M}N.(\nu x)P, (\nu x)(\overline{M}N.P)) : x \sharp \Psi \wedge x \sharp M \wedge x \sharp N\}$

Static equivalence : We must prove that $((\nu x)\mathbf{1}) \otimes \Psi \simeq \mathbf{1} \otimes \Psi$. Since $x \sharp \Psi$ we have that $((\nu x)(\nu \varepsilon)\mathbf{1} \otimes \Psi) \simeq ((\nu \varepsilon)\mathbf{1} \otimes \Psi)$, which is proved by Lemma 28.1.

Simulation: Follows from Lemma 28.7.

Extension: Follows from the definition of $\mathscr{X}$, where x is alpha-converted to avoid the new assertion.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric.

□

Lemma 28.9.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp M \qquad x \sharp \tilde{x} \qquad x \sharp N \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd (\nu x)\underline{M}(\lambda\tilde{x})N.P \hookrightarrow_{\mathscr{R}} \underline{M}(\lambda\tilde{x})N.(\nu x)P}$$

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad x \sharp \Psi \qquad x \sharp M \qquad x \sharp \tilde{x} \qquad x \sharp N \qquad \bigwedge Q.\ (\Psi, Q, Q) \in \mathscr{R}}{\Psi \rhd \underline{M}(\lambda\tilde{x})N.(\nu x)P \hookrightarrow_{\mathscr{R}} (\nu x)\underline{M}(\lambda\tilde{x})N.P}$$

Proof. Follows from $\hookrightarrow$-I, the SCOPE and INPUT inversion rules to derive the possible cases, and the SCOPE and INPUT semantic laws to discharge them. □

Lemma 28.10. *If $x \sharp M$ and $x \sharp \tilde{x}$ and $x \sharp N$ then $\Psi \rhd (\nu x)\underline{M}(\lambda\tilde{x})N.P \mathrel{\dot{\sim}} \underline{M}(\lambda\tilde{x})N.(\nu x)P$.*

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(\Psi, (\nu x)\underline{M}(\lambda\tilde{x})N.P, \underline{M}(\lambda\tilde{x})N.(\nu x)P) : x \sharp \Psi \wedge x \sharp M \wedge x \sharp \tilde{x} \wedge x \sharp N\} \cup$$
$$\{(\Psi, \underline{M}(\lambda\tilde{x})N.(\nu x)P, (\nu x)\underline{M}(\lambda\tilde{x})N.P) : x \sharp \Psi \wedge x \sharp M \wedge x \sharp \tilde{x} \wedge x \sharp N\}$$

In order to prove that $(\nu x)\underline{M}(\lambda\tilde{x})N.P$ and $\underline{M}(\lambda\tilde{x})N.(\nu x)P$ are in $\mathscr{X}$, x must be alpha-converted to avoid Ψ.

Static equivalence : We must prove that $((\nu x)\mathbf{1}) \otimes \Psi \simeq \mathbf{1} \otimes \Psi$. Since $x \sharp \Psi$ we have that $((\nu x)(\nu\varepsilon)\mathbf{1} \otimes \Psi) \simeq ((\nu\varepsilon)\mathbf{1} \otimes \Psi)$, which is proved by Lemma 28.1.

Simulation: Follows from Lemma 28.9.

Extension: Follows from the definition of $\mathscr{X}$, where x is alpha-converted to avoid the new assertion.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric.

□

28.1.5 Restriction is commutative

The proof that restriction is commutative follows its counterpart for the pi-calculus in all cases except the OPEN case. In psi-calculi, we encode the notion that binders on labels behave as sets, by allowing the opened bound name to be inserted anywhere in the already existing binding sequence of the action. Recall that the OPEN-rule is of the form

$$\frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{x}\tilde{y})N} P' \qquad z \in supp\, N \qquad z \sharp \Psi \qquad z \sharp M \qquad z \sharp \tilde{x} \qquad z \sharp \tilde{y}}{\Psi \rhd (\nu z)P \xrightarrow{\overline{M}(\nu\tilde{x}z\tilde{y})N} P'} \text{OPEN}$$

where the binding sequence in the assumptions is split into $\tilde{x}$ and $\tilde{y}$. Since the rule is derived from all possible such splits, the effect is that the bound name x can be inserted anywhere in the sequence. When proving that restriction is commutative we have, when the OPEN-rule is applied twice that

$$\Psi \rhd (\nu x)((\nu y)P) \xrightarrow{\overline{M}(\nu\tilde{z})N} P'$$

where $\tilde{z}$ can either have the form

$$\tilde{x}_1 x(\tilde{x}_2 y \tilde{x}_3)$$

or

$$\tilde{x}_1 y(\tilde{x}_2 x \tilde{x}_3),$$

and when reapplying the OPEN-rule to commute the binders x and y, the sequences $\tilde{x}$ and $\tilde{y}$ have to be instantiated properly for the proof to work. The proof is not particularly difficult, but it is tedious, and demonstrates the drawbacks of encoding sets of binders in this way.

Lemma 28.11.

$$\frac{x \sharp \Psi \qquad y \sharp \Psi \qquad eqvt\, \mathscr{R} \qquad \bigwedge \Psi' Q.\ (\Psi', Q, Q) \in \mathscr{R} \qquad \bigwedge \Psi' a\, b\, Q.\ \frac{a \sharp \Psi' \qquad b \sharp \Psi'}{(\Psi', (\nu a)((\nu b)Q), (\nu b)((\nu a)Q)) \in \mathscr{R}}}{\Psi \rhd (\nu x)((\nu y)P) \hookrightarrow_{\mathscr{R}} (\nu y)((\nu x)P)}$$

Proof. If $x = y$, the proof is straightforward as simulation is reflexive. In the case that $x \neq y$ the introduction rule from Lemma 27.11 is used, ensuring that the bound names of the actions are fresh for x, y, and P, and that x and y are fresh for their subjects. The SCOPE inversion rule from Figure 26.2 is used to strip away the binders.

This lemma is rather complex with the different combinations of the OPEN and RES rules. The requirement that $\mathscr{R}$ is reflexive is used when one or both of x and y are opened, to ensure that the derivatives stay in the candidate relation. The assumption for the commuting binders is used when the RES rule is used twice keeping both binders in the derivative – they must then be commuted in order for the derivative to stay in $\mathscr{R}$. In the case that the OPEN rule is used twice, the strategy discussed above for placing the binders in the correct place in the binding sequence is used. □

Before moving on to bisimilarity, we must prove that frames with commuted binders are statically equivalent.

Lemma 28.12. $((\nu x)(\nu y)F) \simeq ((\nu y)(\nu x)F)$

Proof. Follows immediately from definitions 27.1, 27.3, and 27.4. □

Lemma 28.13. $\Psi \rhd (\nu x)((\nu y)P) \mathrel{\dot{\sim}} (\nu y)((\nu x)P)$

Proof. An auxiliary lemma is proven where x and y are assumed to be fresh for Ψ. The proof is then done by coinduction with $\mathscr{X}$ set to

$$\{(\Psi, (\nu x)((\nu y)P), (\nu y)((\nu x)P)) : x \sharp \Psi \wedge y \sharp \Psi\}$$

Static equivalence: A frame for P is picked such that $\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P$ and $\widetilde{b_P} \sharp \Psi$. We must hence prove that $((\nu xy\widetilde{b_P})\Psi \otimes \Psi_P) \simeq ((\nu yx\widetilde{b_P})\Psi \otimes \Psi_P)$ which follows immediately from Lemma 28.12.

Simulation: Follows immediately from Lemma 28.11, and equivariance and reflexivity of bisimilarity.

Extension: Follows from the definition of $\mathscr{X}$, but the binders x and y must be alpha-converted to avoid the new assertion Ψ'.

Symmetry: Follows directly since bisimilarity is reflexive.

The proof is concluded by alpha-converting the binders to be fresh for Ψ and using the auxiliary lemma. □

We also require a lemma which commutes a sequence of binders with a single binder.

Lemma 28.14. *If* $y \sharp \Psi$ *and* $\widetilde{x} \sharp \Psi$ *then* $\Psi \rhd (\nu y)((\nu \widetilde{x})P) \mathrel{\dot{\sim}} (\nu \widetilde{x})((\nu y)P)$.

Proof. By induction on $\widetilde{x}$ and Lemma 28.13 □

28.2 Bisimulation up-to techniques

We define a function *bisimCompose* which takes a relation, and returns that relation composed with bisimilarity.

Definition 28.15 (*bisimCompose*)**.** $bisimCompose\ X \stackrel{\text{def}}{=} \{(\Psi, P, Q) : \exists P'\ Q'.\ \Psi \rhd P \mathrel{\dot{\sim}} P' \wedge (\Psi, P', Q') \in X \cup \mathrel{\dot{\sim}} \wedge \Psi \rhd Q' \mathrel{\dot{\sim}} Q\}$

The coinduction rule for bisimulation up-to techniques are then defined in the standard way.

Lemma 28.16. *Coinduction rule for bisimilarity using bisimulation up-to techniques.*

$$\frac{\begin{array}{ll}(\Psi, P, Q) \in \mathscr{Y} & eqvt\ \mathscr{Y} \\ \bigwedge \Psi\ P\ Q.\ \dfrac{(\Psi, P, Q) \in \mathscr{Y}}{(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi} & \textsc{StatEq} \\ \bigwedge \Psi\ P\ Q.\ \dfrac{(\Psi, P, Q) \in \mathscr{Y}}{\Psi \rhd P \hookrightarrow_{bisimCompose\ \mathscr{Y}} Q} & \textsc{Simulation} \\ \bigwedge \Psi\ P\ Q\ \Psi'.\ \dfrac{(\Psi, P, Q) \in \mathscr{Y}}{(\Psi \otimes \Psi', P, Q) \in \mathscr{Y} \vee \Psi \otimes \Psi' \rhd P \mathrel{\dot{\sim}} Q} & \textsc{Extension} \\ \bigwedge \Psi\ P\ Q.\ \dfrac{(\Psi, P, Q) \in \mathscr{Y}}{(\Psi, Q, P) \in bisimCompose\ \mathscr{Y}} & \textsc{Symmetry}\end{array}}{\Psi \rhd P \mathrel{\dot{\sim}} Q}$$

Proof. By coinduction with $\mathscr{X}$ set to $bisimCompose\ \mathscr{Y}$. □

Note that the *bisimCompose* function is only used in the simulation and the symmetry cases.

28.2.1 Scope extension for Parallel

Lemma 28.17.

$$\frac{\begin{array}{l} x \sharp P \qquad x \sharp \Psi \qquad \mathit{eqvt}\ \mathscr{R} \\ \bigwedge \Psi'\, R.\ (\Psi', R, R) \in \mathscr{R} \\ \bigwedge y\, \Psi'\, R\, S\, \widetilde{z}.\ \dfrac{y \sharp \Psi' \qquad y \sharp R \qquad \widetilde{z} \sharp \Psi'}{(\Psi', (\nu y)((\nu \widetilde{z})(R \mid S)), (\nu \widetilde{z})(R \mid (\nu y) S)) \in \mathscr{R}} \\ \bigwedge \Psi'\, \widetilde{z}\, R\, y.\ \dfrac{y \sharp \Psi' \qquad \widetilde{z} \sharp \Psi'}{(\Psi', (\nu y)((\nu \widetilde{z}) R), (\nu \widetilde{z})((\nu y) R)) \in \mathscr{R}} \end{array}}{\Psi \rhd (\nu x)(P \mid Q) \hookrightarrow_{\mathscr{R}} P \mid (\nu x) Q}$$

$$\frac{\begin{array}{l} x \sharp P \qquad x \sharp \Psi \qquad \mathit{eqvt}\ \mathscr{R} \\ \bigwedge \Psi'\, R.\ (\Psi, R, R) \in \mathscr{R} \\ \bigwedge y\, \Psi'\, R\, S\, \widetilde{z}.\ \dfrac{y \sharp \Psi' \qquad y \sharp R \qquad \widetilde{z} \sharp \Psi'}{(\Psi', (\nu \widetilde{z})(R \mid (\nu y) S), (\nu y)((\nu \widetilde{z})(R \mid S))) \in \mathscr{R}} \end{array}}{\Psi \rhd P \mid (\nu x) Q \hookrightarrow_{\mathscr{R}} (\nu x)(P \mid Q)}$$

Proof. Follows from $\hookrightarrow$-I, where any new bound name avoids x, Ψ, P, and Q. The PAR and SCOPE inversion rules from Figure 26.2 are then used to generate all possible cases, and the PAR, COMM, SCOPE, and OPEN rules from the operational semantics are used to discharge them. □

Lemma 28.18. *If $x \sharp P$ then $\Psi \rhd (\nu x)(P \mid Q) \mathrel{\dot{\sim}} P \mid (\nu x) Q$.*

Proof. By coinduction using Lemma 28.16 with $\mathscr{Y}$ set to

$$\{(\Psi, (\nu \widetilde{y})((\nu x)(P \mid Q)), (\nu \widetilde{y})(P \mid (\nu x) Q)) : x \sharp \Psi \wedge x \sharp P \wedge \widetilde{y} \sharp \Psi\} \cup$$
$$\{(\Psi, (\nu \widetilde{y})(P \mid (\nu x) Q), (\nu \widetilde{y})((\nu x)(P \mid Q))) : x \sharp \Psi \wedge x \sharp P \wedge \widetilde{y} \sharp \Psi\}$$

Static equivalence: Frames are picked for P and Q such that their binders are sufficiently fresh. The proof then follows from Lemma 27.39.

Simulation: Follows from Lemma 28.17 which has its requisites proven by reflexivity of bisimilarity and bisimulation up-to techniques used in conjunction with Lemma 28.14.

Extension: Follows from the definition of $\mathscr{Y}$, with the binders alpha-converted to not clash with the the new assertion Ψ'.

Symmetry: The candidate relation $\mathscr{Y}$ is symmetric.

□

A corresponding lemma can be proven for binding sequences.

Lemma 28.19. *If* $\tilde{x} \sharp \Psi$ *and* $\tilde{x} \sharp P$ *then* $\Psi \rhd (\nu \tilde{x})(P \mid Q) \stackrel{\cdot}{\sim} P \mid (\nu \tilde{x})Q$.

Proof. By induction on $\tilde{x}$ and Lemma 28.18. □

28.3 Abelian monoid laws for Parallel

28.3.1 Parallel has Nil as unit

Lemma 28.20.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge Q.\ (\Psi, Q \mid \mathbf{0}, Q) \in \mathscr{R}}{\Psi \rhd P \mid \mathbf{0} \hookrightarrow_{\mathscr{R}} P} \qquad \frac{\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge Q.\ (\Psi, Q, Q \mid \mathbf{0}) \in \mathscr{R}}{\Psi \rhd P \hookrightarrow_{\mathscr{R}} P \mid \mathbf{0}}$$

Proof. Follows from the definition of $\hookrightarrow$, the PAR1 semantic rule for the first sub-lemma, and the PAR inversion rule for the second one. □

Lemma 28.21. $\Psi \rhd P \mid \mathbf{0} \stackrel{\cdot}{\sim} P$

Proof. By coinduction with $\mathscr{X}$ set to

$$\{(\Psi, P \mid \mathbf{0}, P) : \mathit{True}\} \cup \{(\Psi, P, P \mid \mathbf{0}) : \mathit{True}\}.$$

Static equivalence: A frame for P is picked such that $\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P$. We must hence prove that $((\nu \widetilde{b_P})\mathbf{1} \otimes \Psi_P) \simeq ((\nu \widetilde{b_P})\Psi_P)$, which follows from Lemma 27.31, the fact that static equivalence is an equivalence relation and the AID and ACOMM static equivalence laws.

Simulation: Follows directly from Lemma 28.20.

Extension: Follows from the definition of $\mathscr{X}$.

Symmetry: Follows from the definition of $\mathscr{X}$.

□

28.3.2 Parallel is commutative

Lemma 28.22.

$$\frac{\mathit{eqvt}\ \mathscr{R} \qquad \bigwedge \Psi' R S.\ (\Psi', R \mid S, S \mid R) \in \mathscr{R} \qquad \bigwedge \Psi' R S \tilde{x}.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \tilde{x} \sharp \Psi'}{(\Psi', (\nu \tilde{x})R, (\nu \tilde{x})S) \in \mathscr{R}}}{\Psi \rhd P \mid Q \hookrightarrow_{\mathscr{R}} Q \mid P}$$

Proof. Follows from the definition of $\hookrightarrow$. The PAR inversion rules from Figure 26.2 are used to derive the relevant cases, and the PAR, COMM, and CLOSE semantic rules are used to discharge them. In all cases the symmetric semantic rule is used from the inversion rule used to derive the case; PAR2 for PAR1 and so on. □

We must also prove that frame composition commutes.

Lemma 28.23. $\Psi \rhd P \mid Q \mathrel{\dot{\sim}} Q \mid P$

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, (\nu\tilde{x})(P \mid Q), (\nu\tilde{x})(Q \mid P)) : \tilde{x} \sharp \Psi\}$

Static equivalence: We must prove that

$$((\nu\tilde{x})(\mathscr{F}\ P \otimes \mathscr{F}\ Q)) \otimes \Psi \simeq ((\nu\tilde{x})(\mathscr{F}\ Q \otimes \mathscr{F}\ P)) \otimes \Psi.$$

- We pick frames for P and Q such that $\mathscr{F}\ P = (\nu\widetilde{b_P})\Psi_P$, $\mathscr{F}\ Q = (\nu\widetilde{b_Q})\Psi_Q$, and $\widetilde{b_P}$ and $\widetilde{b_Q}$ are fresh for everything in the proof context.
- We have that $\Psi \otimes (\Psi_P \otimes \Psi_Q) \simeq \Psi \otimes (\Psi_Q \otimes \Psi_P)$ by the laws of static equivalence.
- Hence $((\nu\widetilde{b_P}\widetilde{b_Q})\Psi \otimes (\Psi_P \otimes \Psi_Q)) \simeq ((\nu\widetilde{b_P}\widetilde{b_Q})\Psi \otimes (\Psi_Q \otimes \Psi_P))$ by Lemma 27.31.
- Hence $((\nu\widetilde{b_P}\widetilde{b_Q})\Psi \otimes (\Psi_P \otimes \Psi_Q)) \simeq ((\nu\widetilde{b_Q}\widetilde{b_P})\Psi \otimes (\Psi_Q \otimes \Psi_P))$ by Lemma 27.40.
- Finally, we have that $((\nu\tilde{x}\widetilde{b_P}\widetilde{b_Q})\Psi \otimes (\Psi_P \otimes \Psi_Q)) \simeq ((\nu\tilde{x}\widetilde{b_P}\widetilde{b_Q})\Psi \otimes (\Psi_Q \otimes \Psi_P))$ by Lemma 27.31, proving the case since $\tilde{x}$, $\widetilde{b_P}$, and $\widetilde{b_Q}$ are sufficiently fresh.

Simulation: We must prove that

$$\Psi \rhd (\nu\tilde{x})(P \mid Q) \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu\tilde{x})(Q \mid P)$$

- Using Lemma 27.34 we have that $\Psi \rhd P \mid Q \hookrightarrow_{\mathscr{X}} Q \mid P$ by Lemma 28.22.
- Since $\tilde{x} \sharp \Psi$ we have that $\Psi \rhd (\nu\tilde{x})(P \mid Q) \hookrightarrow_{\mathscr{X}} (\nu\tilde{x})(Q \mid P)$ by Lemma 27.33
- Finally we have that $\Psi \rhd (\nu\tilde{x})(P \mid Q) \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu\tilde{x})(Q \mid P)$ since $\hookrightarrow$ is monotonic.

Extension: Follows from the definition of $\mathscr{X}$, but the sequence $\tilde{x}$ must be alpha-converted to avoid the new assertion Ψ'

Symmetry: Follows from the definition of $\mathscr{X}$.

□

28.3.3 Parallel is associative

As for the pi-calculus, we use bisimulation up-to techniques to only have to prove the simulation one way. The pi-calculus simulation proof had 18 cases to prove. For psi-calculi, there are only nine. There are two reasons for this. Firstly, the pi-calculus has its semantics split between transitions which do bound and free actions, whereas the coding of residuals for psi-calculi allow us to reason about both types of actions without a case distinction. Secondly, the pi-calculus has two types of communication rules – the COMM and the CLOSE rules, whereas psi-calculi only has a COMM rule.

Lemma 28.24.

$$\frac{\begin{array}{l} \mathit{eqvt}\ \mathscr{R} \\ \bigwedge \Psi'\ S\ T\ U.\ (\Psi, (S \mid T) \mid U, S \mid (T \mid U)) \in \mathscr{R} \\ \bigwedge \tilde{x}\ \Psi'\ S\ T\ U.\ \dfrac{\tilde{x} \sharp \Psi' \qquad \tilde{x} \sharp S}{(\Psi', (\nu \tilde{x})((S \mid T) \mid U), S \mid (\nu \tilde{x})(T \mid U)) \in \mathscr{R}} \\ \bigwedge \tilde{x}\ \Psi'\ S\ T\ U.\ \dfrac{\tilde{x} \sharp \Psi' \qquad \tilde{x} \sharp U}{(\Psi', (\nu \tilde{x})(S \mid T) \mid U, (\nu \tilde{x})(S \mid (T \mid U))) \in \mathscr{R}} \\ \bigwedge \Psi'\ S\ T\ \tilde{x}.\ \dfrac{(\Psi', S, T) \in \mathscr{R} \qquad \tilde{x} \sharp \Psi'}{(\Psi', (\nu \tilde{x})S, (\nu \tilde{x})T) \in \mathscr{R}} \end{array}}{\Psi \rhd (P \mid Q) \mid R \hookrightarrow_{\mathscr{R}} P \mid (Q \mid R)}$$

Proof. Follows from the $\hookrightarrow$-I, and the PAR inversion rules. At all steps newly occurring bound names avoid all other terms under consideration. The cases are then discharged using the PAR and COMM semantic rules, and the requisites on $\mathscr{R}$ ensure that the derivatives remain in the candidate relation. □

Lemma 28.25.

$$\Psi \rhd (P \mid Q) \mid R \dot{\sim} P \mid (Q \mid R)$$

Proof. By coinduction using Lemma 28.16 with $\mathscr{Y}$ set to

$$\{(\Psi, (\nu \tilde{x})((P \mid Q) \mid R), (\nu \tilde{x})(P \mid (Q \mid R))) : \tilde{x} \sharp \Psi\}.$$

Static equivalence: Similar to the proof for the corresponding case for Lemma 28.23, but with AASSOC used to prove associativity of the assertion components of the frames for P, Q, and R.

Simulation: Follows from Lemma 28.24 which has its requisites proven by reflexivity of bisimilarity, and bisimulation up-to techniques using Lemmas 28.23, 27.34 and 28.19.

Extension: Follows from the definition of $\mathscr{Y}$, with the bound names alpha-converted to avoid the new assertion Ψ'.

Symmetry: The candidate relation $\mathscr{Y}$ is not symmetric. We must prove that for all agents S and T, if S and T are in $\mathscr{Y}$, then T and S are in $\mathit{bisimCompose}\,\mathscr{Y}$. By unfolding the definition of $\mathscr{Y}$ we obtain $\tilde{x}$, P, Q, and R such that $S = (\nu\tilde{x})((P \mid Q) \mid R)$ and $T = (\nu\tilde{x})(P \mid (Q \mid R))$, we must hence prove that $(\nu\tilde{x})(P \mid (Q \mid R))$ and $(\nu\tilde{x})((P \mid Q) \mid R)$ are in $\mathit{bisimCompose}\,\mathscr{Y}$.

- Since $\tilde{x} \sharp \Psi$, we have that

$$\Psi \rhd (\nu\tilde{x})(P \mid (Q \mid R)) \mathrel{\dot{\sim}} (\nu\tilde{x})((R \mid Q) \mid P)$$

  by Lemmas 28.23, 27.42, 27.34, and transitivity of bisimilarity.
- Moreover since $\tilde{x} \sharp \Psi$ we have that

$$(\Psi, (\nu\tilde{x})((R \mid Q) \mid P), (\nu\tilde{x})(R \mid (Q \mid P))) \in \mathscr{Y}$$

  by the definition of $\mathscr{Y}$.
- Moreover since $\tilde{x} \sharp \Psi$, we have that

$$\Psi \rhd (\nu\tilde{x})(R \mid (Q \mid P)) \mathrel{\dot{\sim}} (\nu\tilde{x})((P \mid Q) \mid R)$$

  by Lemmas 28.23, 27.42, 27.34, and transitivity of bisimilarity.
- Hence we have that

$$(\Psi, (\nu\tilde{x})(P \mid (Q \mid R)), (\nu\tilde{x})((P \mid Q) \mid R)) \in \mathit{bisimCompose}\,\mathscr{Y}$$

  by Definition 28.15.

□

28.4 The unfolding law

Lemma 28.26.

$$\frac{\mathit{guarded}\ P \qquad \bigwedge \Psi'\, Q.\ (\Psi', Q, Q) \in \mathscr{R}}{\Psi \rhd\ !P \hookrightarrow_{\mathscr{R}} P \mid !P} \qquad \frac{\bigwedge \Psi'\, Q.\ (\Psi', Q, Q) \in \mathscr{R}}{\Psi \rhd P \mid !P \hookrightarrow_{\mathscr{R}} !P}$$

Proof. Follows from the $\hookrightarrow$-I, the REPL inversion rule, and the REPL semantic rule. □

Lemma 28.27. *If guarded P then $\Psi \rhd\ !P \mathrel{\dot{\sim}} P \mid !P$.*

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, !P, P \mid !P) : \mathit{guarded}\ P\} \cup \{(\Psi, P \mid !P, !P) : \mathit{guarded}\ P\}$.

Static equivalence: Since P is guarded we know that it has an empty frame, and the AID axiom proves the case.

Simulation: Follows from Lemma 28.26 and reflexivity of bisimilarity.

Extension: Follows from the definition of $\mathscr{X}$.

Symmetry: The relation $\mathscr{X}$ is symmetric.

□

28.5 Bisimilarity is preserved by Replication

The proof utilises the observation that whenever a transition $\Psi \rhd !P \xrightarrow{\alpha} P'$, the agent $!P$ will be present as a parallel component of the derivative P'; the semantic rule for replication unfolds instances of P as many times as is needed to infer the desired action, but the agent $!P$ is never eliminated from the derivation. This requires that the simulation proof has more constraints on the candidate relation than seen so far.

Lemma 28.28. *Simulation is preserved by Replication. There are a total of 18 premisess for this simulation lemma.*

$$(\Psi, P, Q) \in \mathscr{R} \qquad eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}' \qquad guarded\ P \qquad guarded\ Q$$

$$1 : \bigwedge \Psi'\, \Psi_U\, S\, T\, U\, \widetilde{b_U}.\ \frac{\begin{array}{c}(\Psi' \otimes \Psi_U, S, T) \in \mathscr{R} \\ \mathscr{F}\ U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T\end{array}}{(\Psi', U \mid S, U \mid T) \in \mathscr{R}}$$

$$2 : \bigwedge \Psi\, S\, T\, U.\ \frac{(\Psi, S, T) \in \mathscr{R} \qquad guarded\ S \qquad guarded\ T}{(\Psi, U \mid !S, U \mid !T) \in \mathscr{R}'}$$

$$3 : \bigwedge \Psi'\, S\, T\, \tilde{x}.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad \tilde{x} \sharp \Psi'}{(\Psi', (\nu \tilde{x})S, (\nu \tilde{x})T) \in \mathscr{R}}$$

$$4 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi', S, T) \in \mathscr{R}}{\Psi' \rhd S \hookrightarrow_{\mathscr{R}} T}$$

$$5 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi', S, T) \in \mathscr{R}}{(\Psi' \otimes \Psi'', S, T) \in \mathscr{R}} \qquad 6 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi', S, T) \in \mathscr{R}}{(\Psi', T, S) \in \mathscr{R}}$$

$$7 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', S, T) \in \mathscr{R}}$$

$$8 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad (\Psi', T, U) \in \mathscr{R}}{(\Psi', S, U) \in \mathscr{R}}$$

$$9 : \bigwedge \tilde{x}\, \Psi'\, S\, T.\ \frac{\tilde{x} \sharp \Psi' \qquad \tilde{x} \sharp T}{(\Psi', (\nu \tilde{x})(S \mid T), (\nu \tilde{x})S \mid T) \in \mathscr{R}}$$

$$10 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi', S, T) \in \mathscr{R}}{(\Psi', S \mid U, T \mid U) \in \mathscr{R}}$$

$$11 : \bigwedge \Psi'\, S\, T\, U.\ (\Psi', S \mid (T \mid U), (S \mid T) \mid U) \in \mathscr{R}$$

$$12 : \bigwedge \Psi'\, S\, T\, U\, O.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad (\Psi', T, U) \in \mathscr{R}' \qquad (\Psi', U, O) \in \mathscr{R}}{(\Psi', S, O) \in \mathscr{R}'}$$

$$13 : \bigwedge \Psi'\, S\, \alpha\, S'\, T.\ \frac{\begin{array}{c}\Psi' \rhd !S \xrightarrow{\alpha} S' \\ (\Psi', S, T) \in \mathscr{R} \qquad bn\ \alpha \sharp \Psi' \qquad bn\ \alpha \sharp S \\ bn\ \alpha \sharp T \qquad guarded\ T \qquad bn\ \alpha \sharp subject\ \alpha\end{array}}{\begin{array}{c}\exists T'\, U\, O.\ \Psi' \rhd !T \xrightarrow{\alpha} T' \wedge (\Psi', S', U \mid !S) \in \mathscr{R} \wedge \\ (\Psi', T', O \mid !T) \in \mathscr{R} \wedge (\Psi', U, O) \in \mathscr{R} \wedge supp\ U \subseteq \\ supp\ S' \wedge supp\ O \subseteq supp\ T'\end{array}}$$

$$\Psi \rhd R \mid !P \hookrightarrow_{\mathscr{R}'} R \mid !Q$$

The unnumbered premisess dictate that the agents P and Q must be in $\mathscr{R}$, that the relations are equivariant, and that P and Q must be guarded. Premises 1-3 provide the preservation properties required for $\mathscr{R}$ and $\mathscr{R}'$. Premises 4-6 ensure that $\mathscr{R}$ meets all requirements of strong bisimilarity, except static equivalence. Premise 7 ensures that statically equivalent assertions can be switched in $\mathscr{R}$, and premise 8 that $\mathscr{R}$ is transitive. Premises 9-11 are structural congruence laws. Premise 12 allows bisimulation up-to techniques to be used on the relations.

Premise 13 states that for all possible actions that if $\Psi \rhd !P \xrightarrow{\alpha} P'$, there must be a Q', an R and a T such that $\Psi \rhd !Q \xrightarrow{\alpha} Q'$, and it must be possible to ensure that the pair of agents Q' and $T \mid !Q$, are in the relation $\mathscr{R}$ as well as the agents P' and $R \mid !Q$. Moreover, R and T must be in $\mathscr{R}$. This last premise states that it is possible to extract the replicated component from any derivative and the derivatives are still in the relation.

Proof. The proof follows from $\hookrightarrow$-I, and the PAR inversion rule. When the agent $!Q$ does an action, the last of the assumptions is used to generate a mimicking action by $!P$. The other assumptions are then used to ensure that the derivatives are in $\mathscr{R}'$. Other than that, the proof structure is almost identical to the preservation lemma for Parallel, Lemma 27.38. □

This lemma looks intimidating since there requirements on $\mathscr{R}$ and $\mathscr{R}'$ are extensive. However, all of the requirements except 13 have either already been proven, or are easily derivable from the rules already proven for bisimilarity. For requirement 13, we need an auxiliary lemma.

Lemma 28.29.

$$\frac{\Psi \rhd !P \xrightarrow{\alpha} P' \qquad \Psi \rhd P \dot{\sim} Q \qquad bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp P \qquad bn\,\alpha \sharp Q \qquad bn\,\alpha \sharp subject\,\alpha \qquad guarded\,Q}{\begin{array}{l}\exists\, Q'\, R\, T.\\ \quad \Psi \rhd !Q \xrightarrow{\alpha} Q' \wedge\\ \quad \Psi \rhd P' \dot{\sim} R \mid !P \wedge\\ \quad \Psi \rhd Q' \dot{\sim} T \mid !Q \wedge \Psi \rhd R \dot{\sim} T \wedge supp\,R \subseteq supp\,P' \wedge supp\,T \subseteq supp\,Q'\end{array}}$$

Proof. Proof by induction on $\Psi \rhd !P \xrightarrow{\alpha} P'$ using Lemma 25.6. □

Given two bisimilar processes P and Q, this lemma states that if $!P$ can do an action, $!Q$ can mimic that action, and their derivatives can be transformed using the laws of bisimilarity such that they consist of two bisimilar processes in parallel with the replicated ones.

With this lemma in place we derive the lemma which proves that bisimilarity is preserved by replication.

Lemma 28.30.

If $\Psi \rhd P \dot{\sim} Q$ *and guarded* P *and guarded* Q *then* $\Psi \rhd !P \dot{\sim} !Q$.

Proof. By coinduction using Lemma 28.16, with $\mathscr{Y}$ set to

$$\{(\Psi, R \mid !P, R \mid !Q) : \Psi \rhd P \dot{\sim} Q \wedge \textit{guarded } P \wedge \textit{guarded } Q\}.$$

The proof is done by proving that $\Psi \rhd \mathbf{0} \mid !P \dot{\sim} \mathbf{0} \mid !Q$.

