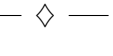
The Applied π-calculus: Type Systems and Expressiveness

Master Thesis

Anders Bloch
Morten V. Frederiksen
Bjørn Haagensen

7th June 2004



Aalborg University
Department of Computer Science · Fredrik Bajers Vej 7E
DK-9220 Aalborg Øst

Aalborg University
The Faculty of Engineering and Science

Department of Computer Science

TITLE:
The Applied π-calculus: Type Systems and Expressiveness

PROJECT PERIOD:
DAT6,
1st February 2004 to 8th June 2004

PROJECT GROUP:
B1-215B

GROUP MEMBERS:
Anders Bloch, *bloch@cs.auc.dk*
Morten V. Frederiksen, *mortenf@cs.auc.dk*
Bjørn Haagensen, *bh@cs.auc.dk*

SUPERVISOR:
Hans Hüttel, *hans@cs.auc.dk*

NUMBER OF COPIES: **5**

NUMBER OF PAGES: **119**

ABSTRACT

In this master thesis we present an encoding of the applied π-calculus (Appπ) [AF01] in the π-calculus and it is shown how Appπ can be instantiated to obtain the spi-calculus [AG97]. Furthermore we define two type systems for Appπ. One is a basic type system capable of capturing errors such as arity mismatch and errounous use of names. The other is a type system equivalent to the type system for the spi-calculus presented in [Aba97]. It is shown that the representation of the spi-calculus preserves types with respect to this type system.

The encoding of the spi-calculus in the instance of Appπ is shown to be sound and complete with respect to operational correspondence. In order to facilitate the proof of this result a semantics for the spi-calculus in the same style as the semantics for App$\pi$is used, i.e. an early labelled transition relation. This modified semantics is shown to be strongly equivalent to that of the original in [AG97] which uses the notion of abstractions and concretions.

We do not directly encode the equational theory used in Appπ, rather we rely on well-known results for transforming such a theory into a confluent and terminating rewrite system on which the encoding is performed. The encoding is shown to be sound with respect to operational correspondence. In light of the sparse amount of work which has followed the original publication of Appπ, one of the major assets of our encoding is that it makes the vast amount of theoretical and practical tools developed for the π-calculus available for reasoning in Appπ.

Preface

This master thesis documents the results obtained by group B1-215B during the spring semester of 2004. The report is written as part of the DAT6-term at the Department of Computer Science at Aalborg University.

The work is done under the distributed systems and semantics group, and the topic is process calculi, specifically the applied π-calculus.

We have strived to stay as close as possible to the notation standard in the field. Abbreviations, most notably for the languages we define, are introduced where the language is first defined. Citations are in bracketed parentheses e.g. [Mil99].

We would like to thank our supervisor Hans Hüttel for providing overview as well as advice on technical details.

Aalborg, June 2004

Anders Bloch

Morten V. Frederiksen

Bjørn Haagensen

Summary

In the π-calculus the only kind of messages that can be communicated are atomic names and this simplicity is appealing from a theoretical point of view. But for many purposes one often finds it necessary or convenient to be able to represent, for instance, data types such as pairs. Various authors have demonstrated that more complex constructs such as objects, functions, and higher-order processes can be encoded in the π-calculus.

However, such encodings can be cumbersome and their correctness can be difficult to establish, and it is not clear whether such encodings can always be established. For this reason a plethora of extensions of the π-calculus exist. These are often aimed at capturing common features and properties of programming languages. The applied π-calculus (Appπ) [AF01], proposed by Abadi and Fournet, is a generalization of the π-calculus which captures several of these extensions. More specifically, messages may be constructed from, not only names, but also function symbols applied to names and other functions. Furthermore, the calculus includes a *let*-like construct which serves to capture the knowledge an environment may have of the values of certain variables.

To our knowledge very little work has succeeded the publication of [AF01] in spite of the fact that the original article leaves a number of aspects open to further investigation. This master thesis focuses on a few of these aspects. The major contributions are the following.

- Definition of type systems for Appπ,
- a representation of the spi-calculus in Appπ,
- an encoding of Appπ in the π-calculus.

We have established a basic type system for Appπ similar to the one in [SW01]. This type system is capable of capturing errors such as arity mismatch and erroneous use of channel names. A subject-reduction theorem, which states that the transition relation preserves typing, has been established for this type system.

As suggested in [AF01] the development of Appπ has been inspired by the desire to have a calculus in which, among others, the spi-calculus [AG97] can be represented. We have formalised this correspondence by establishing a representation of the spi-calculus in an instantiation of Appπ. That is, an instance of Appπ has been constructed by defining a presentation Π and an encoding of spi-processes in this language has been defined.

The encoding is shown to respect the transition relation with regard to actions and derivatives, that is if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{\alpha} \approx [\![P']\!]$ and vice versa. In order to facilitate the proof of this result, we have tailored the semantics of the spi-calculus to be conceptually as close to the semantics of Appπ as possible. Thus minor changes has been made compared to the semantics originally given in [AG97]. Specifically, we use an early labelled transition relation with action labels in the style of $a\langle M\rangle$ for input and $(\nu q)\overline{a}\langle M\rangle$ for output. This contrasts to the semantics given in [AG97] which uses the notion of abstractions and concretions. Furthermore, scope extrusion is handled differently in the original presentation. In order to ensure that our results hold also for the original spi-calculus, we have proved that the modified semantics is strongly equivalent to that of the original.

The encoding of the spi-calculus in Appπ also demonstrates one of the assets of Appπ. If we where to add hashing to the spi-calculus we would have to change the syntax and semantics. In Appπ terms are constructed from a signature and the semantics is extended with an equational theory over the set of terms. The remaining parts of the syntax and semantics, and consequently entire theory, is independent of the exact nature of the equational theory. Hence the above mentioned variations can be captured by merely modifying the signature and its associated equational theory.

An approach to analysing security protocols in the spi-calculus is by using type systems and considerable effort has been put into investigating this topic. Most notably Gordon and Jeffrey has published [Aba97] and [GJ01a] in which secrecy and authenticity properties of security protocols are analysed using a type system for the spi-calculus. We modify our type system for Appπ in order to obtain a type system for the instantiation of spi in Appπ which is equivalent to the one Abadi defines in [Aba97]. The changes made account for the specific message format Abadi defines for messages on secret channels.

The most innovative contribution of this master thesis is an encoding of Appπ in the π-calculus. Although the π-calculus is well-known to be Turing-complete, this encoding is non-trivial. The main difficulty lies in the encoding of the equational theory which is used for comparing terms in Appπ. Our encoding is inspired by a recent proposal for an encoding of the spi-calculus in the π-calculus [BPV03], but differs considerably in the way that terms are encoded. We rely on well-known results for transforming an equational theory into a terminating and confluent rewrite system. Moreover the encoding requires the rules of the rewrite system to be left linear. The idea is that all terms are forced to reduce to their irreducible form before they can be accessed. Since the rewrite system is terminating and confluent, equality then reduces to a matter of checking for syntactic identity. The encoding also imposes a bottom-up evaluation strategy of terms. The choice of evaluation strategy is inessential for confluent rewrite systems. We prove soundness of the encoding of terms.

Resumé

I π-kalkylen er det kun atomiske navne, der kan kommunikeres, hvilket giver en enkelhed, der er appellerende ud fra et teoretisk synspunkt. I mange henseender er det dog ofte nødvendigt eller ønskeligt at kunne repræsentere for eksempel data typer som for eksempel par. Flere forfattere har vist, at komplekse konstruktioner som objekter, funktioner og højere ordens processer kan kodes i π-kalkylen.

Sådanne kodninger kan imidlertid være besværlige at arbejde med, og deres korrekthed kan ligeledes være svær at fastslå og derudover er det ikke altid oplagt at sådanne kodninger overhovedet eksisterer. På grund af dette er der dukket adskillige udvidelser af π-kalkylen op. Disse udvidelsers formål er ofte at udtrykke almindelige egenskaber, der er til stede i programmeringssprog. Den Anvendte π-kalkyle (Appπ) [AF01], som blev foreslået af Abadi og Fournet, er en generalisering af π-kalkylen, der er i stand til at udtrykke mange forskellige sådanne udvidelser. Mere specifikt kan beskeder ikke blot konstrueres ved hjælp af navne, men også ved hjælp at funktionssymboler, der kan anvendes på navne og andre funktionssymboler. Derudover har kalkylen en konstruktion, der minder om *let*, som kan udtrykke et miljøs viden om visse variablers værdier.

Så vidt vi ved, har udgivelsen af [AF01] kun affødt en yderst begrænset mængde forskning og det på trods af, at den oprindelige artikel foreslår flere åbne problemer, der kan studeres. Nærværende specialerapport fokuserer på nogle af disse problemer. Rapportens vigtigste bidrag er følgende.

- Definition af et typesystem for Appπ,
- en repræsentation af spi-kalkylen i Appπ og
- en kodning af Appπ i π-kalkylen.

Vi etablerer et typesystem for Appπ, der på mange måder ligner det typesystem der defineres i [SW01] for π-kalkylen. Dette typesystem garanterer, at der ikke opstår aritetsfejl og forhindrer fejlagtig brug af kanalnavne. I denne forbindelse bevises at transitionsrelationen bevarer veltypethed.

Udviklingen af Appπ har, som det fremhæves i [AF01], været motiveret af et ønske om at opnå en kalkyle hvori blandt andet spi-kalkylen [AG97] kunne udtrykkes. Vi demonstrerer, at dette mål er nået ved at konstruere en instans af Appπ, hvori spi-kalkylen kan kodes. Dette gøres ved at definere en signatur indeholdende de nødvendige funktionssymboler samt en mængde af ligninger og desuden en kodningsfunktion, der koder spi-processer til Appπ-processer.

Kodningen vises så at respektere transitionsrelationen med hensyn til handlinger og afledte processudtryk. Det vil sige at hvis $P \xrightarrow{\alpha} P'$ så $[\![P]\!] \xrightarrow{\alpha} \approx [\![P']\!]$ og omvendt. For at lette beviset for denne sætning, defineres en spi-kalkyle med en semantik, der ligger så tæt op ad semantikken for Appπ som muligt. Dette indebærer blandt andet at semantikken er tidlig og at transitionerne mærkes med handlinger af formen $a\langle M\rangle$ for input og $(\nu q)\overline{a}\langle M\rangle$ for bundet output. I semantikken for den oprindelige spi-kalkyle, der beskrives i [AG97], bruges i stedet abstraktioner og konkretiseringer. Desuden behandles virkefeltsudvidelse anderledes i denne semantik. På grund at disse omstændigheder er det nødvendigt at bevise, at de to semantikker er ækvivalente.

Kodningen af af spi-kalkylen i Appπ demonstrerer en af fordelene ved Appπ. Hvis vi ønskede at tilføje funktionalitet for hashing til spi-kalkylen, ville vi være nødt til at ændre dens syntaks og semantik. I Appπ kan termer konstrueres ved hjælp af en signatur og semantikken udvides ved hjælp af en ligningsteori på mængden af termer. De resterende dele af syntaksen og semantikken forbliver uændrede således at den generelle teori for Appπ også forbliver uændret. Den udvidelse af π-kalkylen, man ønsker, kan således beskrives ved en signatur og en tilhørende ligningsteori.

Typesystemer har i forbindelse med spi-kalkylen været brugt til at analysere sikkerhedsegenskaber for protokoller og en ikke ubetydelig forskningsindsats er blevet lagt for dagen i dette henseende. Her bør især fremhæves Gordon og Jeffreys artikler [Aba97] og [GJ01a], hvori typesystemer til analyse af hemmeligheds- og autenticitetsegenskaber ved sikkerhedsprotokoller beskrives. Ved at tilpasse vores generelle typesystem for Appπ er vi i stand at konstruere et typesystem for instantieringen af spi-kalkylen i Appπ, som er ækvivalent med Abadis. De nødvendige ændringer vedrører det særlige format Abadi bruger i forbindelse med kommunikation af beskeder på hemmelige kanaler.

Det væsentligste bidrag i denne specialerapport er en kodning af Appπ i π-kalkylen. På trods af at π-kalkylen har vist sig at være Turing-fuldstændig, er denne kodning langt fra triviel. Den største vanskelighed er kodningen ligningsteorien, der bruges til at sammenligne termer i Appπ. Vores kodning er inspireret af en kodning af spi-kalkylen i π-kalkylen, som blev præsenteret i den nyligt udgivne artikel [BPV03], men adskiller sig væsentligt herfra ved den måde termer kodes på. Vi benytter os af velkendte metoder til at transformere en ligningsteori til et terminerende og konfluent omskrivningssystem, og dertil fordrer vi at omskrivningssystemet er venstre-lineært. Ideen i kodningen er at kodede termer reducerer til deres entydigt bestemte normal form, og eftersom omskrivningssystemet er konfluent er kontrol af lighed mellem termer nu blot er spørgsmål om at afgøre om de er syntaktisk identiske. For at opnå dette, må kodningen først reducere alle undertermer til deres normal form, men da omskrivningssystemet er konfluent er dette uden betydning. Vi afslutter med at bevise sundheden af kodningen.

Contents

Chapter 1

Introduction

The majority of the computer technology which we surround ourselves with every day, contains some elements of concurrency. Although concurrency is a general concept and not solely a property of computer systems, the study of concurrency has been driven by the desire to understand and develop computer technology. In the world of computers, concurrency appears at many levels, for instance at the hardware level, the operating system level and the network level.

The leap from sequential to concurrent computing is of overwhelming magnitude; by making computing entities able to interact, the size of the state space grows enormously and it is often difficult, or impossible, to predict the behavior of such a compound system without rigorous methods.

Several formalisms for describing concurrent systems have been proposed and a well-studied and consequently also well understood formalism is the π-calculus. The precursor of the π-calculus, the Calculus of Concurrent Systems (CCS) [Mil80] was limited to synchronisation on named channels between processes. The ability to describe communication of names on channels, a concept coined mobility, was introduced in the π-calculus which was developed in the late 1980's by Milner, Parrow and Walker [MPW92].

The literature is abound with calculi derived from the π-calculus and most are motivated by a desire for more high-level primitives and expressive formalism. Many, however, have turned out to be expressible in the π-calculus and therefore the π-calculus is now considered to be a canonical formalism for modelling concurrency. An example is the spi-calculus which among other things adds primitives for encryption and decryption. The spi-calculus is expressible in the π-calculus which was recently proved by Baldamus, Parrow and Victor [BPV03].

1.1 The spi-calculus

The spi-calculus was defined by Gordon and Abadi in [AG97] in 1997. The calculus is an example of an extension of the π-calculus in which the syntax and semantics is often modified to suit specific needs. The spi-calculus is tailored towards being able to verify that certain properties of security protocols are

fulfilled. Hence its syntax and semantics accommodates abstract functionality for encryption and decryption. The reason for the changes in the syntax and semantics is that different cryptographic protocols uses different cryptographic functions, e.g. hashing, shared key encryption and public key encryption. Thus in one variant which uses shared keys we could have a term $\{M\}_k$ which is a message M encrypted under the key k. This term can appear in the process

$$\text{case } \{M\}_k \text{ of } \{x\}_k \text{ in } P$$

which reduces to $P\{\vec{M}/\vec{x}\}$ as the keys match. In another variant with public and secret keys we could have a message M encrypted under a public key N^+, $\{\!|M|\!\}_{N^+}$ and the process

$$\text{case } \{\!|M|\!\}_{N^+} \text{ of } \{\!|x|\!\}_{N^-} \text{ in } P$$

which reduces to $P\{M/x\}$. Since the underlying theory of the calculus changes each time a new cryptographic operation is added to the syntax, it is natural to ask whether a calculus can be defined which captures such changes in a more general manner.

1.2 The applied π-calculus

The need for more specialised and perhaps more expressive calculi is evident from the plethora of variants of the π-calculus that exists. The above mentioned spi-calculus is one example, but in fact it is often necessary or convenient to have more complex data structures than the atomic names available in the π-calculus. An interesting question is therefore whether a more general calculus can be found in which one can easily model different variants of the π-calculus. In 2001 Abadi and Fournet came up with the applied π-calculus (Appπ), [AF01].

Appπ is designed to easily be able to represent some of the many variants of the π-calculus and also many of the various data structures for which encodings in the π-calculus have been proposed, without changing the underlying theory of the calculus. To our knowledge only two articles regarding the calculus have been published. Besides the original presentation in [AF01], Abadi and Fournet themselves have analysed a protocol for private authentication [AF03] using Appπ.

The calculus uses ideas from universal algebra. In particular the terms of Appπ are defined through a signature. Along with the signature a set of equations is defined, hence obtaining a presentation. To alter the calculus, only the signature and equations need to be changed. The theory of the calculus is independent of the actual presentation. As a consequence one data structure can easily be exchanged with another since such changes only affects the presentation.

As an example consider how pairs can be represented in Appπ. The signature should include a binary function symbol pair and two unary function symbols fst and snd. The set of equations should include

$$\mathsf{fst}(\mathsf{pair}(x, y)) = x$$
$$\mathsf{snd}(\mathsf{pair}(x, y)) = y.$$

As is evident from this example it is rather simple to to represent such data structures in Appπ, whereas a direct representation of the same structure in the π-calculus would be significantly more complex.

1.3 Type systems for the spi-calculus

In the spi-calculus protocol properties such as secrecy and authentication can be formulated as process equivalences [AG97]. However, equivalence in process calculi, and specifically the spi-calculus, can in most cases not be characterised in a tractable manner. A solution can be found by utilising types and type systems. Considerable effort has been put into investigating the use of type systems for such purposes. Most notably Gordon and Jeffrey have published several papers [GJ01a, GJ02, GJ01b] on the subject. Perhaps the first paper to appear was [Aba97], where a type system which ensures secrecy in the spi-calculus is given.

1.4 Topics treated in this thesis

The purpose of this thesis is to further develop the theory of Appπ and relate it to the theory of the π-calculus and the spi-calculus.

In [AF01] the authors assume that processes are always well-typed. However, to our knowledge, no detailed treatment of this matter in relation to Appπ exists. Since type systems for the π-calculus have turned out to be useful for several purposes [Gay00], one of the objects in this thesis is to investigate this topic in the framework of Appπ.

The spi-calculus is considered as a quite fundamental process calculus for cryptographic protocols. In the light of the, in [AF01], implied connection between the spi-calculus and Appπ it is therefore interesting to investigate the exact nature of this relationship. Given the importance of type systems for the spi-calculus and the encoding of the spi-calculus, a relevant question is to how can a type system for Appπ be defined that captures the the system for the spi-calculus in [GJ01b].

As Appπ has not been thoroughly investigated it is appealing to pursue an encoding of Appπ in the π-calculus for several reasons. For instance automated analysis of Appπ-processes could be performed by first encoding a process and then use one of the many automated tools that exist for the π-calculus. E.g. the Mobility Workbench [VM94] developed by Victor and Moller, which is an automated tool for manipulating and analysing mobile concurrent systems described in the π-calculus.

The same arguments applies for the substantial amount of theoretical tools that have been developed for the π-calculus. It should be possible to utilise some of these theoretical results in order to obtain a better understanding of Appπ. In fact, due to the generality of Appπ this arguments applies to any calculi expressible in Appπ. Finally the encoding itself also contributes to a better understanding of the π-calculus itself.

Moreover many of the extensions proposed for the π-calculus has turned out

to be expressible in the π-calculus. This is often demonstrated by exhibiting encodings from the extended language back to the π-calculus. In the light of the seemingly great generality of Appπ compared to that of the π-calculus, a fundamental question to be answered is whether it is possible to represent Appπ in the π-calculus. Indeed the results established in this theses indicate that this question has a positive answer.

Basic type system

We define a simple type system for Appπ which is capable of catching errors such as arity mismatch. The type system is similar to existing type systems for the π-calculus. However there is one fundamental difference. A general type rule for functions is defined. In this respect the type system is, like the rest of Appπ, also parameterised by the signature. One of the basic properties of type systems is that types are preserved by transitions, a property commonly coined subject-reduction. Whether this can be established for the basic type system defined herein is one of the topics we investigate.

Encoding the spi-calculus in Appπ

Appπ is a general calculus and is as such, not usable before a concrete presentation has been defined. We define a presentation aimed at capturing the cryptographic functionality of the spi-calculus. Thus an instance of Appπ is obtained in which an encoding of the spi-calculus is given.

The basic type system serves as a basis for the other type system we define. This modified type system is aimed at capturing the type system for the spi-calculus defined by Abadi in [Aba97]. We show that the modified type system and Abadi's type system are equivalent. The changes made to the basic type system is primarily due to the specific message formats Abadi defines in [Aba97].

Appπ encoded in the π-calculus

The most innovative contribution of this master thesis is to exhibit an encoding of Appπ in the π-calculus. This encoding is inspired by the encoding of the spi-calculus in the π-calculus [BPV03]. However it is significantly different since we need to handle the equational theory which is a crucial part underlying the semantics of Appπ. Instead of encoding the equational theory directly an encoding of a corresponding rewrite systems is performed. This encoding is necessary for checking equality of terms, i.e. we need to reduce terms to their irreducible form. The other part of the encoding is relatively simple. Once terms are reduced, matching becomes a matter of checking for syntactic identity.

1.5 Encodings in general

We use the same notion of representability of a source language in a target language as Sangiorgi [San92] where three different phases are identified:

(1) Formal definitions of the semantics of the source and target language.

(2) Encoding from the source to the target language.

(3) Proof of correctness.

With respect to (1) we use transition systems with rules that are inductively defined on the structure of the process expressions. The encoding should be compositional, i.e. the encoding of an expression should only depend on its immediate constituents. (3) entails two aspects. On one hand the encoding is required to respect some equivalence. I.e if two processes P and Q are equivalent in the source language, then the encoding $[\![P]\!]$ and $[\![Q]\!]$ should be equivalent in the target language. We refer to this as soundness since it asserts that whenever two processes are equal, the encoding preserves this equality. Conversely, if $[\![P]\!]$ and $[\![Q]\!]$ are equivalent in the target language, then P and Q should be equivalent in the source language. This we call completeness as it anticipates that all encoded processes that are equal in the target language were originally equal in the source language. An encoding which is sound and complete is called fully abstract. Soundness and completeness with respect to equality does not reveal the whole truth. The other aspect of (3) is that of operational correspondence between P and $[\![P]\!]$. The simplest formulation of this requirement is that whenever $P \xrightarrow{\tau} P'$, then the encoding matches this by going from $[\![P]\!]$ to $[\![P']\!]$. This is soundness with respect to operational correspondence. The converse implication is called completeness with respect to operational correspondence.

We have chosen barbed bisimilarity [MS92] and its induced congruence as our preferred equivalence. (Its counterpart in Appπ is called observational equivalence.) This choice is motivated by its applicability across different calculi. Barbed bisimilarity is very flexible in the sense that it can easily be applied to different process algebras. The principal cause of this flexibility is the observation predicate on which it is based. In order to apply it one basically only needs to come up with a reasonable notion of when a process is to be regarded as exhibiting visible behavior.

1.6 Structure of the thesis

The thesis is structured as follows. Chapter 2 contains a brief review of the syntax and semantics of the spi-calculus, including equivalence theorems relating the semantics we use with that of the original. Chapter 3 introduces Appπ, and is ended with a section on the normal form which is utilised in the encoding of Appπ in the π-calculus. These two chapters serves as background material for the remaining parts of the thesis. In Chapter 4 the topic is the basic type system for Appπ. Chapter 5 contains the encoding of the spi-calculus in the instantiation of Appπ created. It also contains a modification of the type system from Chapter 4, which is equivalent to Abadi's type system in [Aba97]. Chapter 6 treats the encoding of Appπ in the π-calculus. The chapter starts by briefly going over the syntax and semantics for the π-calculus. The encoding itself is then defined, and finally proofs related to the correctness of the encoding is given. Concluding remarks and indications to possible future work is given in Chapter 7.

The appendices contains some useful background material. Appendix A reviews basic terminology related to universal algebra. Appendix B contains background material on rewrite systems crucial to the encoding of Appπ in the π-calculus. Appendix C recapitulate Abadi's original semantics for the spi-calculus. Finally Appendix D contains proofs which were omitted in Chapter 1.

Chapter 2

The spi-calculus

The spi-calculus was first presented in [AG97] and we present a minor variant here for reference. The basic principle governing the spi-calculus is the same as for the π-calculus, namely the idea of communicating information along named channels. The range of information that can be communicated is extended to encompass pairs and encrypted messages so primitives are also added for supporting abstract cryptographic functionality. Nevertheless, the expressiveness of the spi-calculus is not greater than the expressiveness of the π-calculus. This was established recently in [BPV03] which presents an encoding of the spi-calculus in the π-calculus that preserves may-testing. In view of this result the motivation for using the spi-calculus is shifted towards notational convenience.

2.1 Syntax of the spi-calculus

We present the basic spi-calculus that supports symmetric cryptography, though it can easily be extended to cope with asymmetric cryptography as well as hashing. Furthermore, we opt for a polyadic calculus even though polyadic communication can easily be encoded using nested pairs. The abstract syntax for the polyadic spi-calculus is presented below.

Definition 2.1.1 (The polyadic spi-calculus)
The terms of the polyadic spi-calculus are generated by the grammar

$M, N ::=$	*terms*
$a, b, c, \ldots$	*name*
(M, N)	*pair*
0	*zero*
$\mathrm{suc}(M)$	*successor*
$x, y, z, \ldots$	*variable*
$\{\vec{M}\}_N$	*encryption*

The processes of the polyadic spi-calculus are generated by the grammar

$P, Q ::=$	*processes*
$\overline{M}\langle\vec{M}\rangle.P$	*output*
$M(\vec{x}).P$	*input*
$P \mid Q$	*composition*
$(\nu a)P$	*restriction*
$!P$	*replication*
if $M_1 = M_2$ then P	*match*
$\mathbf{0}$	*inactive process*
let $(x, y) = M$ in P	*pair splitting*
case M of $0 : P$ $\mathrm{suc}(x) : Q$	*integer case*
case M of $\{\vec{x}\}_M$ in P	*decryption*

It should be noted that some syntactically correct processes will be given no meaning. For example any input or output prefix α such that $\mathrm{subj}(\alpha)$ is not a name or a coname, will cause no transitions.

Free and bound names and variables of processes are defined as usual. The free names and variables of a term are simply the names and variables appearing in the term. (Names and variables cannot be bound in a term.) Additionally, we assume that free and bound names and variables are distinct. Substitution is defined as usual. We shall consider processes modulo α-conversion, that is, a process represents the equivalence class with respect to α-conversion of which it is a member.

We define the syntactical notion of evaluation context and evaluation congruence below.

Definition 2.1.2 (Evaluation context)
An evaluation context C is generated by the grammar

$$C_E \quad ::= \quad P \mid C_E \quad \Big| \quad C_E \mid P \quad \Big| \quad (\nu q)C_E \quad \Big| \quad [\cdot].$$

If C is an evaluation context and P is a spi-process then $C[P]$ is the process obtained by replacing $[\cdot]$ in C with P with name capturing.

Definition 2.1.3
A binary relation $\mathcal{R}$ on spi-processes is an evaluation congruence if $P\mathcal{R}Q$ implies $C[P]\mathcal{R}C[Q]$ for all evaluation contexts C.

The semantics we define in the following section is a slight variation from Abadi's semantics, given in Appendix C, and we wish to prove it equivalent to another semantics in the literature. For this purpose we will need the following definition.

Definition 2.1.4 (Scope extrusion evaluation congruence)
The relation $\overset{\nu}{=}$ is the smallest evaluation congruence which satisfies

$$\begin{aligned}(\nu q)Q \mid P &\overset{\nu}{=} (\nu q)(Q \mid P)\\ P \mid (\nu q)Q &\overset{\nu}{=} (\nu q)(P \mid Q)\\ (\nu p)(\nu r)P &\overset{\nu}{=} (\nu r)(\nu p)P\end{aligned}$$

for all spi-processes P and Q and names p, q and r such that $q \notin \mathrm{fn}(P)$.

2.2 Semantics of the spi-calculus

Traditionally, no structural congruence is used in the spi-calculus. Instead a reduction relation will be convenient when defining the transition relation.

Definition 2.2.1 (Reduction relation on spi-calculus terms)
The reduction relation $>$ on spi-calculus processes is the smallest relation which satisfy the following

$$\begin{aligned}
!P &> P \mid !P \\
\text{if } M = M \text{ then } P &> P \\
\text{let } (x,y) = (M,N) \text{ in } P &> P\{M/x, N/y\} \\
\text{case } 0 \text{ of } 0 : P \ \text{suc}(x) : Q &> P \\
\text{case } \text{suc}(M) \text{ of } 0 : P \ \text{suc}(x) : Q &> Q\{M/x\} \\
\text{case } \{\vec{M}\}_N \text{ of } \{\vec{x}\}_N \text{ in } P &> P\{\vec{M}/\vec{x}\} \quad \text{if } |\vec{M}| = |\vec{x}|
\end{aligned}$$

The semantics we give is an early labelled transition relation. As actions we consider the silent action τ and input actions $a\langle\vec{M}\rangle$, bound output actions $(\nu\vec{q})\overline{a}\langle\vec{M}\rangle$ where $\text{n}(\vec{q}) \subseteq \text{fn}(\vec{M})$ and $a \notin \text{n}(\vec{q})$.

Definition 2.2.2 (Semantics of the polyadic spi-calculus)
The labelled transition relation $\xrightarrow{\alpha}$ of the polyadic spi-calculus is the smallest relation on spi-processe that satisfies the rules

$$\text{(Output)}\ \frac{}{\overline{a}\langle\vec{M}\rangle.P \xrightarrow{\overline{a}\langle\vec{M}\rangle} P}$$

$$\text{(Input)}\ \frac{}{a\,(\vec{x})\,.P \xrightarrow{a\langle\vec{M}\rangle} P\{\vec{M}/\vec{x}\}} \quad \text{for all } \vec{M} \text{ such that } |\vec{M}| = |\vec{x}|$$

$$\text{(Restriction)}\ \frac{P \xrightarrow{\alpha} P'}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'} \quad \text{if } a \notin \text{n}(\alpha)$$

$$\text{(Open)}\ \frac{P \xrightarrow{(\nu\vec{s})\overline{a}\langle\vec{M}\rangle} P'}{(\nu q)P \xrightarrow{(\nu q,\vec{s})\overline{a}\langle\vec{M}\rangle} P'} \quad \text{if } a, \overline{a} \neq q \text{ and } q \in \text{fn}(\vec{M})$$

$$\text{(Reduction)}\ \frac{P > P' \xrightarrow{\alpha} P''}{P \xrightarrow{\alpha} P''}$$

$$\text{(Parallel}^\sharp\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{(Communication}^\sharp\text{)}\ \frac{P \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} P' \qquad Q \xrightarrow{a\langle\vec{M}\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu\vec{q})(P' \mid Q')}$$

where the rules marked with $\sharp$ has symmetric counterparts.

A reduction relation is obtained by considering only transitions labelled with τ and we denote the reduction relation by $\xrightarrow{\tau}$. Its weak counterpart is the reflexive and transitive closure denoted by $\xrightarrow{\tau}{}^*$.

As usual a barb is a name or a co-name a. A process P has an a barb, denoted by $P \downarrow_a$, if $P \xrightarrow{\alpha}$ where $a = \mathrm{subj}(\alpha)$. The process P has a weak a barb, denoted by $P \downarrow_a^*$, if $P \xrightarrow{\tau}{}^* \downarrow_a$.

Definition 2.2.3 (Strict barbed bisimulation)
A strict barbed bisimulation is a symmetric relation $\mathcal{R}$ on spi-processes, such that $P\mathcal{R}Q$ if

(i) $P \downarrow_a$ implies $Q \downarrow_a$, and

(ii) $P \xrightarrow{\tau} P'$ implies that there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$.

If two processes P and Q are related by a strict barbed bisimulation we write $P \sim_b Q$.

Lemma 2.2.4
Let P and Q be spi-processes, then the following equations hold

$$\begin{aligned} P \mid Q &\sim_b Q \mid P \\ P &\sim_b P \mid \mathbf{0} \\ (\nu q)P &\sim_b P && \text{if } q \notin \mathrm{n}(P) \\ (\nu q)(\nu p)P &\sim_b (\nu p)(\nu q)P \\ P \mid (\nu q)Q &\sim_b (\nu q)(P \mid Q) && \text{if } q \notin \mathrm{n}(P). \end{aligned}$$

Proof
By induction in the structure of the process. ■

The notion of testing equivalence is defined in the two definitions below.

Definition 2.2.5 (Test)
A test is a pair (T, a) where T is a spi-process and a is a barb. A spi-process P passes the test (T, a) if $(P \mid T) \downarrow_a^*$

Definition 2.2.6 (Testing equivalence)
Two spi-processes are testing equivalent, denoted by $P \simeq_t Q$ if for all tests t, P passes t if and only if Q passes t.

Testing equivalence is obviously reflexive, transitive and symmetric and hence it is an equivalence.

2.3 Abadi's semantics

The early labelled semantics for spi-processes given above differs from the semantics given by Abadi in [Aba97], and also given in Appendix C. The semantics given here is a commitment relation which is defined in terms of abstractions and concretions. We want to prove that our semantics is equivalent to the one

in [Aba97] in terms of reduction and barbs. We denote the commitment relation and observation predicate in Appendix C by $\stackrel{\alpha}{\longrightarrow\!\!\!\!\!\!\backslash}$ and $\downarrow_a$ in order to distinguish them from the ones defined in the present report. The reduction relation $>$ is the same in both semantics.

A particular difference between the two semantics, which complicates an otherwise straightforward proof, is the way restriction is handled in relation to output actions. If we consider the process $a(x) \mid (\nu c)\overline{a}\langle b\rangle$ then the two semantics does not yield the same derivatives. We get

$$a(x).P \mid (\nu c)\overline{a}\langle b\rangle.Q \stackrel{\tau}{\longrightarrow} P\{b/x\} \mid (\nu c)Q$$

and

$$a(x).P \mid (\nu c)\overline{a}\langle b\rangle.Q \stackrel{\tau}{\longrightarrow\!\!\!\!\!\!\backslash} (\nu c)(P\{b/x\} \mid Q),$$

because Abadi's semantics always extrudes the scope whereas our semantics only extrudes the scope when required.

To establish the equivalence between the two semantics we establish two operational correspondences.

Lemma 2.3.1
Let P be a spi-process. Then

(i) if $P \stackrel{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle}{\longrightarrow} P'$ then there exists P'' and $\vec{q}$ such that $P \stackrel{\overline{a}}{\longrightarrow\!\!\!\!\!\!\backslash} (\nu\vec{q})\langle\vec{M}\rangle P''$ and $P' \stackrel{\nu}{=} (\nu\vec{q} \setminus \vec{p})P''$,

(ii) if $P \stackrel{a\langle\vec{M}\rangle}{\longrightarrow} P'$ then there exists P'' such that $P \stackrel{a}{\longrightarrow\!\!\!\!\!\!\backslash} (\vec{x})P''$ and $P' = P''\{\vec{M}/\vec{x}\}$, and

(iii) if $P \stackrel{\tau}{\longrightarrow} P'$ then there exists P'' such that $P \stackrel{\tau}{\longrightarrow\!\!\!\!\!\!\backslash} P'' \stackrel{\nu}{=} P'$.

Lemma 2.3.2
Let P be a spi-process. Then

(i) if $P \stackrel{\overline{a}}{\longrightarrow\!\!\!\!\!\!\backslash} (\nu\vec{p})\langle\vec{M}\rangle P'$ then there exists P'' and $\vec{q}$ such that $P \stackrel{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle}{\longrightarrow} P''$ and $P'' \stackrel{\nu}{=} (\nu\vec{p} \setminus \vec{q})P'$,

(ii) if $P \stackrel{a}{\longrightarrow\!\!\!\!\!\!\backslash} (\vec{x})P'$ then $P \stackrel{a\langle\vec{M}\rangle}{\longrightarrow} P'\{\vec{M}/\vec{x}\}$ for all $\vec{M}$ such that $|\vec{M}| = |\vec{x}|$,

(iii) if $P \stackrel{\tau}{\longrightarrow\!\!\!\!\!\!\backslash} P'$ then there exists P'' such that $P \stackrel{\tau}{\longrightarrow} P'' \stackrel{\nu}{=} P'$.