Static equivalence: Follows immediately since both $!P$ and $!Q$ have empty frames, by Definition 24.19.

Simulation: Follows from Lemma 28.28 with $\mathscr{R}$ set to $\dot{\sim}$, and $\mathscr{R}'$ set to *bisimCompose* $\dot{\sim}$. Premises 1-3 follow from Lemmas 27.41, 27.42, and 27.34; for Premise 2, some rewriting up to structural congruence is required. Premises 4-6 are properties of bisimilarity. Premise 7 is proven by Lemma 27.20. Premise 8 is proven by transitivity of bisimilarity. Premise 9-11 are proven by Lemma 28.19, 28.23, and 28.25. Premise 12 follows from Definition 28.15. Finally Premise 13 is proven by Lemma 28.29.

Extension: Follows immediately from the definition of $\mathscr{Y}$

Symmetry: The candidate relation $\mathscr{Y}$ is symmetric.

From $\Psi \rhd \mathbf{0} \mid !P \dot{\sim} \mathbf{0} \mid !Q$ we can derive that $\Psi \rhd !P \dot{\sim} !Q$ by Lemma 28.21, and symmetry and transitivity of $\dot{\sim}$. □

28.6 Main results

With the results from this and the previous chapter we can prove three main theorems – that bisimilarity is preserved by all operators except Input, that strong equivalence is a congruence, and that all structurally congruent agents are bisimilar and strongly equivalent.

Theorem 28.1. *Bisimilarity is preserved by all operators except Input.*

Proof. Follows directly from lemmas 27.25, 27.27, 27.32, 27.42, and 28.30. □

Theorem 28.2. *Strong equivalence is a a congruence*

Proof. That strong equivalence is preserved by Input was proven in Lemma 27.51. That it also is an equivalence relation and preserved by the remaining operators follows immediately from Lemma 27.23, Theorem 28.1, and the definition of $\sim$, where any bound names are alpha-converted to avoid the substitutions. □

We also prove that strong bisimilarity and strong equivalence include structural congruence.

Theorem 28.3. *If* $P \equiv Q$ *then* $\mathbf{1} \rhd P \dot{\sim} Q$.

Abelian monoid laws for Parallel: Follows from Lemmas 28.25, 28.23 and 28.21

Unfolding law: Follows from Lemma 28.27.

Scope extension laws: Follows from Lemmas 28.6, 28.10, 28.3, 28.8, 28.13, and 28.18.

Theorem 28.4. *If* $P \equiv Q$ *then* $\mathbf{1} \rhd P \sim Q$.

Proof. Follows from Theorem 28.3 and Definition 27.48 where any binders in the agents are alpha-converted to avoid the sequential substitution. □

29. Weak bisimilarity

In this chapter we introduce weak bisimilarity, $\dot{\approx}$, with the intuition that τ actions are invisible. We will encode τ-prefixes as a communication over a restricted channel; the exact mechanisms for this encoding will be discussed in Chapter 33. This notion is standard in many variants of the pi-calculus, but in our framework it poses unexpected challenges. As an example, consider the law $P \dot{\approx} \tau.P$. This law looks obvious and indeed holds for weak bisimulation in the pi-calculus. But in psi-calculi in general it would imply that parallel composition does not preserve $\dot{\approx}$. Consider a situation where it holds that $\mathbf{1} \vdash \varphi$ and $\mathcal{F}(P) \not\vdash \varphi$. In other words, $\mathcal{F}(P)$ makes condition φ false. Now consider

$$P \mid \mathbf{if}\ \varphi\ \mathbf{then}\ Q \qquad \text{and} \qquad \tau.P \mid \mathbf{if}\ \varphi\ \mathbf{then}\ Q$$

Here only the right hand side has the possibility of acting like Q. Therefore the left and right hand sides are not in general equivalent. If parallel preserves $\dot{\approx}$ then it follows that P and $\tau.P$ are not always equivalent.

The root of this issue is that the frame of P can *falsify* the condition φ. There are some circumstances where this might happen; an example is if the assertions represent constraint stores and the constraint system admits retracts. Suppose that P represents a retract of φ. A system sitting in parallel with P cannot infer φ, and therefore **if** φ **then** Q will have no action. But a system in parallel with $\tau.P$ might infer φ. Only when this agent executes its action τ and asserts the retract will **if** φ **then** Q become blocked. Thus P and $\tau.P$ cannot be deemed equivalent: the parallel context of **if** φ **then** Q can tell the difference by proceeding only in company with the latter.

29.1 Psi-calculi with weakening

In many natural instances of psi-calculi this situation cannot arise. For example, if the logics involved are monotonic there can be nothing similar to a retract: formally, frame composition $\otimes$ is interpreted as conjunction of information, and a logical weakening law is assumed, saying that a conjunction cannot entail less than its conjuncts. In our framework this is represented as an extra requisite:

$$\text{weakening:} \quad \Psi \vdash \varphi \Rightarrow \Psi \otimes \Psi' \vdash \varphi$$

Since $(\otimes, \mathbf{1})$ is a monoid we have $\mathbf{1}\otimes\Psi \simeq \Psi$ for all Ψ, and with weakening this implies $\mathbf{1} \vdash \varphi \Rightarrow \Psi \vdash \varphi$, in other words, no assertion can falsify any condition. With this requisite the law $P \mathrel{\dot{\approx}} \tau.P$ indeed holds, and it turns out that the definition of weak bisimilarity is significantly simpler. We shall therefore begin by exploring weak bisimilarity for psi-instances with weakening, and later generalise to the situation without weakening.

Our approach is to adjust Definition 22.10 (strong bisimilarity) so that τ actions can be inserted or removed when simulating a transition. Clause 1 in the definition, that P and Q are statically equivalent, is adjusted so that if P can make conditions true, then Q can make them true possibly after performing some τ actions. Clauses 2 and 3 are unchanged. Clause 4 (simulation) is split in two parts. If the action α to be simulated is τ then Q should simulate by doing zero or more τs. If it is a visible (i.e. non-τ) action then Q simulates by doing an arbitrary number of τ actions before and after the α action.

We define $\Psi \rhd P \Longrightarrow P'$ to mean that there exist $P_1, \ldots P_n$ where $P = P_1$, $P' = P_n$, and $\Psi \rhd P_i \xrightarrow{\tau} P_{i+1}$ for all i in $[1, n-1]$, allowing the case where $n = 1$ and $P = P'$. The weak transition $\Psi \rhd P \overset{\alpha}{\Longrightarrow} P'$ is defined as $\Psi \rhd P \Longrightarrow P''$ and $\Psi \rhd P'' \xrightarrow{\alpha} P'''$ and $\Psi \rhd P''' \Longrightarrow P'$. We also define $P \leq_\Psi Q$, pronounced P statically implies Q, to mean that $\forall\varphi.\, \Psi\otimes\mathscr{F}(P) \vdash \varphi \;\Rightarrow\; \Psi\otimes\mathscr{F}(Q) \vdash \varphi$. We write $P \leq Q$ for $P \leq_{\mathbf{1}} Q$.

Definition 29.1 (Simple weak bisimilarity)**.** *A* simple weak bisimilarity $\mathscr{R}$ *is a ternary relation between assertions and pairs of agents such that* $\mathscr{R}(\Psi, P, Q)$ *implies all of*

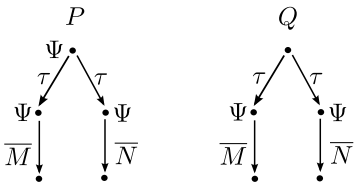
1. *Weak static implication: There exists* Q' *such that*
   $\Psi \rhd Q \Longrightarrow Q'$ *and* $P \leq_\Psi Q'$ *and* $\mathscr{R}(\Psi, P, Q')$.
2. *Symmetry:* $\mathscr{R}(\Psi, Q, P)$
3. *Extension of arbitrary assertion:*
   $\forall\Psi'.\, \mathscr{R}(\Psi\otimes\Psi', P, Q)$
4. *Weak simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha) \sharp \Psi, Q$ *and* $\Psi \rhd P \xrightarrow{\alpha} P'$ *it holds*
   $$\text{if } \alpha = \tau: \quad \exists Q'.\, \Psi \rhd Q \Longrightarrow Q' \wedge \mathscr{R}(\Psi, P', Q')$$
   $$\text{if } \alpha \neq \tau: \quad \exists Q'.\, \Psi \rhd Q \overset{\alpha}{\Longrightarrow} Q' \wedge \mathscr{R}(\Psi, P', Q')$$

We define $\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q$ *to mean that there exists a simple weak bisimulation* $\mathscr{R}$ *such that* $\mathscr{R}(\Psi, P, Q)$*, and write* $P \underset{\text{smp}}{\dot{\approx}} Q$ *for* $\mathbf{1} \rhd P \underset{\text{smp}}{\dot{\approx}} Q$.

The one point which may not be immediately obvious is Clause 1, weak static implication, where the conjunct $\mathscr{R}(\Psi, P, Q')$ may be surprising. It states that Q must evolve to a Q' that is statically implied by P, and also bisimilar to P. This last requirement may seem unnecessarily strong, but in fact without it the resulting simple weak bisimulation equivalence would

not be preserved by the parallel operator. To prove this, let $\dot{\simeq}'$ be defined as simple weak bisimilarity above but without the conjunct $\mathscr{R}(\Psi, P, Q')$ in Clause 1. Let there be an assertion Ψ and condition φ such that $\Psi \vdash \varphi$ and $\mathbf{1} \not\vdash \varphi$, and let L, M, N be distinct terms. Consider the following agents (the diagrams illustrate agents informally):

$$P = (\!|\Psi|\!) \mid (\tau.\overline{M}.\mathbf{0} + \tau.\overline{N}.\mathbf{0})$$
$$Q = \tau.((\!|\Psi|\!) \mid \overline{M}.\mathbf{0}) + \tau.((\!|\Psi|\!) \mid \overline{N}.\mathbf{0})$$
$$R = \mathbf{if}\ \varphi\ \mathbf{then}\ \overline{L}.\mathbf{0}$$



The transitions from P and Q are identical, only their frames differ in that $\mathscr{F}(P) = \Psi$ and $\mathscr{F}(Q) = \mathbf{1}$. With our original definition $P \underset{\text{smp}}{\dot{\simeq}} Q$, since there is no appropriate Q' for Clause 1. In contrast we have $P \dot{\simeq}' Q$ since $Q \xrightarrow{\tau} Q'$ implies $\mathscr{F}(Q') = \mathscr{F}(P)$. But to simulate $P|R \xrightarrow{\overline{L}} P|\mathbf{0}$ from $Q|R$ the only possibilities are $Q|R \overset{\overline{L}}{\Longrightarrow} (\!|\Psi|\!)|\overline{M}.\mathbf{0}|\mathbf{0}$ and $Q|R \overset{\overline{L}}{\Longrightarrow} (\!|\Psi|\!)|\overline{N}.\mathbf{0}|\mathbf{0}$. Neither of these can continue to simulate $P|\mathbf{0}$ which can perform both actions $\overline{M}$ and $\overline{N}$. Therefore $P|R \not\dot{\simeq}' Q|R$.

Simple weak bisimilarity is the natural weak counterpart of Definition 22.10. For all psi-calculi that satisfy the weakening requisite it is sufficient. As we demonstrate in the following section, without weakening simple weak bisimilarity is in general not preserved by parallel composition and also not transitive; therefore a more elaborate definition is required in these cases.

29.2 Psi-calculi without weakening

We now generalise to psi-calculi without the weakening requisite. It turns out that the definition of weak labeled bisimilarity needs to be adjusted in Clauses 1 and 4, where the interplay of assertions and transitions is quite subtle. We proceed to give the full definition of weak labeled bisimilarity followed by a series of examples motivating the need for the added complexities. In Chapter 32 we provide a proof that it coincides with $\underset{\text{smp}}{\dot{\simeq}}$ for psi-calculi with weakening,

Definition 29.2 (Weak bisimilarity)**.** *A* weak bisimilarity $\mathscr{R}$ *is a ternary relation between assertions and pairs of agents such that* $\mathscr{R}(\Psi, P, Q)$ *implies all of*

1. *Weak static implication:*

$$\begin{aligned}&\forall\Psi'\exists Q'',Q'.\\&\quad \Psi \rhd Q \Longrightarrow Q'' \quad\wedge\quad P \leq_\Psi Q'' \quad\wedge\\&\quad \Psi\otimes\Psi' \rhd Q'' \Longrightarrow Q' \quad\wedge\quad \mathscr{R}(\Psi\otimes\Psi', P, Q')\end{aligned}$$

2. *Symmetry:* $\mathscr{R}(\Psi, Q, P)$
3. *Extension of arbitrary assertion:* $\forall\Psi'.\ \mathscr{R}(\Psi\otimes\Psi', P, Q)$
4. *Weak simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha)\sharp\Psi, Q$ *and* $\Psi \rhd P \xrightarrow{\alpha} P'$ *it holds*

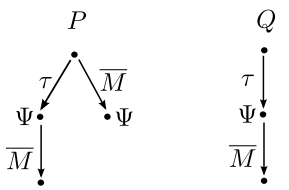
$$\begin{aligned}&\textit{if } \alpha = \tau : \exists Q'.\quad \Psi \rhd Q \Longrightarrow Q' \quad\wedge\quad \mathscr{R}(\Psi, P', Q')\\&\textit{if } \alpha \neq \tau : \forall\Psi'\exists Q'', Q'''.\\&\quad \Psi \rhd Q \Longrightarrow Q''' \quad\wedge\quad P \leq_\Psi Q''' \quad\wedge\\&\quad \Psi \rhd Q''' \xrightarrow{\alpha} Q'' \quad\wedge\\&\quad \exists Q'.\quad \Psi\otimes\Psi' \rhd Q'' \Longrightarrow Q' \quad\wedge\quad \mathscr{R}(\Psi\otimes\Psi', P', Q')\end{aligned}$$

We define $P \dot{\approx}_\Psi Q$ *to mean that there exists a weak bisimulation* $\mathscr{R}$ *such that* $\mathscr{R}(\Psi, P, Q)$ *and write* $P \dot{\approx} Q$ *for* $P \dot{\approx}_{\mathbf{1}} Q$.

We will for the remainder of the chapter motivate the added complexities of this definition.

Example: the use of $P \leq_\Psi Q'''$.
We shall demonstrate that with a simplification omitting $P \leq_\Psi Q'''$ in Clause 4, i.e., if we do not take into account the conditions that hold at the point of executing the visible part of a simulation, then equivalence is not in general preserved by parallel. Let $\dot{\approx}'$ be defined with this simplification. Choose an instance with an assertion Ψ and condition φ such that $\Psi \not\vdash \varphi$ and $\mathbf{1} \vdash \varphi$, i.e., Ψ makes φ false. Consider the agents

$$\begin{aligned}P &= \tau.(\!|\Psi|\!) \mid \overline{M}.\mathbf{0}) + \overline{M}.(\!|\Psi|\!)\\ Q &= \tau.(\!|\Psi|\!) \mid \overline{M}.\mathbf{0})\\ R &= \textbf{if } \varphi \textbf{ then } \underline{M}.\overline{N}.\mathbf{0}\end{aligned}$$



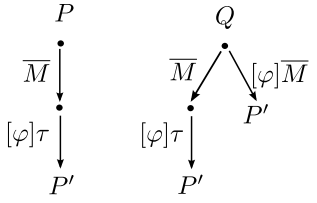
Here $P \dot{\approx}' Q$. To see this, consider the only transition that differs between the agents, namely $P \xrightarrow{\overline{M}} (\!|\Psi|\!)$. This can be simulated by $Q \xrightarrow{\tau} (\!|\Psi|\!) \mid \overline{M}.\mathbf{0} = Q'''$ and $Q''' \xrightarrow{\overline{M}} (\!|\Psi|\!)|\mathbf{0}$. But in composition with R, we have through the second branch of P that $P|R \xrightarrow{\tau} (\!|\Psi|\!)|\overline{N}.\mathbf{0}$. This cannot be weakly simulated by

$Q|R$ since $Q|R \xrightarrow{\tau} (\!|\Psi|\!) \,|\, \overline{M}.\mathbf{0} \,|\, R$ which has no $\overline{N}$ transition. Therefore $P|R \dot{\not\approx}' Q|R$ and $\dot{\approx}'$ is not preserved by parallel.

Example: the quantification $\forall\Psi'$.

Next we motivate the quantification of Ψ' in the sub-clause $\alpha \neq \tau$ of weak simulation, showing that without it, again equivalence would not be preserved by parallel. Let $\dot{\approx}'$ be defined with this simplification. Let Ψ and φ be such that $\mathbf{1} \vdash \varphi$ and $\Psi \not\vdash \varphi$ and let

$$P = \overline{M}.\mathbf{if}\ \varphi\ \mathbf{then}\ \tau.P'$$
$$Q = P + \mathbf{if}\ \varphi\ \mathbf{then}\ \overline{M}.P'$$
$$R = \underline{M}.(\!|\Psi|\!)$$



Here $P \dot{\approx}' Q$. Clearly we have $Q|R \xrightarrow{\tau} P'|(\!|\Psi|\!)$ through the second branch of Q. This cannot be weakly simulated by $P|R$. Here the only transition is $P|R \xrightarrow{\tau} \mathbf{if}\ \varphi\ \mathbf{then}\ \tau.P'\,|\,(\!|\Psi|\!)$ which has no further transition. Therefore $P|R \dot{\not\approx}' Q|R$ and $\dot{\approx}'$ is not preserved by parallel.

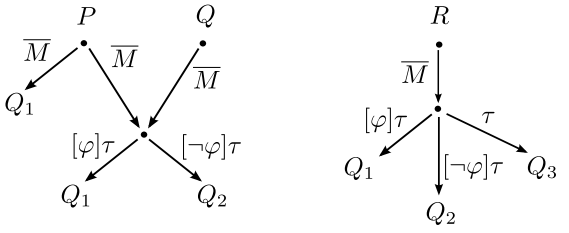
Example: quantifier order of Ψ' *and* Q'.

Next we motivate the order of the quantifiers, showing that if we commute the quantifiers $\forall\Psi'$ and $\exists Q'$ the resulting "equivalence" would not be transitive. Let $\dot{\approx}'$ be defined with these quantifiers commuted. Let all Q_i for $i = 1,2,3$ be distinct but weakly equivalent, and let $\varphi, \neg\varphi$ be two conditions that partition the assertions in two disjoint sets $\{\Psi.\ \Psi \vdash \varphi \wedge \Psi \not\vdash \neg\varphi\}$ and $\{\Psi.\ \Psi \not\vdash \varphi \wedge \Psi \vdash \neg\varphi\}$. Let $\top$ be a condition that is entailed by all assertions, and let

$$U = \mathbf{case}\ \varphi : \tau.Q_1\ []\ \neg\varphi : \tau.Q_2$$
$$V = \mathbf{case}\ \varphi : \tau.Q_1\ []\ \neg\varphi : \tau.Q_2\ []\ \top : \tau.Q_3$$

Here $U \dot{\approx} V$. The rightmost branch in $\Psi \rhd V \xrightarrow{\tau} Q_3$ is simulated by one of the two branches in U (which one depends on Ψ). Let

$$P = \overline{M}.Q_1 + \overline{M}.U$$
$$Q = \overline{M}.U$$
$$R = \overline{M}.V$$

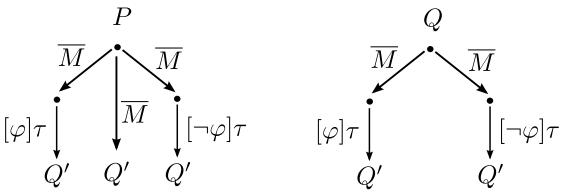


Our point is that although $P \dot{\approx} R \dot{\approx} Q$ we have $P \dot{\approx}' R$ and $R \dot{\approx}' Q$, but not $P \dot{\approx}' Q$. The crucial difference between the equivalences is explained as follows. $P \dot{\approx} Q$ holds because the only nontrivial simulation is for Q to simulate the first branch of P. This is done by first doing $\overline{M}$ leading to U, and then for all Ψ' continuing to either Q_1 or Q_2, depending on whether $\Psi' \vdash \varphi$ or not. Here the quantification order is important. If the final bisimulation clause would read $\exists Q' \forall \Psi' \ldots$ then Q cannot simulate the first branch of P and therefore $P \dot{\not\approx}' Q$. Note that $P \dot{\approx}' R$ since the only nontrivial case is again for R to simulate the first branch of P. This can be done through the third branch leading to Q_3. This holds for any Ψ'.

Example: quantifier order of Ψ' and Q''.

In Clause 4, the quantifier order is $\forall \Psi' \exists Q''$. Let $\dot{\approx}'$ be defined with the alternative order $\exists Q'' \forall \Psi'$. The difference is highlighted by the following example. Let φ and $\neg\varphi$ be two conditions such that for any assertion exactly one of them is entailed, as in the previous example. Let

$$
\begin{aligned}
P = \quad & \overline{M}.Q' + Q \\
Q = \quad & \overline{M}.\mathbf{if}\ \varphi\ \mathbf{then}\ \tau.Q' \\
& + \overline{M}.\mathbf{if}\ \neg\varphi\ \mathbf{then}\ \tau.Q'
\end{aligned}
$$



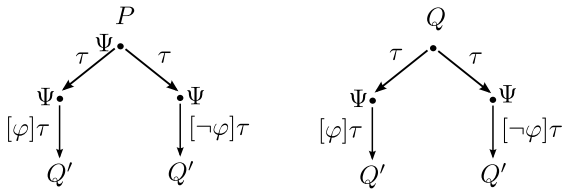
Here $P \dot{\approx} Q$ and $P \dot{\not\approx}' Q$. To see this consider how Q can simulate $P \xrightarrow{\overline{M}} Q'$. Using $\dot{\approx}$, for all Ψ' we must find a Q'' such that $Q \xrightarrow{\overline{M}} Q''$ and $Q'' \implies Q'$. This holds, since the choice of Q'' may depend on Ψ'. Using $\dot{\approx}'$ we must find one Q'' suitable for all Ψ', and there is none.

As it turns out $\dot{\approx}'$ is a viable definition, in the sense that it is transitive and preserves parallel. But from an observational point of view it is hard to argue that P and Q should be different — in essence that would give the observer the power to observe that a conditional branch has been passed. The difference between $\dot{\approx}$ and $\dot{\approx}'$ is reminiscent of the difference between late and early equivalence.

Example: quantifiers in Clause 1.

Keeping the simpler Clause 1 from Definition 29.1 will also yield an equivalence $\dot{\approx}'$ that preserves parallel. A distinguishing example is similar to the one above. Again, let φ and $\neg\varphi$ be two conditions such that for any assertion exactly one of them is entailed. Let Ψ be an assertion such that $\mathbf{1} \leq \Psi$ and $\Psi \not\leq \mathbf{1}$ and $\Psi \otimes \Psi \simeq \mathbf{1}$.

$$
\begin{aligned}
P = \quad & (\!|\Psi|\!) \mid (\tau\,.\,\textbf{if}\ \varphi\ \textbf{then}\ \tau.Q' + \tau\,.\,\textbf{if}\ \neg\varphi\ \textbf{then}\ \tau.Q' \\
Q = \quad & \tau\,.\,((\!|\Psi|\!) \mid \textbf{if}\ \varphi\ \textbf{then}\ \tau.Q') \\
& + \tau\,.\,((\!|\Psi|\!) \mid \textbf{if}\ \neg\varphi\ \textbf{then}\ \tau.Q')
\end{aligned}
$$



Here we assume that Q' is weakly bisimilar to $(\!|\Psi|\!) \mid Q$. Then $P \mathrel{\dot{\approx}} Q$. The critical argument is that in Clause 1, depending on whether $\Psi' \otimes \Psi \vdash \varphi$ or not, Q can evolve to either $(\!|\Psi|\!) \mid \textbf{if}\ \varphi\ \textbf{then}\ \tau.Q'$ or $(\!|\Psi|\!) \mid \textbf{if}\ \neg\varphi\ \textbf{then}\ \tau.Q'$, in either case reaching an agent with a frame Ψ. It can then continue to $(\!|\Psi|\!) \mid Q' \mathrel{\dot{\approx}} (\!|\Psi \otimes \Psi|\!) \mid Q \mathrel{\dot{\approx}} Q$. In contrast $P \mathrel{\dot{\not\approx}'} Q$, since Q cannot evolve to an agent that both has Ψ as frame and is bisimilar to P. Again, it is hard to argue that they should be different from an observational point of view. In [47] we define a barbed bisimilarity for psi-calculi and prove that it is sound and complete with respect to weak bisimilarity. This last example was discovered in the process of doing this proof.

30. Formalising weak bisimilarity

In this chapter we will formalise our results for weak bisimilarity in Isabelle. All proofs will be conducted on psi-calculi without weakening. We will later in Chapter 32 prove that the simple bisimilarity described in the previous chapter coincide for psi-calculi with weakening.

30.1 Tau chains

The encodings of τ-chains for CCS and the pi-calculus use the reflexive transitive closure of τ-actions. For psi-calculi transitions are parametrised with an environment. We define τ-chains in the following way.

Definition 30.1 (τ-chain)**.**

$$\Psi \rhd P \Longrightarrow P' \stackrel{\text{def}}{=} (P, P') \in \{(P, P') : \Psi \rhd P \xrightarrow{\tau} P'\}^*$$

We also define τ-chains which must contain at least one τ-action. This is defined as the transitive closure of τ-actions.

Definition 30.2 (τ-respecting τ-chain)**.**

$$\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P' \stackrel{\text{def}}{=} (P, P') \in \{(P, P') : \Psi \rhd P \xrightarrow{\tau} P'\}^+$$

The definition of τ-chains allows us to use Isabelle's induction rules for reflexive transitive closures and transitive closures.

Lemma 30.3. *Induction rule for τ-chains.*

$$\frac{\left(\begin{array}{l} \Psi \rhd R \Longrightarrow R' \qquad \bigwedge P.\ Prop\ P\ P \\ \\ \bigwedge P\ P'\ P''.\ \dfrac{\Psi \rhd P' \xrightarrow{\tau} P'' \qquad \Psi \rhd P \Longrightarrow P' \qquad Prop\ P\ P'}{Prop\ P\ P''} \end{array}\right)}{Prop\ R\ R'}$$

$$\frac{\left(\begin{array}{l} \Psi \rhd R \stackrel{\tau}{\Longrightarrow} R' \qquad \bigwedge P\, P'.\ \dfrac{\Psi \rhd P \xrightarrow{\tau} P'}{Prop\ P\ P'} \\ \\ \bigwedge P\, P'\, P''.\ \dfrac{\Psi \rhd P' \xrightarrow{\tau} P'' \qquad \Psi \rhd P \stackrel{\tau}{\Longrightarrow} P' \qquad Prop\ P\ P'}{Prop\ P\ P''} \end{array}\right)}{Prop\ R\ R'}$$

Proof. Follows immediately from Isabelle's induction rule for reflexive transitive closures and transitive closures. □

We can now derive basic lemmas for equivariance, freshness, and static equivalence.

Lemma 30.4. *If* $\Psi \rhd P \Longrightarrow P'$ *then* $p \cdot \Psi \rhd p \cdot P \Longrightarrow p \cdot P'$.

Proof. By induction on $\Psi \rhd P \Longrightarrow P'$

Base case ($P = P'$): Follows immediately since $p \cdot \Psi \rhd p \cdot P \Longrightarrow p \cdot P$.

Inductive step ($\Psi \rhd P \Longrightarrow P''$ and $\Psi \rhd P'' \xrightarrow{\tau} P'$): From the induction hypothesis we have that $p \cdot \Psi \rhd p \cdot P \Longrightarrow p \cdot P''$, with $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $p \cdot \Psi \rhd p \cdot P'' \xrightarrow{\tau} p \cdot P'$ by Lemma 25.10, and hence $p \cdot \Psi \rhd p \cdot P \Longrightarrow p \cdot P'$ from Definition 30.1.

□

Lemma 30.5. *If* $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ *then* $p \cdot \Psi \rhd p \cdot P \stackrel{\tau}{\Longrightarrow} p \cdot P'$.

Proof. Similar to Lemma 30.4 □

Lemma 30.6. *If* $\Psi \rhd P \Longrightarrow P'$ *and* $\widetilde{x} \sharp P$ *then* $\widetilde{x} \sharp P'$.

Proof. By induction on $\Psi \rhd P \Longrightarrow P'$

Base case ($P = P'$): Follows immediately since $\widetilde{x} \sharp P$.

Inductive step ($\Psi \rhd P \Longrightarrow P''$ and $\Psi \rhd P'' \xrightarrow{\tau} P'$): From the induction hypothesis we have that $\widetilde{x} \sharp P''$, and with $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $\widetilde{x} \sharp P'$ by Lemma 25.23.

□

Lemma 30.7. *If* $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ *and* $\widetilde{x} \sharp P$ *then* $\widetilde{x} \sharp P'$.

Proof. Similar to Lemma 30.6 □

$$\frac{\Psi \otimes \Psi_Q \rhd P \Longrightarrow P' \quad \mathscr{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \quad \widetilde{b_Q} \sharp \Psi \quad \widetilde{b_Q} \sharp P}{\Psi \rhd P \mid Q \Longrightarrow P' \mid Q}\ \textsc{Par1}$$

$$\frac{\Psi \otimes \Psi_P \rhd Q \Longrightarrow Q' \quad \mathscr{F}\, P = (\nu \widetilde{b_P})\Psi_P \quad \widetilde{b_P} \sharp \Psi \quad \widetilde{b_P} \sharp Q}{\Psi \rhd P \mid Q \Longrightarrow P \mid Q'}\ \textsc{Par2}$$

$$\frac{\Psi \rhd P \Longrightarrow P' \quad x \sharp \Psi}{\Psi \rhd (\nu x)P \Longrightarrow (\nu x)P'}\ \textsc{Scope}$$

Figure 30.1: Lifted operational semantics for τ-chains

Lemma 30.8. *If* $\Psi \rhd P \Longrightarrow P'$ *and* $\Psi \simeq \Psi'$ *then* $\Psi' \rhd P \Longrightarrow P'$.

Proof. By induction on $\Psi \rhd P \Longrightarrow P'$

Base case ($P = P'$)**:** Follows immediately since $\Psi' \rhd P \Longrightarrow P$.

Inductive step ($\Psi \rhd P \Longrightarrow P''$ **and** $\Psi \rhd P'' \xrightarrow{\tau} P'$)**:** From the induction hypothesis we have that $\Psi' \rhd P \Longrightarrow P''$, with $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $\Psi' \rhd P'' \xrightarrow{\tau} P'$ by Lemma 25.11, and hence $\Psi' \rhd P \Longrightarrow P'$ from Definition 30.1.

□

Lemma 30.9. *If* $\Psi \rhd P \Longrightarrow P'$ *and* $\Psi \simeq \Psi'$ *then* $\Psi' \rhd P \Longrightarrow P'$.

Proof. Similar to Lemma 30.8 □

As for pi-calculus and CCS we lift the operational semantics to include τ-chains. The lifted semantics for the relevant rules can be found in Figure 30.1.

30.2 Weak semantics

Recall the simulation requisite of weak bisimilarity

for all α, P' such that $\mathrm{bn}(\alpha) \sharp \Psi, Q$ and $\Psi \rhd P \xrightarrow{\alpha} P'$ it holds

$$
\begin{aligned}
&\text{if } \alpha = \tau : \exists Q'. \quad \Psi \rhd Q \Longrightarrow Q' \quad \wedge \quad \mathscr{R}(\Psi, P', Q') \\
&\text{if } \alpha \neq \tau : \forall \Psi' \exists Q'', Q'''. \\
&\quad \Psi \rhd Q \Longrightarrow Q''' \quad \wedge \quad P \leq_\Psi Q''' \quad \wedge \\
&\quad \Psi \rhd Q''' \xrightarrow{\alpha} Q'' \quad \wedge \\
&\quad \exists Q'. \quad \Psi \otimes \Psi' \rhd Q'' \Longrightarrow Q' \quad \wedge \quad \mathscr{R}(\Psi \otimes \Psi', P', Q')
\end{aligned}
$$

For the case when $\alpha \neq \tau$ there are two things we must take into consideration apart from the τ-chains. The first is that the frame of the agent after the first τ-chain must meet a static equivalence constraints. The second is that the path the τ-chains take depend on the extended assertion Ψ' for the trailing τ-chain.

Weak transitions for psi-calculi are defined by a τ-chain, a static implication, and a strong transition in the following manner:

Definition 30.10 (Weak transition)**.** $\Psi : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P' \stackrel{\text{def}}{=}$
$\exists P''. \Psi \rhd P \Longrightarrow P'' \wedge (\mathscr{F}\, Q) \otimes \Psi \leq (\mathscr{F}\, P'') \otimes \Psi \wedge \Psi \rhd P'' \xrightarrow{\alpha} P'$

A weak transition has one additional parameter to regular transitions, and that is the agent whose frame must statically imply the frame of the agent at the end of the first τ-chain.

The reason that the trailing τ-chain is not included in the weak transitions is that the quantifier order of weak bisimilarity – the assertion Ψ' is universally quantified before the existential quantifiers for the actions. If the trailing τ-chain were included, the weak transition would require an extra parameter solely for the purpose of extending the environment of the trailing τ-chain.