Proofs of the lemmas can be found in Appendix D.

For the rest of this chapter we let $\sim_A$ denote Abadi's definition of strict barbed bisimulation. This definition uses bisimulation as defined in Definition 2.2.3 but with every $\longrightarrow$ replaced by $\longrightarrow\!\!\!\!\!\!\backslash$.

We will prove that our semantics and Abadi's semantics gives rise to the same strict barbed bisimulation. To this end we need the following lemma.

Lemma 2.3.3
$\stackrel{\nu}{=} \subseteq \sim_b$ *and* $\stackrel{\nu}{=} \subseteq \sim_A$.

Proof
We sketch the proof only. We wish to show that $\stackrel{\nu}{=}$ is a strict barbed bisimulation in both semantics. The proof is as usual a proof by structural induction. First assume that $P \stackrel{\nu}{=} Q$. Then prove that $P \downarrow_a$ implies $Q \downarrow_a$ and that whenever $P \stackrel{\tau}{\longrightarrow} P'$ then there exists Q such that $Q \stackrel{\tau}{\longrightarrow} Q'$ and $P' \stackrel{\nu}{=} Q'$. Similarly for $\longrightarrow\!\!\!\!\!\!\rightarrow$. ■

Theorem 2.3.4
$\sim_b = \sim_A$

Proof
To show the equality, we show inclusion in both directions by structural induction. To prove the inclusion $\sim_A \subseteq \sim_b$ assume that $P \sim_b Q$.

Assume that $P \downarrow_a$. By Lemma 2.3.2 this implies that that $P \downarrow_a$ which in turn, by definition of strict barbed bisimulation, implies that $Q \downarrow_a$. Then by Lemma 2.3.1 we have $Q \downarrow_a$.

Assume that $P \stackrel{\tau}{\longrightarrow\!\!\!\!\!\!\rightarrow} P'$. Then by Lemma 2.3.2, $P \stackrel{\tau}{\longrightarrow} \widehat{P} \stackrel{\nu}{=} P'$. Since $P \sim_b Q$ we have that there exists $\widehat{Q}$ such that $Q \stackrel{\tau}{\longrightarrow} \widehat{Q}$ and $\widehat{P} \sim_b \widehat{Q}$. Lemma 2.3.1 then gives us that $Q \longrightarrow\!\!\!\!\!\!\rightarrow Q' \stackrel{\nu}{=} \widehat{Q}$. From Lemma 2.3.3 now have $P' \sim_b Q'$ and by the induction hypothesis this implies that $P' \sim_A Q'$ which concludes the proof.

The inclusion $\sim_b \subseteq \sim_A$ is proven similarly. ■

Chapter 3

The applied π-calculus

Representing various data types and security protocols in the π-calculus can be quite cumbersome as it requires either an encoding of the data types or even an extension of the calculus. As an example consider the spi-calculus. Here it is necessary to define a new syntax and semantics for each cryptographic operation. A contrast to this is the applied π-calculus, denoted by Appπ. This calculus takes advantage of the notion of presentations from universal algebra (see Appendix A) from which an equational theory can be defined. With presentations various data structures can easily be represented without changing anything but the presentation. As a consequence the theory of the calculus remains unchanged. This makes the Appπ-calculus quite appealing.

Appπ also has strong analogies with the ideas presented in [BDP02]. In this article a calculus is presented which accommodates for some of the differences between the π-calculus and the spi-calculus. For instance in spi it is not always possible to use a name which has been acquired as it e.g. could be encrypted under a fresh key. Secondly, in the spi-calculus equivalent processes need not exhibit the same sequence of transitions. Therefore a labelled transition system is presented where the states are configurations $\sigma \rhd P$ where P is a process and σ the current knowledge of the environment. Reasoning about equivalence of processes then includes reasoning about the environments. As we will see this corresponds to some of the concepts of Appπ. In particular the environment σ has analogies with active substitutions and equivalence of environments has similarities with static equivalence.

This chapter, except Section 3.4, is based on [AF01] in which Appπ was first defined. The chapter contains a review of the syntax and semantics of Appπ and a look at some equivalences. In the last section we define a substitution normal form for Appπ-processes.

3.1 Syntax of Appπ

Presently we define the syntax of Appπ-processes. Syntactically, the major difference from the π-calculus comes from the introduction of function names, see Section A.1. These functions names can occur as terms in Appπ-processes which

are defined as follows.

Definition 3.1.1 (Appπ-terms)
Let $\Sigma = (\Omega, \alpha)$ be a signature. The terms of the Appπ-calculus are defined as

$M, N ::=$	*terms*
$a, b, c, \ldots$	*names*
$x, y, z, \ldots$	*variables*
$f(M_1, \ldots, M_n)$	*function applications*

where f ranges over Ω.

Note that names and variables are kept distinct and that Appπ-terms do not necessarily respect the arity of function names. We will use metavariables $u, v, \ldots$ to range over both names and variables. A term is said to be ground if it contains no free variables. Note that a ground term in the term algebra does not contain anything but functions. A ground Appπ-term is then a term in $T_\Sigma(\mathcal{N})$, where N is the set of Appπ names.

With the definition of terms we can now define Appπ-processes.

Definition 3.1.2 (Appπ-processes)
The set $\mathcal{P}_A$ of Appπ-processes is the set generated by the following grammar

$$P \quad ::= \quad \mathbf{0} \;\mid\; P \mid P \;\mid\; !P \;\mid\; (\nu a)P \;\mid\; \text{if } M = N \text{ then } P \text{ else } P \;\mid\; u(\vec{x}).P \;\mid\; \overline{u}\langle\vec{M}\rangle.P.$$

The intention is that we may exchange arbitrary terms and not just names as in the π-calculus, and that $M = N$ denotes equality in an equational theory rather than syntactical identity. Besides that, the operations are quite standard. We will sometimes omit $\mathbf{0}$ in for example $u(x).\mathbf{0}$ and we will abbreviate $(\nu a_1)\cdots(\nu a_n)$ by $(\nu\vec{a})$. Plain processes are extended with active substitutions.

Definition 3.1.3 (Extended processes)
The set $\mathcal{A}$ of extended processes is the set generated by the grammar

$$A \quad ::= \quad P \;\mid\; A \mid A \;\mid\; (\nu a)A \;\mid\; (\nu x)A \;\mid\; \{M/x\}.$$

The active substitution $\{M/x\}$ denotes the replacement of the variable x with the term M. If we consider the extended process $P \mid \{M/x\}$ then this corresponds to substituting x for M in P. Thus active substitutions are like "let $x = M$ in $\ldots$". However, if we consider another process $P \mid Q \mid \{M/x\}$, then the active substitution $\{M/x\}$ also applies to Q. If we only want the substitution to occur in P then we need to restrict the variable x as in

$$(\nu x)(P \mid \{M/x\}) \mid Q.$$

This corresponds exactly to (let $x = M$ in P) $\mid Q$. We will use $\{M_1/x_1, \ldots, M_n/x_n\}$ as shorthand for $\{M_1/x_1\} \mid \cdots \mid \{M_n/x_n\}$. Furthermore, we write $\sigma, \{M/x\}$ and $\{\vec{M}/\vec{x}\}$ for substitutions and $\sigma(x)$ for the image of x under σ.

We assume that substitutions are cycle free and that there is at most one substitution for each variable and exactly one when the variable is restricted.

We let $\mathrm{fv}(A), \mathrm{bv}(A), \mathrm{fn}(A)$ and $\mathrm{bn}(A)$ denote the sets of free and bound variables and names, respectively, of an extended process A. As usual an input $u(x)$ binds the variable x and restriction can bind both names and variables. We will omit the inductive definition of these sets, except for active substitutions:

$$\mathrm{fv}(\{M/x\}) \stackrel{\mathrm{def}}{=} \mathrm{fv}(M) \cup \{x\}$$
$$\mathrm{fn}(\{M/x\}) \stackrel{\mathrm{def}}{=} \mathrm{fn}(M).$$

The sets of bound names and variables of an active substitution $\{M/x\}$ are defined as the sets of bound names and variables of M, respectively. The set of all names of an extended process A is denoted by $\mathrm{n}(A)$ and the set of all variables by $\mathrm{v}(A)$. An extended process is said to be closed if all variables are either bound or defined by an active substitution.

We also follow standard assumptions in order to avoid capturing of names and variables. These assumptions are that the sets of bound and free names and variables of an extended process are always disjoint, and that substitutions do not capture names or variables. These assumptions can be made to hold by α-converting.

At last we need a definition of frames which will become necessary when we later look at equivalences on Appπ.

Definition 3.1.4 (Frame)
A frame φ is an extended process generated by the following grammar

$$A \quad ::= \quad A \mid A \quad \mid \quad (\nu a)A \quad \mid \quad (\nu x)A \quad \mid \quad \{M/x\} \quad \mid \quad \mathbf{0}.$$

The domain of φ, $dom(\varphi)$, is the set of variables that φ exports, i.e. the variables that are defined by an active substitution and not under a restriction.

We can define a mapping from extended processes to frames simply by replacing each plain process with the inactive process $\mathbf{0}$. If A is an extended process then $\varphi(A)$ denotes the frame obtained by this mapping. We then define $\mathrm{dom}(A)$ to be $\mathrm{dom}(\varphi(A))$. We define the range of A, $\mathrm{ran}(A)$, as the terms M where $\{M/x\}$ occurs in $\varphi(A)$ and $x \in \mathrm{dom}(\varphi(A))$. We will also write $\{M/x\} \in A$ if $\{M/x\}$ occurs in A and $x \in \mathrm{dom}(A)$.

3.2 Operational semantics of Appπ

In this section we give the operational semantics of Appπ-processes in which a crucial point is played by the signature Σ. By defining a set Φ of Σ-equations we obtain a presentation $\Pi = (\Sigma, \Phi)$ and an equation in a match is defined to be successful in the semantics if it is an equation in the theory of Π.

Before we can define the reduction relation we need to define contexts and structural equivalence.

Definition 3.2.1 (Appπ-context)
An expression derived from the grammar

$$\begin{array}{rcl} C & ::= & A \mid C \;\;\big|\;\; C \mid A \;\;\big|\;\; !C \;\;\big|\;\; (\nu a)C \\ & & \textit{if } M = N \textit{ then } C \textit{ else } P \;\;\big|\;\; \textit{if } M = N \textit{ then } P \textit{ else } C \\ & & u(\vec{x}).C \;\;\big|\;\; \overline{u}\langle \vec{M} \rangle.C \;\;\big|\;\; [\cdot] \end{array}$$

is called an Appπ-context. If $A \in \mathcal{A}$ then $C[A]$ denotes C with $[\cdot]$ replaced literally with A. An evaluation context, denoted by C_E, is a context whose hole is not under a replication, a conditional, an input or an output. A context C closes A if $C[A]$ is closed.

Definition 3.2.2 (Structural equivalence on Appπ)
The structural equivalence relation $\equiv$ on extended Appπ-processes is the smallest equivalence on Appπ that is closed by α-conversion on both names and variables, and closed under evaluation contexts, such that

$$\begin{array}{rcll} A \mid B & \equiv & B \mid A & \text{(SC-ParCommute)} \\ A \mid \mathbf{0} & \equiv & A & \text{(SC-ParInact)} \\ A \mid (B \mid C) & \equiv & (A \mid B) \mid C & \text{(SC-ParAssoc)} \\ (\nu a)(A \mid B) & \equiv & A \mid (\nu a)B & \textit{if } a \notin \mathrm{fn}(A) \cup \mathrm{fv}(A) \quad \text{(SC-ResPar)} \\ (\nu a)\mathbf{0} & \equiv & \mathbf{0} & \text{(SC-ResInact)} \\ (\nu a)(\nu b)A & \equiv & (\nu b)(\nu a)A & \text{(SC-ResCommute)} \\ !P & \equiv & P \mid !P & \text{(SC-Rep)} \\ (\nu x)\{M/x\} & \equiv & \mathbf{0} & \text{(SC-Alias)} \\ \{M/x\} \mid A & \equiv & \{M/x\} \mid A\{M/x\} & \text{(SC-Subst)} \\ \{M/x\} & \equiv & \{N/x\} & \textit{if } \Phi \vdash_\Sigma M = N \quad \text{(SC-Rewrite)} \end{array}$$

If A and B are α-convertible, we write $A \equiv_\alpha B$.

All axioms, except the last three, are standard. The rule (SC-Alias) makes it possible to introduce an arbitrary substitution, (SC-Subst) describes how an active substitution applies to a process which is in parallel with the substitution and (SC-Rewrite) shows how a substitution can be rewritten using the equational theory. As shown in [AF01] the substitution $A\{M/x\}$ is structurally equivalent to $(\nu x)(\{M/x\} \mid A)$ if $x \notin \mathrm{fv}(M)$ and this is justified by the following equivalences:

$$\begin{aligned} A\{M/x\} &\equiv A\{M/x\} \mid \mathbf{0} \\ &\equiv (\nu x)(\{M/x\}) \mid A\{M/x\} \\ &\equiv (\nu x)(\{M/x\} \mid A\{M/x\}) \\ &\equiv (\nu x)(\{M/x\} \mid A). \end{aligned} \tag{3.1}$$

We are now in position to define the reduction relation.

Definition 3.2.3
The reduction relation of Appπ is the smallest relation on $\mathcal{A} \times \mathcal{A}$ closed by

application of evaluation contexts, generated by the following rules

$$(\textsc{Comm})\ \frac{}{\overline{a}\langle\vec{y}\rangle.P \mid a(\vec{x}).Q \longrightarrow P \mid Q\{\vec{y}/\vec{x}\}} \quad \text{if } |\vec{y}| = |\vec{x}|$$

$$(\textsc{Then})\ \frac{}{\text{if } M = M \text{ then } P \text{ else } Q \longrightarrow P}$$

$$(\textsc{Else})\ \frac{}{\text{if } M = N \text{ then } P \text{ else } Q \longrightarrow Q} \quad \begin{array}{l}\text{for ground terms } M, N \\ \text{such that } \Phi \nvdash_\Sigma M = N\end{array}$$

$$(\textsc{Struct})\ \frac{A \equiv B \longrightarrow B' \equiv A'}{A \longrightarrow A'}$$

We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

Note that despite the simplicity of the (Comm) rule we do not lose any generality as shown by the following lemma.

Lemma 3.2.4

$$\frac{}{\overline{a}\langle\vec{M}\rangle.P \mid a(\vec{x}).Q \longrightarrow P \mid Q\{\vec{M}/\vec{x}\}}$$

Proof
Combine the axioms (SC-Alias) and (SC-Subst), as shown by the equivalences in (3.1) for the unary case, to obtain:

$$\begin{array}{rcl}\overline{a}\langle\vec{M}\rangle.P \mid a(\vec{x}).Q & \equiv & (\nu\vec{y})(\{\vec{M}/\vec{y}\} \mid \overline{a}\langle\vec{y}\rangle.P \mid a(\vec{x}).Q) \\ & \longrightarrow & (\nu\vec{y})(\{\vec{M}/\vec{y}\} \mid P \mid Q\{\vec{y}/\vec{x}\}) \\ & \equiv & P \mid Q\{\vec{M}/\vec{x}\},\end{array}$$

where $\vec{y}$ are fresh names. By the (Struct) we obtain the statement in the lemma. ■

We also see that the process

$$\text{if } M = N \text{ then } P \text{ else } Q$$

reduces to P if $\Phi \vdash_\Sigma M = N$ as justified by the following lemma.

Lemma 3.2.5

$$\frac{}{\text{if } M = N \text{ then } P \text{ else } Q \longrightarrow P} \quad \text{if } \Phi \vdash_\Sigma M = N$$

Proof
As in the previous lemma use the axioms (SC-Alias) and (SC-Subst) to obtain

$$\begin{array}{rl}\text{if } M = N \text{ then } P \text{ else } Q \equiv & (\nu x)(\{N/x\} \mid \text{if } M = x \text{ then } P \text{ else } Q) \\ \equiv & (\nu x)(\{M/x\} \mid \text{if } M = x \text{ then } P \text{ else } Q) \\ \equiv & \text{if } M = M \text{ then } P \text{ else } Q.\end{array}$$

■

We mentioned in the beginning of the chapter that various data structures and cryptographic primitives can easily be represented in Appπ. Two examples of this are given in Sections A.4.1 and A.4.2. If we have a presentation (Ω, Φ), we can for instance represent pairs simply by having a binary function symbol $\mathsf{pair} \in \Omega$ and two unary functions $\mathsf{fst}, \mathsf{snd} \in \Omega$ and appropriate equations in Φ for example

$$\mathsf{fst}(\mathsf{pair}(x, y)) = x.$$

Similarly, in order to represent symmetric encryption we just need the function names enc and dec, representing respectively encryption and decryption functions, and appropriate equations.

3.3 Equivalences

We now define a few equivalences for Appπ. The first equivalence we consider is observational equivalence.

3.3.1 Observational equivalence

The weak observation predicate is defined as follows.

Definition 3.3.1 (Weak observation predicate)
We say that an extended process A has a weak barb a, denoted by $A \Downarrow_a$ if

$$A \longrightarrow^* C_E[\overline{a}\langle \vec{M} \rangle.P]$$

for some evaluation context C_E which does not bind a.

Next follows the definition of observation equivalence.

Definition 3.3.2 (Observation equivalence)
Observational equivalence is the largest symmetric relation $\approx$ between closed extended processes such that $A \approx B$ implies

(i) $dom(A) = dom(B)$.

(ii) $A \Downarrow_a$ *implies* $B \Downarrow_a$.

(iii) If $A \longrightarrow^* A'$ *then* $B \longrightarrow^* B'$ *and* $A' \approx B'$.