30.2.1 Lifting the semantics

The semantics for weak transitions can be found in Figure 30.2, and are lifted in a similar way as is done for the pi-calculus and for CCS. The main difference is the COMM rule. For the previous calculi both transitions in the assumptions of the COMM rules are weak, whereas for psi-calculi only the leftmost transition is weak – the other one is a standard strong transition. The reason for this stems from the proof that weak bisimilarity is preserved by Replication. For the pi-calculus and CCS, a simulation lemma for Parallel is required which states that if P simulates R and Q simulates T, then $P \mid Q$ simulates $R \mid T$. The lemmas in question are Lemmas 9.26 and 15.38. For psi-calculi, we use an alternate strategy to prove that bisimilarity is preserved by Replication, where these types of lemmas for Parallel are not re-

quired, and the significantly simpler weak semantic rules for COMM are sufficient.

30.3 Weak Bisimilarity

Recall that weak bisimilarity has the following requisite for weak static implication:

$$\begin{array}{l}\forall \Psi' \exists Q''\, Q'.\\ \quad \Psi \rhd Q \Longrightarrow Q'' \quad \wedge \quad P \leq_\Psi Q'' \quad \wedge \\ \quad \Psi \otimes \Psi' \rhd Q'' \Longrightarrow Q' \quad \wedge \quad \mathscr{R}(\Psi \otimes \Psi', P, Q')\end{array}$$

This requisite is significantly more complex than the corresponding requisite for strong bisimilarity, which only requires the frames of the agents to be statically equivalent. Hence for weak bisimilarity we define weak implication as a separate definition.

Definition 30.11 (Weak static implication)**.**

$$\begin{array}{l}\Psi \rhd P \lessapprox_{\mathscr{R}} Q \stackrel{\text{def}}{=} \\ \forall \Psi'.\, \exists Q'\, Q''.\\ \qquad \Psi \rhd Q \Longrightarrow Q' \wedge \\ \qquad (\mathscr{F}\, P) \otimes \Psi \leq (\mathscr{F}\, Q') \otimes \Psi \wedge \Psi \otimes \Psi' \rhd Q' \Longrightarrow Q'' \wedge (\Psi \otimes \\ \Psi', P, Q'') \in \mathscr{R}\end{array}$$

As for simulations, a weak static implication is parametrised with a set $\mathscr{R}$ which contains the environment and the agents that are required to be in the candidate relation.

We define weak simulation in a similar way.

Definition 30.12 (Weak simulation)**.** $\Psi \rhd P \mathrel{\widehat{\leadsto}}_{\mathscr{R}} Q \stackrel{\text{def}}{=}$

$$\begin{array}{l}(\forall \Psi'\, \alpha\, Q'.\, \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge bn\, \alpha \sharp \Psi \wedge bn\, \alpha \sharp P \wedge \alpha \neq \tau \longrightarrow \\ \qquad \exists P''.\, \Psi : Q \rhd P \stackrel{\hat{\alpha}}{\Longrightarrow} P'' \wedge \\ \qquad\quad \exists P'.\, \Psi \otimes \Psi' \rhd P'' \Longrightarrow P' \wedge (\Psi \otimes \Psi', P', Q') \in \mathscr{R}) \wedge \\ (\forall Q'.\, \Psi \rhd Q \xrightarrow{\tau} Q' \longrightarrow (\exists P'.\, \Psi \rhd P \Longrightarrow P' \wedge (\Psi, P', Q') \in \mathscr{R}))\end{array}$$

Before we define bisimilarity, we must prove that weak simulation, and weak static implication are monotonic with respect to the candidate relation.

Lemma 30.13. *Weak static implication and weak simulation are monotonic.*

$$\frac{\begin{pmatrix}\Psi \vdash M \leftrightarrow K & distinct\ \tilde{x} & \\ set\ \tilde{x} \subseteq supp\ N & |\tilde{x}| = |\tilde{T}| & (\mathscr{F}\ Q) \otimes \Psi \leq ((\nu\varepsilon)\Psi \otimes \mathbf{1})\end{pmatrix}}{\Psi : Q \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{KN[\tilde{x}:=\tilde{T}]} P[\tilde{x} := \tilde{T}]}\ \textsc{Input}$$

$$\frac{\Psi \vdash M \leftrightarrow K \qquad (\mathscr{F}\ Q) \otimes \Psi \leq ((\nu\varepsilon)\Psi \otimes \mathbf{1})}{\Psi : Q \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}\ \textsc{Output}$$

$$\frac{\begin{pmatrix}\Psi : Q \rhd P \xrightarrow{\alpha} P' & (\varphi, P)\ mem\ \widetilde{C}P & \Psi \vdash \varphi \\ guarded\ P & (\mathscr{F}\ R) \otimes \Psi \leq (\mathscr{F}\ Q) \otimes \Psi & (\mathscr{F}\ R) \otimes \Psi \leq ((\nu\varepsilon)\Psi)\end{pmatrix}}{\Psi : R \rhd Cases\ \widetilde{C}P \xrightarrow{\alpha} P'}\ \textsc{Case}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_Q : R \rhd P \xrightarrow{\alpha} P' & \mathscr{F}\ Q = (\nu\widetilde{b_Q})\Psi_Q & & & \\ bn\ \alpha \sharp Q & \widetilde{b_Q} \sharp \Psi & \widetilde{b_Q} \sharp P & \widetilde{b_Q} \sharp \alpha & \widetilde{b_Q} \sharp R\end{pmatrix}}{\Psi : R \mid Q \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Par1}$$

$$\frac{\begin{pmatrix}\Psi \otimes \Psi_P : R \rhd Q \xrightarrow{\alpha} Q' & \mathscr{F}\ P = (\nu\widetilde{b_P})\Psi_P & & & \\ bn\ \alpha \sharp P & \widetilde{b_P} \sharp \Psi & \widetilde{b_P} \sharp Q & \widetilde{b_P} \sharp \alpha & \widetilde{b_P} \sharp R\end{pmatrix}}{\Psi : P \mid R \rhd P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Par2}$$

$$\frac{\begin{pmatrix}\Psi : Q \rhd P \xrightarrow{\overline{M}(\nu\tilde{x}\tilde{y})N} P' & & & & \\ x \in supp\ N & x \sharp \Psi & x \sharp M & x \sharp \tilde{x} & x \sharp \tilde{y}\end{pmatrix}}{\Psi : (\nu x)Q \rhd (\nu x)P \xrightarrow{\overline{M}(\nu\tilde{x}x\tilde{y})N} P'}\ \textsc{Open}$$

$$\frac{\begin{pmatrix} \Psi \otimes \Psi_Q : R \rhd P \stackrel{MN}{\Longrightarrow} P' & \mathscr{F}\, R = (\nu \widetilde{b_R})\Psi_R \\ \Psi \otimes \Psi_R \rhd Q \stackrel{\overline{K}(\nu \widetilde{x})N}{\longrightarrow} Q' & \mathscr{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q \\ \Psi \otimes (\Psi_R \otimes \Psi_Q) \vdash M \leftrightarrow K & \widetilde{b_R} \sharp \Psi \quad \widetilde{b_R} \sharp P \\ \widetilde{b_R} \sharp Q \quad \widetilde{b_R} \sharp R \quad \widetilde{b_R} \sharp M & \widetilde{b_R} \sharp \widetilde{b_Q} \quad \widetilde{b_Q} \sharp \Psi \\ \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp R & \widetilde{b_Q} \sharp K \quad \widetilde{x} \sharp P \end{pmatrix}}{\Psi \rhd P \mid Q \stackrel{\tau}{\Longrightarrow} (\nu \widetilde{x})(P' \mid Q')}\ \textsc{Comm1}$$

$$\frac{\begin{pmatrix} \Psi \otimes \Psi_Q : R \rhd P \stackrel{\overline{M}(\nu \widetilde{x})N}{\Longrightarrow} P' & \\ \mathscr{F}\, R = (\nu \widetilde{b_R})\Psi_R & \Psi \otimes \Psi_R \rhd Q \stackrel{KN}{\Longrightarrow} Q' \\ \mathscr{F}\, Q = (\nu \widetilde{b_Q})\Psi_Q & \Psi \otimes (\Psi_R \otimes \Psi_Q) \vdash M \leftrightarrow K \\ \widetilde{b_R} \sharp \Psi \quad \widetilde{b_R} \sharp P & \widetilde{b_R} \sharp Q \quad \widetilde{b_R} \sharp R \quad \widetilde{b_R} \sharp M \\ \widetilde{b_R} \sharp \widetilde{b_Q} \quad \widetilde{b_Q} \sharp \Psi & \widetilde{b_Q} \sharp P \quad \widetilde{b_Q} \sharp Q \quad \widetilde{b_Q} \sharp R \\ \widetilde{b_Q} \sharp K \quad \widetilde{x} \sharp Q & \widetilde{x} \sharp M \quad \widetilde{x} \sharp \widetilde{b_Q} \quad \widetilde{x} \sharp \widetilde{b_R} \end{pmatrix}}{\Psi \rhd P \mid Q \stackrel{\tau}{\Longrightarrow} (\nu \widetilde{x})(P' \mid Q')}\ \textsc{Comm2}$$

$$\frac{\Psi : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P' \qquad x \sharp \Psi \qquad x \sharp \alpha}{\Psi : (\nu x)Q \rhd (\nu x)P \stackrel{\alpha}{\Longrightarrow} (\nu x)P'}\ \textsc{Scope}$$

$$\frac{\Psi : R \rhd P \mid !P \stackrel{\alpha}{\Longrightarrow} P' \qquad \mathit{guarded}\, P}{\Psi : R \rhd !P \stackrel{\alpha}{\Longrightarrow} P'}\ \textsc{Repl}$$

Figure 30.2: Lifted semantics for weak transitions.

If $\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} Q$.
If $\Psi \rhd P \mathrel{\dot{\lesssim}}_{\mathscr{R}} Q$ *and* $\mathscr{R} \subseteq \mathscr{R}'$ *then* $\Psi \rhd P \mathrel{\dot{\lesssim}}_{\mathscr{R}'} Q$.

Proof. Follows immediately from the definitions of $\widehat{\rightsquigarrow}$ and $\dot{\lesssim}$. □

Weak bisimilarity can then be defined in the standard way.

Definition 30.14 (Weak bisimilarity)**.** *Weak bisimilarity, denoted* $\dot{\approx}$*, is defined coinductively as the greatest fixpoint satisfying:*

$$\begin{aligned}
\Psi \rhd P \dot{\approx} Q \Longrightarrow\ & \Psi \rhd P \mathrel{\dot{\lesssim}}_{\dot{\approx}} Q && \textsc{StatImp}\\
\wedge\ & \Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\dot{\approx}} Q && \textsc{Simulation}\\
\wedge\ & \forall \Psi'.\, \Psi \otimes \Psi' \rhd P \dot{\approx} Q && \textsc{Extension}\\
\wedge\ & \Psi \rhd Q \dot{\approx} P && \textsc{Symmetry}
\end{aligned}$$

The simulation, symmetry, and extension cases resemble their counterparts from strong bisimilarity. The static implication differs in that the frames of the agents must weakly statically imply each other, rather than be statically equivalent.

30.3.1 Primitive inference rules

We define an introduction rule for weak simulation which allows the user to specify an arbitrary freshness context $\mathscr{C}$ with which no bound names which appear on the label of the transitions may not clash.

Lemma 30.15. *Introduction rule for weak simulation*

$$\frac{\begin{array}{l}\textit{eqvt}\ \mathscr{R}\\ \bigwedge \Psi'\ \alpha\ Q'.\ \dfrac{\begin{pmatrix}\Psi \rhd Q \xrightarrow{\alpha} Q' & bn\,\alpha \sharp \Psi & \\ bn\,\alpha \sharp P & bn\,\alpha \sharp Q & bn\,\alpha \sharp subject\,\alpha \\ bn\,\alpha \sharp \mathscr{C} & distinct\,(bn\,\alpha) & \alpha \neq \tau\end{pmatrix}}{\begin{array}{l}\exists P''.\, \Psi : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P'' \wedge (\exists P'.\, \Psi \otimes \Psi' \rhd P'' \Longrightarrow P' \wedge\\ \qquad (\Psi \otimes \Psi', P', Q') \in \mathscr{R})\end{array}}\\ \bigwedge Q'.\ \dfrac{\Psi \rhd Q \xrightarrow{\tau} Q'}{\exists P'.\, \Psi \rhd P \Longrightarrow P' \wedge (\Psi, P', Q') \in \mathscr{R}}\end{array}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\ \widehat{\rightsquigarrow}\text{-I}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$ where the bound names are alpha-converted to avoid $\mathscr{C}$. □

Lemma 30.16. *Introduction and elimination rules for weak bisimilarity.*

$$\frac{\Psi \rhd P \mathrel{\dot{\lesssim}}_{\dot{\approx}} Q \quad \Psi \rhd P \mathrel{\dot{\widehat{\rightsquigarrow}}}_{\dot{\approx}} Q \qquad \forall \Psi'.\, \Psi \otimes \Psi' \rhd P \mathrel{\dot{\approx}} Q \qquad \Psi \rhd Q \mathrel{\dot{\approx}} P}{\Psi \rhd P \mathrel{\dot{\approx}} Q}\ \dot{\approx}\text{-I}$$

$$\frac{\Psi \rhd P \mathrel{\dot{\approx}} Q}{\Psi \rhd P \mathrel{\dot{\lesssim}}_{\dot{\approx}} Q}\ \dot{\approx}\text{-E1} \qquad \frac{\Psi \rhd P \mathrel{\dot{\approx}} Q}{\Psi \rhd P \mathrel{\dot{\widehat{\rightsquigarrow}}}_{\dot{\approx}} Q}\ \dot{\approx}\text{-E2} \qquad \frac{\Psi \rhd P \mathrel{\dot{\approx}} Q}{\Psi \otimes \Psi' \rhd P \mathrel{\dot{\approx}} Q}\ \dot{\approx}\text{-E3}$$

$$\frac{\Psi \rhd P \mathrel{\dot{\approx}} Q}{\Psi \rhd Q \mathrel{\dot{\approx}} P}\ \dot{\approx}\text{-E4}$$

Proof. Follows from the definition of $\dot{\approx}$ □

Moreover we derive the following coinduction rule from the one generated by Isabelle.

Lemma 30.17. *Coinduction rule for weak bisimilarity.*

$$\frac{\begin{array}{ll} (\Psi, P, Q) \in \mathscr{X} & \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \mathrel{\dot{\lesssim}}_{\mathscr{X} \cup \dot{\approx}} S} & \textsc{StatImp} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \mathrel{\dot{\widehat{\rightsquigarrow}}}_{\mathscr{X} \cup \dot{\approx}} S} & \textsc{Simulation} \\ \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{X} \vee \Psi' \otimes \Psi'' \rhd R \mathrel{\dot{\approx}} S} & \textsc{Extension} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi', S, R) \in \mathscr{X} \vee \Psi' \rhd S \mathrel{\dot{\approx}} R} & \textsc{Symmetry} \end{array}}{\Psi \rhd P \mathrel{\dot{\approx}} Q}$$

Proof. Follows from the automatically generated rule for coinduction. □

As for strong simulation, we derive an introduction rule which allows us to ensure that the subjects of the simulating actions are sufficiently fresh.

Lemma 30.18.

$$\frac{\begin{array}{l} eqvt\ \mathscr{R} \qquad \widetilde{x} \sharp \Psi \qquad \widetilde{x} \sharp P \qquad \widetilde{x} \sharp Q \\ \bigwedge \Psi'\ \alpha\ Q'.\ \dfrac{\begin{pmatrix} \Psi \rhd Q \xrightarrow{\alpha} Q' & bn\,\alpha \sharp \Psi & bn\,\alpha \sharp P \\ bn\,\alpha \sharp Q & \alpha \neq \tau & bn\,\alpha \sharp subject\,\alpha \\ bn\,\alpha \sharp \mathscr{C} & \widetilde{x} \sharp \alpha & \widetilde{x} \sharp Q' \quad \widetilde{x} \sharp \Psi' \end{pmatrix}}{\exists P''.\ \Psi : Q \rhd P \overset{\alpha}{\Longrightarrow} P'' \wedge (\exists P'.\ \Psi \otimes \Psi' \rhd P'' \Longrightarrow P' \wedge (\Psi \otimes \Psi', P', Q') \in \mathscr{R})} \\ \bigwedge Q'.\ \dfrac{\Psi \rhd Q \xrightarrow{\tau} Q' \qquad \widetilde{x} \sharp Q'}{\exists P'.\ \Psi \rhd P \Longrightarrow P' \wedge (\Psi, P', Q') \in \mathscr{R}} \end{array}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\ \widehat{\rightsquigarrow}\text{-I2}$$

Proof. Similar to Lemma 27.11, but the introduction rule used is $\widehat{\rightsquigarrow}$-I. □

Note that this lemma also guarantees that the sequence $\widetilde{x}$ is fresh for the new assertion Ψ'.

30.3.2 Equivariance

When proving that weak simulation and weak static implication are equivariant, extra care has to be taken when reasoning about the assertion extending the environment of the last τ-chain.

Lemma 30.19. *Weak static implication is equivariant*

$$\frac{eqvt\ \mathscr{R} \qquad \Psi \rhd P \lesssim_{\mathscr{R}} Q}{p \cdot \Psi \rhd p \cdot P \lesssim_{\mathscr{R}} p \cdot Q}$$

Proof. We must prove that for all Ψ' there exists an R and an S such that

$$\begin{array}{l} p \cdot \Psi \rhd p \cdot Q \Longrightarrow R \\ (\mathscr{F}\ (p \cdot P)) \otimes (p \cdot \Psi) \leq (\mathscr{F}\ R) \otimes (p \cdot \Psi), \\ (p \cdot \Psi) \otimes \Psi' \rhd R \Longrightarrow S, \text{ and} \\ ((p \cdot \Psi) \otimes \Psi', p \cdot P, S) \in \mathscr{R}. \end{array}$$

From $\Psi \rhd P \lesssim_{\mathscr{R}} Q$ we obtain a Q' and a Q'' such that.

$$\begin{array}{l} \Psi \rhd Q \Longrightarrow Q' \\ (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi, \\ \Psi \otimes (p^{-} \cdot \Psi') \rhd Q' \Longrightarrow Q'', \text{ and} \\ (\Psi \otimes (p^{-} \cdot \Psi'), P, Q'') \in \mathscr{R}. \end{array}$$

- From $\Psi \rhd Q \Longrightarrow Q'$ we have that $p \cdot \Psi \rhd p \cdot Q \Longrightarrow p \cdot Q'$ by Lemma 30.4.
- Moreover from $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ R) \otimes \Psi$ we have that $(\mathscr{F}\ (p \cdot P)) \otimes (p \cdot \Psi) \leq (\mathscr{F}\ (p \cdot Q')) \otimes (p \cdot \Psi)$ by Lemma 27.6.
- Moreover from $\Psi \otimes (p^- \cdot \Psi') \rhd Q' \Longrightarrow Q''$ we have that $p \cdot \Psi \otimes (p^- \cdot \Psi') \rhd p \cdot Q' \Longrightarrow p \cdot Q''$ by Lemma 30.4, and hence that $(p \cdot \Psi) \otimes \Psi' \rhd p \cdot Q' \Longrightarrow p \cdot Q''$ by equivariance.
- Moreover from $(\Psi \otimes (p^- \cdot \Psi')\ ,\ P,\ Q'') \in \mathscr{R}$ and *eqvt* $\mathscr{R}$ we have that $(p \cdot \Psi \otimes (p^- \cdot \Psi')\ ,\ p \cdot P,\ p \cdot Q'') \in \mathscr{R}$ and hence that $((p \cdot \Psi) \otimes \Psi'\ ,\ p \cdot P,\ p \cdot Q'') \in \mathscr{R}$ by equivariance.
- Finally we can finish the proof by setting R to $p \cdot Q'$ and S to $p \cdot Q''$.

□

Lemma 30.20. *Simulation is equivariant*

$$\frac{\textit{eqvt}\ \mathscr{R} \qquad \Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}{p \cdot \Psi \rhd p \cdot P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} p \cdot Q}$$

Proof. The proof follows the same structure as Lemma 30.19. □

With these lemmas in place we can prove that weak bisimilarity is equivariant.

Lemma 30.21. *If* $\Psi \rhd P \mathrel{\dot{\approx}} Q$ *then* $p \cdot \Psi \rhd p \cdot P \mathrel{\dot{\approx}} p \cdot Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(p \cdot \Psi,\ p \cdot P,\ p \cdot Q) : \Psi \rhd P \mathrel{\dot{\approx}} Q\}$. The candidate relation $\mathscr{X}$ is equivariant.

Static implication: Lemma 30.19 proves that $p \cdot \Psi \rhd p \cdot P \mathrel{\dot{\lessapprox}}_{\mathscr{X}} p \cdot Q$, and hence $p \cdot \Psi \rhd p \cdot P \mathrel{\dot{\lessapprox}}_{\mathscr{X} \cup \dot{\approx}} p \cdot Q$ since $\mathrel{\dot{\lessapprox}}$ is monotonic.

Simulation: Lemma 30.20 proves that $p \cdot \Psi \rhd p \cdot P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X}} p \cdot Q$, and hence $p \cdot \Psi \rhd p \cdot P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} p \cdot Q$ since $\mathrel{\widehat{\rightsquigarrow}}$ is monotonic.

Extension: Follows by choosing the extended assertion to be $p^- \cdot \Psi'$ – the permutation then cancels out when p is applied.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as $\dot{\approx}$ is symmetric.

□

30.3.3 Preserved by static equivalence

Lemma 30.22. *Weak static implication and weak simulation are preserved by static equivalence*

$$\frac{\Psi \rhd P \lesssim_{\mathscr{R}} Q \qquad \Psi \simeq \Psi' \qquad \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', R, S) \in \mathscr{R}'}}{\Psi' \rhd P \lesssim_{\mathscr{R}'} Q}$$

$$\frac{\begin{array}{l}\Psi \ \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} \ Q \\ \mathit{eqvt}\ \mathscr{R}' \qquad \Psi \simeq \Psi' \qquad \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', R, S) \in \mathscr{R}'}\end{array}}{\Psi' \ \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} \ Q}$$

Proof. Follows from the definitions of $\mathrel{\widehat{\rightsquigarrow}}$ and $\lesssim$, where any bound names on the transitions avoid Ψ and Ψ'. The subgoals can then be proved using Lemmas 25.11, 30.8 □

Lemma 30.23.

$$\textit{If } \Psi \rhd P \mathrel{\dot{\approx}} Q \textit{ and } \Psi \simeq \Psi' \textit{ then } \Psi' \rhd P \mathrel{\dot{\approx}} Q\,.$$

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi', P, Q) : \exists \Psi.\ \Psi \rhd P \mathrel{\dot{\approx}} Q \wedge \Psi \simeq \Psi'\}$.

Static implication: Follows from Lemma 30.22

Simulation: Follows from Lemma 30.22

Extension: Follows from the axioms of static equivalence.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as $\mathrel{\dot{\approx}}$ is symmetric.

□

30.4 Weak bisimulation is an equivalence relation

Lemma 30.24. *Weak static implication and weak simulation are reflexive*

$$\frac{\{(\Psi, P, P) : \mathit{True}\} \subseteq \mathscr{R}}{\Psi \rhd P \lesssim_{\mathscr{R}} P} \qquad \frac{\{(\Psi, P, P) : \mathit{True}\} \subseteq \mathscr{R}}{\Psi \ \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} \ P}$$

Proof. By the definitions of $\mathrel{\widehat{\rightsquigarrow}}$ and $\lesssim$, where the τ-chains are set to be empty. □

Lemma 30.25.

$$\frac{\Psi \rhd Q \Longrightarrow Q' \qquad (\Psi, P, Q) \in \mathscr{R} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{\Psi' \rhd R \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} S}}{\exists P'.\ \Psi \rhd P \Longrightarrow P' \wedge (\Psi, P', Q') \in \mathscr{R}}$$

Proof. By induction on $\Psi \rhd Q \Longrightarrow Q'$

Base case ($Q = Q'$): Follows immediately by setting P' to P.

Inductive step ($\Psi \rhd Q \Longrightarrow Q'$ and $\Psi \rhd Q' \xrightarrow{\tau} Q''$):

- From the induction hypothesis we obtain a P' such that $\Psi \rhd P \Longrightarrow P'$ and $(\Psi, P', Q') \in \mathscr{R}$.
- From $(\Psi, P', Q') \in \mathscr{R}$ we have that $\Psi \rhd P' \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q'$ by the assumptions
- With $\Psi \rhd Q' \xrightarrow{\tau} Q''$ we obtain a P'' such that $\Psi \rhd P' \Longrightarrow P''$ and $(\Psi, P'', Q'') \in \mathscr{R}$.
- From $\Psi \rhd P \Longrightarrow P'$ and $\Psi \rhd P' \Longrightarrow P''$ we have that $\Psi \rhd P \Longrightarrow P''$.

□

Lemma 30.26.

$$\frac{(\Psi, P, Q) \in \mathscr{R} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{\Psi' \rhd R \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} S} \qquad \Psi : R \rhd Q \overset{\alpha}{\Longrightarrow} Q' \qquad bn\,\alpha \sharp \Psi \qquad bn\,\alpha \sharp P \qquad \alpha \neq \tau}{\exists P'' P'.\ \Psi : R \rhd P \overset{\alpha}{\Longrightarrow} P'' \wedge \Psi \otimes \Psi' \rhd P'' \Longrightarrow P' \wedge (\Psi \otimes \Psi', P', Q') \in \mathscr{R}}$$

Proof. From $(\Psi, P, Q) \in \mathscr{R}$ we obtain a Q'' such that $\Psi \rhd Q \Longrightarrow Q''$, $(\mathscr{F}\ R) \otimes \Psi \leq (\mathscr{F}\ Q'') \otimes \Psi$ and $\Psi \rhd Q'' \xrightarrow{\alpha} Q'$.

- From $\Psi \rhd Q \Longrightarrow Q''$ and the assumptions we obtain a P'' where $\Psi \rhd P \Longrightarrow P''$ and $(\Psi, P'', Q'') \in \mathscr{R}$ using Lemma 30.25.
- From $(\Psi, P'', Q'') \in \mathscr{R}$ we have that $\Psi \rhd P'' \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q''$ by the assumptions.
- With $\Psi \rhd Q'' \xrightarrow{\alpha} Q'$, $bn\,\alpha \sharp \Psi$, $bn\,\alpha \sharp P$, and $\alpha \neq \tau$ we obtain P''' and a P' where $\Psi : Q'' \rhd P'' \overset{\alpha}{\Longrightarrow} P'''$, $\Psi \otimes \Psi' \rhd P''' \Longrightarrow P'$, and $(\Psi \otimes \Psi', P', Q') \in \mathscr{R}$.
- From $\Psi : Q'' \rhd P'' \overset{\alpha}{\Longrightarrow} P'''$ we obtain a P'''' where $\Psi \rhd P'' \Longrightarrow P''''$, $(\mathscr{F}\ Q'') \otimes \Psi \leq (\mathscr{F}\ P'''') \otimes \Psi$, and $\Psi \rhd P'''' \xrightarrow{\alpha} P'''$ from Definition 30.10.
- From $\Psi \rhd P \Longrightarrow P''$ and $\Psi \rhd P'' \Longrightarrow P''''$ we have that $\Psi \rhd P \Longrightarrow P''''$.
- Moreover from $(\mathscr{F}\ R) \otimes \Psi \leq (\mathscr{F}\ Q'') \otimes \Psi$ and $(\mathscr{F}\ Q'') \otimes \Psi \leq (\mathscr{F}\ P'''') \otimes \Psi$ we have that $(\mathscr{F}\ R) \otimes \Psi \leq (\mathscr{F}\ P'''') \otimes \Psi$ since $\leq$ is transitive.

- Finally with $\Psi \rhd P'''' \xrightarrow{\alpha} P'''$ we have that $\Psi : R \rhd P \overset{\alpha}{\Longrightarrow} P'''$ by Definition 30.10.
- With $(\Psi \otimes \Psi', P', Q') \in \mathscr{R}$ we prove the goal.

□

Lemma 30.27. *Simulation is transitive*

$$\frac{\begin{array}{l}(\Psi, P, Q) \in \mathscr{R} \qquad \Psi \rhd Q \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}'} R \qquad eqvt\,\mathscr{R}'' \\ \{(\Psi, P, R) : \exists Q.\, (\Psi, P, Q) \in \mathscr{R} \wedge (\Psi, Q, R) \in \mathscr{R}'\} \subseteq \mathscr{R}'' \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{\Psi' \rhd R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} S}\end{array}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}''} R}$$

Proof. The introduction rule $\widehat{\rightsquigarrow}$-I is used such that the bound names of the transitions avoid Q. The weak transition case is then discharged using Lemmas 30.26 and 30.25, and the τ-chain case is discharged using Lemma 30.25. □

30.5 Equivalence correspondences

As with the weak bisimilarities for CCS and the pi-calculus, we prove that weak bisimilarity includes strong bisimilarity. This is useful in a variety of proofs, and establishes that weak bisimilarity preserves structural congruence.

Lemma 30.28. *Static implication implies weak static implication.*

$$\frac{(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q) \otimes \Psi \qquad \bigwedge \Psi'.\, (\Psi \otimes \Psi', P, Q) \in \mathscr{R}}{\Psi \rhd P \lesssim_{\mathscr{R}} Q}$$

Proof. For all Ψ' we have that

$$\begin{array}{l}\Psi \rhd Q \Longrightarrow Q, \\ (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q) \otimes \Psi, \\ \Psi \otimes \Psi' \rhd Q \Longrightarrow Q, and \\ (\Psi \otimes \Psi', P, Q) \in \mathscr{R}\end{array}$$

□

Lemma 30.29. *Strong simulation implies weak simulation.*

$$\frac{(\Psi, P, Q) \in \mathscr{R} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{(\mathscr{F}\ S) \otimes \Psi' \leq (\mathscr{F}\ R) \otimes \Psi'} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{\Psi' \rhd R \hookrightarrow_{\mathscr{R}} S} \qquad \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{R}}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}$$

Proof. Follows from the definitions of $\widehat{\rightsquigarrow}$ and $\hookrightarrow$. □

Lemma 30.30. *If* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *then* $\Psi \rhd P \mathrel{\dot{\approx}} Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\mathrel{\dot{\sim}}$

Static implication: Follows from $\mathrel{\dot{\sim}}$-E1, $\mathrel{\dot{\sim}}$-E3, the definition of $\simeq$ and Lemma 30.28.

Simulation: Follows from $\mathrel{\dot{\sim}}$-E1, $\mathrel{\dot{\sim}}$-E2, $\mathrel{\dot{\sim}}$-E3, the definition of $\simeq$, Lemma 30.29, and monotonicity of $\widehat{\rightsquigarrow}$.

Extension: Follows directly from $\mathrel{\dot{\sim}}$-E3.

Symmetry: Follows directly from $\mathrel{\dot{\sim}}$-E4.

□

We can now prove the theorem that structurally congruent agents are also weakly bisimilar.

Theorem 30.1. *If* $P \equiv Q$ *then* $\mathbf{1} \rhd P \mathrel{\dot{\approx}} Q$.

Proof. Follows directly from Theorem 28.3 and Lemma 30.30. □

30.6 Preservation properties

Weak bisimilarity is preserved by all operators except for Case and Input, for the same reason as the pi-calculus. In Section 22.2.3 we demonstrated how the Sum is encoded in psi-calculi, and hence the counterexample for the pi-calculus can be used to prove that weak bisimilarity is not preserved by Case.

The preservation proofs for weak bisimilarity are similar to their strong counterparts, with the exception that for simulation, there must be a requirement on the candidate relation that it is preserved by extension of arbitrary assertions. This requisite has up until now only been present for the bisimilarity proofs, but as assertions are added before the final τ-chain when mimicking a strong action, this requisite is also required for the simulation proofs.

30.6.1 Output

To prove that weak static implication and weak simulation is preserved by Output, the agents under the prefix must be in the candidate relation for all possible extensions of the environment.

Lemma 30.31. *Weak static implication is preserved by Output.*

$$\frac{\bigwedge \Psi'.\ (\Psi \otimes \Psi', \overline{M}N.P, \overline{M}N.Q) \in \mathscr{R}}{\Psi \rhd \overline{M}N.P \lessapprox_{\mathscr{R}} \overline{M}N.Q}$$

Proof. From the definition of $\lessapprox$ we have to find for all assertions Ψ', agents Q' and Q'' such that

$$\begin{aligned}
&\Psi \rhd \overline{M}N.Q \Longrightarrow Q',\\
&(\mathscr{F}\ (\overline{M}N.P)) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi,\\
&\Psi \otimes \Psi' \rhd Q' \Longrightarrow Q'', \text{ and}\\
&(\Psi \otimes \Psi', \overline{M}N.P, Q'') \in \mathscr{R}.
\end{aligned}$$

This follows directly from the assumptions, and by setting Q' and Q'' to $\overline{M}N.Q$. □

Lemma 30.32. *Weak simulation is preserved by Output.*

$$\frac{\bigwedge \Psi'.\ (\Psi \otimes \Psi', P, Q) \in \mathscr{R}}{\Psi \rhd \overline{M}N.P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} \overline{M}N.Q}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$, the OUTPUT inversion rule from Figure 26.2, and the weak OUTPUT semantic rule from Figure 30.2. □

Lemma 30.33. *If* $\Psi \rhd P \mathrel{\dot{\approx}} Q$ *then* $\Psi \rhd \overline{M}N.P \mathrel{\dot{\approx}} \overline{M}N.Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, \overline{M}N.P, \overline{M}N.Q) : \Psi \rhd P \mathrel{\dot{\approx}} Q\}$.