(iv) $C_E[A] \approx C_E[B]$ *for all closing evaluation contexts* C_E.

This definition is a straightforward extension of barbed congruence in the π-calculus.

One obvious disadvantage of observation equivalence is that the definition contains a universal quantification over evaluation contexts. Therefore we also define a labelled equivalence. This, however, depends on static equivalence which follows next.

3.3.2 Static equivalence

Static equivalence regards equivalence of frames when applied to terms. As mentioned in Section 3.1 each extended process A can be mapped to a frame $\varphi(A)$. This frame can be viewed as the static knowledge that A reveals to its environment, as $\varphi(A)$ contains the active substitutions that the environment can come into contact with.

The next example serves to motivate and explain Definition 3.3.5 of static equivalence.

Example 3.3.3 ([AF01])
Consider two functions f *and* g *with no equations and the three frames*

$$\varphi_0 = (\nu k)\{k/x\} \mid (\nu s)\{s/y\}$$
$$\varphi_1 = (\nu k)\{\mathsf{f}(k)/x, \mathsf{g}(k)/y\}$$
$$\varphi_2 = (\nu k)\{k/x, \mathsf{f}(k)/y\}$$

In φ_0 *the two variables* x *and* y *are mapped to two unrelated new values. Similarly, for* φ_1 *no context can distinguish between* $\mathsf{f}(k)$ *and* $\mathsf{g}(k)$*. Thus a context that obtains the values for* x *and* y *cannot distinguish* φ_0 *and* φ_1*. However, for the last frame* φ_2*, a context can check the predicate* $\mathsf{f}(x) = y$*. Thus we would like to define* $\varphi_0 \approx_s \varphi_1 \not\approx_s \varphi_2$*, where* $\approx_s$ *denotes static equivalence.* △

A closed frame φ is always structurally equivalent to a process of the form

$$\varphi \equiv (\nu \vec{n})\{\vec{M}/\vec{x}\},$$

where $\mathrm{fv}(\vec{M}) = \emptyset$ and $\vec{n} \subseteq \mathrm{fn}(\vec{M})$. The domain of φ is $\vec{x}$.

Definition 3.3.4 (Equality of Appπ-terms)
Two Appπ-terms M *and* N *are said to be equal in the closed frame* $\varphi \equiv (\nu \vec{n})\sigma$*, denoted by* $(M = N)\varphi$*, if an only if* $M\sigma = N\sigma$ *and* $\vec{n} \cap (\mathrm{fn}(M) \cup \mathrm{fn}(N)) = \emptyset$*.*

Now we can define static equivalence.

Definition 3.3.5 (Static equivalence)
Two closed frames φ *and* ψ *are statically equivalent when*

(i) $dom(\varphi) = dom(\psi)$,

(ii) $(M = N)\varphi$ *if and only if* $(M = N)\psi$ *for all terms* M *and* N*.*

Two extended processes are statically equivalent when their frames are statically equivalent.

We write $A \approx_s B$ if the two extended processes A and B are statically equivalent.

How difficult it is to find out whether two extended processes are statically equivalent depends on the signature Σ. One advantage is, however, that it does not depend on the reductions of the two processes. We restate the following two lemmas from [AF01].

Lemma 3.3.6
Static equivalence is closed by structural equivalence and by reduction.

Lemma 3.3.7
Observational equivalence and static equivalence coincide on frames. Observational equivalence is strictly finer than static equivalence on extended processes, i.e. $\approx \subset \approx_s$.

3.3.3 A labelled equivalence

In order to define a labelled equivalence relation we need a transition relation of $\mathcal{A}$ which we define with an early semantics. The set *Act* contains actions of the form:

$\overline{a}\langle\vec{u}\rangle$**:** Output action; only output of channel names or variables is permitted.

$(\nu\vec{u})\overline{a}\langle\vec{v}\rangle$**:** Bound output action; output $\vec{v}$ on a, extruding the scope of $\vec{u}$. It holds that $\vec{u} \subseteq v$.

$a\langle\vec{M}\rangle$**:** Input action; input the terms $\vec{M}$ on the channel a.

τ**:** Internal action.

Notice that terms can not be directly output as in the process $\overline{a}\langle\vec{M}\rangle.P$. Terms that are not channel names or variables of a base type can only be output by "reference" as in the process $(\nu\vec{x})(\{\vec{M}/\vec{x}\} \mid \overline{a}\langle\vec{x}\rangle.P)$. Now the transition relation can be defined by extending the reduction relation from Section 3.2.3.

Definition 3.3.8
The transition relation of $\mathcal{A}$ *is the smallest relation on* $\mathcal{A} \times Act \times \mathcal{A}$ *generated by the following rules*

$$(\textsc{Input})\ \frac{}{a(\vec{x}).P \xrightarrow{a\langle\vec{M}\rangle} P\{\vec{M}/\vec{x}\}}$$

$$(\textsc{Output})\ \frac{}{\overline{a}\langle\vec{u}\rangle.P \xrightarrow{\overline{a}\langle\vec{u}\rangle} P}$$

$$(\textsc{Open})\ \frac{A \xrightarrow{(\nu\vec{u})\overline{a}\langle\vec{v}\rangle} A'}{(\nu w)A \xrightarrow{(\nu w,\vec{u})\overline{a}\langle\vec{v}\rangle} A'}\ \text{if } w \neq a \text{ and } w \in \vec{v}$$

$$(\textsc{Res})\ \frac{A \xrightarrow{\alpha} A' \qquad u \notin \mathrm{n}(\alpha) \cup \mathrm{v}(\alpha)}{(\nu u)A \xrightarrow{\alpha} (\nu u)A'}$$

$$(\textsc{Par})\ \frac{A \xrightarrow{\alpha} A'}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$(\textsc{Struct})\ \frac{A \equiv B \xrightarrow{\alpha} B' \equiv A'}{A \xrightarrow{\alpha} A'}$$

and the rules (COMM), (THEN) *and* (ELSE) *from definition 3.2.3 with* $\longrightarrow$ *replaced by* $\xrightarrow{\tau}$.

Again we write $\xrightarrow{\tau}{}^*$ for the reflexive and transitive closure of $\xrightarrow{\tau}$ and $\xrightarrow{\alpha}{}^*$ for $\xrightarrow{\tau}{}^* \xrightarrow{\alpha} \xrightarrow{\tau}{}^*$.

Definition 3.3.9 (Labelled bisimulation)
Labelled bisimilarity is the largest symmetric relation $\approx_l$ *between closed extended processes such that* $A \approx_l B$ *implies*

(i) $A \approx_s B$,

(ii) if $A \xrightarrow{\tau} A'$ *then* $B \xrightarrow{\tau}{}^* B'$ *and* $A' \approx_l B'$ *and*

(iii) if $A \xrightarrow{\alpha} A'$, $\mathrm{fv}(\alpha) \subseteq dom(A)$ *and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset$ *then* $B \xrightarrow{\alpha}{}^* B'$ *and* $A' \approx_l B'$.

In the definition, only condition (i) is non standard. The main result proved in [AF01, Thm. 1] is that labelled bisimilarity and observational equivalence coincide.

Theorem 3.3.10
Observational equivalence and labelled bisimilarity coincide, i.e. $\approx = \approx_l$.

One of the lemmas needed for proving this theorem says that $\approx_l$ is closed under evaluation contexts. Hence labelled bisimilarity permits simpler proofs than observational equivalence does.

3.4 A normal form for Appπ-processes

In this section we present a normal form for Appπ-processes which we will refer to as the substitution normal form. This normal form will be particularly useful when we in Chapter 6 encode Appπ in the π-calculus. The idea is that we factor terms, other than variables, out of processes by using active substitutions and furthermore we factor sub-terms out of these active substitutions so that the parse tree of each term has height one or zero. If a term occurs in the range of the process it is a part of, the term is not factored out. Instead we replace the term for the variable occurring in the active substitution it is a part of. We show how an extended process can be put on substitution normal form by defining an encoding $\{\!|\,|\!\}_A$, where A is the process encoded. It is necessary to parameterise the encoding by the process we encode as we need sometimes need to check if a term $M \in \mathrm{ran}(A)$. We will say that $A \in \mathcal{A}$ is on substitution normal form if $\{\!|A|\!\}_P = A$.

3.4.1 The encoding $\{\!|\,|\!\}_A$

We now define the encoding $\{\!|\,|\!\}_A$ which is an encoding from $\mathcal{A} \times \mathcal{A}$ to $\mathcal{A}$ and it is an encoding which produces a process on substitution normal form. First of all note that we still use the convention that $x, y, z, \ldots$ represent variables, $a, b, \ldots$

names and $u, v, \ldots$ either names or variables. The encoding is homomorphic on parallel composition, restriction and replication and the identity on the inactive process, $\mathbf{0}$. It is also the identity on active substitutions involving terms whose parse tree already has height one or zero, i.e.

$$\{\!|\{f(z_1,\ldots,z_n)/x\}|\!\}_A \stackrel{\text{def}}{=} \{f(z_1,\ldots,z_n)/x\}$$
$$\{\!|\{u/x\}|\!\}_A \stackrel{\text{def}}{=} \{u/x\}.$$

Similarly, if a prefix only contains variables, then the encoding is also the identity on the prefix.

$$\{\!|y(\vec{x}).P|\!\}_A \stackrel{\text{def}}{=} y(\vec{x}).\{\!|P|\!\}_A$$
$$\{\!|\overline{y}\langle\vec{x}\rangle.P|\!\}_A \stackrel{\text{def}}{=} \overline{y}\langle\vec{x}\rangle.\{\!|P|\!\}_A$$

In the following we will write $u : var$ if u is a variable and $\neg u : var$ if u is not a variable. The encoding on the remaining process constructs is defined as follows.

$$\{\!|\text{if } M = N \text{ then } P \text{ else } Q|\!\}_A \stackrel{\text{def}}{=} \begin{cases} (\nu x, y)(\{\!|\{M/x\}|\!\}_A \mid \{\!|\{N/y\}|\!\}_A \mid \text{if } x = y \text{ then} \\ \{\!|P|\!\}_A \text{ else } \{\!|Q|\!\}_A) & \text{if } M, N \notin \text{ran}(A) \\ (\nu x)(\{\!|\{M/x\}|\!\}_A \mid \text{if } x = y \text{ then } \{\!|P|\!\}_A \text{ else } \{\!|Q|\!\}_A) & \\ & \text{if } M \notin \text{ran}(A) \text{ and } \{N/y\} \in A \\ \text{if } x = y \text{ then } \{\!|P|\!\}_A \text{ else } \{\!|Q|\!\}_A) & \\ & \text{if } \{M/x\}, \{N/y\} \in A \end{cases}$$

$$\{\!|u(\vec{x}).P|\!\}_A \stackrel{\text{def}}{=} \begin{cases} (\nu y)(\{u/y\} \mid y(\vec{x}).\{\!|P|\!\}_A) & \text{if } \neg u : var \text{ and } u \notin \text{ran}(A) \\ y(\vec{x}).\{\!|P|\!\}_A) & \text{if } \neg u : var \text{ and } \{u/y\} \in A \\ u(\vec{x}).\{\!|P|\!\}_A & \text{if } u : var \end{cases}$$

$$\{\!|\overline{u}\langle\vec{z}, M_1, \vec{M}\rangle.P|\!\}_A \stackrel{\text{def}}{=} \begin{cases} (\nu y)(\{u/y\} \mid \{\!|\overline{y}\langle\vec{z}, M_1, \vec{M}\rangle.P|\!\}_A) & \\ & \text{if } \neg u : var \text{ and } u \notin \text{ran}(A) \\ \{\!|\overline{y}\langle\vec{z}, M_1, \vec{M}\rangle.P|\!\}_A) & \text{if } \neg u : var \text{ and } \{u/y\} \in A \\ (\nu y)(\{\!|\{M_1/y\}|\!\}_A \mid \{\!|\overline{u}\langle\vec{z}, y, \vec{M}\rangle.P|\!\}_A) & \\ & \text{if } u : var \text{ and } M \notin \text{ran}(A) \\ \{\!|\overline{u}\langle\vec{z}, y, \vec{M}\rangle.P|\!\}_A) & \text{if } u : var \text{ and } \{M/y\} \in A \end{cases}$$

At last we encode active substitutions. We simply factor out the terms, which are not variables, one at a time.

$$\{\!|\{f(\vec{z}, M_1, \vec{M})/x\}|\!\}_A \stackrel{\text{def}}{=} (\nu y)(\{\!|\{M_1/y\}|\!\}_A \mid \{\!|\{f(\vec{z}, y, \vec{M})/x\}|\!\}_A) \quad \text{if } M_1 \notin \text{ran}(A)$$
$$\{\!|\{f(\vec{z}, M_1, \vec{M})/x\}|\!\}_A \stackrel{\text{def}}{=} \{\!|\{f(\vec{z}, y, \vec{M})/x\}|\!\}_A \quad \text{if } \{M_1/y\} \in A$$

where, in the encoding of output and active substitutions, we allow the vector $\vec{z}$ to be empty. Next we give an example of how the encoding works.

Example 3.4.1
Consider the process

$$A \stackrel{\text{def}}{=} a(x).\overline{x}\langle b\rangle \mid \{a/y\} \mid (\nu z)(\overline{y}\langle z\rangle \mid \{b/z\}).$$

Then the encoding of A is

$$\{\!|A|\!\}_A = (\nu z')(y(x).\overline{x}\langle z'\rangle \mid \{b/z'\}) \mid \{a/y\} \mid (\nu z)(\overline{y}\langle z\rangle \mid \{b/z\}).$$

We have that $A \equiv \{\!|A|\!\}_A$ which is generally proved by the next lemma. △

At last we prove that an extended process A and its encoding $\{\!|A|\!\}_A$ are structurally equivalent.

Lemma 3.4.2
Let $A \in \mathcal{A}$. Then $\{\!|A|\!\}_A \equiv A$.

Proof
The proof is by structural induction. Therefore we go through each of the cases in the definition of the encoding $\{\!|\,|\!\}_A$. The cases where $\{\!|\,|\!\}_A$ is homomorphic or the identity either follows by applying the induction hypothesis or trivially, thus these cases are omitted.

The first case we consider is $A =$ if $M = N$ then P else Q. We need to remember which process we originally encoded, say B, thus we look at $\{\!|A|\!\}_B$. This case is very similar to the proof of Lemma 3.2.5, i.e. we use that

$$A\{M/x\} \equiv (\nu x)(\{M/x\} \mid A) \tag{3.2}$$

as shown by the equivalences in (3.1).

$$\begin{aligned}\text{if } M = N \text{ then } P \text{ else } Q &\equiv (\nu x)(\{N/x\} \mid \text{if } M = x \text{ then } P \text{ else } Q)\\ &\equiv (\nu x, y)(\{N/x\} \mid \{M/y\} \mid \text{if } y = x \text{ then } P \text{ else } Q)\end{aligned}$$

The conclusion follows by the induction hypothesis. This proves the case where M and N are not in the range of B. If one or both are we use the (SC-Subst) axiom to obtain the conclusion.

Both input and output are very similar to the previous case so these are omitted.

The last case is when $A = \{f(z_1, \ldots, z_k, M_1, \ldots, M_l)/x\}$. Again assume that we originally encoded the process B. Now assume that $\neg M_1 : var$ and that M_1 is not in the range of B. Then we use (3.2) again and we get

$$\begin{aligned}\{f(z_1, \ldots, z_k, M_1, \ldots, M_l)/x\} \equiv (\nu y)(\{M_1/y\} \mid\\ \{f(z_1, \ldots, z_k, y, M_2, \ldots M_l)/x\})\end{aligned}$$

We can continue in this way, factoring out all the terms M_i which are not variables and in the range of B. Then we obtain the statement in the lemma by applying the induction hypothesis and the (SC-Subst) axiom. ■

Chapter 4

Typed Appπ

In this chapter we define a simple type system for Appπ. This includes giving type rules for well-formed environments, for Appπ-terms and for Appπ-processes and we prove a subject reduction theorem for the type system. The type system will catch the most elementary errors such as arity errors and matching of terms of different type. We have also included a rule for subtyping in the type system. We do not, however, include recursive types so we are not able type processes such as $\overline{a}\langle a\rangle.P$. We start with some basic definitions.

4.1 Basic definitions and terminology

We now introduce some basic terminology related to type systems. The definitions are standard so the reader may want to skip this section.

Definition 4.1.1 (Type assignment)
A type assignment is an assignment of a type T to a name or variable v, denoted by $v : T$

Given some type assignments one can form a type environment.

Definition 4.1.2 (Well-formed type environment)
A well-formed type environment Γ is a finite set of type assignments where all names and variables are distinct.

The domain of Γ is $\mathrm{dom}(\Gamma) = \{v \mid \exists T.\{v : T\} \in \Gamma\}$. It will sometimes be necessary to extend a type environment Δ with another one, Γ.

Definition 4.1.3 (Extension of type environments)
Let Γ and Δ be type environments. We then say that Γ extends Δ if the following holds.

- $dom(\Delta) \subseteq dom(\Gamma)$
- *If $v \in dom(\Delta)$ such that $\{v : T\} \in \Delta$, then $\{v : T\} \in \Gamma$.*

If $\Delta = \{v_1 : T_1, \ldots, v_n : T_n\}$ and $\Gamma = \{u_1 : S_1, \ldots, u_n : T_n\}$ are type environments we say that we extend Δ with Γ when taking the union of the two. When extending an environment Γ with another Δ we assume that $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$, and denote the resulting environment by $\Gamma \uplus \Delta$.

Let Γ be a type environment and E an expression which may be either a term, a process, or an extended process. A type judgment $\Gamma \vdash E : T$ is then an assertion that the expression E has type T under the assumptions given in Γ. The meaning of the type judgments will be specified in greater detail after we have introduced types in the next section. If $\Gamma \vdash E : T$ we sometimes simply say that E is well typed if Γ and T are not important.

4.2 A simple type system for Appπ

In this section we describe a simple type system for Appπ. The type system is a small extension of a type system for the π-calculus as defined in [SW01]. In addition to the type rules in [SW01] we introduce rules for typing terms and extended processes.