Static implication: Follows from Lemma 30.31.

Simulation: Follows from Lemma 30.32.

Extension: Follows directly from $\dot{\approx}$-E3.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric since weak bisimilarity is symmetric.

□

30.6.2 Restriction

Weak static implication and weak simulation introduce universally quantified assertions for the trailing τ-chains. Bound names are required to be fresh for these extended environments, and must hence be manually alpha-converted.

Lemma 30.34. *Weak static implication is preserved by restriction.*

$$\frac{\Psi \rhd P \lessapprox_{\mathscr{R}} Q \qquad \mathit{eqvt}\ \mathscr{R} \qquad x \sharp \Psi \qquad \bigwedge \Psi' R S y.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad y \sharp \Psi'}{(\Psi', (\nu y)R, (\nu y)S) \in \mathscr{R}'}}{\Psi \rhd (\nu x)P \lessapprox_{\mathscr{R}'} (\nu x)Q}$$

Proof. From the definition of $\lessapprox$ we have to find for all assertions Ψ', agents R and S such that

$$\begin{aligned}
&\Psi \rhd (\nu x)Q \Longrightarrow R,\\
&(\mathscr{F}\ ((\nu x)P)) \otimes \Psi \leq (\mathscr{F}\ R) \otimes \Psi,\\
&\Psi \otimes \Psi' \rhd R \Longrightarrow S, \text{ and}\\
&(\Psi \otimes \Psi', (\nu x)P, S) \in \mathscr{R}.
\end{aligned}$$

- We obtain a name y such that y is sufficiently fresh.
- From $\Psi \rhd P \lessapprox_{\mathscr{R}} Q$ and $\mathit{eqvt}\ \mathscr{R}$ we have that

  $$(x\ y) \cdot \Psi \rhd (x\ y) \cdot P \lessapprox_{\mathscr{R}} (x\ y) \cdot Q$$

  by Lemma 30.19, and hence that

  $$\Psi \rhd (x\ y) \cdot P \lessapprox_{\mathscr{R}} (x\ y) \cdot Q$$

  since $x \sharp \Psi$ and $y \sharp \Psi$.
- From the definition of $\lessapprox$ we obtain a Q' and a Q'' such that

  $$\begin{aligned}
  &\Psi \rhd (x\ y) \cdot Q \Longrightarrow Q',\\
  &(\mathscr{F}\ (x\ y) \cdot P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi,\\
  &\Psi \otimes \Psi' \rhd Q' \Longrightarrow Q'', \text{ and}\\
  &(\Psi \otimes \Psi', (x\ y) \cdot P, Q'') \in \mathscr{R}.
  \end{aligned}$$

- From $\Psi \rhd (x\ y) \cdot Q \Longrightarrow Q'$ and $y \sharp \Psi$ we have that

  $$\Psi \rhd (\nu y)(x\ y) \cdot Q \Longrightarrow (\nu y)Q'$$

  by the SCOPE law from Figure 30.1, and hence that

  $$\Psi \rhd (\nu x)Q \Longrightarrow (\nu y)Q'$$

by alpha-conversion.

- Moreover, from $(\mathscr{F}\ (x\,y)\cdot P)\otimes\Psi \leq (\mathscr{F}\ Q')\otimes\Psi$ we have that

$$((\nu y)(\mathscr{F}\ (x\,y)\cdot P)\otimes\Psi) \leq ((\nu y)(\mathscr{F}\ Q')\otimes\Psi)$$

by Lemma 27.29, and hence that

$$(\mathscr{F}\ ((\nu x)P))\otimes\Psi \leq (\mathscr{F}\ ((\nu y)Q'))\otimes\Psi$$

since $y\sharp\Psi$ and by alpha-conversion.

- Moreover from $\Psi\otimes\Psi'\ \rhd\ Q' \Longrightarrow Q''$ we have that

$$\Psi\otimes\Psi'\ \rhd\ (\nu y)Q' \Longrightarrow (\nu y)Q'',$$

by the SCOPE rule from Figure 30.1, and that $y\sharp\Psi$ and $y\sharp\Psi'$.

- Moreover from $(\Psi\otimes\Psi', (x\,y)\cdot P, Q')\in\mathscr{R}$ we have that

$$(\Psi\otimes\Psi', (\nu y)(x\,y)\cdot P, (\nu y)Q')\in\mathscr{R}$$

by the assumptions and that $y\sharp\Psi$ and $y\sharp\Psi'$, and hence that

$$(\Psi\otimes\Psi', (\nu x)P, (\nu y)Q')\in\mathscr{R}$$

by alpha-conversion.

- Finally, we prove the lemma by setting R to $(\nu y)Q'$ and S to $(\nu y)Q''$

□

Lemma 30.35. *Weak simulation is preserved by restriction.*

$$\frac{\Psi\ \rhd\ P\ \widehat{\rightsquigarrow}_{\mathscr{R}}\ Q \qquad eqvt\ \mathscr{R}' \qquad x\sharp\Psi \qquad \mathscr{R}\subseteq\mathscr{R}' \qquad \bigwedge\Psi' R S y.\ \dfrac{(\Psi', R, S)\in\mathscr{R} \qquad y\sharp\Psi'}{(\Psi', (\nu y)R, (\nu y)S)\in\mathscr{R}'}}{\Psi\ \rhd\ (\nu x)P\ \widehat{\rightsquigarrow}_{\mathscr{R}'}\ (\nu x)Q}$$

Proof. Since $x\sharp\Psi$, $x\sharp(\nu x)P$, and $x\sharp(\nu x)Q$, Lemma 30.18 can be used to ensure that x is fresh for any action or derivative of $(\nu x)Q$.

The SCOPE inversion rule is then used to generate two possible transitions, which are discharged by the SCOPE and OPEN rules from Figure 30.2 respectively. The trailing τ-chains are discharged using the SCOPE rule from Figure 30.1 when applicable. □

We also require a corresponding lemma for binding sequences.

Lemma 30.36.

$$\frac{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad \mathit{eqvt}\ \mathscr{R} \qquad \tilde{x} \sharp \Psi \qquad \bigwedge \Psi' R S \tilde{y}.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \tilde{y} \sharp \Psi'}{(\Psi', (\nu\tilde{y})R, (\nu\tilde{y})S) \in \mathscr{R}}}{\Psi \rhd (\nu\tilde{x})P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} (\nu\tilde{x})Q}$$

Proof. By induction on $\mathscr{R}$ and Lemma 30.35. □

Lemma 30.37. *If* $\Psi \rhd P \mathrel{\dot{\approx}} Q$ *and* $x \sharp \Psi$ *then* $\Psi \rhd (\nu x)P \mathrel{\dot{\approx}} (\nu x)Q$.

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, (\nu x)P, (\nu x)Q) : \Psi \rhd P \mathrel{\dot{\approx}} Q \wedge x \sharp \Psi\}$

Static implication: Follows from $\dot{\approx}$-E1 and Lemma 30.34.

Simulation: Follows from $\dot{\approx}$-E2 and Lemma 30.35.

Extension: Given $\Psi \rhd P \mathrel{\dot{\approx}} Q$, we must prove that $(\Psi \otimes \Psi', (\nu x)P, (\nu x)Q) \in \mathscr{X}$ for all possible Ψ', including those containing names that clash with x.

- A fresh name y is chosen such that $y \sharp \Psi$, $y \sharp \Psi'$, $y \sharp P$, and $y \sharp Q$.
- From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have that $\Psi \otimes (x\,y) \cdot \Psi' \rhd P \mathrel{\dot{\approx}} Q$ by $\dot{\approx}$-E3.
- Since weak bisimilarity is equivariant we have that $(x\,y) \cdot \Psi \otimes (x\,y) \cdot \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\approx}} (x\,y) \cdot Q$.
- Since $x \sharp \Psi$ and $y \sharp \Psi$ we have that $\Psi \otimes \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\approx}} (x\,y) \cdot Q$.
- Finally, since $y \sharp \Psi$ and $y \sharp \Psi'$, we can derive that $(\Psi \otimes \Psi', (\nu y)(x\,y) \cdot P, (\nu y)(x\,y) \cdot Q) \in \mathscr{X}$, and since $y \sharp P$ and $y \sharp Q$ we have by alpha-conversion that $(\Psi \otimes \Psi', (\nu x)P, (\nu x)Q) \in \mathscr{X}$.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as weak bisimilarity is symmetric.

□

30.6.3 Parallel

The proof that weak bisimilarity is preserved by Parallel is more involved than the corresponding proof for strong bisimilarity. The main reason for this is that the frames of two weakly bisimilar agents are not required to be statically equivalent, but for each pair of bisimilar agents, there must exist a τ-chain from one of the agents to an agent such that the frame of the other agent statically implies the one at the end of the τ-chain.

Lemma 30.38. *Weak static implication is preserved by Parallel.*

$$\bigwedge \widetilde{b_R}\ \Psi_R.\ \frac{\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q}{\Psi \otimes \Psi_R \rhd P \lesssim_{\mathscr{R}} Q}$$

$\widetilde{x} \sharp \Psi \qquad eqvt\ \mathscr{R}$

$$\bigwedge \Psi'\ S\ T\ \widetilde{b_U}\ \Psi_U\ U.\ \frac{(\Psi' \otimes \Psi_U, S, T) \in \mathscr{R} \quad \mathscr{F}\ U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T}{(\Psi', S \mid U, T \mid U) \in \mathscr{R}'}$$

$$\bigwedge \Psi'\ S\ T\ \widetilde{y}.\ \frac{(\Psi', S, T) \in \mathscr{R}' \qquad \widetilde{y} \sharp \Psi'}{(\Psi', (\nu \widetilde{y})S, (\nu \widetilde{y})T) \in \mathscr{R}'}$$

$$\bigwedge \Psi'\ S\ T\ \Psi''.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', S, T) \in \mathscr{R}}$$

$$\Psi \rhd (\nu \widetilde{x})(P \mid R) \lesssim_{\mathscr{R}'} (\nu \widetilde{x})(Q \mid R)$$

Proof. Follows from the definition of $\lesssim$. The binding sequence $\widetilde{x}$ must be alpha-converted to avoid the new assertion Ψ'. □

Lemma 30.39. *Weak simulation is preserved by Parallel.*

$eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}'$

$$1 : \bigwedge \widetilde{b_R}\ \Psi_R.\ \frac{\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q}{(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}}$$

$$2 : \bigwedge \Psi'\ S\ T\ \widetilde{b_U}\ \Psi_U\ U.\ \frac{(\Psi' \otimes \Psi_U, S, T) \in \mathscr{R} \quad \mathscr{F}\ U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T}{(\Psi', S \mid U, T \mid U) \in \mathscr{R}'}$$

$$3 : \bigwedge \Psi'\ S\ T.\ \frac{(\Psi', S, T) \in \mathscr{R}}{\Psi' \rhd S \lesssim_{\mathscr{R}} T} \qquad 4 : \bigwedge \Psi'\ S\ T.\ \frac{(\Psi', S, T) \in \mathscr{R}}{\Psi' \rhd S\ \widehat{\rightsquigarrow}_{\mathscr{R}}\ T}$$

$$5 : \bigwedge \Psi'\ S\ T.\ \frac{(\Psi', S, T) \in \mathscr{R}}{(\Psi', T, S) \in \mathscr{R}} \qquad 6 : \bigwedge \Psi'\ S\ T\ \Psi' a.\ \frac{(\Psi' a, S, T) \in \mathscr{R}}{(\Psi' a \otimes \Psi'', S, T) \in \mathscr{R}}$$

$$7 : \bigwedge \Psi'\ S\ T\ \widetilde{x}.\ \frac{(\Psi', S, T) \in \mathscr{R}' \qquad \widetilde{x} \sharp \Psi'}{(\Psi', (\nu \widetilde{x})S, (\nu \widetilde{x})T) \in \mathscr{R}'}$$

$$8 : \bigwedge \Psi'\ S\ T\ \Psi''.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', S, T) \in \mathscr{R}}$$

$$\Psi \rhd P \mid R\ \widehat{\rightsquigarrow}_{\mathscr{R}'}\ Q \mid R$$

The premises for this lemma are nearly identical to the ones for strong bisimilarity, Lemma 27.38. Premise 1 states that $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$ *for all sufficiently fresh frames* $(\nu \widetilde{b_R})\Psi_R$ *of* R*. If this were not the case, it would not be possible to alpha-convert the frame of* R *without falling outside the relation* $\mathscr{R}$*. Premise 2 states the preservation property of* $\mathscr{R}'$*, again, the frame*

of the parallel agent must be sufficiently fresh. Premises 3-6 are properties of weak bisimilarity. Premise 7 states that the relation $\mathscr{R}'$ must be preserved by Restriction. Premise 8 states that it must be possible to switch the assertion component of the elements of $\mathscr{R}$ for statically equivalent ones.

Proof. From $\widehat{\rightsquigarrow}$-I we obtain two goals which need to be proven – one where $P \mid R$ does a τ-action, and one for the other actions.

For the first case we have to prove that for all α and T' where $\Psi \rhd Q \mid R \xrightarrow{\alpha} T'$ and $\alpha \neq \tau$, then for all Ψ', there must exist an S'' and an S' such that $\Psi : Q \mid R \rhd P \mid R \stackrel{\alpha}{\Longrightarrow} S''$, $\Psi \otimes \Psi' \rhd S'' \stackrel{\tau}{\Longrightarrow} S'$ and $(\Psi \otimes \Psi', S', T') \in \mathscr{R}$. We apply the PAR-inversion rule on the transition $\Psi \rhd Q \mid R \xrightarrow{\alpha} T'$ and ensure that any bound names avoid Ψ, Ψ', P, Q, and R, and get the following two cases.

PAR**1** ($\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$ **and** $T' = Q' \mid R$)**:**

Moreover we know that $\mathscr{F} R = (\nu \widetilde{b_R})\Psi_R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, and $\widetilde{b_R} \sharp Q$.

- From $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$ we have from the assumptions that $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$, and hence that $\Psi \otimes \Psi_R \rhd P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}}} Q$.
- With $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, $bn\,\alpha \sharp \Psi$, $bn\,\alpha \sharp P$, and $\alpha \neq \tau$ we obtain a P'' and a P' such that $\Psi \otimes \Psi_R : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P''$, $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$, and $((\Psi \otimes \Psi_R) \otimes \Psi', P', Q') \in \mathscr{R}$.
- From $\Psi \otimes \Psi_R : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P''$, $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$, $\widetilde{b_R} \sharp P$, and $\widetilde{b_R} \sharp Q$ we have that $\widetilde{b_R} \sharp P'$, $\widetilde{b_R} \sharp P''$ and $\widetilde{b_R} \sharp Q'$ by Lemmas 25.25 and 30.6
- From $\Psi \otimes \Psi_R : Q \rhd P \stackrel{\alpha}{\Longrightarrow} P''$, $bn\,\alpha \sharp R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$, and $\widetilde{b_R} \sharp \alpha$ we have that $\Psi : Q \mid R \rhd P \mid R \stackrel{\alpha}{\Longrightarrow} P'' \mid R$ by PAR1.
- Moreover from $(\Psi \otimes \Psi_R) \otimes \Psi' \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ we have that $(\Psi \otimes \Psi') \otimes \Psi_R \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ by Lemma 30.8 and the static equivalence laws for commutativity and associativity of $\simeq$.
- hence $\Psi \otimes \Psi' \rhd P'' \mid R \stackrel{\tau}{\Longrightarrow} P' \mid R$ by the PAR1 rule from Figure 30.1.
- Moreover from $((\Psi \otimes \Psi_R) \otimes \Psi', P', Q') \in \mathscr{R}$ we have that $((\Psi \otimes \Psi') \otimes \Psi_R, P', Q') \in \mathscr{R}$ since the assumptions allow us to switch static equivalent assertions in $\mathscr{R}$, and static equivalence laws for commutativity and associativity of $\simeq$.
- Since $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp \Psi'$, $\widetilde{b_R} \sharp P'$, and $\widetilde{b_R} \sharp Q'$ we have by the assumptions that $(\Psi, P' \mid R, Q' \mid R) \in \mathscr{R}'$.
- Finally we prove the goal by instantiating S'' to $P'' \mid R$, and S' to $P' \mid R$.

PAR**2** ($\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ **and** $T' = Q \mid R'$)**:**

Moreover we know that $\mathscr{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q$, $\widetilde{b_Q} \sharp \Psi$, $\widetilde{b_Q} \sharp P$, and $\widetilde{b_Q} \sharp Q$.

- We pick frames for P and R such that $\mathscr{F}\ P = (\nu \widetilde{b_P})\Psi_P$, $\mathscr{F}\ R = (\nu \widetilde{b_R})\Psi_R$ such that $\widetilde{b_P}$ and $\widetilde{b_R}$ are fresh for everything in the proof context.
- From $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ and derived freshness conditions we obtain a p, a Ψ'', a $\widetilde{b_{R'}}$, and a ${\Psi_R}'$ such that $\mathit{set}\ p \subseteq \mathit{set}\ (\mathit{bn}\ \alpha) \times \mathit{set}\ (\mathit{bn}\ (p \cdot \alpha))$, $(p \cdot \Psi_R) \otimes \Psi'' \simeq {\Psi_R}'$, $\mathscr{F}\ R' = (\nu \widetilde{b_{R'}}){\Psi_R}'$, and that $\mathit{bn}\ (p \cdot \alpha)$ and $\widetilde{b_{R'}}$ are sufficiently fresh, using Lemma 27.37.
- From $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$ we have from the assumptions that $(\Psi \otimes \Psi_R, P, Q) \in \mathscr{R}$, and thus we obtain a P' and a P'' such that

$$\begin{aligned}
&\Psi \otimes \Psi_R \rhd P \Longrightarrow P',\\
&(\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R) \leq (\mathscr{F}\ P') \otimes (\Psi \otimes \Psi_R),\\
&(\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')) \rhd P' \Longrightarrow P'', \text{ and}\\
&((\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')), P'', Q) \in \mathscr{R}.
\end{aligned}$$

- We pick a frame for P' such that $\mathscr{F}\ P' = (\nu \widetilde{b_{P'}}){\Psi_P}'$, and $\widetilde{b_{P'}}$ is fresh for everything in the proof context.
- From $\Psi \otimes \Psi_R \rhd P \Longrightarrow P'$, $\widetilde{b_R} \sharp \Psi$, and $\widetilde{b_R} \sharp P$ we have that $\Psi \rhd P \mid R \Longrightarrow P' \mid R$ by the PAR1 rule from Figure 30.1.
- Moreover from

$$(\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R) \leq (\mathscr{F}\ P') \otimes (\Psi \otimes \Psi_R)$$

  and the derived freshness conditions, we have that

$$(\mathscr{F}\ (Q \mid R)) \otimes \Psi \leq (\mathscr{F}\ (P' \mid R)) \otimes \Psi.$$

- Moreover from $(\mathscr{F}\ Q) \otimes (\Psi \otimes \Psi_R) \leq (\mathscr{F}\ P') \otimes (\Psi \otimes \Psi_R)$, the definitions of the frames of P' and Q, and the laws of static equivalence we have that $((\nu \widetilde{b_Q})(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu \widetilde{b_P})(\Psi \otimes {\Psi_P}') \otimes \Psi_R)$.
- With $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$ we have that $\Psi \otimes {\Psi_P}' \rhd R \xrightarrow{\alpha} R'$ using Lemma 27.35.
- With $\mathit{bn}\ \alpha \sharp P'$, $\widetilde{b_P} \sharp \Psi$, $\widetilde{b_P} \sharp R$, and $\widetilde{b_P} \sharp \alpha$ we have that $\Psi \rhd P' \mid R \xrightarrow{\alpha} P' \mid R'$ by PAR2.
- Finally we have that $\Psi : Q \mid R \rhd P \mid R \stackrel{\alpha}{\Longrightarrow} P' \mid R'$ by Definition 30.10.
- Moreover from

$$(\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')) \rhd P' \Longrightarrow P''$$

we have by equivariance that

$$p \cdot (\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')) \ \rhd\ p \cdot P' \Longrightarrow p \cdot P'',$$

and by equivariance and the freshness conditions that

$$(\Psi \otimes (p \cdot \Psi_R)) \otimes (\Psi'' \otimes \Psi') \ \rhd\ P' \Longrightarrow P''.$$

- With $(p \cdot \Psi_R) \otimes \Psi'' \simeq {\Psi_R}'$ and the laws of static equivalence we have that
$$(\Psi \otimes \Psi') \otimes {\Psi_R}' \ \rhd\ P' \Longrightarrow P''$$
using Lemma 30.8.
- With $\widetilde{b_{R'}} \sharp \Psi$, $\widetilde{b_{R'}} \sharp \Psi'$, and $\widetilde{b_{R'}} \sharp P'$ we have that
$$\Psi \otimes \Psi' \ \rhd\ P' \mid R' \Longrightarrow P'' \mid R'$$
using the PAR1-rule from Figure 30.1.
- Moreover from
$$((\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')), P'', Q) \in \mathscr{R}$$
we have that
$$p \cdot ((\Psi \otimes \Psi_R) \otimes ((p \cdot \Psi'') \otimes (p \cdot \Psi')), p \cdot P'', p \cdot Q) \in \mathscr{R}$$
since $\mathscr{R}$ is equivariant, and hence
$$((\Psi \otimes (p \cdot \Psi_R)) \otimes (\Psi'' \otimes \Psi'), P'', Q) \in \mathscr{R}$$
by equivariace and the freshness conditions.
- With the assumptions and $(p \cdot \Psi_R) \otimes \Psi'' \simeq {\Psi_R}'$ we have that
$$((\Psi \otimes \Psi') \otimes {\Psi_R}', P'', Q) \in \mathscr{R}$$
using the static equivalence laws.
- With $\widetilde{b_{R'}} \sharp \Psi$, $\widetilde{b_{R'}} \sharp \Psi'$, $\widetilde{b_{R'}} \sharp P''$, and $\widetilde{b_{R'}} \sharp Q$ we have that
$$(\Psi \otimes \Psi', P'' \mid R', Q \mid R') \in \mathscr{R}'$$
using the assumptions
- Finally we can prove the case by setting S'' to $P' \mid R'$ and S' to $P'' \mid R'$.

We now have to prove the case where $Q \mid R$ does a τ-action i.e. for all T' where $\Psi \rhd Q \mid R \xrightarrow{\tau} T'$ then there must exist an S' such that $\Psi \ \rhd\ P \mid R \stackrel{\tau}{\Longrightarrow} S'$, and $(\Psi, S', T') \in \mathscr{R}$. We apply the PAR-inversion rule on the transition $\Psi \rhd Q \mid R \xrightarrow{\tau} T'$ and ensure that any bound names avoid Ψ, P, Q, and

R. We get four cases, where the PAR1 and PAR2 are very similar to the ones we have already proven for the non τ-actions, and hence we focus on the communication case.

COMM**1** ($\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q', \Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\tilde{x})N} Q'$, **and** $T' = (\nu\tilde{x})(Q' \mid R')$**):**

Moreover we know that $\mathscr{F}\ Q = (\nu\widetilde{b_Q})\Psi_Q$, $\widetilde{b_Q} \sharp \Psi$, $\widetilde{b_Q} \sharp P$, $\widetilde{b_Q} \sharp Q$, $\mathscr{F}\ R = (\nu\widetilde{b_R})\Psi_R$, $\widetilde{b_R} \sharp \Psi$, $\widetilde{b_R} \sharp P$, $\widetilde{b_R} \sharp Q$, and that $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$.

One crucial observation is that we cannot guarantee that $\widetilde{b_R} \sharp M$, and this complicates the proof considerably.

- We pick a frame for P such that $\mathscr{F}\ P = (\nu\widetilde{b_P})\Psi_P$ and that $\widetilde{b_P}$ is fresh for everything in the proof context.
- From $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\tilde{x})N} R'$ and derived freshness conditions we obtain a p, a Ψ', a $\widetilde{b_{R'}}$, and a ${\Psi_R}'$ such that $set\ p \subseteq set\ \tilde{x} \times set\ (p \cdot \tilde{x})$, $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$, $\mathscr{F}\ R' = (\nu\widetilde{b_{R'}}){\Psi_R}'$, and that $p \cdot \tilde{x}$ and $\widetilde{b_{R'}}$ are sufficiently fresh, using Lemma 27.37. Note that since we do not have that $\widetilde{b_R} \sharp M$ we also do not have that $(p \cdot \tilde{x}) \sharp M$.
- From $\Psi \otimes \Psi_R \rhd Q \xrightarrow{MN} Q'$, $\tilde{x} \sharp Q$, and $(p \cdot \tilde{x}) \sharp Q$ we have that $p \cdot \Psi \otimes \Psi_R \rhd Q \xrightarrow{(p \cdot M)N} Q'$ using Lemma 25.15, and hence $\Psi \otimes (p \cdot \Psi_R) \rhd Q \xrightarrow{(p \cdot M)N} Q'$ by equivariance and that $\tilde{x} \sharp \Psi$ and $(p \cdot \tilde{x}) \sharp \Psi$.
- From $\mathscr{F}\ R = (\nu\widetilde{b_R})\Psi_R$, $\tilde{x} \sharp R$, and $(p \cdot \tilde{x}) \sharp R$ we have that $\mathscr{F}\ R = (\nu(p \cdot \widetilde{b_R}))(p \cdot \Psi_R)$
- The freshness conditions are extended such that for all contexts $\mathscr{C}$ such that $\widetilde{b_R} \sharp \mathscr{C}$, $\tilde{x} \sharp \mathscr{C}$, and $(p \cdot \tilde{x}) \sharp \mathscr{C}$ we have that $(p \cdot \widetilde{b_R}) \sharp \mathscr{C}$.
- Since $(p \cdot \widetilde{b_R}) \sharp \Psi$, $(p \cdot \widetilde{b_R}) \sharp P$, $(p \cdot \widetilde{b_R}) \sharp Q$ we have from the assumptions that $(\Psi \otimes (p \cdot \Psi_R), P, Q) \in \mathscr{R}$, and hence that $\Psi \otimes (p \cdot \Psi_R) \rhd P \hookrightarrow_{\mathscr{R}} Q$.
- With $\Psi \otimes (p \cdot \Psi_R) \rhd Q \xrightarrow{(p \cdot M)N} Q'$ we obtain a P'' and a P' such that $\Psi \otimes (p \cdot \Psi_R) : Q \rhd P \xRightarrow{(p \cdot M)N} P''$, $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd P'' \Longrightarrow P'$ and $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in \mathscr{R}$.
- From $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$ we have that $(p \cdot \Psi \otimes (\Psi_Q \otimes \Psi_R)) \vdash (p \cdot M) \leftrightarrows (p \cdot K)$, and hence by the freshness conditions that $\Psi \otimes (\Psi_Q \otimes (p \cdot \Psi_R)) \vdash (p \cdot M) \leftrightarrows K$.
- With $\Psi \otimes (p \cdot \Psi_R) : Q \rhd P \xRightarrow{(p \cdot M)N} P''$, $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\tilde{x})N} R'$ and the freshness conditions we have that $\Psi \rhd P \mid R \Longrightarrow (\nu\tilde{x})(P'' \mid R')$ using the COMM1 rule from Figure 30.2.
- Moreover from $(\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi' \rhd P'' \Longrightarrow P'$ and $(p \cdot \Psi_R) \otimes \Psi' \simeq {\Psi_R}'$ we have that $\Psi \otimes {\Psi_R}' \rhd P'' \Longrightarrow P'$, by Lemma 30.8.

- With $\widetilde{b_{R'}} \sharp \Psi$ and $\widetilde{b_{R'}} \sharp P''$ we have that $\Psi \rhd P'' \mid R' \Longrightarrow P' \mid R'$, and since $\tilde{x} \sharp \Psi$ that $\Psi \rhd (\nu\tilde{x})(P'' \mid R') \Longrightarrow (\nu\tilde{x})(P' \mid R')$ by the PAR1 and SCOPE rules from Figure 30.1.
- Moreover, from $((\Psi \otimes (p \cdot \Psi_R)) \otimes \Psi', P', Q') \in \mathscr{R}$ and the assumptions we have that $(\Psi \otimes \Psi_R{}', P', Q') \in \mathscr{R}$ since $(p \cdot \Psi_R) \otimes \Psi' \simeq \Psi_R{}'$, and the static equivalence laws.
- With $\widetilde{b_{R'}} \sharp \Psi$, $\widetilde{b_{R'}} \sharp P'$, and $\widetilde{b_{R'}} \sharp Q'$ we have by the assumptions that $(\Psi, P' \mid R', Q' \mid R') \in \mathscr{R}'$ and hence that $(\Psi, (\nu\tilde{x})(P' \mid R'), (\nu\tilde{x})(Q' \mid R')) \in \mathscr{R}'$ since $\tilde{x} \sharp \Psi$.
- Finally we prove the goal by instantiating S'' to $(\nu\tilde{x})(P'' \mid R')$ and S' to $(\nu\tilde{x})(P' \mid R')$.

COMM**2:** Symmetric version of COMM1, but considerably simpler as the permutation p simplifies away completely since input actions have no bound names.

□

As with strong bisimilarity, we first prove that weak bisimilarity is preserved by Parallel for agents with sufficiently fresh frames.

Lemma 30.40.

$$\frac{\Psi \otimes \Psi_R \rhd P \dot{\approx} Q \qquad \mathscr{F}\, R = (\nu\widetilde{b_R})\Psi_R \qquad \widetilde{b_R} \sharp \Psi \qquad \widetilde{b_R} \sharp P \qquad \widetilde{b_R} \sharp Q}{\Psi \rhd P \mid R \dot{\approx} Q \mid R}$$

Proof. By coinduction with $\mathscr{X}$ set to

$$\begin{aligned}\{(\Psi, (\nu\tilde{x})(P \mid R), (\nu\tilde{x})(Q \mid R)) : \tilde{x} \sharp \Psi \wedge \\ \forall \widetilde{b_R}\, \Psi_R.\, \mathscr{F}\, R = (\nu\widetilde{b_R})\Psi_R \wedge \widetilde{b_R} \sharp \Psi \wedge \widetilde{b_R} \sharp P \wedge \widetilde{b_R} \sharp Q \longrightarrow \\ \Psi \otimes \Psi_R \rhd P \dot{\approx} Q\}.\end{aligned}$$

Static implication: Follows from Lemma 30.38.

Simulation: Given that $\Psi \otimes \Psi_R \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X}} Q$ for all possible frames of R such that $\widetilde{b_R}$ is fresh for Ψ, P, and Q, we must prove that $\Psi \rhd (\nu\tilde{x})(P \mid R) \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} (\nu\tilde{x})(Q \mid R)$, where $\tilde{x} \sharp \Psi$

- Using Lemma 30.39 we prove that $\Psi \rhd P \mid R \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{X} \cup \dot{\approx}} Q \mid R$.
- Moreover we have that for all Ψ, P, Q, and x such that $x \sharp \Psi$, $(\Psi, P, Q) \in \mathscr{X} \cup \dot{\sim}$ implies $(\Psi, (\nu x)P, (\nu x)Q) \in \mathscr{X} \cup \dot{\sim}$, by the definition of $\mathscr{X}$ and Lemma 30.35.
- Finally we have that $\Psi \rhd (\nu\tilde{x})(P \mid R) \hookrightarrow_{\mathscr{X} \cup \dot{\approx}} (\nu\tilde{x})(Q \mid R)$ using Lemma 30.36.

Extension: Follows from the definition of $\mathscr{X}$. The binding sequence $\tilde{x}$ is alpha-converted to avoid the extended assertion Ψ'.

Symmetry: The candidate relation $\mathscr{X}$ is symmetric as weak bisimilarity is symmetric.

□

We can now prove that weak bisimilarity is preserved by Parallel.

Lemma 30.41. *If* $\Psi \rhd P \mathrel{\dot{\approx}} Q$ *then* $\Psi \rhd P \mid R \mathrel{\dot{\approx}} Q \mid R$.

Proof. Follows from Lemma 30.40 by choosing a sufficiently fresh frame for R. □

30.6.4 Replication

The proof that weak bisimilarity is preserved by Replication follows a similar pattern to the corresponding proof for strong bisimilarity. The bisimiluation up-to technique required for the proof is bisimulation up-to weak bisimilarity, in a similar manner as is done for CCS or the pi-calculus.

We define the function *weakBisimCompose* which takes a relation as an argument and composes that relation with weak and strong bisimilarity.

Definition 30.42 (*weakBisimCompose*)**.**

$$weakBisimCompose\ X \stackrel{\text{def}}{=} \{(\Psi, P, Q) : \exists P'\, Q'.\ \Psi \rhd P \mathrel{\dot{\approx}} P' \wedge (\Psi, P', Q') \in X \cup \mathrel{\dot{\approx}} \wedge \Psi \rhd Q' \mathrel{\dot{\sim}} Q\}$$

Lemma 30.43. *Bisimulation up-to weak bisimilarity.*

$$\frac{\begin{array}{l}(\Psi, P, Q) \in \mathscr{Y} \\ eqvt\ \mathscr{Y} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{Y}}{\Psi' \rhd R \mathrel{\dot{\lesssim}}_{\mathscr{Y} \cup \mathrel{\dot{\approx}}} S} \quad \textsc{StatImp} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{Y}}{\Psi' \rhd R \mathrel{\dot{\rightsquigarrow}}_{weakBisimCompose\,\mathscr{Y}} S} \quad \textsc{Simulation} \\ \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{Y}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{Y} \vee \Psi' \otimes \Psi'' \rhd R \mathrel{\dot{\approx}} S} \quad \textsc{Extension} \\ \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{Y}}{(\Psi', S, R) \in \mathscr{Y}} \quad \textsc{Symmetry}\end{array}}{\Psi \rhd P \mathrel{\dot{\approx}} Q}$$

Proof. By coinduction with $\mathscr{X}$ set to $\{(\Psi, P, Q) : \exists P'\, Q'.\ \Psi \rhd P \dot{\approx} P' \wedge (\Psi, P', Q') \in \mathscr{Y} \cup \dot{\approx} \wedge \Psi \rhd Q' \dot{\approx} Q\}$. □

The proof that weak simulations are preserved by Replication follows the same structure as the one for strong simulations. We have the agents P and Q which are weakly bisimilar, and we must prove that $!P$ and $!Q$ are weakly bisimilar. Every derivative P' of $!P$ must have $!P$ as a parallel component, and hence P' must be structurally congruent to an agent $T \mid !P$ for some T. The agent $!Q$ can then mimic any action of $!Q$ to a derivative Q', which is structurally congruent to an agent $S \mid !Q$ for some S where T is weakly bisimilar to S; the simulation lemma hence needs three relations – weak bisimilarity for T and S, strong bisimilarity for applying the structural congruence laws and extracting the replicated component of the derivatives, and the candidate relation for the simulation proof. The corresponding lemma for strong simulations only required two simulations, which makes this lemma slightly more involved.

Lemma 30.44. *Weak simulations preserved by Replication. The lemma can be found in Figure 30.3.*

Proof. The proof follows from $\widehat{\rightsquigarrow}$-I, and the PAR inversion rule. When the agent $!Q$ does an action α, case analysis is done on whether or not α is a τ-action, and Premises 22 or 23 are used for the different cases and a simulating action can be derived. Premise 24 is used to when the mimicking action is of the form $S \mid S \Longrightarrow_\tau S'$. Other than that, the proof structure is almost identical to the preservation lemma for Parallel, Lemma 28.28. □

The proof that weak bisimilarity is preserved by Replication will instantiate the relation $\mathscr{R}$ in Figure 30.3 to $\dot{\approx}$, $\mathscr{R}'$ to $\dot{\sim}$ and $\mathscr{R}''$ to *weakBisimCompose* $\dot{\approx}$. Hence all premises required for the simulation lemma has already been proven in previous chapters, except Premises 22-24. We will now prove them in turn.

Lemma 30.45. *Proof of Premise 22*

$$\frac{\Psi \rhd !P \xrightarrow{\alpha} P' \qquad bn\,\alpha \sharp subject\,\alpha \qquad guarded\,P \qquad \alpha \neq \tau \qquad bn\,\alpha \sharp P}{\exists Q.\ \Psi \rhd P \xrightarrow{\alpha} Q \wedge \mathbf{1} \rhd P' \dot{\sim} Q \mid !P}$$

Proof. By induction on $\Psi \rhd !P \xrightarrow{\alpha} P'$. The agent P which does the action α is obtained, and the derivative rewriten such that $!P$ is to the far right using the laws of structural congruence. □

$(\Psi, P, Q) \in \mathscr{R}$
$\mathit{eqvt}\ \mathscr{R} \qquad \mathit{eqvt}\ \mathscr{R}' \qquad \mathit{eqvt}\ \mathscr{R}'' \qquad \mathit{guarded}\ P \qquad \mathit{guarded}\ Q \qquad \mathscr{R}' \subseteq \mathscr{R}$

$$1 : \bigwedge \Psi'\, \Psi_U\, S\, T\, U\, \widetilde{b_U}.\ \frac{(\Psi' \otimes \Psi_U,\, S,\, T) \in \mathscr{R} \qquad \mathscr{F}\ U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T}{(\Psi',\, U \mid S,\, U \mid T) \in \mathscr{R}}$$

$$2 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R} \qquad \mathit{guarded}\ S \qquad \mathit{guarded}\ T}{(\Psi',\, U \mid !S,\, U \mid !T) \in \mathscr{R}''}$$

$$3 : \bigwedge \Psi'\, S\, T\, \widetilde{x}.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R} \qquad \widetilde{x} \sharp \Psi'}{(\Psi',\, (\nu \widetilde{x})S,\, (\nu \widetilde{x})T) \in \mathscr{R}}$$

$$4 : \bigwedge \Psi'\, S\, T\, \widetilde{x}.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}' \qquad \widetilde{x} \sharp \Psi'}{(\Psi',\, (\nu \widetilde{x})S,\, (\nu \widetilde{x})T) \in \mathscr{R}'}$$

$$5 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'',\, S,\, T) \in \mathscr{R}}$$

$$6 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}' \qquad \Psi' \simeq \Psi''}{(\Psi'',\, S,\, T) \in \mathscr{R}'}$$

$$7 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R} \qquad (\Psi',\, T,\, U) \in \mathscr{R}}{(\Psi',\, S,\, U) \in \mathscr{R}}$$

$$8 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}' \qquad (\Psi',\, T,\, U) \in \mathscr{R}'}{(\Psi',\, S,\, U) \in \mathscr{R}'}$$

$$9 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}}{\Psi' \rhd S \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} T} \qquad 10 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}}{(\Psi' \otimes \Psi'',\, S,\, T) \in \mathscr{R}}$$

$$11 : \bigwedge \Psi'\, S\, T\, \Psi''.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}'}{(\Psi' \otimes \Psi'',\, S,\, T) \in \mathscr{R}'} \qquad 12 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}}{(\Psi',\, T,\, S) \in \mathscr{R}}$$

$$13 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}'}{(\Psi',\, T,\, S) \in \mathscr{R}'} \qquad 14 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}}{(\Psi',\, S \mid U,\, T \mid U) \in \mathscr{R}}$$

$$15 : \bigwedge \Psi'\, S\, T\, U.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}'}{(\Psi',\, U \mid S,\, U \mid T) \in \mathscr{R}'}$$

$$16 : \bigwedge \Psi'\, S\, T.\ \frac{(\Psi',\, S,\, T) \in \mathscr{R}}{(\Psi',\, S \mid S,\, T \mid T) \in \mathscr{R}}$$

$$17: \bigwedge \Psi'\, S\, T\, U.\ (\Psi', S \mid (T \mid U), (S \mid T) \mid U) \in \mathscr{R}$$

$$18: \bigwedge \Psi'\, S\, T\, U.\ (\Psi', S \mid (T \mid U), (S \mid T) \mid U) \in \mathscr{R}'$$

$$19: \bigwedge \widetilde{x}\, \Psi'\, T\, S.\ \frac{\widetilde{x} \sharp \Psi' \qquad \widetilde{x} \sharp T}{(\Psi', (\nu \widetilde{x})(S \mid T), (\nu \widetilde{x}) S \mid T) \in \mathscr{R}}$$

$$20: \bigwedge \widetilde{x}\, \Psi'\, T\, S.\ \frac{\widetilde{x} \sharp \Psi' \qquad \widetilde{x} \sharp T}{(\Psi', (\nu \widetilde{x})(S \mid T), (\nu \widetilde{x}) S \mid T) \in \mathscr{R}'}$$

$$21: \bigwedge \Psi'\, S\, T\, U\, O.\ \frac{(\Psi', S, T) \in \mathscr{R} \qquad (\Psi', T, U) \in \mathscr{R}'' \qquad (\Psi', U, O) \in \mathscr{R}'}{(\Psi', S, O) \in \mathscr{R}''}$$

$$22: \bigwedge \Psi'\, S\, \alpha\, S'.\ \frac{\Psi' \rhd !S \xrightarrow{\alpha} S' \qquad \mathit{guarded}\ S \qquad \mathit{bn}\ \alpha \sharp S \qquad \alpha \neq \tau \qquad \mathit{bn}\ \alpha \sharp \mathit{subject}\ \alpha}{\exists T.\ \Psi' \rhd S \xrightarrow{\alpha} T \wedge (\mathbf{1}, S', T \mid !S) \in \mathscr{R}'}$$

$$23: \bigwedge \Psi'\, S\, S'.\ \frac{\Psi' \rhd !S \xrightarrow{\tau} S' \qquad \mathit{guarded}\ S}{\exists T.\ \Psi' \rhd S \mid S \xrightarrow{\tau} T \wedge (\mathbf{1}, S', T \mid !S) \in \mathscr{R}'}$$

$$24: \bigwedge \Psi'\, S\, S'.\ \frac{\Psi' \rhd S \mid S \Longrightarrow S' \qquad \mathit{guarded}\ S}{\exists T.\ \Psi' \rhd !S \Longrightarrow T \wedge (\Psi', T, S' \mid !S) \in \mathscr{R}'}$$

$$\frac{}{\Psi \rhd R \mid !P \mathrel{\widehat{\rightsquigarrow}_{\mathscr{R}''}} R \mid !Q}$$

Figure 30.3: Weak simulations which are preserved by Replication. The lemma contains three equivariant relations $\mathscr{R}$, $\mathscr{R}'$, and $\mathscr{R}''$.
Premise 1 states that $\mathscr{R}$ must be preserved by Parallel if the bound names of the frame of the parallel agent are sufficiently fresh.
Premise 2 states that $\mathscr{R}''$ must be preserved by Replication and Parallel.
Premises 3 and 4 state that $\mathscr{R}$ and $\mathscr{R}'$ must be preserved by Restriction.
Premises 5 and 6 state that the assertion components of $\mathscr{R}$ and $\mathscr{R}'$ must be interchangeable with statically equivalent assertions.
Premises 7 and 8 state that $\mathscr{R}$ and $\mathscr{R}'$ must be transitive.
Premises 9-13 are properties of strong and weak bisimilarity for $\mathscr{R}$ and $\mathscr{R}'$.
Requsites 14 and 15 state that $\mathscr{R}$ and $\mathscr{R}'$ must be preserved by Parallel.
Premises 16-20 are structural congruence properties of $\mathscr{R}$ and $\mathscr{R}'$.
Requsite 21 allows bisimulation up-to techniques to be used on $\mathscr{R}''$.
Premise 22 states that for all environments Ψ', if $!S$ does a non τ-action α to S', then S must be able to do the same action to a T such that $(\mathbf{1}, S', T \mid !S) \in \mathscr{R}'$
Premise 23 states that for all environments Ψ', if $!S$ does a τ-action to S', then $S \mid S$ must be able to do a τ-action to a T such that $(\mathbf{1}, S', T \mid !S) \in \mathscr{R}'$
Premise 24 states that for all environments Ψ', if $S \mid S$ does a τ-chain to S', then $!S$ must be able to do a τ-chain to a T such that $(\Psi', S', T \mid !S) \in \mathscr{R}'$

Lemma 30.46. *Proof of Premise 23*

$$\frac{\Psi \rhd !P \xrightarrow{\tau} P' \qquad \textit{guarded } P}{\exists Q.\, \Psi \rhd P \mid P \xrightarrow{\tau} Q \wedge \mathbf{1} \rhd P' \mathrel{\dot{\sim}} Q \mid !P}$$

Proof. By induction on $\Psi \rhd !P \xrightarrow{\tau} P'$. The Parallel cases are discharged in the same way as Lemma 30.45, but with an inert copy of P added to complete the lemma. For the communication case, two copies of P are extracted to communicate with each other. The derivatives are then rewritten such that $!P$ is to the far right using the laws of structural congruence. □

To prove Premise 24, an auxiliary lemma is required.

Lemma 30.47.

$$\frac{\Psi \rhd P \mid P \xrightarrow{\tau} P' \qquad \textit{guarded } P}{\exists Q.\, \Psi \rhd !P \xrightarrow{\tau} Q \wedge \mathbf{1} \rhd Q \mathrel{\dot{\sim}} P' \mid !P}$$

Proof. Follows from the PAR inversion rule. All cases are discharged by adding $!P$ as a parallel component using the PAR2 rule from the operational semantics. The parallel copies of P are then folded into $!P$ using the REPL rule. □

Lemma 30.48. *Proof of Premise 24*

$$\frac{\Psi \rhd P \mid P \Longrightarrow P' \qquad \textit{guarded } P}{\exists Q.\, \Psi \rhd !P \Longrightarrow Q \wedge \Psi \rhd Q \mathrel{\dot{\sim}} P' \mid !P}$$

Proof. By induction on $\Psi \rhd P \mid P \Longrightarrow P'$.

Base case ($P' = P \mid P$): Follows immediately by setting Q to $!P$ since $\Psi \rhd !P \stackrel{\tau}{\Longrightarrow} !P$ and $\Psi \rhd !P \mathrel{\dot{\sim}} (P \mid P) \mid !P$ by the laws of structural congruence.

Inductive step ($\Psi \rhd P \mid P \stackrel{\tau}{\Longrightarrow} P''$ and $\Psi \rhd P'' \xrightarrow{\tau} P'$): We have to find a Q such that $\Psi \rhd !P \stackrel{\tau}{\Longrightarrow} Q$ and $\Psi \rhd Q \mathrel{\dot{\sim}} P' \mid !P$.

- From the induction hypothesis we obtain an R such that $\Psi \rhd !P \stackrel{\tau}{\Longrightarrow} R$ and $\Psi \rhd R \mathrel{\dot{\sim}} P'' \mid !P$.
- From $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $\Psi \otimes \mathbf{1} \rhd P'' \xrightarrow{\tau} P'$ by Lemma 25.11 and the AID axiom.
- Hence $\Psi \rhd P'' \mid !P \xrightarrow{\tau} P' \mid !P$ by PAR2.
- With $\Psi \rhd R \mathrel{\dot{\sim}} P'' \mid !P$ we obtain an agent R' such that $\Psi \rhd R \xrightarrow{\tau} R'$ and $\Psi \rhd R' \mathrel{\dot{\sim}} P' \mid !P$.
- From $\Psi \rhd !P \stackrel{\tau}{\Longrightarrow} R$ and $\Psi \rhd R \xrightarrow{\tau} R'$ we have that $\Psi \rhd !P \stackrel{\tau}{\Longrightarrow} R'$

- We prove the goal by setting Q to R'.

□

With these lemmas in place we can prove that weak bisimilarity is preserved by Replication.

Lemma 30.49. *If* $\Psi \rhd P \dot{\approx} Q$ *and guarded* P *and guarded* Q *then* $\Psi \rhd !P \dot{\approx} !Q$.

Proof. □

By coinduction using Lemma 30.43 with $\mathscr{Y}$ set to

$$\{(\Psi, R \mid !P, R \mid !Q) : \textit{guarded } P \wedge \textit{guarded } Q \wedge \Psi \rhd P \dot{\approx} Q\}$$

Static implication: Follows directly from the definition of $\dot{\lessapprox}$ and $\dot{\approx}$-E3 since $!P$ and $!Q$ have empty frames.

Simulation: Follows from Lemma 30.44 with $\mathscr{R}$ set to $\dot{\approx}$, $\mathscr{R}'$ set to $\dot{\sim}$ and $\mathscr{R}''$ set to *weakBisimCompose* $\dot{\approx}$.

Premise 1 follows from Lemma 30.40. Premise 2 follows from the definition of *weakBisimCompose*. Premise 3 and 4 follows from Lemmas 30.37 and 27.32. Premises 5 and 6 follow from Lemmas 30.23 and 27.20. Premises 7 and 8 follow from transitivity of weak bisimilarity and strong bisimilarity. Premises 9-13 are properties of weak and/or strong bisimilarity. Premises 14 and 16 follow from Lemmas 30.41 and 27.42. Premises 17-20 are structural congrence laws for strong and weak bisimilarity. Premise 21 allows weak bisimulation up-to techniques to be used on $\mathscr{R}''$. Premises 22-24 follow from Lemmas 30.45, 30.46, and 30.48.

In some of the cases laws of structural congruence are used to rewrite the rules to match the premises.

Extension: Follows directly from $\dot{\approx}$-E3

Symmetry: The candidate relation $\mathscr{Y}$ is symmetric.

We can now prove a main theorem for weak bisimilarity.

Theorem 30.2. *Weak bisimilarity is preserved by all opertors except Case and Input.*

Proof. Follows from Lemmas 30.33, 30.37, 30.41, and 30.49. □

31. Weak congruence

Weak bisimilarity is not a congruence, as it is preserved neither by Input nor Case. We obtain a congruence in a similar manner as is done for the pi-calculus and for CCS. We first obtain a weak equivalence by disallowing agents to mimic a τ-action by doing nothing. A congruence is then obtained by closing weak equivalence under sequential substitutions, in the same way as for strong bisimilarity.

In the next section we will define weak τ-bisimilarity, sometimes denoted τ-bisimilarity, which does not allow an agent to mimic a τ-action by doing nothing. We then obtain a congruence by closing τ-bisimilarity under substitutions.

31.1 Weak τ-bisimilarity

Before defining τ-bisimilarity, we must define its simulation. A τ-simulation only mimics τ-actions – any other action is mimicked in the same way as for weak simulations, and hence will not be redefined.

Definition 31.1 (τ-simulation)**.**

$$\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q \stackrel{\text{def}}{=} \forall Q'.\, \Psi \rhd Q \xrightarrow{\tau} Q' \longrightarrow (\exists P'.\, \Psi \rhd P \stackrel{\tau}{\Longrightarrow} P' \ \wedge \ (\Psi, P', Q') \in \mathscr{R})$$

Note that the mimicking τ-chain is the transitive closure, and not the reflexive transitive closure of τ-actions, and hence any τ-action must be mimicked by at least one τ-action.

We now define τ-bisimilarity.

Definition 31.2 (weak τ-bisimilarity)**.**

$$\Psi \rhd P \cong Q \stackrel{\text{def}}{=} \Psi \rhd P \dot{\approx} Q \wedge \Psi \rhd P \rightsquigarrow_{\dot{\approx}} Q \wedge \Psi \rhd Q \rightsquigarrow_{\dot{\approx}} P$$

We do not require that weak τ-bisimilarity is closed by arbitrary assertions. We will discuss the absence of this requirement when defining weak congruence.

31.1.1 Primitive inference rules

The introduction rule for τ-simulation is significantly simpler than the previous simulation rules – since it only refers to τ-actions and no new binders are introduced.

Lemma 31.3. *Introduction and elimination rules for τ-simulation*

$$\frac{\bigwedge Q'.\ \dfrac{\Psi \rhd Q \xrightarrow{\tau} Q'}{\exists P'.\ \Psi \rhd P \overset{\tau}{\Longrightarrow} P' \wedge (\Psi, P', Q') \in \mathscr{R}}}{\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q}\ \rightsquigarrow\text{-I}$$

$$\frac{\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q \qquad \Psi \rhd Q \xrightarrow{\tau} Q'}{\exists P'.\ \Psi \rhd P \overset{\tau}{\Longrightarrow} P' \wedge (\Psi, P', Q') \in \mathscr{R}}\ \rightsquigarrow\text{-E}$$

Proof. Follows immediately from the definition of $\rightsquigarrow$. □

Lemma 31.4. *Introduction and elimination rules for τ-bisimilarity.*

$$\frac{\Psi \rhd P \dot{\approx} Q \qquad \Psi \rhd P \rightsquigarrow_{\dot{\approx}} Q \qquad \Psi \rhd Q \rightsquigarrow_{\dot{\approx}} P}{\Psi \rhd P \cong Q}\ \cong\text{-}I$$

$$\frac{\Psi \rhd P \cong Q}{\Psi \rhd P \dot{\approx} Q}\ \cong\text{-}E1 \qquad \frac{\Psi \rhd P \cong Q}{\Psi \rhd P \rightsquigarrow_{\dot{\approx}} Q}\ \cong\text{-}E2 \qquad \frac{\Psi \rhd P \cong Q}{\Psi \rhd Q \rightsquigarrow_{\dot{\approx}} P}\ \cong\text{-}E3$$

Proof. Follows from the definition of $\cong$. □

The symmetric introduction rule for τ-bisimilarity follows the same format as for CCS and the pi-calculus. It has one extra case for the bisimilarity case.

Lemma 31.5. *Symmetric introduction rule for τ-bisimilarity.*

$$\frac{\begin{array}{c}Prop\ P\ Q \qquad \bigwedge P\ Q.\ \dfrac{Prop\ P\ Q}{Prop\ Q\ P} \\ \bigwedge P\ Q.\ \dfrac{Prop\ P\ Q}{\Psi \rhd F\ P \dot{\approx} F\ Q} \qquad \bigwedge P\ Q.\ \dfrac{Prop\ P\ Q}{\Psi \rhd F\ P \rightsquigarrow_{\dot{\approx}} F\ Q}\end{array}}{\Psi \rhd F\ P \cong F\ Q}$$

Proof. Follows from the definition of $\cong$. □

31.2 Weak τ-bisimilarity is an equivalence relation

Lemma 31.6. *Weak τ-simulation is reflexive*

$$\frac{\{(\Psi, P, P) : \mathit{True}\} \subseteq \mathscr{R}}{\Psi \rhd P \rightsquigarrow_{\mathscr{R}} P}$$

Proof. Follows from the definition of $\rightsquigarrow$. □

Lemma 31.7.

$$\frac{\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q' \qquad \Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q \qquad \bigwedge \Psi\, P\, Q.\ \dfrac{(\Psi, P, Q) \in \mathscr{R}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}}{\exists P'.\ \Psi \rhd P \stackrel{\tau}{\Longrightarrow} P' \wedge (\Psi, P', Q') \in \mathscr{R}}$$

Proof. By induction on $\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q'$.

Base case ($\Psi \rhd Q \xrightarrow{\tau} Q'$): The proof follows directly from $\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q$ and $\rightsquigarrow$-E.

Inductive step ($\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q'$ and $\Psi \rhd Q' \xrightarrow{\tau} Q''$):

- From the induction hypothesis we obtain a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ and $(P', Q') \in \mathscr{R}$.
- From $(P', Q') \in \mathscr{R}$ and the assumptions we have that $\Psi \rhd P' \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q'$.
- With $\Psi \rhd Q' \xrightarrow{\tau} Q''$ we obtain a P'' such that $\Psi \rhd P' \Longrightarrow P''$ and $(\Psi, P'', Q'') \in \mathscr{R}$.
- Finally with $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ we have that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P''$ and $(\Psi, P'', Q'') \in \mathscr{R}$.

□

Lemma 31.8. *Weak τ-simulation is transitive*

$$\frac{\begin{array}{l}(\Psi, P, Q) \in \mathscr{R} \qquad \Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q \qquad \Psi \rhd Q \rightsquigarrow_{\mathscr{R}'} R \\ \{(\Psi, P, R) : \exists Q.\ (\Psi, P, Q) \in \mathscr{R} \wedge (\Psi, Q, R) \in \mathscr{R}'\} \subseteq \mathscr{R}'' \\ \bigwedge \Psi\, P\, Q.\ \dfrac{(\Psi, P, Q) \in \mathscr{R}}{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q}\end{array}}{\Psi \rhd P \rightsquigarrow_{\mathscr{R}''} R}$$

Proof. From $\rightsquigarrow$-I we have that given that $\Psi \rhd R \xrightarrow{\tau} R'$, we must find a P' such that $\Psi \rhd P \Longrightarrow P'$ and $(\Psi, P', R') \in \mathscr{R}''$.

- From $\Psi \rhd Q \rightsquigarrow_{\mathscr{R}'} R$ and $\Psi \rhd R \xrightarrow{\tau} R'$ we obtain a Q' such that $\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q'$ and $(\Psi, Q', R') \in \mathscr{R}'$ by $\rightsquigarrow$-E.
- From $\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q$ and $\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q'$ we obtain a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}$ by Lemma 31.7.
- Finally, from $(\Psi, P', Q') \in \mathscr{R}$ and $(\Psi, Q', R') \in \mathscr{R}'$ we have from the assumptions that $(\Psi, P', R') \in \mathscr{R}''$.

□

Lemma 31.9. *Weak τ-bisimilarity is an equivalence relation.*

Proof.

Reflexivity: Follows from Lemma 31.6 and reflexivity of weak bisimilarity.

Symmetry: Follows from $\cong$-E and symmetry of weak bisimilarity.

Transitivity: Follows from Lemma 31.8 and transitivity of weak bisimilarity.

□

31.3 Equivalence correspondences

That τ-bisimilarity includes weak bisimilarity follows directly from its definition. The remaining proof is that it also includes strong bisimilarity. We begin with proving the correspondence of simulations.

Lemma 31.10. *If $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$ and $\mathscr{R} \subseteq \mathscr{R}'$ then $\Psi \rhd P \rightsquigarrow_{\mathscr{R}'} Q$.*

Proof. Follows immediately from the definitions of $\hookrightarrow$ and $\rightsquigarrow$. The only actions involved are τ-actions, and the mimicking τ-action can be converted to a τ-chain. □

Lemma 31.11. *If $\Psi \rhd P \dot{\sim} Q$ then $\Psi \rhd P \cong Q$.*

Proof. We use the symmetric introduction rule from Lemma 31.5 with *Prop* set to $\lambda R\ S.\ R \dot{\sim} S$ and F set to the identity function.

Symmetry: Follows directly since strong bisimilarity is symmetric.

Simulation: Follows from Lemma 31.10.

Bisimilarity: Follows from Lemma 30.30.

□

We can now prove that τ-bisimilarity includes structural congruence.

Theorem 31.1. *If* $P \equiv Q$ *then* $\mathbf{1} \rhd P \cong Q$.

Proof. Follows directly from Theorem 28.3 and Lemma 31.11. □

31.4 Preservation properties

In order to facilitate the preservation proofs we create an alternative introduction rule specially tailored for proving congruence properties.

Lemma 31.12. *Symmetric introduction rule for weak congruence.*

$$\frac{\Psi \rhd P \cong Q \qquad \bigwedge P\,Q.\ \dfrac{\Psi \rhd P \cong Q}{\Psi \rhd F\,P \mathrel{\dot{\approx}} F\,Q} \qquad \bigwedge P\,Q.\ \dfrac{\Psi \rhd P \cong Q}{\Psi \rhd F\,P \rightsquigarrow_{\dot{\approx}} F\,Q}}{\Psi \rhd F\,P \cong F\,Q}\ \cong\text{-I2}$$

Proof. Follows from Lemma 31.5 with *Prop* set to $\lambda P\,Q.\ \Psi \rhd P \cong Q$. □

Whenever this introduction rule is used two cases have to be proven – one for the τ-simulation, and one for the weak bisimilarity. This rule is also the one used for the remaining preservation proofs for τ-bisimilarity.

31.4.1 Output

As an agent with an output prefix cannot do any τ-actions, the τ-simulation is not applicable for this preservation proof.

Lemma 31.13. *If* $\Psi \rhd P \cong Q$ *then* $\Psi \rhd \overline{M}N.P \cong \overline{M}N.Q$.

Proof. Follows from $\approx$-I2

Simululation: Trivially true since none of the agents can do any τ-actions.

Bisimilarity: Follows directly from Lemma 30.33.

□

31.4.2 Case

The first part of the proof is to prove that weak bisimilarity is preserved by Case as long as τ-actions may not be mimicked by doing nothing.

Lemma 31.14.

$$\bigwedge \varphi\ Q.\ \frac{(\varphi,\ Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi,\ P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge Eq\ \Psi\ P\ Q}$$

$$\bigwedge \Psi'\ P\ Q.\ \frac{(\Psi',\ P,\ Q) \in \mathscr{R}}{\Psi' \rhd P\ \widehat{\rightsquigarrow}_{\mathscr{R}}\ Q} \qquad \bigwedge \Psi'\ P\ Q.\ \frac{Eq\ \Psi'\ P\ Q}{(\Psi',\ P,\ Q) \in \mathscr{R}}$$

$$\frac{\bigwedge \Psi'\ P\ Q.\ \dfrac{Eq\ \Psi'\ P\ Q}{\Psi' \rhd P \rightsquigarrow_{\mathscr{R}} Q}}{\Psi \rhd Cases\ \widetilde{C_P}\ \widehat{\rightsquigarrow}_{\mathscr{R}}\ Cases\ \widetilde{C_Q}}$$

The function *mem* checks whether or not an element is a member of a list. The *Eq* predicate is an arbitrary predicate meeting the constraints of the lemma, but will later be instantiated to τ-bisimilarity.

Proof. Similar to Lemma 27.26, but a case analysis is done on whether or not the action being mimicked is a τ-action. If a τ-action is being mimicked then the τ-simulation is used, which guarantees that the mimicking agent does at least one τ-action, otherwise regular weak simulation is used. □

Lemma 31.15.

$$\bigwedge \varphi\ P.\ \frac{(\varphi,\ P)\ mem\ \widetilde{C_P}}{\exists Q.\ (\varphi,\ Q)\ mem\ \widetilde{C_Q} \wedge guarded\ Q \wedge (\forall \Psi.\ \Psi \rhd P \cong Q)}$$

$$\frac{\bigwedge \varphi\ Q.\ \dfrac{(\varphi,\ Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi,\ P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge (\forall \Psi.\ \Psi \rhd P \cong Q)}}{\Psi \rhd Cases\ \widetilde{C_P} \stackrel{\cdot}{\approx} Cases\ \widetilde{C_Q}}$$

Proof. By coinduction with $\mathscr{X}$ set to

$$\begin{aligned}
&\{(\Psi,\ Cases\ \widetilde{C_P},\\
&\ \ Cases\ \widetilde{C_Q}) : (\forall \varphi\ P.\ (\varphi,\ P)\ mem\ \widetilde{C_P} \longrightarrow\\
&\qquad\qquad (\exists Q.\ (\varphi,\ Q)\ mem\ \widetilde{C_Q} \wedge\\
&\qquad\qquad\quad guarded\ Q \wedge (\forall \Psi.\ \Psi \rhd P \cong Q))) \wedge\\
&\qquad (\forall \varphi\ Q.\ (\varphi,\ Q)\ mem\ \widetilde{C_Q} \longrightarrow\\
&\qquad\qquad (\exists P.\ (\varphi,\ P)\ mem\ \widetilde{C_P} \wedge\\
&\qquad\qquad\quad guarded\ P \wedge (\forall \Psi.\ \Psi \rhd P \cong Q)))\}
\end{aligned}$$

Static implication: Similar to Lemma 30.31 – agents with Case as their top-most operator have empty frames.

Simulation: Follows from Lemma 31.14

Extension: Follows directly from the definition of $\mathscr{X}$

Symmetry: Follows directly since τ-bisimilarity is symmetric.

□

We can now prove the corresponding lemma for weak congruence.

Lemma 31.16.

$$\bigwedge \varphi\ Q.\ \frac{(\varphi, Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi, P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge Eq\ \Psi\ P\ Q}$$

$$\frac{\bigwedge \Psi'\ P\ Q.\ \dfrac{Eq\ \Psi'\ P\ Q}{\Psi' \rhd P \leadsto_{\mathscr{R}} Q}}{\Psi \rhd Cases\ \widetilde{C_P} \leadsto_{\mathscr{R}} Cases\ \widetilde{C_Q}}$$

Proof. Similar to Lemma 31.14, but only the mimicking of τ-actions must be considered. □

Lemma 31.17.

$$\bigwedge \varphi\ P.\ \frac{(\varphi, P)\ mem\ \widetilde{C_P}}{\exists Q.\ (\varphi, Q)\ mem\ \widetilde{C_Q} \wedge guarded\ Q \wedge (\forall \Psi.\ \Psi \rhd P \cong Q)}$$

$$\frac{\bigwedge \varphi\ Q.\ \dfrac{(\varphi, Q)\ mem\ \widetilde{C_Q}}{\exists P.\ (\varphi, P)\ mem\ \widetilde{C_P} \wedge guarded\ P \wedge (\forall \Psi.\ \Psi \rhd P \cong Q)}}{\Psi \rhd Cases\ \widetilde{C_P} \cong Cases\ \widetilde{C_Q}}$$

Proof. Proof using $\approx$-I2

Simulation: Follows from Lemma 31.16

Bisimilarity: Follows directly from Lemma 31.15

□

31.4.3 Restriction

The proof that τ-bisimilarity is preserved by restriction is significantly simpler than its counterpart for strong and weak bisimilarity, where the simulation lemmas require the renaming of the subjects of the transitions such that they do not clash with the bound names. Since τ-actions have no subjects, this is not required for τ-simulation.