We distinguish between channel types and value types. Channel types are assigned to subjects of prefixes, while value types are assigned to the objects of prefixes. Value types can be basic types. In order to type processes capable of mobility it is necessary to allow value types to also be channel types. Given a value type V, channel types are constructed by the unary type constructor $\mathrm{ch}(V)$.

Let Γ be a type environment and E an expression which may be either a term, a process, or an extended process. A type judgment $\Gamma \vdash E : T$ then asserts the following depending on E. If E is a term, then T is value type, i.e. a basic type or a channel type. Thus $\Gamma \vdash M : V$ asserts that M has value type under the assumptions of Γ. For processes we introduce the behavior type $\checkmark$. $\Gamma \vdash P : \checkmark$ then asserts that P respects the type assertions in Γ. Similarly for A. Finally the judgment ($\vdash \Gamma$ well-formed) means that Γ is a well-formed type environment. We now define the types system for Appπ, starting with the types.

Definition 4.2.1 (Types)
Let B be a set of basic types. The types for Appπ are generated by the grammar:

$$\begin{aligned} S, T &::= V \mid \checkmark && \text{(Types)} \\ V &::= L \mid b_1 \mid \cdots \mid b_n && \text{(Value types)} \\ L &::= \mathrm{ch}(V_1, \ldots, V_n), && \text{(Channel types)} \end{aligned}$$

where $b_1, \ldots, b_n \in B$ are basic types.

We write $\vec{x} : \vec{T}$ for $x_1 : T_1, \ldots, x_n : T_n$. The syntax for typed Appπ-calculus remains unchanged from the one for untyped Appπ, see Section 3.1. The reduction relation for typed Appπ is also defined in the same way as in Section 3.2, and the transition relation is defined in Section 3.3.3.

Note that a value type can also be a channel type. On the other hand a channel type is always constructed from value types. This conforms to the intuitive idea

that value types are more general than channel types. For instance, one would not expect a value of basic type to be capable of carrying other values, although it certainly should be possible to send or receive such a value. The rules for well-formed environments are as follows.

Definition 4.2.2 (Well-formed environment)

$$\text{(E-Empty)}\frac{}{\vdash \emptyset \ \textit{well-formed}}$$

$$\text{(E-NamVar)}\frac{\vdash \Gamma \ \textit{well-formed} \qquad u \notin \textit{dom}(\Gamma)}{\vdash \Gamma \uplus u : T \ \textit{well-formed}}$$

Being able to form type environments one can define the type rules for the terms. We do this as follows. For the rest of this chapter fix a presentation $\Pi = (\Sigma, \Phi)$.

Note the following. If $\Phi \vdash_\Sigma M = N$ and P is a process containing M, then the (SC-rewrite) rule can be applied to rewrite P to a structurally equivalent process P' containing N, and vice versa. Therefore, in order to ensure that structural equivalence preserves well-typedness, we require that the type system assigns equal types to terms that are equated by the equational theory. In specific instances of Appπ using the type system we now give, this can be ensured by inspection of the rules.

Definition 4.2.3 (Type rules for terms)
Let M be a term, T a type, and Γ a well-formed type environment. Then the type judgment $\Gamma \vdash_{\mathcal{M}} M : T$ holds if it can be derived by application of the following rules.

$$\text{(T-NamVar)}\frac{\vdash \Gamma \ \textit{well-formed} \qquad \{u : T\} \in \Gamma}{\Gamma \vdash_{\mathcal{M}} u : V}$$

For each function name $f \in \Omega$ a type rule of the form

$$\text{(T-Func)}\frac{\Gamma \vdash_{\mathcal{M}} M_1 : V_1 \cdots \Gamma \vdash_{\mathcal{M}} M_n : V_n \qquad \alpha(f) = n}{\Gamma \vdash_{\mathcal{M}} f(M_1, \ldots, M_n) : V}$$

The type rules for processes are standard, we only comment on the rules (T-Ite), (T-Inp), and (T-Out). In the rule (T-Ite) the types for M and N must be equal. Furthermore, the two branches P and Q must also be well typed. In a process $v(x).P$ the value v must have channel type constructed from the type of x and since x is a binding occurrence in P, it is natural to also require P to be well typed in Γ appended with a type assignment for x. The last rule we comment on is (T-Out). In a process $\overline{v}\langle M\rangle.P$ we require v to have channel type, capable of carrying values having the type of M and that P is well typed, all in the same environment Γ. The rules for processes are as following.

Definition 4.2.4 (Type rules for processes)
Let $P \in \mathcal{A}$ be a process, Γ a well-formed type environment, and $\checkmark$ the behavior type. Then the judgment $\Gamma \vdash_{\mathcal{A}} P : \checkmark$ holds if it can be derived by application of the following rules.

$$(\text{T-Nil})\frac{\Gamma \text{ well-formed}}{\Gamma \vdash_{\mathcal{A}} \mathbf{0} : \checkmark}$$

$$(\text{T-Par})\frac{\Gamma \vdash_{\mathcal{A}} P : \checkmark \quad \Gamma \vdash_{\mathcal{A}} Q : \checkmark}{\Gamma \vdash_{\mathcal{A}} P \mid Q : \checkmark}$$

$$(\text{T-Rep})\frac{\Gamma \vdash_{\mathcal{A}} P : \checkmark}{\Gamma \vdash_{\mathcal{A}} !P : \checkmark}$$

$$(\text{T-Res})\frac{\Gamma \uplus \{u : V\} \vdash_{\mathcal{A}} P : \checkmark}{\Gamma \vdash_{\mathcal{A}} (\nu u)P : \checkmark}$$

$$(\text{T-Ite})\frac{\Gamma \vdash_{\mathcal{M}} M : S \quad \Gamma \vdash_{\mathcal{M}} N : S \quad \Gamma \vdash_{\mathcal{A}} P : \checkmark \quad \Gamma \vdash_{\mathcal{A}} Q : \checkmark}{\Gamma \vdash_{\mathcal{A}} \text{if } M = N \text{ then } P \text{ else } Q : \checkmark}$$

$$(\text{T-Inp})\frac{\Gamma \uplus \{\vec{x} : \vec{V}\} \vdash_{\mathcal{A}} P : \checkmark \quad \Gamma \vdash_{\mathcal{M}} u : \mathrm{ch}(\vec{V})}{\Gamma \vdash_{\mathcal{A}} u(\vec{x}).P : \checkmark}$$

$$(\text{T-Out})\frac{\Gamma \vdash_{\mathcal{M}} \vec{M} : \vec{V} \quad \Gamma \vdash_{\mathcal{M}} u : \mathrm{ch}(\vec{V}) \quad \Gamma \vdash P : \checkmark}{\Gamma \vdash_{\mathcal{A}} \overline{u}\langle \vec{M} \rangle.P : \checkmark}$$

In order to type extended processes we extend the rules above with the following rule for active substitutions.

$$(\text{T-Subs})\frac{\Gamma \vdash_{\mathcal{M}} M : V \quad \Gamma \vdash_{\mathcal{M}} x : V}{\Gamma \vdash_{\mathcal{A}} \{M/x\} : \checkmark}$$

Note that since $\Gamma \uplus \{x : T\}$ implies that $x \notin \mathrm{dom}(\Gamma)$, this again implies, by the rules (T-Inp) and (T-Res), that bound names are not in Γ. Since it is not hard to determine which of the judgments $\vdash_{\mathcal{A}}$ or $\vdash_{\mathcal{M}}$ we are referring to, we will usually omit the index and simply write $\vdash$.

We end this section with a few examples.

Example 4.2.5
Consider the following process:

$$b(y) \mid \overline{a}\langle b \rangle \mid a(x).\overline{x}\langle c \rangle.$$

This process is well-typed under the environment $\Gamma = \{b : \mathrm{ch}(T), c : T, a : \mathrm{ch}(\mathrm{ch}(T)\}$ *as is demonstrated by the derivation below.*

$$\frac{\dfrac{\Gamma \vdash y : T, b : \mathrm{ch}(T)}{\Gamma \vdash b(y) : \checkmark} \qquad \dfrac{\Gamma \vdash b : \mathrm{ch}(T), a : \mathrm{ch}(\mathrm{ch}(T))}{\Gamma \vdash \overline{a}\langle b \rangle : \checkmark} \qquad A}{\Gamma \vdash b(y) \mid \overline{a}\langle b \rangle \mid a(x).\overline{x}\langle c \rangle : \checkmark},$$

where A is the inference

$$\frac{\dfrac{\Gamma \uplus \{x : \mathrm{ch}(T)\} \vdash x : \mathrm{ch}(T), c : T}{\Gamma \uplus \{x : \mathrm{ch}(T)\} \vdash \overline{x}\langle c \rangle} \qquad \Gamma \vdash a : \mathrm{ch}(\mathrm{ch}(T))}{\Gamma \vdash a(x).\overline{x}\langle c \rangle}$$

$\triangle$

Example 4.2.6
Consider the process

$$\overline{a}\langle b\rangle \mid a(x).\overline{x}\langle c\rangle \mid b(y).\overline{y}\langle c\rangle.$$

We show that this process can not be well-typed.

$$\dfrac{\dfrac{\Gamma \vdash a : \mathrm{ch}(\mathrm{ch}(T)), \mathbf{b} : ???}{\Gamma \vdash \overline{a}\langle b\rangle : \checkmark} \quad A \quad B}{\Gamma \vdash \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle c\rangle \mid b(y).\overline{y}\langle c\rangle : \checkmark}$$

where A is the inference

$$\dfrac{\dfrac{\Gamma \uplus \{y : \mathrm{ch}(T)\} \vdash y : \mathrm{ch}(T), c : T}{\Gamma \uplus \{y : \mathrm{ch}(T)\} \vdash \overline{y}\langle c\rangle : \checkmark} \qquad \Gamma \vdash b : \mathrm{ch}(\mathrm{ch}(T))}{\Gamma \vdash b(y).\overline{y}\langle c\rangle : \checkmark}$$

and B is the inference

$$\dfrac{\dfrac{\Gamma \uplus \{x : \mathrm{ch}(T)\} \vdash x : \mathrm{ch}(T), c : T}{\Gamma \uplus \{x : \mathrm{ch}(T)\} \vdash \overline{x}\langle c\rangle : \checkmark} \qquad \Gamma \vdash a : \mathrm{ch}(\mathrm{ch}(T))}{\Gamma \vdash a(x).\overline{x}\langle c\rangle : \checkmark}$$

Clearly from this inference one sees that there can be no consistent type assignment to the names in this process which make the process well-typed. $\triangle$

4.3 Subject reduction for typed Appπ

In this section we prove an important property of the type system commonly referred to as subject reduction. Type checking is performed by a static analysis of the source program. However, we often want a typed judgment about a process to express something about its dynamic behavior. In other words subject reduction relates the type rules with the operational semantics.

Before we set out to prove subject reduction, we need a few lemmas whose statements and proofs are similar to the corresponding ones in [SW01, p.244].

Lemma 4.3.1
Let E be a term or an extended process. If $\Gamma \vdash E : T$ and $u \notin dom(\Gamma)$. Then $u \notin \mathrm{fv}(E) \cup \mathrm{fn}(E)$.

Proof
The proof is carried out by induction in the height of the derivation tree of the type assertion.

Induction basis: We treat the case where the assertion was derived directly from one of the axioms. If $\Gamma \vdash E : T$ was derived using (T-NamVar). Then the theorem is vacuously true since the assumption that $u \notin \mathrm{dom}(\Gamma)$ is not satisfied. In the latter case the theorem is also trivially fulfilled since $\mathrm{fv}(\mathbf{0}) = \emptyset = \mathrm{fn}(\mathbf{0})$.

Induction hypothesis: Assume the lemma holds for derivations of height $n-1$.

Induction step: We prove this step by a case by case analysis of the last rule used in the derivation. Most of the cases are immediate, so we only give details for some of them.

(T-Func). Suppose the root of the derivation tree is

$$\frac{\Gamma \vdash M_1 : V_1 \cdots \Gamma \vdash M_n : V_n \qquad \alpha(f) = n}{\Gamma \vdash f(M_1, \ldots, M_n) : V}.$$

By the induction hypothesis we can then assume that if $u \notin \mathrm{dom}(\Gamma)$, then $u \notin \mathrm{fv}(M_i) \cup \mathrm{fn}(M_i)$ for all $1 \leq i \leq n$. But then $u \notin \mathrm{fv}(f(M_1, \ldots, M_n)) \cup \mathrm{fn}(f(M_1, \ldots, M_n))$.

Suppose the last rules is (T-Res), and choose $v \neq u$ such that $v \notin \mathrm{dom}(\Gamma)$. Then $v \notin \mathrm{dom}(\Gamma \uplus \{u : T\})$ so by the induction hypothesis we conclude that $v \notin \mathrm{fv}(P) \cup \mathrm{fn}(P)$, which implies that $v \notin \mathrm{fv}((\nu u)P) \cup \mathrm{fn}((\nu u)P)$. If $v = u$, then clearly $v \notin \mathrm{fv}((\nu u)P) \cup \mathrm{fn}((\nu u)P)$.

If the root of the derivation tree was obtained by (T-Ite), then clearly the result holds. The case for (T-Inp) is similar to the one for (T-Res). So take v so $v \notin \mathrm{dom}(\Gamma)$. Since by assumption $u(\vec{x}).P$ is well typed, this implies that $u \neq v$. Furthermore, if $v \in \vec{x}$, then $v \notin \mathrm{fv}(u(x).P)$. So assume that $v \neq u$ and $v \notin \vec{x}$. Then also $v \notin \mathrm{dom}(\Gamma \uplus \{x : T\})$ is undefined. So by the induction hypothesis we know that $v \notin \mathrm{fv}(P) \cup \mathrm{fn}(P)$, hence $v \notin \mathrm{fv}(u(x).P) \cup \mathrm{fn}(u(x).P)$.

For the rule (T-Out). The reasoning is similar to the one above, noting that if $v \notin \mathrm{dom}(\Gamma)$, then this implies, by the well typedness of $u\langle\vec{M}\rangle$, that $u \neq v$. The conclusion then follows by the induction hypothesis.

Finally if the last rule in the derivation is (T-Subs) the lemma follows by the induction hypothesis and the definition of free names and variables for an active substitution. That is $\mathrm{fv}(\{M/x\}) = \mathrm{fv}(M) \cup \{x\}$ and $\mathrm{fn}(\{M/x\}) = \mathrm{fn}(M)$.

■

The proofs of the next three lemmas are similar to the one for the previous lemma. Therefore we just give a short description of the proof technique for these lemmas.

Lemma 4.3.2
For every Γ and u there is at most one type T such that $\Gamma \vdash u : T$.

Proof
Induction in the height of the derivation of the judgment. ■

The next lemma states that one may remove superfluous type assignments from a type environment, i.e. strengthen the assumptions.

Lemma 4.3.3 (Strengthening)
If $\Gamma \uplus \{u : S\} \vdash E : T$, and u is not free in E. Then $\Gamma \vdash E : T$.

Proof
Induction in the height of the derivation of the judgment. ■

Conversely, adding type assignments to the type environment does not influence the well typedness of an expression.

Lemma 4.3.4 (Weakening)
If $\Gamma \vdash E : T$ *then* $\Gamma \uplus \{u : S\} \vdash E : T$ *for any type* S *and* u *such that* $u \notin dom(\Gamma)$.

Proof
Induction in the height of the derivation of the judgment. ■

The strengthening as well as the weakening lemma easily generalize to the polyadic case. I.e. in strengthening we assume that if $\Gamma \uplus \{\vec{u} : \vec{S}\} \vdash E : T$ and $\vec{u} \notin \mathrm{fv}(E)$, then $\Gamma \vdash E : T$. Similarly the additional assumption in the weakening lemma is that $\vec{u} \notin \mathrm{dom}(\Gamma)$.

Lemma 4.3.5
Suppose $\Gamma \vdash P : \checkmark$ *and* $\Gamma \uplus \Delta \vdash (\nu\vec{x})\{\vec{M}/\vec{x}\} : \checkmark$, *where* $dom(\Delta) = \mathrm{fv}(\vec{M}) \cup \mathrm{fn}(\vec{M})$. *Then*

$$\Gamma \uplus \Delta \vdash P \mid (\nu\vec{x})\{\vec{M}/\vec{x}\} : \checkmark .$$

Proof
The statements is obtained by applying weakening. ■

Thus since type environments only make assertions about the free names in a process, a consequence of Lemma 4.3.5, and the requirement that (SC-Rewrite) is sound with respect to typing, is that structural equivalence preserves well typing.

Lemma 4.3.6 (Substitution lemma)
Suppose the following hold:

(i) $\Gamma \vdash E : T$

(ii) $\Gamma \vdash x : S$

(iii) $\Gamma \vdash v : S$.

Then $\Gamma \vdash E\{v/x\} : T$.

Proof
The derivation for $E\{v/x\} : T$ is obtained from the derivation of $E : T$ as follows. Consider all leaves of the derivation tree of E of the form

$$\frac{\{x : S\} \in \Delta \uplus \{x : S\} \qquad \vdash \Delta \uplus \{x : S\} \text{ well-formed}}{\Delta \uplus \{x : S\} \vdash x : S}, \tag{4.1}$$

where $\Delta \vdash v : S$. For each of these leaves do the following. For each node on the path from the leaf to the root, both including, replace an occurrence of x to the right of a $\vdash$ with v. We prove by induction in the height of the derivation that this algorithm provides a derivation tree for $E\{v/x\}$.

Induction basis: The only case to consider, besides the (T-Nil) rule which is trivial, is when the derivation is of the form in (4.1). By assumption we know that $\Delta \vdash v : S$ and thus by the Weakening lemma, Lemma 4.3.4, we have $\Delta \uplus x : S \vdash v : S$.

Induction hypothesis: Assume the described algorithm works for all derivations of height $n-1$, where $n > 1$.

Induction step: Consider a derivation of height n. There are eight possible different roots of the derivation tree. We only prove three of the cases as they are all quite similar.