Lemma 31.18.

$$\frac{\Psi \rhd P \rightsquigarrow_{\mathscr{R}} Q \qquad eqvt\ \mathscr{R}' \qquad x \sharp \Psi \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \bigwedge \Psi' R S x.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad x \sharp \Psi'}{(\Psi', (\nu x)R, (\nu x)S) \in \mathscr{R}'}}{\Psi \rhd (\nu x)P \rightsquigarrow_{\mathscr{R}'} (\nu x)Q}$$

Proof. Follows from $\rightsquigarrow$-I and the CASE inversion rule from Figure 26.2. The only applicable action is where a τ-action is done, which can be mimicked using the SCOPE rule from Figure 30.2. □

Lemma 31.19. *If $\Psi \rhd P \cong Q$ and $x \sharp \Psi$ then $\Psi \rhd (\nu x)P \cong (\nu x)Q$.*

Proof. Proof using $\approx$-I2

Simulation: Follows from Lemma 31.18

Bisimilarity: Follows directly from Lemma 30.37

□

31.4.4 Parallel

Lemma 31.20. *Weak τ-simulation is preserved by Parallel.*

$$\frac{\begin{array}{l} eqvt\ \mathscr{R} \qquad eqvt\ \mathscr{R}' \\ 1: \bigwedge \Psi'.\ \Psi' \rhd P \rightsquigarrow_{\mathscr{R}} Q \\ 2: \bigwedge \Psi'.\ \Psi' \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad 3: \bigwedge \Psi'.\ \Psi' \rhd Q \lesssim\!\!\approx_{\mathscr{R}} P \\ 4: \bigwedge \Psi' S T \widetilde{b_U} \Psi_U U.\ \dfrac{\mathscr{F} U = (\nu \widetilde{b_U})\Psi_U \qquad \widetilde{b_U} \sharp \Psi' \qquad \widetilde{b_U} \sharp S \qquad \widetilde{b_U} \sharp T}{(\Psi', S \mid U, T \mid U) \in \mathscr{R}'} \\ (\Psi' \otimes \Psi_U, S, T) \in \mathscr{R} \\ 5: \bigwedge \Psi' S T.\ \dfrac{(\Psi', S, T) \in \mathscr{R}}{(\Psi', T, S) \in \mathscr{R}} \qquad 6: \bigwedge \Psi' S T \Psi''.\ \dfrac{(\Psi', S, T) \in \mathscr{R}}{(\Psi' \otimes \Psi'', S, T) \in \mathscr{R}} \\ 7: \bigwedge \Psi' S T \widetilde{x}.\ \dfrac{(\Psi', S, T) \in \mathscr{R}' \qquad \widetilde{x} \sharp \Psi'}{(\Psi', (\nu \widetilde{x})S, (\nu \widetilde{x})T) \in \mathscr{R}'} \\ 8: \bigwedge \Psi' S T \Psi''.\ \dfrac{(\Psi', S, T) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', S, T) \in \mathscr{R}} \end{array}}{\Psi \rhd P \mid R \rightsquigarrow_{\mathscr{R}'} Q \mid R}$$

Premises 1-3 states that P must weakly simulate, τ-simulate, and weakly statically imply Q in all possible environments. Premise 4 states the preservation properties for Parallel required of $\mathscr{R}$ and $\mathscr{R}'$. Premises 5 and 6 are properties of weak bisimilarity for $\mathscr{R}$. Premise 7 states that $\mathscr{R}'$ must be preserved by Restriction. Premise 8 states that the assertion component of $\mathscr{R}$ must be exchangeable for a statically equivalent one.

Proof. The proof is similar to Lemma 30.39, with one key difference – weak τ-bisimilarity is not required to be preserved by extensions of the environment. For this proof, the agents P and Q must be τ-bisimilar in all environments, allowing the appropriate environment to be picked for different parts of the proof. Once a transition has been made, the derivatives are weakly bisimilar, and weak bisimilarity is preserved by extensions of the environment, as required by Premise 6. □

Lemma 31.21. *If* $\forall \Psi.\ \Psi \rhd P \cong Q$ *then* $\Psi \rhd P \mid R \cong Q \mid R$.

Proof. Follows from the symmetric introduction rule from Lemma 31.5 with *Prop* set to $\lambda S\ T.\ \forall \Psi.\ \Psi \rhd S \cong T$ and F set to the $\lambda S.\ S \mid R$.

Symmetry: Follows directly since τ-bisimilarity is symmetric.

Bisimilarity: Follows from $\cong$-E1 and Lemma 30.41.

Simulation: Proved using Lemma 31.20. Premise 1-3 are properties of τ-bisimilarity and weak bisimilarity. Premise 4 follows from Lemma 30.40. Premises 5-6 are properties of weak bisimilarity. Premise 7 follows from Lemma 30.37.

□

31.4.5 Replication

The proof that τ-bisimilarity is preserved by Replication is similar to its counterpart for weak bisimilarity. The cases where only one parallel agent does an action are simpler, as we only have to consider τ-actions, but the cases where they communicate require that we reason about input and output actions in the standard way.

Lemma 31.22.

If $\forall \Psi.\ \Psi \rhd P \cong Q$ *and guarded* P *and guarded* Q *then* $\Psi \rhd\ !P \cong\ !Q$.

Proof. Similar to Lemma 30.49. □

We can now prove that weak τ-bisimilarity is preserved by all operators except Input.

Theorem 31.2. *Weak τ-bisimilarity is preserved by all operators except Input.*

Proof. Follows from Lemmas 31.13, 31.17, 31.19, 31.21, and 31.22. □

31.5 Weak congruence

Weak congruence is defined as a binary predicate. Two agents are weakly congruent if they are τ-bisimilar for all environments Ψ and for all possible substitutions. The proof is similar to that of strong equivalence presented in Section 27.5.3.

Definition 31.23.

$$P \approx Q \stackrel{\text{def}}{=} \forall \Psi\ \sigma.\ wellFormedSubst\ \sigma \longrightarrow \Psi \rhd P\sigma \cong Q\sigma$$

Weak congruence is defined without the extra requisite that it is closed under arbitrary assertions. This property is implicit since at the top level, the environments are universally quantified, and for the derivatives as they are weakly bisimilar. Agents which are weakly bisimilar can have their environments extended by definition.

31.5.1 Primitive inference rules

The only inference rules required for weak congruence is an introduction, and an elimination rule.

Lemma 31.24. *Introduction and elimination rules for weak congruence*

$$\frac{\bigwedge \Psi\ \sigma.\ \dfrac{wellFormedSubst\ \sigma}{\Psi \rhd P\sigma \cong Q\sigma}}{P \approx Q}\ \approx\text{-}I \qquad \frac{P \approx Q \qquad wellFormedSubst\ \sigma}{\Psi \rhd P\sigma \cong Q\sigma}\ \approx\text{-}E$$

Proof. Follows immediately from the definition of $\approx$ □

31.5.2 Preservation properties

The only remaining preservation property is that weak congruence is preserved by Input.

Lemma 31.25. *Weak static implication is preserved by Input.*

$$\frac{\bigwedge \Psi'.\ (\Psi \otimes \Psi',\ \underline{M}(\lambda \tilde{x})N.P,\ \underline{M}(\lambda \tilde{x})N.Q) \in \mathscr{R}}{\Psi \rhd \underline{M}(\lambda \tilde{x})N.P \lesssim\!\!\!\approx_{\mathscr{R}} \underline{M}(\lambda \tilde{x})N.Q}$$

Proof. From the definition of $\lesssim$ we have to find for all assertions Ψ', agents Q' and Q'' such that

$$\begin{aligned}
&\Psi \ \rhd\ \underline{M}(\lambda\widetilde{x})N.Q \Longrightarrow Q',\\
&(\mathscr{F}\ \underline{M}(\lambda\widetilde{x})N.P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi,\\
&\Psi \otimes \Psi' \ \rhd\ Q' \Longrightarrow Q'', \text{ and}\\
&(\Psi \otimes \Psi', \underline{M}(\lambda\widetilde{x})N.P, Q'') \in \mathscr{R}.
\end{aligned}$$

This follows directly from the assumptions, and by setting Q' and Q'' to $\underline{M}(\lambda\widetilde{x})N.Q$. □

Lemma 31.26. *Weak simulation is preserved by Input.*

$$\frac{\bigwedge \widetilde{T}\ \Psi'.\ \dfrac{|\widetilde{x}| = |\widetilde{T}|}{(\Psi \otimes \Psi', P[\widetilde{x} := \widetilde{T}], Q[\widetilde{x} := \widetilde{T}]) \in \mathscr{R}}}{\Psi \ \rhd\ \underline{M}(\lambda\widetilde{x})N.P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. Follows from the definition of $\widehat{\rightsquigarrow}$, the INPUT inversion rule from Figure 26.2, and the weak INPUT semantic rule from Figure 30.2. □

Lemma 31.27.

$$\frac{\bigwedge \widetilde{T}.\ \dfrac{|\widetilde{x}| = |\widetilde{T}|}{\Psi \ \rhd\ P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\approx}} Q[\widetilde{x} := \widetilde{T}]}}{\Psi \ \rhd\ \underline{M}(\lambda\widetilde{x})N.P \mathrel{\dot{\approx}} \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. Similar to Lemma 27.50 □

Lemma 31.28.

$$\frac{\bigwedge \widetilde{T}.\ \dfrac{|\widetilde{x}| = |\widetilde{T}|}{\Psi \ \rhd\ P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\approx}} Q[\widetilde{x} := \widetilde{T}]}}{\Psi \ \rhd\ \underline{M}(\lambda\widetilde{x})N.P \cong \underline{M}(\lambda\widetilde{x})N.Q}$$

Proof. Follows from the symmetric introduction rule, Lemma 31.5 with *Prop* set to $\lambda R\ S.\ \forall\, \widetilde{T}.\ |\widetilde{x}| = |\widetilde{T}| \longrightarrow \Psi \ \rhd\ R[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\approx}} S[\widetilde{x} := \widetilde{T}]$ and F set to $\lambda R.\ \underline{M}(\lambda\widetilde{x})N.R$.

Symmetry: Follows from symmetry of weak bisimilarity.

Bisimilarity: Follows from Lemma 31.27.

Simulation: This case is trivially true as τ-simulations do not simulate input actions.

□

Lemma 31.29.

$$\textit{If } P \approx Q \textit{ and distinct } \tilde{x} \textit{ then } \underline{M}(\lambda\tilde{x})N.P \approx \underline{M}(\lambda\tilde{x})N.Q\,.$$

Proof. Similar to Lemma 27.51 □

We can now prove that weak congruence is a congruence.

Theorem 31.3. *Weak congruence is a congruence.*

Proof. That weak congruence is preserved by Input follows from Lemma 31.29; that it is preserved by all other operators follows from ≈-I, ≈-E and Theorem 31.2, where all bound names are alpha-converted to avoid the substitution σ. □

32. Psi-calculi with weakening

Given an entailment relation which satisfies logical weakening, a simpler version of weak bisimilarity suffices. In this chapter we formally demonstrate this using Isabelle's support for locales.

Creating a locale with the additional requirement that

$$\Psi \leq \Psi \otimes \Psi' \qquad \text{WEAKEN}$$

is straightforward, and is done with a single command in Isabelle. For the rest of this chapter we will reason about sub-calculi of psi-calculi where the weakening requisite is satisfied, and the results will in general not hold for calculi without weakening.

The main result that weakening allows, and that the results bellow build on, is that a transition cannot be blocked by extending its environment.

Lemma 32.1. *If* WEAKEN *holds then*

$$\textit{If } \Psi \rhd P \longmapsto V \textit{ then } \Psi \otimes \Psi' \rhd P \longmapsto V.$$

Proof. By induction on $\Psi \rhd P \longmapsto V$, where the bound names avoid Ψ'. The proof then follows from the standard operational semantic rules, where any condition enabled by Ψ must also be enabled by $\Psi \otimes \Psi'$ by WEAKEN.

□

32.1 Weak transitions

As was discussed in Section 29.1 any weak transition must be mimicked by the same action with preceding and succeeding τ-chains. Weak transitions can thus be defined in the same way as for CCS and the early semantics of the pi-calculus.

Definition 32.2 (weak transition)**.**

$$\begin{aligned}&\Psi \ \rhd P \stackrel{\hat{\alpha}}{\Longrightarrow} P' \stackrel{\text{def}}{=} \\ &\quad (\exists P''' P''.\ \Psi \ \rhd \ P \Longrightarrow P''' \wedge \Psi \rhd P''' \xrightarrow{\alpha} P'' \wedge \Psi \ \rhd \ P'' \Longrightarrow P') \vee \\ &\quad (P = P' \wedge \alpha = \tau)\end{aligned}$$

We define the following case analysis rule for weak transitions.

Lemma 32.3.

$$\frac{\Psi \rhd P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \qquad Prop\,(\tau)\ P}{\bigwedge P'''\,P''.\ \dfrac{\Psi \rhd P \Longrightarrow P''' \qquad \Psi \rhd P''' \xrightarrow{\alpha} P'' \qquad \Psi \rhd P'' \Longrightarrow P'}{Prop\ \alpha\ P'}}{Prop\ \alpha\ P'}$$

Proof. Follows directly from Definition 32.2 □

32.2 Simple bisimilarity

Simple bisimilarity is defined in the standard way.

Definition 32.4 (simple simulation)**.**

$$\Psi \rhd P \underset{\mathrm{smp}_{\mathscr{R}}}{\widehat{\rightsquigarrow}} Q \stackrel{\mathrm{def}}{=} \forall \alpha Q'.\ \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge bn\ \alpha \sharp \Psi \wedge bn\ \alpha \sharp P \longrightarrow \exists P'.\ \Psi \rhd P \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge (\Psi, P', Q') \in \mathscr{R}$$

Definition 32.5 (Simple static implication)**.** $\Psi \rhd P \underset{\mathrm{smp}_{\mathscr{R}}}{\lessapprox} Q \stackrel{\mathrm{def}}{=}$

$$\exists Q'.\ \Psi \rhd Q \Longrightarrow Q' \wedge (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi \wedge (\Psi, P, Q') \in \mathscr{R}$$

Definition 32.6 (simple bisimilarity)**.** *Simple bisimilarity, denoted* $\underset{\mathrm{smp}}{\dot{\approx}}$*, is defined coinductively as the greatest fixpoint satisfying:*

$$\begin{aligned}
&\Psi \rhd P \underset{\mathrm{smp}}{\overset{\mathrm{smp}}{\lessapprox}} Q \Longrightarrow \Psi \rhd P \underset{\mathrm{smp}}{\overset{\mathrm{smp}}{\lessapprox}} Q && \textsc{StatImp}\\
&\quad \wedge \Psi \rhd P \underset{\mathrm{smp}}{\overset{\mathrm{smp}}{\widehat{\rightsquigarrow}}} Q && \textsc{Simulation}\\
&\quad \wedge \forall \Psi'.\ \Psi \otimes \Psi' \rhd P \underset{\mathrm{smp}}{\dot{\approx}} Q && \textsc{Extension}\\
&\quad \wedge \Psi \rhd Q \underset{\mathrm{smp}}{\dot{\approx}} P && \textsc{Symmetry}
\end{aligned}$$

32.2.1 Primitive inference rules

Previous introduction rules for simulation have been custom tailored to ensure that any bound names appearing in the transition to be sufficiently fresh. Even though the same is possible for simple simulation, it is not necessary for the results of this chapter.

Lemma 32.7. *Introduction and elimination rules for simple simulation.*

$$\frac{\bigwedge \alpha\ Q'.\ \dfrac{\Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\ \alpha \sharp \Psi \qquad bn\ \alpha \sharp P}{\exists P'.\ \Psi \rhd P \xRightarrow{\widehat{\alpha}} P' \wedge (\Psi, P', Q') \in \mathscr{R}}}{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\widehat{\rightsquigarrow}} Q}\ \underset{\text{smp}}{\widehat{\rightsquigarrow}}\text{-I}$$

$$\frac{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\widehat{\rightsquigarrow}} Q \qquad \Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\ \alpha \sharp \Psi \qquad bn\ \alpha \sharp P}{\exists P'.\ \Psi \rhd P \xRightarrow{\widehat{\alpha}} P' \wedge (\Psi, P', Q') \in \mathscr{R}}\ \underset{\text{smp}}{\widehat{\rightsquigarrow}}\text{-E}$$

Proof. Follows directly from the definition of $\underset{\text{smp}}{\widehat{\rightsquigarrow}}$. □

Lemma 32.8. *Introduction and elimination rules for simple static implication.*

$$\frac{\Psi \rhd Q \Longrightarrow Q' \qquad (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi \qquad (\Psi, P, Q') \in \mathscr{R}}{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\lessapprox} Q}\ \underset{\text{smp}}{\lessapprox}\text{-I}$$

$$\frac{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\lessapprox} Q}{\exists Q'.\ \Psi \rhd Q \Longrightarrow Q' \wedge (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi \wedge (\Psi, P, Q') \in \mathscr{R}}\ \underset{\text{smp}}{\lessapprox}\text{-E}$$

Proof. Follows directly from the definition of $\underset{\text{smp}}{\lessapprox}$. □

Lemma 32.9. *Elimination rules for simple bisimilarity.*

$$\frac{\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q}{\Psi \rhd P \underset{\text{smp}}{\lessapprox} \underset{\text{smp}}{\dot{\approx}} Q}\ \underset{\text{smp}}{\dot{\approx}}\text{-E1} \qquad \frac{\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q}{\Psi \rhd P \underset{\text{smp}}{\widehat{\rightsquigarrow}} \underset{\text{smp}}{\dot{\approx}} Q}\ \underset{\text{smp}}{\dot{\approx}}\text{-E2}$$

$$\frac{\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q}{\Psi \otimes \Psi' \rhd P \underset{\text{smp}}{\dot{\approx}} Q}\ \underset{\text{smp}}{\dot{\approx}}\text{-E3} \qquad \frac{\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q}{\Psi \rhd Q \underset{\text{smp}}{\dot{\approx}} P}\ \underset{\text{smp}}{\dot{\approx}}\text{-E4}$$

Proof. Follows from the coinduction rule which Isabelle generates automatically from Definition 32.6. □

Lemma 32.10. *Coinduction rule for simple bisimilarity.*

$$\frac{(\Psi, P, Q) \in \mathscr{X} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \underset{\text{smp}\mathscr{X} \cup \dot{\approx}_{\text{smp}}}{\lesssim} S}\ \textsc{StatImp} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \underset{\text{smp}\mathscr{X} \cup \dot{\approx}_{\text{smp}}}{\widehat{\rightsquigarrow}} S}\ \textsc{Simulation} \qquad \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{X} \vee \Psi' \otimes \Psi'' \rhd R \dot{\underset{\text{smp}}{\approx}} S}\ \textsc{Extension} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi', S, R) \in \mathscr{X} \vee \Psi' \rhd S \dot{\underset{\text{smp}}{\approx}} R}\ \textsc{Symmetry}}{\Psi \rhd P \dot{\underset{\text{smp}}{\approx}} Q}$$

32.3 Weak and simple bisimilarity coincide

The trailing τ-chain of a weak simulation is extended with an arbitrary assertion, whereas the trailing τ-chain for a simple simulation runs in the same environment as before. The main difficulty of the proof is to handle these assertion extensions. To prove that weak bisimilarity includes simple bisimilarity is reasonably straightforward, as the assertion extensions can be set to $\mathbf{1}$, and then discharged using the AID axiom.

The proof in the other direction is more involved, and requires extra lemmas on how the WEAKEN axiom affects transitions and simulations.

32.3.1 Weak bisimilarity includes simple bisimilarity

Before proving that weak bisimilarity includes simple bisimilarity, we need an auxiliary lemma which states that the frame of an agent statically implies the frame of any τ-chain derivative of that agent. We begin by proving this result for single τ-actions.

Lemma 32.11. *If* WEAKEN *holds then:*

$$\textit{If } \Psi \rhd P \xrightarrow{\tau} P' \textit{ then } (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ P') \otimes \Psi.$$

Proof. By $\leq$-I we must prove that for all φ if $(\mathscr{F}\ P) \otimes \Psi \vdash \varphi$ then $(\mathscr{F}\ P') \otimes \Psi \vdash \varphi$.

- We pick a frame $(\nu b_P)\Psi_P$ of P such that $b_P \sharp \Psi$ and $b_P \sharp \varphi$.
- With $\Psi \rhd P \xrightarrow{\tau} P'$ we obtain a $b_P{}'$, a $\Psi_P{}'$, and a Ψ' such that $\mathscr{F}\ P' = (\nu b_P{}')\Psi_P{}'$, $b_P{}' \sharp \varphi$, $b_P{}' \sharp \Psi$ and $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$, by Lemma 27.37.
- From $(\mathscr{F}\ P) \otimes \Psi \vdash \varphi$ and $b_P \sharp \varphi$ we have that $\Psi_P \otimes \Psi \vdash \varphi$
- Hence $(\Psi_P \otimes \Psi') \otimes \Psi \vdash \varphi$ by WEAKEN, ACOMM and AASSOC, and hence $\Psi_P{}' \otimes \Psi \vdash \varphi$ since $\Psi_P \otimes \Psi' \simeq \Psi_P{}'$.
- Finally we have that $(\mathscr{F}\ P') \otimes \Psi \vdash \varphi$ since $b_P{}' \sharp \varphi$ and $b_P{}' \sharp \Psi$.

□

We can then proceed to prove the corresponding lemma for τ-chains.

Lemma 32.12. *If* WEAKEN *holds then:*

$$\textit{If } \Psi \rhd P \Longrightarrow P' \textit{ then } (\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ P') \otimes \Psi.$$

Proof. By induction on $\Psi \rhd P \Longrightarrow P'$

Base case $P = P'$: Since $\leq$ is reflexive we have that $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ P) \otimes \Psi$.

Inductive step $\Psi \rhd P \xRightarrow{\tau} P''$ and $\Psi \rhd P'' \xrightarrow{\tau} P'$:

- By the induction hypothesis we have that $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ P'') \otimes \Psi$.
- Moreover from $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $(\mathscr{F}\ P'') \otimes \Psi \leq (\mathscr{F}\ P') \otimes \Psi$ by Lemma 32.11.
- Finally we have that $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ P') \otimes \Psi$ since $\leq$ is transitive.

□

The only requirement we impose on the candidate relation for the static implication and simulation proofs is that it is preserved by static equivalence.

Lemma 32.13. *If* WEAKEN *holds then:*

$$\frac{\Psi \rhd P \lesssim\!\!\approx_{\mathscr{R}} Q \qquad \bigwedge \Psi'\, R\, S\, \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', R, S) \in \mathscr{R}}}{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\lesssim\!\!\approx} Q}$$

Proof. By $\underset{\text{smp}}{\lesssim\!\!\approx}$-I we have to prove that there exists an R such that $\Psi \rhd Q \xRightarrow{\tau} R$, $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ R) \otimes \Psi$, and $(\Psi, P, R) \in \mathscr{R}$.

- From $\Psi \rhd P \lesssim\!\!\approx_{\mathscr{R}} Q$ we obtain a Q'' and a Q' such that $\Psi \rhd Q \xRightarrow{\tau} Q''$, $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q'') \otimes \Psi$, $\Psi \otimes \mathbf{1} \rhd Q'' \xRightarrow{\tau} Q'$, and $(\Psi \otimes \mathbf{1}, P, Q') \in \mathscr{R}$.

- From $\Psi \otimes \mathbf{1} \rhd Q'' \stackrel{\tau}{\Longrightarrow} Q'$ we have that $\Psi \rhd Q'' \stackrel{\tau}{\Longrightarrow} Q'$ by Lemma 30.8 and IDENTITY.
- Hence $(\mathscr{F}\ Q'') \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi$ by Lemma 32.12.
- From $\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q''$ and $\Psi \rhd Q'' \stackrel{\tau}{\Longrightarrow} Q'$ we have that $\Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q'$.
- Moreover from $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q'') \otimes \Psi$ and $(\mathscr{F}\ Q'') \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi$ we have that $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi$ by transitivity of $\leq$.
- Moreover from $(\Psi \otimes \mathbf{1}, P, Q') \in \mathscr{R}$ we have that $(\Psi, P, Q') \in \mathscr{R}$ by IDENTITY and the assumptions.
- Finally the goal is proven by setting R to Q'.

□

Lemma 32.14.

$$\frac{\Psi \rhd P \mathrel{\widehat{\rightsquigarrow}}_{\mathscr{R}} Q \qquad \bigwedge \Psi'\, R\, S\, \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad \Psi' \simeq \Psi''}{(\Psi'', R, S) \in \mathscr{R}}}{\Psi \rhd P \mathrel{\underset{\mathrm{smp}}{\widehat{\rightsquigarrow}}}_{\mathscr{R}} Q}$$

Proof. By $\underset{\mathrm{smp}}{\widehat{\rightsquigarrow}}$-I, we assume $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and we must prove that there exists a P' such that $\Psi \rhd P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}$

The proof is by case analisys if $\alpha = \tau$

$\alpha = \tau$**:** From $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\alpha = \tau$ we obtain from $\widehat{\rightsquigarrow}$-E2 a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}$.

$\alpha \neq \tau$**:** From $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\alpha \neq \tau$ we obtain from $\widehat{\rightsquigarrow}$-E1 a P''', a P'', and a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'''$, $\Psi \rhd P''' \xrightarrow{\alpha} P''$, $\Psi \otimes \mathbf{1} \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$, and $(\Psi \otimes \mathbf{1}, P', Q') \in \mathscr{R}$.

- From $\Psi \otimes \mathbf{1} \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ we have that $\Psi \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ by AID and Lemma 30.8.
- With $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'''$ and $\Psi \rhd P''' \xrightarrow{\alpha} P''$ we have that $\Psi \rhd P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$, proving the transition part of the simulation.
- Finally, from $(\Psi \otimes \mathbf{1}, P', Q') \in \mathscr{R}$ we have that $(\Psi, P', Q') \in \mathscr{R}$ by the assumptions.

□

We can now prove that weak bisimilarity includes simple bisimilarity.

Lemma 32.15. *If* WEAKEN *holds then:*

$$\textit{If } \Psi \rhd P \dot{\approx} Q \textit{ then } \Psi \rhd P \underset{\mathrm{smp}}{\dot{\approx}} Q.$$

Proof. By coinduction with $\mathscr{X}$ set to $\dot{\approx}$.

Static implication: We need to prove that if $\Psi \rhd P \mathrel{\dot{\approx}} Q$ then $\Psi \rhd P \underset{\text{smp}_{\dot{\approx}}}{\lesssim\!\!\approx} Q$.

- From $\Psi \rhd P \mathrel{\dot{\approx}} Q$ we have that $\Psi \rhd P \mathrel{\dot{\lesssim\!\!\approx}} Q$ by $\dot{\approx}$-E1.
- Hence $\Psi \rhd P \underset{\text{smp}_{\dot{\approx}}}{\lesssim\!\!\approx} Q$ using lemmas 30.23 and 32.13.

Simulation: We need to prove that if $\Psi \rhd P \mathrel{\dot{\approx}} Q$ then $\Psi \rhd P \underset{\text{smp}_{\dot{\approx}}}{\widehat{\rightsquigarrow}} Q$.

- From $\Psi \rhd P \mathrel{\dot{\approx}} Q$ we have that $\Psi \rhd P \mathrel{\dot{\widehat{\rightsquigarrow}}} Q$ by $\dot{\approx}$-E2.
- Hence $\Psi \rhd P \underset{\text{smp}_{\dot{\approx}}}{\widehat{\rightsquigarrow}} Q$ using lemmas 30.23 and 32.14.

Static extension: Follows immediately from $\dot{\approx}$-E3.

Symmetry: Follows immediately from $\dot{\approx}$-E4.

□

32.3.2 Simple bisimilarity includes weak bisimilarity

To prove that simple static implication implies weak static implication we must know that the candidate relation for weak static implication can be extended by an arbitrary assertion.

Lemma 32.16.

$$\frac{\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\lesssim\!\!\approx} Q \qquad \bigwedge \Psi' R S \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{R}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{R}}}{\Psi \rhd P \mathrel{\lesssim\!\!\approx_{\mathscr{R}}} Q}$$

Proof. From $\underset{\text{smp}}{\lesssim\!\!\approx}$-I we have to prove that for all Ψ' there exists an R and an S such that $\Psi \rhd Q \overset{\tau}{\Longrightarrow} R$, $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ R) \otimes \Psi$, $\Psi \otimes \Psi' \rhd R \overset{\tau}{\Longrightarrow} S$ and $\Psi \otimes \Psi' \rhd P \underset{\text{smp}}{\dot{\approx}} S$.

- From $\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\lesssim\!\!\approx} Q$ we obtain a Q' such that $\Psi \rhd Q \overset{\tau}{\Longrightarrow} Q'$, $(\mathscr{F}\ P) \otimes \Psi \leq (\mathscr{F}\ Q') \otimes \Psi$, and $\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q'$.
- We have that $\Psi \otimes \Psi' \rhd Q' \overset{\tau}{\Longrightarrow} Q'$.
- Moreover from $(\Psi, P, Q') \in \mathscr{R}$ we have that $(\Psi \otimes \Psi', P, Q') \in \mathscr{R}$ by the assumptions.
- Finally we solve the goal by instantiating R and S to Q'.

□

Before proving that simple simulation includes weak simulation we prove that we can extend the environment of a τ-chain by an arbitrary assertion.

Lemma 32.17. *If* WEAKEN *holds then:*

$$\textit{If } \Psi \rhd P \Longrightarrow P' \textit{ then } \Psi \otimes \Psi' \rhd P \Longrightarrow P'.$$

Proof. By induction on $\Psi \rhd P \Longrightarrow P'$

Base case $P = P'$: Follows immediately since $\Psi \otimes \Psi' \rhd P \Longrightarrow P$.

Inductive step $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P''$ and $\Psi \rhd P'' \xrightarrow{\tau} P'$:

- By the induction hypothesis we have that $\Psi \otimes \Psi' \rhd P \Longrightarrow P''$.
- Moreover from $\Psi \rhd P'' \xrightarrow{\tau} P'$ we have that $\Psi \otimes \Psi' \rhd P'' \Longrightarrow P'$ by Lemma 32.1.
- Finally we have that $\Psi \otimes \Psi' \rhd P \Longrightarrow P'$.

□

The candidate relation for the weak simulation must meet all the requirements of a simple bisimilarity.

Lemma 32.18. *If* WEAKEN *holds then:*

$$\frac{(\Psi, P, Q) \in \mathscr{R} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi, R, S) \in \mathscr{R}}{\Psi \rhd R \underset{\text{smp}_{\mathscr{R}}}{\lessapprox} S} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi, R, S) \in \mathscr{R}}{\Psi \rhd R \underset{\text{smp}_{\mathscr{R}'}}{\widehat{\rightsquigarrow}} S}}{\bigwedge \Psi' R S \Psi' a.\ \dfrac{(\Psi, R, S) \in \mathscr{R}'}{(\Psi \otimes \Psi' a, R, S) \in \mathscr{R}'} \qquad \bigwedge \Psi' R S.\ \dfrac{(\Psi, R, S) \in \mathscr{R}}{(\Psi, S, R) \in \mathscr{R}}}$$
$$\Psi \rhd P \widehat{\rightsquigarrow}_{\mathscr{R}'} Q$$

Proof. By $\widehat{\rightsquigarrow}$-I, we assume $\Psi \rhd Q \xrightarrow{\alpha} Q'$. There are two cases to be proved.

$\alpha = \tau$: We must prove that there exists a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}$

- From $(\Psi, P, Q) \in \mathscr{R}$ we have from the assumptions that $\Psi \rhd P \underset{\text{smp}_{\mathscr{R}}}{\widehat{\rightsquigarrow}} Q$.
- With $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\alpha = \tau$ we obtain a P' such that $\Psi \rhd P \stackrel{\hat{\tau}}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}$.

$\alpha \neq \tau$: We must prove that for all Ψ' there exists a P''', a P'', and a P' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'''$, $(\mathscr{F}\ Q) \otimes \Psi \leq (\mathscr{F}\ P''') \otimes \Psi$, $\Psi \rhd P''' \xrightarrow{\alpha} P''$, $\Psi \otimes \Psi' \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$, and $(\Psi \otimes \Psi', P', Q') \in \mathscr{R}$.

- From $(\Psi, P, Q) \in \mathscr{R}$ and the assumptions we obtain a P'''' such that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P''''$, $(\mathscr{F}\ Q) \otimes \Psi \leq (\mathscr{F}\ P'''') \otimes \Psi$, and $(\Psi \otimes \Psi', P'''', Q) \in \mathscr{R}$.
- From $(\Psi \otimes \Psi', P'''', Q) \in \mathscr{R}$ and the assumptions we have that $\Psi \rhd P'''' \underset{\text{smp}_{\mathscr{R}'}}{\widehat{\rightarrow}} Q$.
- With $\Psi \rhd Q \xrightarrow{\alpha} Q'$ we obtain a P' such that $\Psi \rhd P'''' \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$ and $(\Psi, P', Q') \in \mathscr{R}'$.
- Hence we can obtain a P''' and a P'' such that $\Psi \rhd P'''' \stackrel{\tau}{\Longrightarrow} P'''$, $\Psi \rhd P''' \xrightarrow{\alpha} P''$, $\Psi \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$.
- From $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P''''$ and $\Psi \rhd P'''' \stackrel{\tau}{\Longrightarrow} P'''$ we have that $\Psi \rhd P \stackrel{\tau}{\Longrightarrow} P'''$.
- Moreover from $\Psi \rhd P'''' \stackrel{\tau}{\Longrightarrow} P'''$ we have that $(\mathscr{F}\ P'''') \otimes \Psi \leq (\mathscr{F}\ P''') \otimes \Psi$ by Lemma 32.12.
- Hence with $(\mathscr{F}\ Q) \otimes \Psi \leq (\mathscr{F}\ P'''') \otimes \Psi$ we have that $(\mathscr{F}\ Q) \otimes \Psi \leq (\mathscr{F}\ P''') \otimes \Psi$ since $\leq$ is transitive.
- Moreover from $\Psi \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ we have that $\Psi \otimes \Psi' \rhd P'' \stackrel{\tau}{\Longrightarrow} P'$ by Lemma 32.17.
- Moreover from $(\Psi, P', Q') \in \mathscr{R}'$ we have that $(\Psi \otimes \Psi', P', Q') \in \mathscr{R}'$ by the assumptions.
- Finally with $\Psi \rhd P''' \xrightarrow{\alpha} P''$ all the existential quantifiers have been instantiated, and the lemma proved.

□

Lemma 32.19. *If* WEAKEN *holds then:*

$$\textit{If } \Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q \textit{ then } \Psi \rhd P \dot{\approx} Q\,.$$

Proof. By coinduction using Lemma 30.17 with $\mathscr{X}$ set to $\underset{\text{smp}}{\dot{\approx}}$.

Static implication: We need to prove that if $\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q$ then $\Psi \rhd P \underset{\approx}{\lesssim} \underset{\text{smp}}{\dot{\approx}} Q$.

- From $\Psi \rhd P \underset{\text{smp}}{\dot{\approx}} Q$ we have that $\Psi \rhd P \underset{\text{smp}\ \underset{\text{smp}}{\dot{\approx}}}{\underset{\approx}{\lesssim}} Q$ by $\underset{\text{smp}}{\dot{\approx}}$-E1.
- Hence $\Psi \rhd P \underset{\approx}{\lesssim} \underset{\text{smp}}{\dot{\approx}} Q$ using $\underset{\text{smp}}{\dot{\approx}}$-E1 and Lemma 32.16.

Simulation: Follows immediately from Lemma 32.18 and the elimination rules for $\underset{\text{smp}}{\dot{\approx}}$

Static extension: Follows immediately from $\dot{\approx}$-E3.

Symmetry: Follows immediately from $\dot{\approx}$-E4.

□

32.3.3 Weak and simple bisimilarity coincide

We can now prove the main theorem.

Theorem 32.1. $\underset{\text{smp}}{\dot{\approx}} = \dot{\approx}$

Proof. Follows directly from lemmas 32.15 and 32.19. □

33. Extending psi-calculi

One way to extend psi-calculi is to encode new operators from already existing ones. In Chapter 22 we discussed how to encode Sum, which requires the existence of a universally true $\top$-condition. In Chapter 29 we discussed how the τ-prefix (Tau) is encoded by communication over a restricted channel; one way to achieve this is by introducing a term with at least one name which is channel equivalent to itself only. In this chapter we will show how the locales in Isabelle can be used to extend the psi-calculi formalisation to include these two operators. For both, we require additional property of substitution, i.e., an axiom in the substitution locale:

$$\frac{\tilde{x} \sharp M}{M[\tilde{x} := \tilde{T}] = M}\ \textsc{SubstId}$$

This axiom is required in order for substitution to behave in the expected way – to propagate past the constructs of the encoded term, and affecting only the agents, and not the constructor itself.

33.1 Encoding Sum

With this infrastructure in place we can extend the psi-calculi locales with the requisite that there exists a condition $\top$ with the following constraints.

$$\Psi \vdash \top \qquad \textsc{Top}$$
$$p \cdot \top = \top \qquad \textsc{TopEqvt}$$

The condition $\top$ must be derivable from all assertions, and it must be equivariant. The equivariance of $\top$ allows us to derive that it has empty support, and hence any name must be fresh for it.

Lemma 33.1.

$$supp\,(\top) = \emptyset$$
$$x \sharp \top$$
$$\tilde{x} \sharp \top$$

Proof. Follows from the definition of freshness and support. □

We can now define Sum as a **case**-expression with two guards using $\top$.

Definition 33.2 (Sum)**.**

$$P + Q \stackrel{\text{def}}{=} \mathit{Cases}\,[(\top, P), (\top, Q)]$$

From this definition we can derive the standard inference rules for Sum.

Lemma 33.3. *Semantic inference rules for Sum*

$$\frac{\Psi \rhd P \longmapsto V \qquad \mathit{guarded}\ P}{\Psi \rhd P + Q \longmapsto V}\ \textsc{Sum1} \qquad \frac{\Psi \rhd Q \longmapsto V \qquad \mathit{guarded}\ Q}{\Psi \rhd P + Q \longmapsto V}\ \textsc{Sum2}$$

Proof. Follows from the CASE inversion rule and the TOP axiom. □

An inversion rule for Sum is also derivable.

Lemma 33.4. *Inversion rule for Sum*

$$\frac{\left[\begin{array}{l}\Psi \rhd P + Q \longmapsto V \\ \\ \dfrac{\left(\Psi \rhd P \longmapsto V \qquad \mathit{guarded}\ P\right)}{\mathit{Prop}} \qquad \dfrac{\left(\Psi \rhd Q \longmapsto V \qquad \mathit{guarded}\ Q\right)}{\mathit{Prop}}\end{array}\right]}{\mathit{Prop}}\ \textsc{Sum}$$

Proof. Follows from the CASE inversion rule and the TOP axiom. □

Finally, we derive a lemma which states that substitutions, both parallel and sequential, propagate over Sum like the other constructors of the framework.

Lemma 33.5.

If $|\tilde{x}| = |\tilde{T}|$ *and distinct* $\tilde{x}$ *then* $(P + Q)[\tilde{x} := \tilde{T}] = P[\tilde{x} := \tilde{T}] + Q[\tilde{x} := \tilde{T}]$.

Proof.

- $(P + Q)[\tilde{x} := \tilde{T}] = (\mathit{Cases}\,[(\top, P), (\top, Q)])[\tilde{x} := \tilde{T}]$ by Definition 33.2
- Moreover

$$\begin{array}{l}\mathit{Cases}\,[(\top, P), (\top, Q)][\tilde{x} := \tilde{T}] = \\ \mathit{Cases}\,[(\top[\tilde{x} := \tilde{T}],\ P[\tilde{x} := \tilde{T}]), (\top[\tilde{x} := \tilde{T}],\ Q[\tilde{x} := \tilde{T}])]\end{array}$$

  by Definition 24.7.
- Moreover

$$\begin{array}{l}\mathit{Cases}\,[(\top[\tilde{x} := \tilde{T}],\ P[\tilde{x} := \tilde{T}]), (\top[\tilde{x} := \tilde{T}],\ Q[\tilde{x} := \tilde{T}])] = \\ \mathit{Cases}\,[(\top,\ P[\tilde{x} := \tilde{T}]), (\top,\ Q[\tilde{x} := \tilde{T}])]\end{array}$$

by Lemma 33.1 and the SUBSTID axiom.
- Moreover $\mathit{Cases}\,[(\top,\ P[\tilde{x} := \tilde{T}]), (\top,\ Q[\tilde{x} := \tilde{T}])] \ = \ P[\tilde{x} := \tilde{T}] + Q[\tilde{x} := \tilde{T}]$ by Definition 33.2.
- Hence $(P+Q)[\tilde{x} := \tilde{T}] = P[\tilde{x} := \tilde{T}] + Q[\tilde{x} := \tilde{T}]$.

□

Lemma 33.6. *If* $\mathit{wellFormedSubst}\ \sigma$ *then* $(P + Q)\sigma = P\sigma + Q\sigma$.

Proof. By induction on σ and Lemma 33.5 □

33.2 Encoding Tau

We encode τ-prefixes in psi-calculi through an instance with a injective equivariant function nt from names to terms satisfying $\mathit{supp}\ (nt\ a) = \{a\}$. We extend the standard set of constraints with following.

$$p \cdot nt\, x = (nt\, p \cdot x) \qquad \text{NTEQVT}$$
$$\mathit{supp}\ (nt\, x) = \{x\} \qquad \text{NTSUPP}$$
$$\Psi \vdash (nt\, x) \leftrightarrow M \Leftrightarrow M = (nt\, x) \qquad \text{NTEQ}$$

The NTEQ axiom is crucial as it ensures that a term generated by nt is only channel equivalent to itself. We can now define the τ-prefix in the following way, where the *THE* operator is the choice operator – the expression *THE x. Prop x* chooses any x such that *Prop x* holds.

Definition 33.7.

$$\tau.P \stackrel{\text{def}}{=} \mathit{THE}\ P'.\ \exists x.\ x \sharp P \wedge P' = (\nu x)(\underline{nt\ x}(\lambda\varepsilon)nt\, x.\mathbf{0} \mid \overline{(nt\ x)}(nt\ x).P)$$

The following lemma can then be derived, which allows us to unfold the definition of the τ-prefix, such that the bound name x is sufficiently fresh.

Lemma 33.8.

$$\exists x.\ x \sharp P \wedge x \sharp \mathscr{C} \wedge \tau.P = (\nu x)(\underline{nt\ x}(\lambda\varepsilon)nt\, x.\mathbf{0} \mid \overline{(nt\ x)}(nt\ x).P)$$

Proof. A sufficiently fresh name is chosen. Definition 33.7 is then unfolded and the bound name it provides is alpha-converted to the new name. □

This lemma can then be used to derive the following lemmas which reason about how substitution, permutation, and freshness operate on τ-prefixed terms. Moreover, we have that any τ-prefixed agent is guarded.

Lemma 33.9.

$$p \cdot \tau.P = \tau.(p \cdot P)$$
$$\textit{If distinct } \tilde{x} \textit{ and } |\tilde{x}| = |\tilde{T}| \textit{ then } (\tau.P)[\tilde{x} := \tilde{T}] = \tau.P[\tilde{x} := \tilde{T}].$$
$$\tilde{x} \sharp \tau.P = \tilde{x} \sharp P$$
$$\textit{guarded } (\tau.P)$$

Proof. Follows by unfolding the τ-prefixes using Lemma 33.8 ensuring that the bound names are fresh for everything in the proof context. The axiom NTSUBST is used to remove the substitutions of all terms of the form $nt\,x$, as we know that x is sufficiently fresh. □

The next step is to derive the semantics of the τ-prefix. Ideally, we would like a rule similar to the INPUT and OUTPUT rules – a prefixed agent has that prefix on the label and the agent under the prefix as its derivative. However, the COMM-rule provides the derivation

$$\Psi \rhd \tau.P \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid P) \qquad \text{where } x \sharp P$$

as the trailing $\mathbf{0}$ and the bound name x of Definition 33.7 remain after the derivation. The derivative is not P, but $(\nu x)(\mathbf{0} \mid P)$, which is structurally congruent to P as $x \sharp P$. More formally, we have the following semantic rule for τ-prefixes.

Lemma 33.10. *Semantic rule for the τ-prefix.*

$$\frac{}{\exists P'.\ \Psi \rhd \tau.P \xrightarrow{\tau} P' \wedge \Psi \rhd P \mathrel{\dot{\sim}} P'}\ \textsc{Tau}$$

Proof. The τ-prefix is unfolded using Lemma 33.8 and the existential quantifier P' is set to $(\nu x)(\mathbf{0} \mid P)$, which is strongly bisimilar to P by Lemma 28.21. □

We also need an elimination rule for τ-prefixes.

Lemma 33.11.

$$\frac{\Psi \rhd \tau.P \xrightarrow{\tau} P'}{\Psi \rhd P \mathrel{\dot{\sim}} P'}\ \textsc{TauE1} \qquad \frac{\Psi \rhd \tau.P \xrightarrow{\tau} P'}{\mathit{supp}\ P' = \mathit{supp}\ P}\ \textsc{TauE2}$$

The τ-prefix is unfolded using Lemma 33.8. The SCOPE, PAR, INPUT, and OUTPUT inversion rules are used to derive the derivative that P' must be equal to $(\nu x)(\mathbf{0} \mid P)$ where $x \sharp P$, which is bisimilar to P by Lemma 28.21, and also has the same support.

These rules ensure that any derivative of a τ-prefixed agent is bisimilar to the agent under the prefix and has equal support.

Moreover, we need to prove that a τ-prefixed agent cannot do any other actions than a τ-action.

Lemma 33.12. *A τ-prefix can never generate an input or an output action.*

If $\Psi \rhd \tau.P \xrightarrow{MN} P'$ *then False.* *If* $\Psi \rhd \tau.P \xrightarrow{\overline{M}(\nu\tilde{x})N} P'$ *then False.*

Proof. Follows by unfolding the τ-prefixes using Lemma 33.8. The SCOPE, PAR, INPUT and OUTPUT inversion rules then ensure that there are no possible transitions. □

33.3 Proving the τ-laws

In this section we will prove the standard τ-laws from the pi-calculus.

1. $P \dot{\approx} \tau.P$ in psi-calculi with weakening.
2. $P + \tau.P \approx \tau.P$.
3. $\alpha.\tau.P \approx \alpha.P$ in psi-calculi with weakening.
4. $\alpha.P + \alpha.(\tau.P + Q) \approx \alpha.(\tau.P + Q)$.

It is immediately clear that all τ-laws require the τ-prefix, and that τ-laws 2 and 4 require Sum. Moreover, as was discussed in the beginning of Chapter 29 it is required that weakening holds for τ-law 1 to be satisfied. The same is true for law 3 to hold, as whenever the α-prefix has been discharged, the derivatives P and $\tau.P$ must be weakly bisimilar.

33.3.1 Encoding prefixes

The τ-laws 3 and 4 quantify all possible prefixes over α, in a similar way as is done for the labels of the transition system. In order to encode this into Isabelle we use a similar technique as in Section 25.1.

Definition 33.13. *An agent prefix, denoted α can either be an input, an output, or a τ-prefix.*

```
nominal_datatype β prefix = pInput β "name list" β
                          | pOutput β β
                          | pTau
```

We define a function which takes a prefix and an agent and creates an agent with that prefix. In the case that the prefix is an input-prefix, the sequence of names is bound into the agent.

Definition 33.14. *The bindprefix function takes a prefix and an agent and appends that prefix to that agent.*

$$\begin{aligned} \mathit{bindPrefix}\ (\mathit{pInput}\ M\ \tilde{x}\ N)\ P &= \underline{M}(\lambda\tilde{x})N.P \\ \mathit{bindPrefix}\ (\mathit{pOutput}\ M\ N)\ P &= \overline{M}N.P \\ \mathit{bindPrefix}\ \mathit{pTau}\ P &= \tau.P \end{aligned}$$

We will use the notation $\alpha.P$ for $\mathit{bindPrefix}\ \alpha\ P$. We will also use the standard notation for prefixes, rather than using the *pInput*, *pOutput*, and *pTau* constructors.

This function allows us to reason uniformly about arbitrary prefixes on agents, and hence lemmas which refer to prefixes need only be declared once, and not once for each prefix type. The semantics of the prefixes do differ, and an inversion rule for prefixes is required for some proofs.

Lemma 33.15. *Inversion rule for prefixed agents.*

$$\frac{\left[\begin{array}{l} \Psi \rhd \alpha.P \xrightarrow{\beta} P' \\ \\ \bigwedge M\ \tilde{x}\ N\ K\ \tilde{T}.\ \dfrac{\begin{pmatrix}\Psi \vdash M \leftrightarrow K & \mathit{set}\ \tilde{x} \subseteq \mathit{supp}\ N \\ |\tilde{x}| = |\tilde{T}| & \mathit{distinct}\ \tilde{x}\end{pmatrix}}{\mathit{Prop}\ (\underline{M}(\lambda\tilde{x})N)\ (\underline{K}N[\tilde{x} := \tilde{T}])\ (P[\tilde{x} := \tilde{T}])} \\ \\ \bigwedge M\ N\ K.\ \dfrac{\left(\Psi \vdash M \leftrightarrow K\right)}{\mathit{Prop}\ (\overline{M}N)\ (\overline{K}N)\ P} \qquad \dfrac{\left(\Psi \rhd P \mathrel{\dot{\sim}} P'\right)}{\mathit{Prop}\ (\tau)\ (\tau)\ P'} \end{array}\right]}{\mathit{Prop}\ \alpha\ \beta\ P'}\ \textsc{Prefix}$$

With these lemmas in place, we can prove the τ-laws from the pi-calculus.

Theorem 33.1. *The following τ-laws hold:*

1. $P \mathrel{\dot{\approx}} \tau.P$ *in psi-calculi with weakening.*
2. $P + \tau.P \approx \tau.P$.
3. $\alpha.\tau.P \approx \alpha.P$ *in psi-calculi with weakening.*
4. $\alpha.P + \alpha.(\tau.P + Q) \approx \alpha.(\tau.P + Q)$.

Proof. The proofs are done in the standard way – simulation and static implication lemmas are proved using the required inversion and semantic rules; a candidate relation is chosen for the bisimilarity proofs which use bisimulation up-to techniques as the derivative of the τ-prefix is bisimilar but not identical to the agent under the prefix; finally the lemmas are closed under substitution to obtain the congruence results. □

The τ-laws which require weakening are laws 1 and 3, and the only place where the WEAKEN axiom from Chapter 32 is required is when proving one of the weak static implication lemmas for τ-law 1.

Lemma 33.16. *If* $(\Psi, \tau.P, P) \in \mathscr{R}$ *then* $\Psi \rhd \tau.P \underset{\text{smp}_{\mathscr{R}}}{\lessapprox} P$

Proof.

- We have that $(\mathscr{F}\ (\tau.P)) \otimes \Psi \leq ((\nu\varepsilon)\Psi)$ since τ-prefixed agents are guarded, and thus have empty frames.
- Moreover we have that $((\nu\varepsilon)\Psi) \leq (\mathscr{F}\ P) \otimes \Psi$ by WEAKEN.
- Finally we have that $(\mathscr{F}\ (\tau.P)) \otimes \Psi \leq (\mathscr{F}\ P) \otimes \Psi$ by transitivity of static implication.
- Since $\Psi \rhd P \Longrightarrow P$ and $(\Psi, \tau.P, P) \in \mathscr{R}$ we have that $\Psi \rhd \tau.P \underset{\text{smp}_{\mathscr{R}}}{\lessapprox} P$ by Definition 32.5.

□

This lemma is the only one which explicitly uses the WEAKEN axiom, hence τ-law 1 requires weakening, and also τ-law 3 since its proof depends on τ-law 1.

34. Conclusions

In this part of the thesis we introduced and formalised psi-calculi, a general framework which captures a wide variety of existing calculi. A few examples are the applied pi-calculus, by Abadi and Fournet [7], the spi-calculus by Abadi and Gordon [8], the explicit fusion calculus by Wischik and Gardner [40], CC-Pi by Buscemi and Montanari [28], and extended pi-calculi by ourselves [48]. For a more detailed exposition, see [17]. The formalisation is substantially more complex than for CCS or the pi-calculus, which is a price for the high expressiveness of the framework.

A fully formalised framework has several benefits. The most obvious is that we know with absolute certainty that our theorems are correct. This claim is not to be taken lightly since there is a clear need for robust theories. As the complexity of the calculi increases, so does the complexity of their proofs, and during our formalisation efforts we found bugs in the meta-theory of several process calculi. Another benefit is that formalised theories are extensible – the ramification of changes made to the theories are instantly apparent, making it completely safe to modify the theories without risking inconsistencies.

34.1 Inconsistent process calculi

Previous attempts to create uniform frameworks are most notably the applied pi-calculus, CC-Pi, and the extended pi-calculus. All of these have turned out to have mistakes in their meta-theory.

34.1.1 The Applied pi-calculus

The applied pi-calculus is an extension of the pi-calculus parametrised by a signature Σ for data terms and an equational theory $\vdash_\Sigma$ over Σ, and introduces *active substitutions* $\{^M/_x\}$ of data terms for variables. These can be introduced by the inferred structural rule $(\nu x)(\{^M/_x\} \mid A) \equiv A[x := M]$. There are names a, b, c distinct from variables x, y, z where only variables can be substituted, and a simple type system to distinguish names and variables of channel type from other terms of base type. Only names of channel type can be used as communication channels.

The labeled semantics for the applied pi-calculus turns out to be non-compositional. Consider the closed (extended) applied pi-calculus agents

$$A = (\nu a)(\{^a/_x\} \mid x.b.\mathbf{0}) \qquad B = (\nu a)(\{^a/_x\} \mid \mathbf{0})$$

where we omit the objects of the prefixes. They have the same frame and no transitions, and are thus semantically equivalent. But a context can contain x and can therefore use the active substitution to communicate with A. Formally, let $R = \overline{x}.\mathbf{0}$ and $\Downarrow b$ the usual weak observation or barb. We have by scope extension that $A|R \equiv (\nu a)(\{^a/_x\} \mid x.b.\mathbf{0} \mid \overline{x}.0) \Downarrow b$, but it is not the case that $B|R \Downarrow b$. Therefore, *no* observational equivalence that is preserved by all contexts and satisfies scope extension can be captured by the labeled semantics. In this, Theorem 1 of [7] is incorrect; the labeled and observational equivalences do in fact not coincide, nor is labeled equivalence a congruence. This is relevant for other papers that use or develop the labeled semantics, e.g. [43, 51, 34, 32].

34.1.2 The concurrent constraints pi-calculus

The concurrent constraint pi-calculus (CC-Pi) [29] can be seen as a process calculus with **ask** and **tell** statements, parametrised by a constraint system in the form of a named c-semiring. Such a constraint system contains names, name fusion constraints and a name hiding operator ν, and supports general constraint semirings, e.g. Herbrand constraints. The semantics is given by a structural congruence and a reduction relation. There is also a labeled operational semantics, but it is in fact not compositional. Consider the CC-Pi agents

$$P = (\nu b, x)(x = b \mid \overline{a}x\,.\,b\,.\,c) \qquad Q = (\nu b, x)(x = b \mid \overline{a}x)$$

(where insignificant objects are omitted). They have the same constraint store, $x = b$, and the same transitions in all constraint contexts. However, they do not have the same transitions in all process contexts: a parallel context $R = a(y).\overline{y}$ tells the difference:

$$P \mid R \xrightarrow{\tau} \xrightarrow{\tau} (\nu b)(x = b \mid x = y \mid c) \xrightarrow{c}$$

while $Q \mid R$ of course has no such c transition. Thus Theorem 1 of [29] is incorrect: open bisimilarity is not a congruence.

34.1.3 Extended pi-calculi

Our earlier work includes the extended pi-calculi – a framework similar to the applied pi-calculus, but simpler and more expressive [48]. The applied pi-calculus has two levels of agents (pure and extended), two types of

binders (variables and names), and a semantics which makes use of structural congruence. Extended pi-calculi has only one level of agents, only use names as binders, and has a semantics which does not use structural congruence. Like the applied pi-calculus extended pi-calculi has the notion of an alias, written $\{^M/_x\}$, which can syntactically appear in the agent.

As in the applied pi-calculus, the frame of an agent is all of its aliases and binders not under a prefix. Frames can be used to derive equivalence of terms. Exactly how equivalence is derived must be specified when an instance of extended pi-calculi is formulated – a ternary relation, called an EF-relation, of frames and two terms must be supplied (just as the relation $\vdash$ must be supplied in a psi-calculus instance) where $F \vdash M = N$ states that the frame F implies that M and N are equivalent. This EF-relation must meet a certain set of constraints, such as idempotence of frames. For a complete list see Section 4 of [48].

As it turns out, the constraints required for the EF-relation are not enough. This fact was discovered after the publication of [48]. Here is an example to demonstrate that scope extension does not hold. Define an EF-relation $\vdash$ such that $\{^{f(a)}/_a\} \vdash f(a) = a$ for all a. Moreover, let $\{^{f(a)}/_a\} \cup \{^{f(b)}/_b\} \vdash a = b$. By closing this relation under the requirements of EF-relations, we obtain a valid instance of extended pi-calculi.

Let N be any term such that $a \sharp N$ and $b \sharp N$, and consider the agent P defined as:

$$P \stackrel{\text{def}}{=} (\nu a,\ b)(\{^{f(a)}/_a\} \mid \{^{f(b)}/_b\} \mid \overline{a}N.\mathbf{0} \mid b(N).\mathbf{0})$$

The agent P can do a τ-action since the derivation

$$\{^{f(a)}/_a\} \cup \{^{f(b)}/_b\} \rhd \overline{a}N.\mathbf{0} \mid b(N).\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}$$

can be derived.

Theorem 1 from [48] states that all structurally congruent agents are also bisimilar; hence the agent

$$Q \stackrel{\text{def}}{=} (\nu a)(\{^{f(a)}/_a\} \mid \overline{a}N.\mathbf{0}) \mid (\nu b)(\{^{f(b)}/_b\} \mid b(N).\mathbf{0})$$

is strongly congruent to P by the laws of scope extension and associativity and commutativity of Parallel. However, the agent Q cannot do a τ-action, there is no way for the names a and b to be equated as they are blocked by the binders, and hence Theorem 1 from [48] is incorrect. Theorem 2, which states that bisimilarity is a congruence, builds on results from Theorem 1 and is most likely also incorrect.

A solution to this problem is to add an additional requirement on EF-relations.

Suppose that $a \sharp M$, F and $b \sharp N$, G and $F \otimes G \vdash M = N$. Then there exists a K such that a, $b \sharp K$ and $F \otimes G \vdash M = K$.

This requisite is called interpolation, and it states that if a frame which equates two terms M and N can be split into two parts F and G, the name a fresh for F and M, and the name b fresh for G and N, then there must exist a term K equal to both M and N, but which contains neither a or b.

With this fix, the counterexample above does not hold – a τ-action can be derived from Q by communicating over K. Theorems 1 and 2 of [48] with interpolation have been verified in Isabelle.

Even though this patches the calculus, it does not make it easy to work with. Determining whether or not a relation is an EF-relation turns out to be very difficult even for simple cases. Simple properties such as frame idempotence are difficult to prove as the same frame has different alpha-variants of its aliases which can interact in unexpected ways. Also, the interpolation requisite turns out to be difficult to establish for candidate instances.

Psi-calculi also impose requisites on its logical entailment relation, but these requisites only reason about the assertions of the frames and not their binders; entailments are formulated on the assertions, which correspond to aliases in extended pi-calculi. This design decision pays off – in psi-calculi we end up with an abelian monoid parametrised with static equivalence and assertion composition. This makes the requisites for psi-calculi much easier to prove for any given instance. Moreover, psi-calculi can express all of extended pi-calculi, and more. For these reasons development of extended pi-calculi has been abandoned.

34.2 The Psi-calculi formalisation

The formalisation of psi-calculi took nearly two years, starting with the formalisation of extended pi-calculi. What sets this formalisation apart from the ones for CCS and the pi-calculus is that the theories for psi-calculi were developed simultaneously with the formalisation. This has advantages and disadvantages.

Machine checking proofs take time, and the theories being developed on paper will invariably be further into the development stage than their machine checked counterparts. Any bugs found in the theories, either as a part of the formalisation efforts or as a result of the standard development cycle, must be corrected and this can potentially lead to large amounts of work being thrown away, and large amounts of work to reimplement the theories. However, the chances of finding bugs early in the theoretical developments increase with the use of a theorem prover, as the proofs are constantly being verified. In this there is a similarity to rapid prototyping in software development, where design bugs are weeded out by experiments

Part	Lines of code
Agents	1336
Frames	2180
Semantics	9373
Strong bisimilarity	2733
Structural congruence	3032
Weak bisimilarity	7111
Weak Congruence	2395
Weakening	754
Total	28914

Figure 34.1: The size of the different parts of the Isabelle formalisation of the psi-calculi meta-theory.

as early as possible. The amount of backtracks required for psi-calculi were not too numerous. One backtrack was severe. We had finished our formalisation; all proofs were done; strong bisimilarity was proven to be a congruence. At the time, requisites on entailment were formulated in terms of frames rather than assertions, and for the same reasons as for extended pi-calculi, it turned out to be too difficult to develop instances of the framework. This led to the design change that the requirements on the entailment relation are formulated on assertions, and not frames, which led to a complete rewrite of the semantics and all of the Isabelle theories. The lesson learned is that a theorem prover will only prove theorems correct, it will not help determining their relevance.

Machine checked formalisations encourage developers to keep theories simple – the simpler the theories, the simpler they are to formalise, and the simpler they are to use. A good example of this is how extended pi-calculi and psi-calculi treat the entailments of their conditions. Another example is that we prefer calculi without structural congruence in the semantics. All of the bugs described in Section 34.1 involve structural congruence in one way or other.

The size of the psi-calculi formalisation is on par with the size for the pi-calculus, but for the pi-calculus we formalised two semantics – the late and the early versions, and for psi-calculi we have only formalised an early semantics. The size of the different parts of the formalisation can be found in Figure 34.1. In particular, the formalisation of the operational semantics is significantly larger than for the previous formalisations. The reason is that all of the extra infrastructure for induction and inversion rules, which Isabelle derives automatically for the simpler calculi, must be done manually

for psi-calculi. Another observation is that the formalisation for weak congruence is not as large as for weak bisimilarity, which strengthens our claim that weak congruence allows for more reuse of the results than in our formalisation of the pi-calculus and CCS.

The time spent on the formalisation was roughly two years, including the work with extended pi-calculi, the psi-calculi attempt where we had the wrong equivalence, and early attempts to reason about binding sequences. An estimate of the time required to develop the psi-calculi formalisation we have today is about a year.

34.2.1 Example of a variant

There is an alternative way to define bisimilarity for psi-calculi as a binary relation preserved by parallel contexts.

Definition 34.1 (Context bisimilarity)**.** *A context bisimilarity $\mathscr{R}$ is a binary relation on agents such that $\mathscr{R}(P,Q)$ implies all of*

1. *Static equivalence: $\mathscr{F}(P) \simeq \mathscr{F}(Q)$*
2. *Symmetry: $\mathscr{R}(Q,P)$*
3. *Extension of contextual assertion:*
   $\forall \Psi.\ \mathscr{R}(\langle\!|\Psi|\!\rangle \mid P,\ \langle\!|\Psi|\!\rangle \mid Q)$
4. *Simulation: for all α, P' such that $\mathrm{bn}(\alpha) \sharp Q$ there exists a Q' such that*

$$\mathbf{1} \rhd P \xrightarrow{\alpha} P' \quad \Longrightarrow \quad \mathbf{1} \rhd Q \xrightarrow{\alpha} Q' \wedge \mathscr{R}(P', Q')$$

We define $P \mathrel{\dot{\sim}}_{\mathrm{ctx}} Q$ to mean that there exists a context bisimulation $\mathscr{R}$ such that $\mathscr{R}(P,Q)$.

Such a definition is more in line with standard contextual bisimulations, and also the way bisimilarity is defined in the applied pi-calculus. The drawback is that it relies on the Parallel operator of the calculus for its definition. For conducting formal proofs our experience is that Definition 22.10 is preferable. We have shown that these bisimilarities coincide, i.e., the definitions result in the same bisimulation equivalence:

Theorem 34.1 (Bisimilarity and context bisimulation coincide)**.** $\mathrel{\dot{\sim}} = \mathrel{\dot{\sim}}_{\mathrm{ctx}}$

We assigned one person to the paper proof, and another to formalise it in Isabelle. The paper proof is around 70 lines of text and took around two hours to write. It is a thorough proof, but contains hand waving remarks such as “follows trivially from the definitions”, or “proof by intimidation :)”. The Isabelle proof took around eight hours to write. One main advantage of having completely formalised theories is that additions such as these do not require a particularly large effort – the ground work has already been done.

From this it would appear that using Isabelle requires an increased effort by a factor of four. In reality, for different proofs this factor varies wildly. In some cases the paper proof took a very long time. An example is the preservation property for strong congruence where the precise formulation of Lemmas like 27.36 turned out to be elusive. Also, the main effort of the development have been on other things than proofs of any kind. Overall we estimate that the Isabelle formalisation has required roughly the same effort as the non-Isabelle development, in other words, the price to have completely formal theories was to double the development effort.

34.2.2 Weak equivalences

The weak equivalences for psi-calculi were difficult to get right. On several occasions we thought we had a good definition of weak bisimilarity which turned out either to be non compositional or non transitive. Most of the motivating examples in Section 29.2 were found as a result of an unsuccessful formalisation attempt, motivating the search for a counter-example.

It is important to remember that even though theorem provers provide absolute certainty that a proof is correct, they do not guarantee that the right thing is being proven. After a few months, we had arrived at a weak bisimulation relation which is a congruence. To check that it relates precisely the right agents we provided an independent definition in the form of barbed congruence. As it turned out the equivalences did not coincide, and the last example of Section 29.2 was found. The proof that barbed bisimilarity coincides with the final definition of weak bisimilarity has not yet been machine checked.

34.3 Extensibility

One of the main advantages of having fully mechanised theories is that the effects of any changes to the formalisation is instantly apparent. This section will discuss a few changes that have been made to the formalisation after all theorems were proven.

34.3.1 Case

In a previous version of psi-calculi, the CASE rule looked as follows:

$$\textsc{Old-Case}\ \frac{\Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\tau} P_i}$$

In this rule, the choice of which branch to take in a **case** statement yields an internal action, after which the process P evaluates as usual. An implication is that the requirement that P is guarded can be omitted. We initially adopted this rule since it admits simpler induction proofs. At a quite late stage we decided to change it to the present rule in Figure 22.1, since this more closely resembles what is used in similar calculi. The change prompted a rework of the entire proof tree from the semantics and up. The total effort was approximately eight hours, after which we had complete certainty that the new rule does not cause any problems.

34.3.2 The empty process

In the original paper on psi-calculi we defined the deadlocked agent to be the unit assertion, i.e.

$$\mathbf{0} \stackrel{\text{def}}{=} (\!|\mathbf{1}|\!)$$

An unintuitive side effect of this definition is that the empty agent is not guarded, as guarded agents must not have top level assertions which are not under a prefix. Two possible fixes are to either add the empty agent to the primitive psi-calculi agents, or to encode the empty agent as a **case** with no choices. We chose the former.

Again, this is a change at the top of the proof tree, and hence all proofs have to be rechecked. This took about an hour to implement, and a coffee break while Isabelle rechecked the proofs.