First consider the rule (T-Par)

$$\frac{\Gamma \vdash P : \checkmark \quad \Gamma \vdash Q : \checkmark}{\Gamma \vdash P \mid Q : \checkmark}.$$

The induction hypothesis applies to both P and Q, thus by the (T-Par) rule we get as desired

$$\frac{\Gamma \vdash P\{v/x\} : \checkmark \quad \Gamma \vdash Q\{v/x\} : \checkmark}{\Gamma \vdash P\{v/x\} \mid Q\{v/x\} : \checkmark}.$$

As the second case consider the (T-Out) rule

$$\frac{\Gamma \vdash u' : V \quad \Gamma \vdash u : ch(V) \quad \Gamma \vdash P : \checkmark}{\Gamma \vdash \overline{u}\langle u' \rangle.P : \checkmark}.$$

Now if $x = u$ or $x = u'$ the statement follows by the induction basis and the induction hypothesis, otherwise only the induction hypothesis is needed to obtain

$$\frac{\Gamma \vdash u'\{v/x\} : V \quad \Gamma \vdash u\{v/x\} : ch(V) \quad \Gamma \vdash P\{v/x\} : \checkmark}{\Gamma \vdash \overline{u}\{v/x\}\langle u'\{v/x\} \rangle.P\{v/x\} : \checkmark}.$$

At last consider the (T-Func) rule

$$\frac{\Gamma \vdash M_1 : V_1 \cdots \Gamma \vdash M_n : V_n \quad \alpha(f) = n}{\Gamma \vdash f(M_1, \ldots, M_n) : V}.$$

Again the induction hypothesis applies to each of the terms $M_1, \ldots, M_n$ and hence we get

$$\frac{\Gamma \vdash M_1\{v/x\} : V_1 \cdots \Gamma \vdash M_n\{v/x\} : V_n \quad \alpha(f) = n}{\Gamma \vdash f(M_1\{v/x\}, \ldots, M_n\{v/x\}) : V}.$$

■

As a corollary to the substitution lemma one gets that the result of the lemma is also valid for substitutions of the form $\{\vec{x}/\vec{y}\}$.

Lemma 4.3.7
Suppose the following holds:

(i) $\Gamma \vdash E : T$

(ii) $\Gamma \vdash \vec{x} : \vec{S}$

(iii) $\Gamma \vdash \vec{v} : \vec{S}$.

Then $\Gamma \vdash E\{\vec{v}/\vec{x}\} : T$.

Note that, by strengthening, we also have that the conclusion in the corollary is valid for a type environment Δ where $\Delta \uplus \vec{x} : \vec{S} = \Gamma$.

With these lemmas we are now ready to prove the subject reduction theorem.

Theorem 4.3.8 (Subject reduction)
Suppose $\Gamma \vdash A : \checkmark$ *and* $A \xrightarrow{\alpha} A'$. *Then one of the following holds.*

(i) *If* $\alpha = \tau$. *Then* $\Gamma \vdash A' : \checkmark$.

(ii) *If* $\alpha = a(\vec{M})$. *Then there is* $\vec{T}$ *such that*

  (a) $\Gamma \vdash a : \mathrm{ch}(\vec{T})$.

  (b) *if* $\Gamma \vdash \vec{M} : \vec{T}$ *then* $\Gamma \vdash A' : \checkmark$.

(iii) *If* $\alpha = (\nu\vec{u})\overline{a}\langle\vec{v}\rangle$ *then there exists* $\vec{T}$ *such that*

  (a) $\Gamma \vdash a : \mathrm{ch}(\vec{T})$.

  (b) $\Gamma \uplus \{\vec{u} : \vec{S}\} \vdash \vec{v} : \vec{T}$.

  (c) $\Gamma \uplus \{\vec{u} : \vec{S}\} \vdash A' : \checkmark$.

Proof
The proof is by induction on the height of the inference of $A \xrightarrow{\alpha} A'$.

Induction basis: We consider the rules (Comm),(Then), (Else), (Input), and (Output). For the (Comm) rule assume $P = \overline{a}\langle\vec{x}\rangle.P_1 \mid a(\vec{y}).P_2 \xrightarrow{\tau} P_1 \mid P_2\{\vec{x}/\vec{y}\} = P'$. We argue that case (i) of the theorem is satisfied.

$$\dfrac{\dfrac{\Gamma \vdash \vec{x} : \vec{V} \quad \Gamma \vdash P_1 : \checkmark}{\Gamma \vdash a : \mathrm{ch}(\vec{V})}}{\Gamma \vdash \overline{a}\langle\vec{x}\rangle.P_1 : \checkmark} \qquad \dfrac{\Gamma \uplus \{\vec{y} : \vec{V}\} \vdash P_2 : \checkmark \quad \Gamma \vdash a : \mathrm{ch}(\vec{V})}{\Gamma \vdash a(\vec{y}).P_2 : \checkmark}$$

$$\Gamma \vdash \overline{a}\langle\vec{x}\rangle.P_1 \mid a(\vec{y}).P_2 : \checkmark$$

Now observe the following. By the above derivation $\Gamma \uplus \{\vec{y} : \vec{V}\} \vdash P_2 : \checkmark$ and by the weakening lemma we know that $\Gamma \uplus \{\vec{y} : \vec{V}\} \vdash \vec{x} : \vec{V}$, and of course $\Gamma \uplus \{\vec{y} : \vec{V}\} \vdash \vec{y} : \vec{V}$. Thus the substitution lemma yields $\Gamma \uplus \{\vec{y} : \vec{V}\} \vdash P_2\{\vec{x}/\vec{y}\} : \checkmark$. Since $\vec{y} \notin \mathrm{fv}(P_2\{\vec{x}/\vec{y}\})$ using strengthening yields $\Gamma \vdash P_2\{\vec{x}/\vec{y}\} : \checkmark$. Putting it all together, by the derivation above we know that $\Gamma \vdash P_1 : \checkmark$, thus an application of the (T-Par) rule yields $\Gamma \vdash P_1 \mid P_2\{\vec{x}/\vec{y}\} : \checkmark$.

If $P \xrightarrow{\alpha} P'$ was derived using (Then) or (Else), case (i) is immediately satisfied. If $P = a(\vec{x}).P_1 \xrightarrow{a(\vec{M})} P_1\{\vec{M}/\vec{x}\} = P'$ we get.

$$\frac{\Gamma \uplus \{\vec{x} : \vec{T}\} \vdash P_1 : \checkmark \qquad \Gamma \vdash a : \mathrm{ch}(\vec{T})}{\Gamma \vdash a(\vec{x}).P_1 : \checkmark}$$

We now show that if $\Gamma \vdash \vec{M} : \vec{T}$ then $\Gamma \vdash P_1\{\vec{M}/\vec{x}\} : \checkmark$. Since $\vec{x} \notin \mathrm{fv}(\vec{M})$, applying weakening to obtain $\Gamma \uplus \{\vec{x} : \vec{T}\} \vdash a : \mathrm{ch}(\vec{T})$. Furthermore, $\Gamma \uplus \{\vec{x} : \vec{T}\} \vdash P_1 : \checkmark$ and $\Gamma \uplus \{\vec{x} : \vec{T}\} \vdash \vec{x} : \vec{T}$. Hence case (ii) is fulfilled by the substitution lemma. The rule (Output) is proved in a similar manner.

Induction hypothesis: Assume the statement of the theorem holds for shorter derivations.

Induction step: Consider the rule (Open), so $\alpha = (\nu\vec{u})\overline{a}\langle\vec{v}\rangle$. We show that case (iii) is fulfilled. Typing $(\nu u_1)A$ yields

$$\frac{\Gamma \uplus \{u_1 : S_1\} \vdash A : \checkmark}{\Gamma \vdash (\nu u_1)A : \checkmark}.$$

The (Open) rule states the following.

$$\frac{A \xrightarrow{(\nu u_2,\ldots,u_n)\overline{a}\langle\vec{v}\rangle} A'}{(\nu u_1)A \xrightarrow{(\nu\vec{u})\overline{a}\langle\vec{v}\rangle} A'}.$$

Since $\Gamma \uplus \{u_1 : S_1\} \vdash A : \checkmark$ the induction hypothesis yields the following.

- $\Gamma \uplus \{u_1 : S_1\} \vdash a : \mathrm{ch}(\vec{T})$
- $\Gamma \uplus \{u_1 : S_1\} \uplus \{u_2 : S_2, \ldots, u_n : S_n\} \vdash \vec{v} : \vec{T}$
- $\Gamma \uplus \{u_1 : S_1\} \uplus \{u_2 : S_2, \ldots, u_n : S_n\} \vdash A' : \checkmark$.

Hence since $u \neq a$ apply strengthening to the first bullet to obtain the first statement in case (iii). The other cases are clearly fulfilled.

Now assume the last rule is (Res). That is, assume $\Gamma \vdash (\nu u)A : \checkmark$, and that $(\nu u)A \xrightarrow{\alpha} (\nu u)A'$. Then by (T-Res) we know that $\Gamma \uplus \{u : T\} \vdash A : \checkmark$. Thus consider the following sub-cases depending on α.

- If $\alpha = \tau$, the induction hypothesis yields that $\Gamma \uplus \{u : T\} \vdash A' : \checkmark$. Applying (T-Res) now yields that $\Gamma \vdash (\nu u)A' : \checkmark$.
- If $\alpha = \overline{a}\langle\vec{v}\rangle$, $u \notin \vec{v}$ the induction hypothesis yields $\Gamma \uplus \{u : T\} \vdash a : \mathrm{ch}(\vec{T})$, $\Gamma \uplus \{u : T\} \vdash \vec{v} : \vec{T}$, and $\Gamma \uplus \{u : T\} \vdash A' : \checkmark$. Since $u \neq a$ we can apply strengthening to infer $\Gamma \vdash a : \mathrm{ch}(\vec{T})$, similarly since $u \notin \vec{v}$ we obtain $\Gamma \vdash v : \vec{T}$. Finally the (T-Res) rule yields $\Gamma \vdash (\nu u)A' : \checkmark$.
- If $\alpha = a(\vec{M})$ applying the induction hypothesis yields $\Gamma \uplus \{u : T\} \vdash a : \mathrm{ch}(\vec{T})$ and if $\Gamma \uplus \{u : T\} \vdash \vec{v} : \vec{T}$ then $\Gamma \uplus \{u : T\} \vdash A' : \checkmark$. By the same reasoning as above we can conclude that $\Gamma \vdash a : \mathrm{ch}(\vec{T})$ and if $\Gamma \vdash v : T$ then $\Gamma \vdash (\nu u)A' : \checkmark$.
- If $\alpha = (\nu\vec{v})\overline{a}\langle\vec{w}\rangle$ for $\vec{v} \subseteq \vec{w}$, the induction hypothesis yields the following

- $\Gamma \uplus \{u : T\} \vdash a : \mathrm{ch}(\vec{T})$,
- $\Gamma \uplus \{u : T\} \uplus \{\vec{v} : \vec{S}\} \vdash \vec{w} : \vec{T}$, and
- $\Gamma \uplus \{u : T\} \uplus \{\vec{v} : \vec{S}\} \vdash A' : \checkmark$.

Again, since $u \notin \mathrm{n}(\alpha)$, we reason by using strengthening to conclude that $\Gamma \vdash a : \mathrm{ch}(\vec{T})$, $\Gamma \uplus \{\vec{v} : \vec{S}\} \vdash \vec{w} : \vec{T}$. And by the (T-Res) rule we have $\Gamma \uplus \{\vec{v} : \vec{T}\} \vdash (\nu u)A' : \checkmark$.

Now, suppose the last rule in the derivation is (Struct). The difficult cases is when the (SC-Alias) or the (SC-Rewrite) axiom has been used. But by convention the encoding preserves types with respect to the (SC-Rewrite)-axiom. If the (SC-Alias) axiom was applied directly, the theorem holds vacuously since neither $\mathbf{0}$ or $(\nu\vec{x})\{\vec{M}/\vec{x}\}$ has any transitions. The problematic case is when $P = P' \mid \mathbf{0}$. But by Lemma 4.3.5 the theorem follows by extending Γ with assignments for the free names in M.

The last rule is the (Par) rule. The proof of this case is carried out by using the same argument as we did for the (Res) rule, hence the details are omitted.

■

4.4 Subtyping

Subtyping is similar to inclusion of sets of values of certain types. For instance we have that $\mathbb{N} \subset \mathbb{R}$ and one could argue that $\mathbb{N}$ is a subtype of $\mathbb{R}$, since if a function f is well defined on an expression of type $\mathbb{R}$, then specifically it should be well defined on an expression of type $\mathbb{N}$. It is worth noticing that the reverse is not always possible, i.e. to replace a value u of type $\mathbb{N}$ with a value v of type $\mathbb{R}$. We could for instance ask for $suc(u)$ which of course is undefined for the value v.

Now let S and T be types. Then if $S \leq T$ we say that S is a subtype of T and that T is a super-type of S. We extend Definition 4.2.3 with the following rule which captures the intuition described above.

$$(\text{T-Subsumption})\ \frac{\Gamma \vdash M : S \qquad S \leq T}{\Gamma \vdash M : T}$$

With this rule added to Definition 4.2.3 we should of course make sure that the lemmas and theorem in Section 4.3 are still valid.

Proof (of Lemma 4.3.1)
By the proof of Lemma 4.3.1 it is only necessary to check the case where the last rule in the derivation is (T-Subsumption), hence suppose the root of the tree is

$$\frac{\Gamma \vdash M : S \qquad S \leq T}{\Gamma \vdash M : T}$$

Then by the induction hypothesis we have that $x \notin \mathrm{fv}(M)$ which is what we wanted. ■

Proof (of Lemma 4.3.6)
Suppose the root of the tree is

$$\frac{\Gamma \vdash M : S \qquad S \leq T}{\Gamma \vdash M : T}$$

By the induction hypothesis and the (T-Subsumption) rule we have

$$\frac{\Gamma \vdash M\{v/x\} : S \qquad S \leq T}{\Gamma \vdash M\{v/x\} : T}$$

as desired. ■

Subtyping can be used when describing many common programming language constructs; it is indeed an essential feature in object-oriented languages. We need little of the vast amount of theory on this topic, but include a short discussion here on subtyping and input/output types in the π-calculus. These types is a refinement of the pure channel type which we have introduced. The need for these types can be illustrated by considering how one could introduce subtyping on the channel types. This example is taken from [SW01]. A type constructor t is called covariant if $S \leq T$ implies that $t(S) \leq t(T)$ and contravariant if $S \leq T$ implies that $t(T) \leq s(T)$.

Suppose the type constructor $\mathrm{ch}(T)$ is covariant, so we have the rule

$$\frac{S \leq T}{\mathrm{ch}(S) \leq \mathrm{ch}(T)},$$

and the basic types Int and $Real$ with the ordering $Int \leq Real$. Let $suc(x)$ be the successor function on integers. Using this rule and (T-Subsumption) one can now type the two processes

$$a(x).\overline{b}\langle suc(x)\rangle.\mathbf{0} \qquad \text{and} \qquad \overline{a}\langle 3.5\rangle.\mathbf{0}$$

under the same environment $\Gamma = \{a : \mathrm{ch}(Int), b : \mathrm{ch}(Int)\}$. However by the (T-Par)-rule the composition of the two processes should also be well-typed. But this can clearly not be the case since

$$a(x).\overline{b}\langle suc(x)\rangle.\mathbf{0} \mid \overline{a}\langle 3.5\rangle.\mathbf{0} \longrightarrow \overline{b}\langle suc(3.5)\rangle.\mathbf{0} \mid \mathbf{0}.$$

A similar example can be made if $\mathrm{ch}(T)$ is taken to be contravariant. The solution to this problem is to use input and output types. Then $\mathsf{o}S$ is the type of a name capable only of outputting values of type S, and $\mathsf{i}S$ is the type of a name capable only of inputting values of type S. If o is taken to be contravariant and i is covariant, the resulting type system will be sound.

We leave this issue for future work. The notion of subtyping we have already introduced is sufficient for the purpose of this report.

Chapter 5

Representing the spi-calculus in Appπ

In this chapter we will take a closer look at how the spi-calculus can be represented in Appπ. The spi-calculus is defined in Chapter 2. We will first define a presentation to obtain the necessary functions and data structures whereafter we show how various process constructs such as "case L of $\{x_1, \ldots, x_n\}_N$ in P" can be encoded. We prove full abstraction with respect to operational correspondence of this instantiation of spi in Appπ. The encoding we define here is also aimed at representing the type system for the spi-calculus proposed by Abadi [Aba97]. The type system for the spi-calculus presented in this article is aimed at representing security protocols which use shared-key cryptography. The main theorem states that if a protocol type-checks then the desired secrecy property of the protocol is guaranteed. In Section 5.4 we give and explain Abadi's type system whereafter we modify the type system from the previous chapter in order to obtain a type system which is equivalent to Abadi's.

5.1 Definition of a presentation

We now define a presentation $\Pi = (\Sigma, \Phi)$ where $\Sigma = (\Omega, \alpha)$ is a signature and Φ is a set of equations. See Appendix A for background on universal algebra. In [Aba97] Abadi works with three types of data, public, secret, and any where any is a super type of public and secret. Now the set Ω of function symbols is constituted by the function symbols in Table 5.1. Note that if f is a function symbol in Ω then f_α denotes the three functions $\mathsf{f}_{Pub}, \mathsf{f}_{Sec}$ and f_{Any}. The arity function α is also listed for each function name.

The fmt and cast functions are actually only needed for representing Abadi's type system. fmt is a special kind of tuple and the cast functions extracts the elements of an fmt tuple. If we only wanted to instantiate spi in Appπ we could have used the pair, fst and snd instead. Similarly we could have used just one decryption and encryption function. The need for these functions will become clear in Section 5.5.