34.3.3 Axioms for substitution

The substitution axioms are presented in Section 24.2.1 have been subject to change. Originally there were four additional axioms, which we today know are not required. The four axioms were the following:

1. If $a \sharp X[\tilde{x} := \tilde{T}]$ and $a \sharp \tilde{x}$ then $a \sharp X$
2. If $a \sharp X$ and $a \sharp \tilde{T}$ then $a \sharp X[\tilde{x} := \tilde{T}]$
3. If $\tilde{x} \sharp X$ then $X[\tilde{x} := \tilde{T}] = X$
4. If $\tilde{x} \sharp \tilde{y}$ and $\tilde{y} \sharp \tilde{T}$ then $X[\tilde{x}\tilde{y} := \tilde{T}\tilde{U}] = (X[\tilde{x} := \tilde{T}])[\tilde{y} := \tilde{U}]$

In all cases, the lengths of the name vector and the term vector are required to be the same, and the name vector distinct.

Axiom 1 represents a lower bound of the support of the substitution function – it says that substitution may not remove names which are not affected by the substitution. This axiom is used to determine that names are fresh for the object and the derivative in derivations using the INPUT rule.

Axiom 2 represents an upper bound of the support of the substitution – the result of a substitution may not have greater support than the originating term and the terms being substituted into it. This axiom turned out to be derivable since substitution is equivariant, as is proved in Lemma 24.5. This lemma states that the upper bound includes the names being substituted away, but this is unproblematic. We can always assume that the names $\widetilde{x}$ of any substitution $X[\widetilde{x} := \widetilde{T}]$ are sufficiently fresh, as they always originate from the bound names of an input prefix.

Axioms 3 and 4 were needed for a previous definition of strong equivalence and weak congruence. Strong equivalence was defined as strong bisimilarity closed under a single parallel substitution, and not a sequence as in Definition 27.48. The proof of Lemma 27.51 then requires Axiom 4 – the definition of strong equivalence requires that bisimilarity is closed under a single substitution. In order for bisimilarity to be preserved by Input it must be preserved by an additional substitution, and these must be joined into a single substitution. Axiom 4 ensures this. With the current definition of sequential substitution, Definition 27.43, we have this property by definition.

Axiom 3 was required to ensure that strongly equivalent agents are also strongly bisimilar – without it, we cannot prove that an empty substitution has no effect. Again, this is a property we get for free with the current definition of strong equivalence. This axiom is still needed when we encode Sum or the τ-prefix, as described in Chapter 33, to allow substitution to propagate past the new constructors.

From the original seven axioms for substitution, there following three remain:

$$p \cdot X[\widetilde{x} := \widetilde{T}] = (p \cdot X)[p \cdot \widetilde{x} := p \cdot \widetilde{T}] \qquad \textsc{SubstEqvt}$$

$$\frac{|\widetilde{x}| = |\widetilde{T}| \qquad \mathit{distinct}\ \widetilde{x} \qquad \mathit{set}\ \widetilde{x} \subseteq \mathit{supp}\ X \qquad y \sharp X[\widetilde{x} := \widetilde{T}]}{y \sharp \widetilde{T}} \qquad \textsc{SubstFresh}$$

$$\frac{|\widetilde{x}| = |\widetilde{T}| \qquad \mathit{distinctPerm}\ p \qquad \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x}) \qquad (p \cdot \widetilde{x}) \sharp X}{X[\widetilde{x} := \widetilde{T}] = (p \cdot X)[p \cdot \widetilde{x} := \widetilde{T}]} \qquad \textsc{SubstAlpha}$$

The requirement of the SubstEqvt axiom is clear – all constructors in a nominal formalism must be equivariant, or permutations would not propagate past them. The SubstAlpha axiom is required to allow arguments about input prefixes up to alpha-equivalence – if the bound names of a prefix are alpha-converted, then the corresponding conversion must be possi-

ble to apply to the substitutions of the INPUT rule as well. The final axiom is the SUBSTFRESH axiom. It turns out that without it, bisimilarity in psi-calculi would not satisfy scope extension. Consider the agent

$$\underline{M}(\lambda x)N.x$$

where names are terms and can be used as channels, and the agent x abbreviates $\underline{x}(\lambda\epsilon)K.\mathbf{0}$ One possible transition for this agent is

$$\underline{M}(\lambda x)N.x \xrightarrow{\underline{M}N[x:=b]} x[x:=b]$$

where b is any name. We know from the INPUT rule that x must be in the support of N. Let $N' = N[x:=b]$. Assuming that SUBSTFRESH does not hold, it is possible that b is fresh for N', even though b appears in the derivative of the transition as $x[x:=b] = b$.

Now consider the agents:

$$P \stackrel{\text{def}}{=} \underline{M}(\lambda x)N.\underline{x} \mid (\nu b)\overline{b}$$
$$Q \stackrel{\text{def}}{=} (\nu b)(\underline{M}(\lambda x)N.\underline{x} \mid \overline{b})$$

Clearly P and Q should be bisimilar, since bisimilarity should satisfy scope extension. However, we have that

$$Q \xrightarrow{\underline{M}N'} (\nu b)(\underline{b} \mid \overline{b}) \xrightarrow{\tau} (\nu b)(\mathbf{0} \mid \mathbf{0}),$$

where the first transitions is inferred by the INPUT and the SCOPE rules, and the second transition by the COMM and the SCOPE rules.

$$P \xrightarrow{\underline{M}N'} \underline{b} \mid (\nu b)\overline{b}$$

which has no further tau action. Therefore P and Q cannot be bisimilar, and scope extension fails. If the SUBSTFRESH axiom holds, then we know that b must occur in N', and hence because of the side condition in SCOPE, Q cannot do the input action without first alpha-converting its restricted name b. The agent Q', which is the agent Q with b alpha-converted for c, then has the transition

$$Q' \xrightarrow{\underline{M}N'} (\nu c)(\underline{b} \mid \overline{c}),$$

and from here, there are no further τ-actions, and P, Q, and Q' are bisimilar.

The realisation that the four axioms mentioned at the beginning of this section are superfluous dawned on us three days prior to the printing of this thesis. Axioms 1 and 2 were unproblematic to remove, whereas Axioms 3

and 4 required a slight adjustment of the definition of strong equivalence and weak congruence. The total amount of work to make these changes was about six hours. If our formalism had just existed on paper, we would never have dared make this change so close to the deadline, once again the machine checked proofs ensure that we can make these types of changes and be completely certain that our results still hold.

Axioms 3 and 4 are intuitive, and any reasonable substitution function should satisfy them. It is possible to think of a substitution function which in addition to substituting the names also rewrite the agents to some normal form using the laws of structural congruence, but such substitutions are non-standard. If the substitution functions for an instance of psi-calculi satisfies Axioms 3 and 4, then strong equivalence and weak congruence can de defined by closing their respective bisimulation relations under a single parallel substitution, rather than a sequential one. The proof that they are congruences still holds.

34.4 Future work

There are two main areas for future work, the first is to extend the theories of psi-calculi, and mechanise the remaining parts, the other is to create automatic support for bisimilarity checking and instance verification.

34.4.1 Barbed congruence

In [47] we defined weak barbed bisimilarity and proved that it coincides with weak bisimilarity. Barbed bisimilarity has not been formalised for any of the calculi presented in this thesis.

The reduction semantics of barbed bisimilarity contains structural congruence – a construction which we go to great lengths to avoid for the calculi we formalise, but for barbed equivalences it is necessary. Formalising the proof that barbed bisimilarity and weak bisimilarity coincide would require that the reduction semantics is defined in Isabelle.

34.4.2 Automatic instance verification

When creating a psi-calculus instance, a user has to provide terms, assertions, and conditions; define substitution functions for each of these which meet the requirements presented in Section 24.2.1; the three equivariant nominal morphisms presented in Section 24.3 must be provided, and the entailment relation and static equivalence meet the requisites listed in Section 24.6. We are currently working on having Isabelle automatically verify if a candidate instance is a psi-calculus instance or not. The generality of

the framework naturally makes this undecidable, but Isabelle should try to prove as much as possible, and leave the unproved subgoals for the user to handle manually. The proofs for the substitution functions in particular require quite simple heuristics, as long as the terms, assertions, and conditions are inductively defined or nominal datatypes.

34.4.3 Types

All process calculi covered in this thesis are untyped. Type systems are very useful for writing specifications or programs in process calculi. One area of concurrency which has had a significant benefit from types is security. The idea of using types to verify security properties of protocols was first introduced by Abadi [6] and subsequent work has verified a variety of protocols. The idea is that if a program type checks, it fulfills the desired security properties.

We have begun work on typing psi-calculi. Our goal is that the type system should be as general as psi-calculi themselves – in the same way as for terms, assertions, and conditions, types should be members of nominal sets, a typing judgment should be provided, which similarly to the entailment relation must have a set of constraints imposed on it. This set should be kept as small as possible such that the typing system meets the typical properties, e.g. subject reduction.

Part V:
Conclusions

35. Conclusions

This thesis presents the most extensive formalisations of process calculi ever done in a theorem prover. Related approaches were discussed in Chapter 3, but these are small in comparison.

The formalisations of CCS and the pi-calculus means that we have complete confidence in their meta-theories. We did not prove any new results or find any bugs in existing proofs, but having their meta-theory checked by a theorem prover ensures that nothing has been missed. We did find parts of the theories where the proofs are vague in the sence that it is not obvious how to make them completely rigorous.

Formalising the proofs for psi-calculi in parallel with the development has turned out to be invaluable, and we would most likely not have finished successfully without it. Throughout the development we have uncountable times stumbled over slightly incorrect definitions and not quite correct lemmas, prompting frequent changes in the framework. For example, our mistake in [48] mentioned in Section 34.1.3 was found during proof mechanisation and would probably not have been found at all without it; at that time we had completed a manual “proof" which turned out incorrect.

This chapter concludes the thesis. We will discuss the impact of our work, and elaborate on what features we would like to see in the theorem provers of tomorrow to facilitate the types of formalisations presented in this thesis.

35.1 Nominal Isabelle

All of the theories in this thesis have been formalised using the nominal datatype package of Isabelle/HOL. When we started our formalisation of the pi-calculus, Nominal Isabelle existed only as a handful of theory files, created by Urban, demonstrating a proof of concept by formalising the Church-Rosser theorem of the lambda-calculus. At the time there were no notions of nominal datatypes, nominal functions, or nominal induction or inversion rules. Our original formalisation of the pi-calculus encoded all of these from the bottom up using the techniques provided in Urban's theories.

The automatic tactics of Nominal Isabelle gradually became better, and with each release we discarded thousands of lines of proof code, as the latest release could prove them automatically. With psi-calculi we find our-

selves in a similar situation as when we started on the pi-calculus; a lot of the proofs for the theoretical infrastructure currently have to be proven by hand, but we expect developments of Isabelle to prove more parts automatically.

35.1.1 The future of binders

Nominal Isabelle does not currently have support for binding sequences. These must be coded by hand, as discussed in Chapter 23. Moreover, providing support for binding sequences is not enough in the general case. A simple example is if we were to add types to psi-calculi. An input prefix would then have the form

$$M(\lambda\ \widetilde{x_i} : \widetilde{T_i})N.P$$

where $\widetilde{x_i} : \widetilde{T_i}$ is a sequence of pairs of names and types, where the names bind into N and P as usual, but possibly also into the types. Another example is residuals, which are formally defined in Definition 25.3. Residuals are paired with a wrapper function in Definition 25.5 to allow us to reason about them without splitting the actions into the types with binders and those without. These two examples demonstrate that we need something more general than binding sequences. A framework should allow a nominal datatype with an input prefix to be coded along the lines of

$$\mathit{Input}\ \alpha\ ''(\text{«}\mathit{name}\text{»} \times \mathit{psiType})\ \mathit{list}''\ \alpha\ (\alpha,\ \beta,\ \gamma)\ \mathit{psi}$$

where *name* binds into the types, the object and the agent. The bound output residual could be coded like

$$\mathit{BoundOutput}\ \alpha\ ''\text{«}\mathit{name}\text{»}\ \mathit{list}''\ \alpha\ (\alpha,\ \beta,\ \gamma)\ \mathit{psi}$$

where *name* binds into the object and the agent. Isabelle can then provide notation to reason about the prefixes or the actions as separate objects, as described in Section 25.1. A function like the *bn* function for residuals must be created, and an alpha-equivalence lemma like Lemma 25.8 must be derived. Neither of these are complicated, and the techniques to generate Lemma 25.8 do not require the structure of the terms to be of a particular form – all that is required is that the bound names can be retrieved using a *bn* function, and that these bound names do not appear outside their scope.

35.1.2 Induction and inversion rules

Isabelle has support for deriving induction rules for calculi with more advanced binding schemes than just single bindders. When the induction rules are derived the user must specify which names should be allowed to be alpha-converted, and prove a set of proof obligations provided by Isabelle ensuring that these names can actually be alpha-converted.

Isabelle does not currently derive inversion rules for terms with arbitrary binding schemes; only inversion over terms with single binders is supported and more advanced inversion rules must currently be derived manually. In Chapter 26 is described how to create nominal inversion rules for calculi with arbitrary binding schemes. The requirements to derive the rule are modest; it must be possible to alpha-convert the terms, to obtain their bound names, and to calculate the number of binders present in a structure. We believe our techniques to be general enough to be applied for calculi with arbitrary binding schemes.

35.1.3 Current developments

A new version of Nominal Isabelle is currently being developed, which will have support for general binding schemes [74]. It allows the user to create datatypes and supply a *bn* function to tell where in the datatype the binders are. Moreover, three types of binding schemes are supported, and all three of them are useful when writing a formalism such as the one for psi-calculi.

The first binding scheme is what we have in this thesis – names are bound sequentially, their order matters, and two terms are alpha-equivalent if they are structurally equivalent, their binding sequences have the same length, and there is an alpha-converting permutation which equates the two.

The second type of binding scheme uses binding sets – i.e. the order of the sequence does not matter. For our purposes, this would be ideal for modeling bound output transitions as we want the equality

$$\Psi \rhd P \xrightarrow{\overline{M}(\nu xyz)N} P' = \Psi \rhd P \xrightarrow{\overline{M}(\nu zxy)N} P'$$

to hold. The intuition for psi-calculi is that the binders for bound output actions should be sets – that is the reason the OPEN rule inserts the opened binder at an arbitrary place in the binding sequence. This is one of the most awkward part of our formalisation, as demonstrated in the proof in Section 28.1.5 which states that agents with commuting binders are bisimilar.

The third and final binding scheme binds sets, and ignore binders where the bound name does not occur. For example, consider an instance of psi-calculi where assertions are sets. We want the frame equality

$$(\nu xy)\{x\} = (\nu x)\{x\}$$

as y does not appear in the assertion $\{x\}$ at all, and can be removed.

Possible future work includes adapting the current formalisation to the next version of Nominal Isabelle. We expect that significant simplifications can be made. The infrastructure to reason about binding sequences, induction, and inversion in our formalisation well exceeds ten thousand lines of code. The new version of Nominal Isabelle still does not support inversion rules; we think that the strategy presented in Chapter 26 is applicable at least to the binding schemes which do not use binding sets.

35.2 Impact

In this thesis we have demonstrated techniques to formalise process algebras in a theorem prover. We have used nominal logic to represent binders, and Isabelle to to verify our proofs, but our methods are general enough to be used with other theorem provers, and to use other formalisms to represent binders.

We have formalised strong and weak equivalences of three major process calculi, from CCS which was the first process calculus, to the pi-calculus which introduced modeling of mobility in concurrent systems, to psi-calculi which represents the current state of the art. To the best of our knowledge, weak equivalences for process calculi have never before been formalised in a theorem prover. The techniques we use, by lifting strong semantics to a weak level, effectively reduces the complexity of disregarding the internal actions of agents. In short, the presence of τ-chains and weak transitions does not increase the complexity of the formalisation. The techniques have been used to formalise all calculi in this thesis, and we believe them to be general enough for the process calculi of tomorrow as well.

The psi-calculi framework is the most advanced process calculus framework to date. It is expressive, it is general, and it has a simple semantics. The meta-theory has been proven correct in Isabelle. The next step will be to develop tools to allow the verification of programs and protocols. Preferably the output of these tools should be verified by a theorem prover.

For our formalisation efforts nominal logic has worked exceptionally well. One of its main benefits is that it provides reasoning about binders without referring to any particular structures of the data types. The arbitrary binding schemes which we touched on in the previous sections also follow this notion since alpha-equivalence and alpha-conversion lemmas can be established independently of the exact structure of the nominal datatypes. Reasoning about alpha-equivalence with single binders is reasonably well understood – there are de Bruijn indices, higher order abstract syntax, nominal logic, and recently the locally nameless framework [10], but for future calculi the abstraction must go one step

higher and reason about how to alpha-convert terms with arbitrary binding schemes. We have shown that nominal logic and Nominal Isabelle are exceptionally well suited for this task. Ultimately reasoning about binders should not add to the complexity of the proofs. There is still work to be done, but we believe our formalisation to be a proof of concept of how these goals can be attained.

Acknowledgments

First of all I would like to thank my supervisor, Joachim Parrow. When I first knocked on his door, wearing my stylish red student jump-suit, asking "Whom do I talk to for a Ph. D. in theoretical computer science?", I doubt any of us had any idea what we were getting ourselves into. I know I didn't. Later I've come to learn what most graduate students will tell you: A good supervisor will make, or break your Ph. D. Joachim has helped me tremendously as I've trudged along, trying to formalise all of our theorems; his door has always been open; his patience nearly infinite. I read somewhere that the average amount of people reading a Ph. D. thesis is one and a half. I don't know if this is true, but Joachim has read in full, and provided ample feedback on this thesis more than twice that many times already. It would not be in the state it is now without his help.

My second home during my graduate studies has been the Technische Universität in München, with the Theorem Proving Group. When I started my thesis, I was the only one in Uppsala that I know of who used the Isabelle theorem prover. That has not changed much, but we do have a few interested students who are keen to pick up the torch. The people in the Isabelle group taught me the skills required to write my formalisations, but also provided an abundance of social activities. The München opera house practically gave away tickets to students – during a three month visit I attended on average one and a half productions a week. A special thanks to Professor Tobias Nipkow for his hospitality, and for allowing me the use his very spacious office during this visit; although the fact that people kept calling me "boss", or "Herr Professor", was a bit unnerving. I also want to thank Stefan Berghofer, who has been my Isabelle mentor during these last five years. Also thanks to Steven Obua, Norbert Schirmer, Alexander Krauss, Clemens Ballarin, Amine Chaieb, and Jasmin Blanchette for the fun times inside and outside of the office. Jasmin also helped proof read this thesis, for which I am very greatful.

I would also like to thank my colleagues at the department, especially the members of my group. Björn Victor has been my co-supervisor, and a source of inspiration and constructive criticism. Magnus Johansson has been my fellow Ph. D. student for the entire journey, and he beat me to the punch by having his defense four days prior to mine. Congratulations to you, Erika, and "Plopp".

The art in this thesis was painted by Lisa Sohlberg. Thank you so much. At the time it felt like you were more worried about the thesis than I was, but I'm very happy with the result. Erik Lindahl made the transition from canvas to printer possible. Thank you for your professionalism and friendship.

Family and friends have been very supportive. Thanks for being there. A special thank you to the Indian Kitchen group and Krocket-Sigrid.

Finally I want to thank the better of my two birds, the other one is also adorable, but mostly loud and annoying. Eva has been a pillar of strength during these stressful times. Your patience, love, and care has kept me going through the writing process. You have some quality time to cash in.

Summary in Swedish

Att formalisera processalgebror

Vi är vana vid att datorer inte fungerar. Att en dator hänger sig, kraschar, tappar bort filer eller bara beter sig konstigt är något som inte gör oss särskilt förvånade. Ganska ofta är felen användarrelaterade, men inte sällan är det fel på programvaran. Vad vi ofta inte tänker på när vi motvilligt accepterar att datorn inte fungerar är att förhållandevis få datorer är sådana vi hittar på skrivbord. De allra flesta datorer sitter i bilar, tåg, flygplan, medicinsk utrustning eller andra inbyggda system – de är små, förhållandevis enkla, och ser till att servostyrningen på bilen funkar, att växlarna på spåren aldrig kommer att vara ställda så att tågen krockar, att autopiloten på flygplanen landar på banan med några centimeters felmarginal. Att ens arbetsdator kraschar och tappar bort några filer må vara hänt, men om krockkudden på ens bil fälls ut efter fem sekunder istället för fem hundradels sekunder så blir konsekvenserna mycket värre. Det är helt enkelt inte acceptabelt att vissa datorer inte fungerar.

Hur kan vi då garantera att ett program fungerar? Svaret är att det kan vi inte göra. Inte helt och hållet. Vi saknar de verktyg som behövs för att matematiskt kunna bevisa att ett program gör exakt vad det är tänkt att göra och ingenting annat. Vi vet hur man kontrollerar enstaka egenskaper hos program, som att en viss kodsekvens kommer att exekvera inom en viss tid, eller att vissa förutbestämda olämpliga saker inte kommer att hända, men ingen av dessa garanterar avsaknaden av fel i programmen. Det blir inte lättare av att moderna operativsystem möjliggör att flera program kör parallelt med varandra – mängden sätt som program kan växelvärka på är helt enkelt för stor för att kunna få en samlad överblick av. Den här avhandlingen behandlar hur man skaffar sig den överblicken genom att konstruera matematisk modeller av program som kör parallelt med varandra. Vi skapar språk för att resonera om den typen av system, och vi bevisar att dessa språk är korrekta – det vill säga att alla grammatiskt korrekta program kommer att exekvera på ett förutsägbart sätt.

Den typ av språk vi använder oss av kallas processalgebror. En processalgebra är en matematisk modell som analyserar system genom deras kommunikationer med omvärlden – eventuella interna beräkningar som varje enskilt program gör tas ingen hänsyn till. Processalgebror introducerades åren runt 1980 och deras fokus har i huvudsak legat i att analysera protokoll

som datorer använder för att kommunicera med varandra. De frågor man har ställt sig har varit om protokollen fungerar, vilket oftast innebär att de meddelanden som skickas kommer fram och att de inblandande programmen beter sig som de ska. På senare tid har processalgebror också använts för att säkerställa att de meddelanden som skickas enbart kan läsas av den tilltänkte mottagaren. Det ska inte vara möjligt för någon att snappa upp privat kommunikation, till exempel mellan en kund och dess bank.

I takt med att komplexiteten av programmen ökar har också komplexiteten hos processalgebror ökat – den typ av egenskaper vi vill kontrollera går inte att formulera på ett enkelt sätt med de gamla språken. Detta har lett till att mängden dialekter av processalgebror har ökat dramatiskt. Fördelen är att vi får kraftfullare språk som är mer lämpade att lösa de nya uppgifterna, men det finns några nackdelar. Till att börja med är många av dessa språk väldigt snarlika varandra, och en rimlig fråga är om man inte kan hitta ett genemsamt ramverk för processalgebror. Desustom, ju mer komplicerade språken blir, desto svårare blir det att bevisa att språken beskriver det de ska – att grammatiken är korrekt, och att programmen skrivna i språket gör det vi har tänkt oss. Ytterligare ett problem är att när man gör en ändring i ett språk måste alla bevis som säkerställer att språket verkligen fungerar göras om, och även små ändringar kan ha svåröverskådliga konsekvenser.

Den här avhandlingen är ett steg på vägen att lösa dessa problem. Vi introducerar ett ramverk för processalgebror som vi kallar för psi-kalkyl. Detta ramverk är generellt nog att lösa samma problem som en stor del av redan existerande processalgebrorna, men den gör det på ett enkelt och intuitivt sätt. Vi bevisar att kalkylen är korrekt, det vill säga att alla program har ett förutsägbart beteende.

För att förvissa oss om att våra bevis verkligen är korrekta använder vi oss av den interaktiva teorembevisaren Isabelle. Isabelle är ett datorprogram som kontrollerar matematiska bevis. Bevis som görs med papper och penna brukar ofta innehålla oprecisa argument som “vi inser lätt från definitionen av X att Y är sant”, eller “följer direkt genom induktion över de naturliga talen”. Den här typen av resonemang är inte nödvändigtvis fel, men det finns gott om fall där matematiska bevis har gjorts, som senare visat sig vara inkorrekta. Teorembevisare hjälper oss eftersom de tvingar oss att vara tydliga – en dator inser ingenting lätt, och ingenting är uppenbart. Allting måste förklaras i mycket tydliga och konkreta termer. Nackdelen med att använda teorembevisare är att en hel del jobb går åt till att skriva beviset på ett sådant sätt att datorn förstår det. Fördelarna är många. Vi vet när vi har formaliserat våra bevis att de faktiskt är korrekta. Vi har inte missat några fall, vi har inte gjort en felaktig förenkling någonstans, och alla våra bevisstrategier är korrekta. Vi kan dessutom ändra i våra formalismer när som helst och direkt se vilka effekter detta får för våra bevis. Effekten

av alla ändringar blir direkt uppenbar, och vi kan stämma av våra bevis och göra ändringar där så behövs.

Psi-kalkyl är det mest avancerade ramverket för processalberor idag. Grammatiken är enkel, kraftfull, och den fångar beteendet hos ett stort antal redan existerande processalgebror.

Index

Bibliography

[1] Mars Climate Orbiter Mishap Inevstigation Board Phase I Report. `ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf`, Novermber 1999.

[2] 2009-2010 Toyota vehicle recalls. `http://en.wikipedia.org/wiki/2009-2010_Toyota_vehicle_recalls`, April 2010.

[3] Agda: An interactive theorem prover. `http://unit.aist.go.jp/cvs/Agda`, April 2010.

[4] Laboratory for Reliable Software. `http://lars-lab.jpl.nasa.gov/`, April 2010.

[5] The Verisoft project. `http://www.verisoft.de`, April 2010.

[6] Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[7] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *SIGPLAN Notices*, 36(3):104–115, 2001.

[8] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:36–47, 1999.

[9] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, New York, NY, USA, 1996.

[10] Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. *SIGPLAN Notes*, 43(1):3–15, 2008.

[11] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, Nathan J. Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, August 2005.

[12] Michael Baldamus, Jesper Bengtson, Gianluigi Ferrari, and Roberto Raggi. Web services as a new approach to distributing and coordinating semantics-based veri cation toolkits. *Electron. Notes Theor. Comput. Sci.*, 105:11–20, 2004.

[13] H. P. Barendregt. *The lambda calculus : its syntax and semantics*. North-Holland Pub. Co, 1981.

[14] Gertrud Bauer and Tobias Nipkow. The 5 colour theorem in Isabelle/Isar. In *Theorem Proving in Higher Order Logics*, volume 2410 of *Lecture Notes in Compture Science*, pages 67–82. Springer-Verlag, 2002.

[15] Jesper Bengtson. Generic implementations of process calculi in Isabelle. In *The 16th Nordic Workshop on Programming Theory (NWPT'04)*, pages 74–78, 2004.

[16] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 17–32, Washington, DC, USA, 2008. IEEE Computer Society.

[17] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of LICS 2009*, pages 39–48. IEEE, 2009.

[18] Jesper Bengtson and Joachim Parrow. A completeness proof for bisimulation in the pi-calculus using Isabelle. *Electronic Notes in Theoretical Computer Science*, 192(1):61–75, 2007.

[19] Jesper Bengtson and Joachim Parrow. Formalising the pi-Calculus using Nominal Logic. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 4423 of *LNCS*, pages 63–77, 2007.

[20] Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2008.

[21] Jesper Bengtson and Joachim Parrow. Psi-calculi in Isabelle. In *Proceedings of TPHOLs 2009*, volume 5674 of *LNCS*, pages 99–114. Springer, 2009.

[22] Stefan Berghofer. Simply-typed lambda-calculus with let and tuple patterns. `http://isabelle.in.tum.de/repos/isabelle/file/tip/src/HOL/Nominal/Examples/Pattern.thy`, February 2010.

[23] Stefan Berghofer and Christian Urban. Nominal inversion principles. In *TPHOLs '08: Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics*, pages 71–85, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] Gérard Berry and Laurent Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In *Seminar on Concurrency*, pages 389–448, 1984.

[25] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[26] Johannes Borgström and Uwe Nestmann. On bisimulations for the spi calculus. *Mathematical. Structures in Comp. Sci.*, 15(3):487–552, 2005.

[27] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[28] Maria Grazia Buscemi and Ugo Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *Proceedings of ESOP 2007*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.

[29] Maria Grazia Buscemi and Ugo Montanari. Open bisimulation for the concurrent constraint pi-calculus. In Sophia Drossopoulou, editor, *Proceedings of ESOP 2008*, volume 4960 of *LNCS*, pages 254–268. Springer, 2008.

[30] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[31] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ, 1986.

[32] Véronique Cortier, Michaël Rusinowitch, and Eugen Zalinescu. Relating two standard notions of secrecy. *Logical Methods in Computer Science*, 3(3), 2007.

[33] N. G. de Bruijn. Lambda calculus notation with nameless dummies. A tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[34] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic bisimulation for the applied pi-calculus. In V. Arvind and Sanjiva Prasad, editors, *Proceedings of FSTTCS'07*, volume 4855 of *LNCS*, pages 133–145. Springer, December 2007.

[35] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[36] Edsger W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[37] Mark Dowson. The Ariane 5 software failure. *SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[38] M. J. Gabbay. The π-calculus in FM. In Fairouz Kamareddine, editor, *Thirty-five years of Automath*. Kluwer, 2003.

[39] Murdoch J. Gabbay. Automating fraenkel-mostowski syntax. In *TPHOLs, 15th International Conference on Theorem Proving in Higher Order Logics*, number CP-2002-211736, pages 60–70–. NASA, August 2002.

[40] Philippa Gardner and Lucian Wischik. Explicit fusions. In *MFCS '00: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 373–382, London, UK, 2000. Springer-Verlag.

[41] G. Gonthier. A computer-checked proof of the four colour theorem. Technical report, Microsoft Research Cambridge, 2004.

[42] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, New York, NY, USA, 1993.

[43] J. Goubault-Larrecq, C. Palamidessi, and A. Troina. A probabilistic applied pi-calculus. In *Proceedings of APLAS'07*, volume 4807 of *LNCS*, pages 175–190. Springer, 2007.

[44] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[45] Daniel Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In *TPHOLs '97: Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics*, pages 153–169, London, UK, 1997. Springer-Verlag.

[46] Furio Honsell, Marino Miculan, and Ivan Scagnetto. π-calculus in (co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001.

[47] Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Proceedings of LICS 2010 (to appear)*. IEEE, 2010.

[48] Magnus Johansson, Joachim Parrow, Björn Victor, and Jesper Bengtson. Extended pi-calculi. In *Proceedings of ICALP 2008*, volume 5126 of *LNCS*, pages 87–98. Springer, July 2008.

[49] Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales - a sectioning concept for isabelle. In *TPHOLs '99: Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, pages 149–166, London, UK, 1999. Springer-Verlag.

[50] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 207–220, Big Sky, MT, USA, October 2009. ACM.

[51] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Proceedings of ESOP'05*, volume 3444 of *LNCS*, pages 186–200. Springer, April 2005.

[52] Huimin Lin. Complete inference systems for weak bisimulation equivalences in the pi-calculus. In *TAPSOFT*, pages 187–201, 1995.

[53] Huimin Lin. Complete inference systems for weak bisimulation equivalences in the pi-calculus. *Information and Computation*, 180(1):1–29, 2003.

[54] Thomas F. Melham. A mechanized theory of the pi-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, 1994.

[55] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[56] R. Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[57] Robin. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980.

[58] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I/II. *Information and Computation*, 100(1):1–77, 1992.

[59] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[60] Otmane Aït Mohamed. Mechanizing a pi-calculus equivalence in HOL. In *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, pages 1–16, London, UK, 1995. Springer-Verlag.

[61] Faron Moller and Perdita Stevens. Edinburgh Concurrency Workbench user manual (version 7.1). Available from `http://homepages.inf.ed.ac.uk/perdita/cwb/`.

[62] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *In Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[63] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[64] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[65] Tobias Nipkow, Gertrud Bauer, and Paula Schultz. Flyspeck I: Tame graphs. In U. Furbach and N. Shankar, editors, *Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNCS*, pages 21–35, 2006.

[66] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[67] Joachim Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier Science Inc., New York, NY, USA, 2001.

[68] Joachim Parrow and Davide Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):174–197, 1995.

[69] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[70] Christine Röckl and Daniel Hirschkoff. A fully adequate shallow embedding of the π-calculus in Isabelle/HOL with mechanized syntax analysis. *Journal of Functional Programming*, 13(2):415–451, 2003.

[71] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[72] Natarajan Shankar. *Metamathematics, Machines and Gödel's Proof*. Cambridge University Press, 1994.

[73] Christian Urban, Stefan Berghofer, and Michael Norrish. Barendregt's variable convention in rule inductions. In *CADE-21: Proceedings of the 21st international conference on Automated Deduction*, pages 35–50, Berlin, Heidelberg, 2007. Springer-Verlag.

[74] Christian Urban and Cezary Kaliszyk. General Bindings in Nominal Isabelle, or How to Formalise Core-Haskell. *Submitted to ICFP*, 2010.

[75] Christian Urban and Tobias Nipkow. Nominal verification of algorithm W. In G. Huet, J.-J. Lévy, and G. Plotkin, editors, *From Semantics to Computer Science. Essays in Honour of Gilles Kahn*, pages 363–382. Cambridge University Press, 2009.

[76] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In *CADE*, pages 38–53, 2005.

[77] Björn Victor and Faron Moller. The mobility workbench - a tool for the pi-calculus. In *CAV '94: Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818, pages 428–440, London, UK, 1994. Springer-Verlag.

[78] Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs, Thorsten Engel, Enno Keen, Christian Theobalt, and Dalibor Topić. System description: SPASS version 1.0.0. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 314–318, Trento, Italy, 1999. Springer.

[79] Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *Theorem Proving in Higher Order Logics*, pages 167–184, 1999.