$\mathsf{suc},\ \alpha(\mathsf{suc}) = 1$	$\mathsf{pred},\ \alpha(\mathsf{pred}) = 1$
$\mathsf{fmt},\ \alpha(\mathsf{fmt}) = 4$	$\mathsf{enc\text{-}s},\ \alpha(\mathsf{enc\text{-}s}) = 2$
$\mathsf{enc\text{-}p},\ \alpha(\mathsf{enc\text{-}p}) = 2$	$\mathsf{dec\text{-}s},\ \alpha(\mathsf{dec\text{-}s}) = 2$
$\mathsf{dec\text{-}p}_\alpha,\ \alpha(\mathsf{dec\text{-}p}_\alpha) = 2$	$\mathsf{pair},\ \alpha(\mathsf{pair}) = 2$
$\mathsf{fst},\ \alpha(\mathsf{fst}) = 1$	$\mathsf{snd},\ \alpha(\mathsf{snd}) = 1$
$\mathsf{cast\text{-}s},\ \alpha(\mathsf{cast\text{-}s}) = 1$	$\mathsf{cast\text{-}a},\ \alpha(\mathsf{cast\text{-}a}) = 1$
$\mathsf{cast\text{-}p},\ \alpha(\mathsf{cast\text{-}p}) = 1$	$\mathsf{cast}_\alpha,\ \alpha(\mathsf{cast}_\alpha) = 1$
$0,\ \alpha(0) = 0$	

Table 5.1: The signature Σ.

Finally we are in position to give the equations. The set Φ of equations are given in Table 5.2.

$$\begin{aligned}
\mathsf{pred}(\mathsf{suc}(x)) &= x \\
\mathsf{dec\text{-}p}_\alpha(\mathsf{enc\text{-}p}(x,k),k) &= x \\
\mathsf{dec\text{-}s}(\mathsf{enc\text{-}s}(x,k),k) &= x \\
\mathsf{fst}(\mathsf{pair}(x,y)) &= x \\
\mathsf{snd}(\mathsf{pair}(x,y)) &= y \\
\mathsf{cast\text{-}s}(\mathsf{fmt}(x,y,z,n)) &= x \\
\mathsf{cast\text{-}a}(\mathsf{fmt}(x,y,z,n)) &= y \\
\mathsf{cast\text{-}p}(\mathsf{fmt}(x,y,z,n)) &= z \\
\mathsf{cast}_\alpha(\mathsf{fmt}(x,y,z,n)) &= n
\end{aligned}$$

Table 5.2: The equations Φ.

The equations ensure that for instance the equation $\mathsf{suc}(\mathsf{pred}(M)) = M$ is only provable if there exists an N such that $M = \mathsf{suc}(N)$. Using Π the next section describes how to represent spi in Appπ.

5.2 Encoding of process constructs

The syntax for spi-calculus-processes, Definition 2.1.1, contains some constructs which are not present in the syntax of Appπ-processes. In this section we give an encoding $[\![\cdot]\!]$ of spi-calculus processes as Appπ-processes. The encoding also anticipates that it is utilised in Section 5.5 where Abadi's type system is represented in Appπ.

The encoding is a homomorphism on most of the process expressions, for instance $[\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$. Thus we only give explicit definitions for those that are not homomorphisms. Using the presentation Π, and in particular the functions in Ω encoding of terms is as follows:

$$
\begin{aligned}
[\![(M,N)]\!] &\stackrel{\text{def}}{=} \mathsf{pair}([\![M]\!],[\![N]\!]) \\
[\![suc(M)]\!] &\stackrel{\text{def}}{=} \mathsf{suc}([\![M]\!]) \\
[\![\{M_1,\ldots,M_n\}_N]\!] &\stackrel{\text{def}}{=} \mathsf{enc\text{-}p}(\mathsf{pair}([\![M_1]\!],\ldots,[\![M_n]\!]),[\![N]\!]) \\
[\![\{M_1,\ldots,M_4\}_N]\!] &\stackrel{\text{def}}{=} \mathsf{enc\text{-}s}(\mathsf{fmt}([\![M_1]\!],\ldots,[\![M_4]\!]),[\![N]\!]),
\end{aligned}
$$

where $\mathsf{pair}(M_1,M_2,\ldots,M_n)$ is shorthand for $\mathsf{pair}(M_1,\mathsf{pair}(M_2,\ldots,M_n))$. Sometimes we will just write $[\![\vec{M}]\!]$ for this term. Note that we have two definitions of the encoding of an encrypted message. Again this is needed for representing Abadi's type system. For now it should be clear from the context which encoding to use.

Encoding of names and variables is done with the identity mapping. The processes are encoded as shown below.

$$
\begin{aligned}
[\![M(\vec{x}).P]\!] &\stackrel{\text{def}}{=} [\![M]\!](\vec{x}).[\![P]\!] \\
[\![\overline{M}\langle\vec{N}\rangle.P]\!] &\stackrel{\text{def}}{=} (\nu\vec{x})\left(\{[\![\vec{N}]\!]/\vec{x}\} \mid [\![\overline{M}]\!]\langle\vec{x}\rangle.[\![P]\!]\right) \\
[\![\text{let } (x,y)=M \text{ in } P]\!] &\stackrel{\text{def}}{=} \text{if } \mathsf{pair}(\mathsf{fst}([\![M]\!]),\mathsf{snd}([\![M]\!])) = [\![M]\!] \text{ then} \\
&\qquad [\![P]\!]\{\mathsf{fst}([\![M]\!])/x,\mathsf{snd}([\![M]\!])/y\} \text{ else } \mathbf{0} \\
[\![\text{case } M \text{ of } 0:P \;\; suc(x):Q]\!] &\stackrel{\text{def}}{=} \text{if } [\![M]\!]=\mathsf{0} \text{ then } [\![P]\!] \text{ else if } [\![M]\!]=\mathsf{suc}(\mathsf{pred}([\![M]\!])) \\
&\qquad \text{then } [\![Q]\!]\{\mathsf{pred}([\![M]\!])/x\} \text{ else } \mathbf{0} \\
[\![\text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P]\!] &\stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}p}(\mathsf{dec\text{-}p}_\alpha([\![L]\!],[\![N]\!]),[\![N]\!]) = [\![L]\!] \text{ then} \\
&\qquad [\![P]\!]\{\mathsf{dec\text{-}p}([\![L]\!],[\![N]\!])/\vec{x}\} \text{ else } \mathbf{0} \\
[\![\text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P]\!] &\stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}s}(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]),[\![N]\!]) = [\![L]\!] \text{ then} \\
&\qquad [\![P]\!]\{\mathsf{cast}(\mathsf{dec}_s([\![L]\!],[\![N]\!]))/\vec{x}\} \text{ else } \mathbf{0}
\end{aligned}
$$

where $[\![P]\!]\{\mathsf{dec\text{-}p}([\![L]\!],[\![N]\!])/\vec{x}\}$ is shorthand for

$$
\begin{aligned}
[\![P]\!]\{&\mathsf{fst}(\mathsf{dec\text{-}p}_\alpha([\![L]\!],[\![N]\!]))/x_1,\mathsf{fst}(\mathsf{fst}(\mathsf{dec\text{-}p}_\alpha([\![L]\!],[\![N]\!])))/x_2,\ldots, \\
&\qquad \mathsf{fst}(\cdots(\mathsf{fst}(\mathsf{snd}(\mathsf{dec\text{-}p}_\alpha([\![L]\!],[\![N]\!])))))/x_n\},
\end{aligned}
$$

and furthermore $[\![P]\!]\{\mathsf{cast}(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]))/[\![\vec{x}]\!]\}$ is shorthand for

$$
\begin{aligned}
[\![P]\!]\{&\mathsf{cast\text{-}s}(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]))/x_1,\mathsf{cast\text{-}a}(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]))/x_2, \\
&\qquad \mathsf{cast\text{-}p}(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]))/x_3,\mathsf{cast}_\alpha(\mathsf{dec\text{-}s}([\![L]\!],[\![N]\!]))/x_4\}
\end{aligned}
$$

In [Aba97] Abadi assumes that processes are checked for certain execution errors at runtime. For instance in let $(x,y)=M$ in P the process is stuck if M is not a pair. We check for this by using the if-then-else clauses. A consequence of this runtime check is also that we need two definitions of the encoding of the decryption case. Output is encoded directly to the format used for communication in Appπ. The choice is a matter of taste, we could just as well have chosen to encode the prefix subject, object and process since

$$
(\nu\vec{x})\left(\{[\![\vec{N}]\!]/\vec{x}\} \mid [\![\overline{M}]\!]\langle\vec{x}\rangle.[\![P]\!]\right) \equiv [\![\overline{M}]\!]\langle[\![N]\!]\rangle.[\![P]\!].
$$

A few final notes on notation. In order to avoid complicating expressions unnecessarily we will in the rest of the report often omit writing $[\![]\!]$ when encoding terms, if its use is clear from the context. For instance in an output prefix we often assume that M is a name or variable. In this case the encoding is the identity mapping, in which case we omit writing $[\![]\!]$. Furthermore, if the type is inessential we sometimes write f instead of f_α.

5.3 Operational correspondence of $[\![\cdot]\!]$

In this section we state and prove an operational correspondence of the encoding $[\![\cdot]\!]$. With respect to soundness we prove that silent transitions can be matched by a sequence of silent transitions. The reason why we need a sequence of transitions to match a single transition is that, if a spi-process P reduces to another spi-process, then the encoding of P will possibly need a τ-transition to match the reduction, see Lemma 5.3.2. We also prove that if $P \downarrow_a$ then this barb can be matched by by a $[\![P]\!] \xrightarrow{\alpha}$ transition, $\mathrm{subj}(\alpha) = a$. This formulation is necessary as we only have output barbs in Appπ. With respect to completeness we prove that if the encoding of a spi-process P has a τ-transitions, then the derivative is weakly observation equivalent to the encoding of another spi-process Q. Furthermore, the transition can either be matched or else P and Q are weakly barbed congruent.

The following is a standard lemma asserting that the encoding is an homomorphism with respect to substitution. We only state and prove the lemma for process expressions, but it also holds for terms. This is a necessary property in the proof of the lemma, and can easily be verified by a case by case analysis of the terms.

Lemma 5.3.1 (Substitution)
Let $P \in \mathcal{S}$, and let $M \in \mathcal{S}$. Then $[\![P]\!]\{[\![\vec{M}]\!]/\vec{x}\} = [\![P\{\vec{M}/\vec{x}\}]\!]$.

Proof
The proof is carried out by induction in the structure of $P \in \mathcal{S}$.

Induction basis: Suppose $P = \mathbf{0}$. The theorem follows immediately.

Induction hypothesis: The theorem holds for the immediate constituents of P.

Induction step: There are nine cases to consider.

- If $P \stackrel{\text{def}}{=} \overline{L}\langle\vec{N}\rangle.Q$, then

$$\begin{aligned}
[\![\overline{L}\langle\vec{N}\rangle.Q]\!]\{[\![\vec{M}]\!]/\vec{x}\} &= ((\nu\vec{y})(\{[\![\vec{N}]\!]/\vec{y}\} \mid [\![\overline{L}]\!]\langle\vec{y}\rangle.[\![Q]\!]))\{[\![\vec{M}]\!]/\vec{x}\} \\
&= (\nu\vec{y})(\{[\![\vec{N}]\!]\{[\![\vec{M}]\!]/\vec{x}\}/\vec{y}\} \mid \\
&\qquad ([\![\overline{L}]\!]\langle\vec{y}\rangle)\{[\![\vec{M}]\!]/\vec{x}\}.[\![Q]\!]\{[\![\vec{M}]\!]/\vec{x}\}) \\
&= [\![\overline{L}\langle\vec{N}\rangle.Q\{[\![\vec{M}]\!]/\vec{x}\}]\!].
\end{aligned}$$

The third equality follows from the induction hypothesis.

- Suppose $P \stackrel{\text{def}}{=} L(\vec{x}).Q$. This case is similar to the first recalling that by convention the names in the substitution are different from the names in $\vec{x}$.
- Parallel composition follows immediately since $[\![\cdot]\!]$ is an homomorphism and by applying the induction hypothesis.
- A restriction $P \stackrel{\text{def}}{=} (\nu a)Q$ is similar to the previous case. Again recall that by convention substitutions avoid bound names.
- If $P \stackrel{\text{def}}{=} !Q$ we get $[\![!Q]\!]\{]\vec{M}[\![/\vec{x}\} = !{[\![Q]\!]}\{[\![\vec{M}]\!]/\vec{x}\} = !{[\![Q\{\vec{M}/\vec{x}\}]\!]} = [\![!Q\{\vec{M}/\vec{x}\}]\!]$, by an application of the induction hypothesis.
- Now suppose $P \stackrel{\text{def}}{=} \text{if } L = L \text{ then } Q$. We encode and obtain the following

$$\begin{aligned}
&[\![\text{if } L = L \text{ then } Q]\!]\{[\![\vec{M}]\!]/\vec{x}\} \\
&\qquad = (\text{if } [\![L]\!] = [\![L]\!] \text{ then } [\![Q]\!] \text{ else } \mathbf{0})\{[\![\vec{M}]\!]/\vec{x}\} \\
&\qquad = (\text{if } ([\![L]\!] = [\![L]\!])\{[\![\vec{M}]\!]/\vec{x}\} \text{ then } [\![Q\{\vec{M}/\vec{x}\}]\!] \text{ else } \mathbf{0}) \\
&\qquad = [\![(\text{if } L = L \text{ then } Q)\{\vec{M}/\vec{x}\}]\!].
\end{aligned}$$

  Again the second equation is obtained by applying the induction hypothesis.
- We now consider the case when P is a let construct.

$$\begin{aligned}
&[\![\text{let } (z, y) = L \text{ in } Q]\!]\{[\![\vec{M}]\!]/\vec{x}\} \\
&\qquad = (\text{if } \mathsf{pair}(\mathsf{fst}([\![L]\!]), \mathsf{snd}([\![L]\!])) = [\![L]\!] \text{ then} \\
&\qquad\quad [\![Q]\!]\{\mathsf{fst}([\![L]\!])/z, \mathsf{snd}([\![L]\!])/y\} \text{ else } \mathbf{0})\{[\![\vec{M}]\!]/\vec{x}\} \\
&\qquad = \text{if } (\mathsf{pair}(\mathsf{fst}([\![L]\!]), \mathsf{snd}([\![L]\!])) = [\![L]\!])\{[\![\vec{M}]\!]/\vec{x}\} \text{ then} \\
&\qquad\quad [\![Q]\!]\{\mathsf{fst}([\![L]\!])/z, \mathsf{snd}([\![L]\!])/y\}\{[\![\vec{M}]\!]/\vec{x}\} \text{ else } \mathbf{0}.
\end{aligned}$$

  By applying the induction hypothesis one readily sees that the result equals the process

$$[\![(\text{let } (x, y) = L \text{ in } Q)\{\vec{M}/\vec{x}\}]\!].$$

  Noting that the names in $\vec{x}$ are different from x and y.
- Suppose now $P \stackrel{\text{def}}{=} \text{case } L \text{ of } \{\vec{x}\}_N \text{ in } Q$, where we assume secret encryption and decryption.

$$\begin{aligned}
&[\![\text{case } L \text{ of } \{\vec{y}\}_N \text{ in } Q]\!]\{[\![\vec{M}]\!]/\vec{y}\}\{[\![\vec{M}]\!]/\vec{x}\} \\
&\qquad = (\text{if } \mathsf{enc\text{-}s}(\mathsf{dec\text{-}s}(L, N), N) = L \text{ then} \\
&\qquad\quad [\![P]\!]\{\mathsf{cast}(\mathsf{dec\text{-}s}(L, N))/\vec{y}\} \text{ else } \mathbf{0})\{[\![\vec{M}]\!]/\vec{x}\}.
\end{aligned}$$

  Again one sees that by applying the induction hypothesis and by the comments preceding the lemma, the conclusion follows.

The remaining cases are all similar to the above ones, hence the details are omitted.

■

Next follows the aforementioned Reduction lemma.

Lemma 5.3.2 (Reduction lemma)
Let $P, Q \in \mathcal{S}$. If $P > Q$ then either

(i) there exists $\widehat{Q} \in \mathcal{P}$ such that $[\![P]\!] \xrightarrow{\tau} \widehat{Q} \equiv [\![Q]\!]$, or

(ii) $[\![P]\!] \equiv [\![Q]\!]$.

Proof
We go through each of the reduction rules given in Definition 2.2.1 and prove that either (i) or (ii) is fulfilled. If case (i) is fulfilled then we just write $P \xrightarrow{\tau} \equiv [\![Q]\!]$ when $\widehat{Q}$ is not important.

$!P$: Then encoding is a homomorphism on replication and therefore we have

$$[\![!P]\!] \stackrel{\text{def}}{=} !\,[\![P]\!] \equiv [\![P]\!] \mid !\,[\![P]\!].$$

Thus case (ii) is fulfilled.

if $M = M$ then P: The definition of the encoding gives us

$$[\![\text{if } M = M \text{ then } P]\!] \stackrel{\text{def}}{=} \text{if } [\![M]\!] = [\![M]\!] \text{ then } [\![P]\!] \text{ else } \mathbf{0} \xrightarrow{\tau} [\![P]\!].$$

let $(x, y) = (M, N)$ in P: If we apply the encoding to this spi-process we get

$$\begin{aligned}[\![\text{let } (x,y) = (M,N) \text{ in } P]\!] &\stackrel{\text{def}}{=} \text{if } \mathsf{pair}(\mathsf{fst}(\mathsf{pair}(M,N)), \mathsf{snd}(\mathsf{pair}(M,N))) \\ &= \mathsf{pair}(M,N) \text{ then } [\![P]\!]\{\mathsf{fst}(\mathsf{pair}(M,N))/x, \mathsf{snd}(\mathsf{pair}(M,N))/y\} \text{ else } \mathbf{0}.\end{aligned}$$

This process has a τ-transition to a process which is structurally equivalent to $[\![P\{M/x, N/y\}]\!]$, by Lemma 5.3.1.

case 0 of $0 : P$ $\mathsf{suc}(x) : Q$: Again use the definition of the encoding to obtain

$$\begin{aligned}[\![\text{case } 0 \text{ of } 0 : P \ \mathsf{suc}(x) : Q]\!] \stackrel{\text{def}}{=} &\text{if } \mathsf{0} = \mathsf{0} \text{ then } [\![P]\!] \text{ else if} \\ &\mathsf{0} = \mathsf{suc}(\mathsf{pred}(\mathsf{0})) \text{ then } [\![Q]\!]\{\mathsf{pred}(\mathsf{0})/x\} \text{ else } \mathbf{0} \xrightarrow{\tau} [\![P]\!].\end{aligned}$$

case $\mathsf{suc}(M)$ of $0 : P$ $\mathsf{suc}(x) : Q$: This case is very similar to the previous one. We have

$$\begin{aligned}[\![\text{case } \mathsf{suc}(M) \text{ of } 0 : P \ \mathsf{suc}(x) : Q]\!] \stackrel{\text{def}}{=} &\text{if } \mathsf{suc}([\![M]\!]) = \mathsf{0} \text{ then } [\![P]\!] \text{ else if} \\ &\mathsf{suc}([\![M]\!]) = \mathsf{suc}(\mathsf{pred}(\mathsf{suc}([\![M]\!]))) \text{ then} \\ &[\![Q]\!]\{\mathsf{pred}(\mathsf{suc}([\![M]\!]))/x\} \text{ else } \mathbf{0} \xrightarrow{\tau} \equiv [\![Q]\!]\{[\![M]\!]/x\}.\end{aligned}$$

Case (i) is then fulfilled by Lemma 5.3.1.

case $\{\vec{M}\}_N$ of $\{\vec{x}\}_N$ in P**:** There are two definitions of this encoding. First consider the case where $\mathsf{enc\text{-}p}$ and $\mathsf{dec\text{-}p}_\alpha$ are used.

$$\begin{aligned}[\![\text{case } \{\vec{M}\}_N \text{ of } \{\vec{x}\}_N \text{ in } P]\!] &\stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}p}(\mathsf{dec\text{-}p}_\alpha(\mathsf{enc\text{-}p}(\vec{M},N),N),N) \\ &= \mathsf{enc\text{-}p}(\vec{M},N) \text{ then } [\![P]\!]\{\mathsf{dec\text{-}p}_\alpha(\mathsf{enc\text{-}p}(\vec{M},N),N)/\vec{x}\} \text{ else } \mathbf{0} \\ &\xrightarrow{\tau}\equiv [\![P\{\vec{M}/\vec{x}\}]\!].\end{aligned}$$

The other case is almost identical and is omitted.

■

Before we can state the operational correspondence we need a lemma that generalizes some of the transition rules from Definition 3.3.8 for τ-transitions.

Lemma 5.3.3
For Appπ-processes the following rules hold.

$$(\textsc{Res}^*)\ \frac{A \xrightarrow{\tau}{}^* A'}{(\nu a)A \xrightarrow{\tau}{}^* (\nu a)A'}$$

$$(\textsc{Par}^*)\ \frac{A \xrightarrow{\tau}{}^* A'}{A \mid B \xrightarrow{\tau}{}^* A' \mid B}$$

$$(\textsc{Struct}^*)\ \frac{A \equiv B \xrightarrow{\tau}{}^* B' \equiv A'}{A \xrightarrow{\tau}{}^* A'}$$

Proof
We prove the rules in order of appearance. Assume that $A \xrightarrow{\tau}{}^* A'$. The proof is by induction in the length n of $\xrightarrow{\tau}{}^*$. If $n = 1$ then the conclusion follows by the original (Res) rule. Otherwise we can split the transition into two transitions and then the induction hypothesis applies to both of these transitions:

$$(\nu a)A \xrightarrow{\tau}{}^* (\nu a)\widehat{A} \xrightarrow{\tau}{}^* (\nu a)A'.$$

Since $\xrightarrow{\tau}{}^*$ is transitive this can also be written as $(\nu a)A \xrightarrow{\tau}{}^* (\nu a)A'$ which is what we wanted.

Now again assume that $A \xrightarrow{\tau}{}^* A'$. Again the induction basis is fulfilled by the (Par) rule from Definition 3.3.8. Otherwise we again split the transition into two transitions each of shorter length

$$A \mid B \xrightarrow{\tau}{}^* \widehat{A} \mid B \xrightarrow{\tau}{}^* A' \mid B.$$

From this we conclude that $A \mid B \xrightarrow{\tau}{}^* A' \mid B$.

At last consider the ($\textsc{Struct}^*$) rule. Again the induction basis is trivially fulfilled. Otherwise the transition is split into two and we apply the induction hypothesis:

$$\frac{A \equiv B \xrightarrow{\tau}{}^* \widehat{B} \equiv \widehat{B}}{A \xrightarrow{\tau}{}^* \widehat{B}} \quad \text{and} \quad \frac{\widehat{B} \equiv \widehat{B} \xrightarrow{\tau}{}^* B' \equiv A'}{\widehat{B} \xrightarrow{\tau}{}^* A'}.$$

From these two derivations we get that $A \xrightarrow{\tau}{}^* A'$. ■

Proposition 5.3.4
Let $P \in \mathcal{S}$ and assume that $P \xrightarrow{\tau} Q$. Then there exists $\widehat{Q}$ such that $[\![P]\!] \xrightarrow{\tau}{}^{*} \widehat{Q} \equiv [\![Q]\!]$.

Proof
The proof is by induction in the height n of the derivation of $P \xrightarrow{\tau} Q$. As induction basis we consider derivations of height one. When the derivation tree has height one the only possible way $P \xrightarrow{\tau} Q$ could have been deduced is by using the two axioms (Input) and (Output) and then one of the (Communication) rules

$$\frac{\dfrac{}{a(\vec{x}).P_1 \xrightarrow{a\langle \vec{M}\rangle} P_1\{\vec{M}/\vec{x}\}} \qquad \dfrac{}{\overline{a}\langle \vec{M}\rangle.P_2 \xrightarrow{\overline{a}\langle \vec{M}\rangle} P_2}}{a(\vec{x}).P_1 \mid \overline{a}\langle \vec{M}\rangle.P_2 \xrightarrow{\tau} P_1\{\vec{M}/\vec{x}\} \mid P_2}$$

First consider the (Input) axiom. If

$$a(\vec{x}).P_1 \xrightarrow{a\langle \vec{M}\rangle} P_1\{\vec{M}/\vec{x}\},$$

then

$$[\![a(\vec{x}).P_1]\!] = a(\vec{x}).[\![P_1]\!] \xrightarrow{a\langle [\![\vec{M}]\!]\rangle} [\![P_1]\!]\{[\![\vec{M}]\!]/\vec{x}\}.$$

Next consider the (Output) axiom

$$\overline{a}\langle \vec{M}\rangle.P_2 \xrightarrow{\overline{a}\langle \vec{M}\rangle} P_2.$$

We use the encoding to obtain

$$[\![\overline{a}\langle \vec{M}\rangle.P_2]\!] \stackrel{\text{def}}{=} (\nu \vec{y})(\{[\![\vec{M}]\!]/\vec{y}\} \mid \overline{a}\langle \vec{y}\rangle.[\![P_2]\!]) \xrightarrow{(\nu \vec{y})\overline{a}\langle \vec{y}\rangle} \{[\![\vec{M}]\!]/\vec{y}\} \mid [\![P_2]\!].$$

Now as $\vec{y}$ are fresh names variables we can use the (SC-ResPar) axiom

$$(\nu \vec{y})(a(\vec{x}).[\![P_1]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\} \mid \overline{a}\langle \vec{y}\rangle.[\![P_2]\!]) \equiv \tag{5.1}$$

$$a(\vec{x}).[\![P_1]\!] \mid (\nu \vec{y})(\{[\![\vec{M}]\!]/\vec{y}\} \mid \overline{a}\langle \vec{y}\rangle.[\![P_2]\!]). \tag{5.2}$$

And we finally deduce the communication:

$$\frac{\dfrac{\dfrac{\dfrac{}{a(\vec{x}).[\![P_1]\!] \mid \overline{a}\langle \vec{y}\rangle.[\![P_2]\!] \xrightarrow{\tau} [\![P_1\{\vec{y}/\vec{x}\}]\!] \mid [\![P_2]\!]}}{a(\vec{x}).[\![P_1]\!] \mid \overline{a}\langle \vec{y}\rangle.[\![P_2]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\} \xrightarrow{\tau} [\![P_1\{\vec{y}/\vec{x}\}]\!] \mid [\![P_2]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\}}}{(5.1) \xrightarrow{\tau} (\nu \vec{y})([\![P_1\{\vec{y}/\vec{x}\}]\!] \mid [\![P_2]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\})}}{(5.2) \xrightarrow{\tau} (\nu \vec{y})([\![P_1\{\vec{y}/\vec{x}\}]\!] \mid [\![P_2]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\})}$$

where the derivations follow by the (Comm), (Par), (Res) and (Struct) rules.

Now the conclusion follows by Lemma 5.3.1 and by equivalences similar to those shown in (3.1):

$$(\nu \vec{y})([\![P_1\{\vec{y}/\vec{x}\}]\!] \mid [\![P_2]\!] \mid \{[\![\vec{M}]\!]/\vec{y}\}) \equiv [\![P_1\{\vec{M}/\vec{x}\}]\!] \mid [\![P_2]\!].$$

The induction hypothesis is that the theorem holds for all derivations of height less than n.

The inductive step is actually quite easy. First note that (Open) can not be the root of the tree as this rule only concerns outputs. If the last rule applied was the (Restriction) rule:

$$\frac{P \xrightarrow{\tau} P'}{(\nu a)P \xrightarrow{\tau} (\nu a)P'},$$

Then by the induction hypothesis we have $[\![P]\!] \xrightarrow{\tau}{}^* \widehat{Q} \equiv [\![P']\!]$. By the (Res*) rule we conclude

$$\frac{[\![P]\!] \xrightarrow{\tau}{}^* \widehat{Q}}{[\![(\nu a)P]\!] \stackrel{\text{def}}{=} (\nu a)[\![P]\!] \xrightarrow{\tau}{}^* (\nu a)\widehat{Q} \equiv (\nu a)[\![P']\!] \stackrel{\text{def}}{=} [\![(\nu a)P']\!]}.$$

The case where (Parallel) is the root of the derivation tree is almost identical to this as the encoding is also a homomorphism on this operator.

Next assume the last rule applied was (Reduction):

$$\frac{P > P' \xrightarrow{\tau} P''}{P \xrightarrow{\tau} P''}$$

The induction hypothesis applies to $P' \xrightarrow{\tau} P''$, thus $[\![P']\!] \xrightarrow{\tau}{}^* \equiv [\![P'']\!]$. By the Reduction lemma, Lemma 5.3.2 we have that $[\![P]\!] \xrightarrow{\tau}{}^* \equiv [\![P']\!]$ and by applying the (Struct*) rule we get

$$[\![P]\!] \xrightarrow{\tau}{}^* \equiv [\![P'']\!].$$

At last suppose the root of the derivation tree is one of the two (Communication) rules

$$\frac{P \xrightarrow{(\nu\vec{q})\bar{a}\langle\vec{M}\rangle} P' \qquad Q \xrightarrow{a\langle\vec{M}\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu\vec{q})(P' \mid Q')}.$$

We can not obtain the conclusion by simply applying the induction hypothesis to the premises as, even though we also have an (Open) in the transition relation for Appπ, bound output actions can not directly interact with inputs. Instead we apply the axioms for ν in the definition of $\equiv$, Definition 3.2.2, and subsequently apply (Struct). This completes the proof. ■

We also have the following lemma regarding barbs.

Lemma 5.3.5
Let $P \in \mathcal{S}$. If $P \downarrow_a$ then $[\![P]\!] \xrightarrow{\alpha}{}^*$, where $\text{subj}(\alpha) = a$.

Proof
The lemma can be proven by structural induction. The proof is omitted as most of it has been proved in the induction basis in the proof of the previous proposition. ■

Note that a consequence of the Reduction lemma is that we can not obtain a stronger result than in Lemma 5.3.5, i.e. if $P \downarrow_a$ then $[\![P]\!] \downarrow_a$.

Next we look at completeness of the encoding. First we need a lemma regarding the derivatives of encoded spi-processes.

Lemma 5.3.6
Let $P \in \mathcal{S}$ and assume that $[\![P]\!] \xrightarrow{\tau} \widehat{Q}$. Then $Q \in \mathcal{S}$ exists such that $\widehat{Q} \approx [\![Q]\!]$.

Proof
By structural induction. The induction basis is when $P \stackrel{\text{def}}{=} \mathbf{0}$ and since $[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$ the induction basis is proved.

Now P can not be of the form $a(\vec{x}).P'$ or $\overline{a}\langle\vec{M}\rangle.P'$ because then no τ-transition can occur.

Now assume that $P \stackrel{\text{def}}{=} P_1 \mid P_2$. If the τ-transition originates from either P_1 or P_2 the lemma is fulfilled by the induction hypothesis. Otherwise assume that $P_1 \stackrel{\text{def}}{=} a(\vec{x}).P'$ and $P_2 \stackrel{\text{def}}{=} \overline{a}\langle\vec{M}\rangle.P''$. The encoding of these two processes are

$$[\![P_1]\!] \stackrel{\text{def}}{=} a(\vec{x}).[\![P']\!]$$
$$[\![P_2]\!] \stackrel{\text{def}}{=} (\nu\vec{y})(\{\vec{M}/\vec{y}\} \mid \overline{a}\langle\vec{y}\rangle.[\![P'']\!])$$

From the transition rules for Appπ we can then deduce

$$\begin{aligned}[\![P_1 \mid P_2]\!] &\stackrel{\text{def}}{=} [\![P_1]\!] \mid [\![P_2]\!] \xrightarrow{\tau} (\nu\vec{y})(\{\vec{M}/\vec{y}\} \mid [\![P'\{\vec{y}/\vec{x}\}]\!] \mid [\![P'']\!]) \\ &\equiv [\![P'\{\vec{M}/\vec{x}\} \mid P'']\!],\end{aligned}$$

where Lemma 5.3.1 was applied.

Next assume that $P \stackrel{\text{def}}{=} (\nu x)P'$. We can apply the induction hypothesis and obtain

$$[\![P]\!] \stackrel{\text{def}}{=} (\nu x)[\![P']\!] \xrightarrow{\tau} (\nu x)\widehat{Q} \approx (\nu x)[\![Q]\!],$$

for some $Q \in \mathcal{S}$. These case where $P \stackrel{\text{def}}{=} !P'$ follows by a similar argument.

Next suppose $P \stackrel{\text{def}}{=}$ if $M = N$ then P'. The conclusion easily follows by the induction hypothesis as the encoding of P either has a τ-transition to $[\![P']\!]$ or $\mathbf{0}$.

The case where $P \stackrel{\text{def}}{=}$ let $(x, y) = M$ in P' also follows easily from the induction hypothesis.

If $P \stackrel{\text{def}}{=}$ case M of $0 : P'$ $suc(x) : Q$ then the encoding of P either has a τ-transition to $[\![P']\!]$, in which case the lemma is fulfilled, or to

$$R \stackrel{\text{def}}{=} \text{if } M = \mathsf{suc}(\mathsf{pred}(M)) \text{ then } [\![Q]\!]\{\mathsf{pred}(M)/x\} \text{ else } \mathbf{0}.$$

We have that either $R \approx [\![Q]\!]\{\mathsf{pred}(M)/x\}$ or $R \approx \mathbf{0}$. The first case occurs precisely when $M = \mathsf{suc}(N)$ for some term N in which case we get $R \approx [\![Q\{N/x\}]\!]$.

At last the two cases where $P \stackrel{\text{def}}{=}$ case L of $\{\vec{x}\}_N$ in P' follows by the induction hypothesis by using equational rewriting on the substitutions. ■

This lemma says that the assumption in the following proposition is reasonable.

Proposition 5.3.7
Let $P \in \mathcal{S}$ and assume that $[\![P]\!] \xrightarrow{\tau} \widehat{Q}$. Then there exists $Q \in \mathcal{S}$ such that $\widehat{Q} \approx [\![Q]\!]$ and either $P \xrightarrow{\tau} Q$ or $P \approx Q$.

Proof
By structural induction in P. We actually expand the proof given to Lemma 5.3.6. Again the induction basis is trivially fulfilled.

The cases where $P \stackrel{\text{def}}{=} (\nu x)P'$ and $P \stackrel{\text{def}}{=} !P'$ are again fulfilled by the induction hypothesis.

If $P \stackrel{\text{def}}{=} P_1 \mid P_2$ then if the transition originates from either P_1 or P_2 we apply the induction hypothesis to obtain the statement in the theorem. Otherwise again assume that $P_1 \stackrel{\text{def}}{=} a(\vec{x}).P'$ and $P_2 \stackrel{\text{def}}{=} \overline{a}\langle\vec{M}\rangle.P''$. Then

$$[\![P_1 \mid P_2]\!] \xrightarrow{\tau} \widehat{Q} \approx [\![P'\{\vec{M}/\vec{x}\} \mid P'']\!]$$

Obviously we also have $P_1 \mid P_2 \xrightarrow{\tau} P'\{\vec{M}/\vec{x}\} \mid P''$.

Now if $P \stackrel{\text{def}}{=} \text{let } (x,y) = M \text{ in } P'$ then $[\![P]\!] \xrightarrow{\tau} [\![P']\!]\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\}$ if M is a pair or $[\![P]\!] \xrightarrow{\tau} \mathbf{0}$ if M is not a pair. Now if $M = (L,N)$ then $[\![P]\!] \xrightarrow{\tau} \widehat{Q} \equiv [\![P'\{L/x, N/y\}]\!]$ and $P > P'\{L/x, N/y\}$. Otherwise P is deadlocked and therefore $P \approx \mathbf{0}$.

The case where $P \stackrel{\text{def}}{=} \text{if } M = N \text{ then } P'$ is similar and is omitted.

If $P \stackrel{\text{def}}{=} \text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P'$ then there are two sub cases to consider as the encoding of this expression is overloaded. We first consider the case where N has type Pub

$$[\![P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}p}(\mathsf{dec\text{-}p}_\alpha(L,N),N) \;=\; L \text{ then } [\![P']\!]\{\mathsf{dec\text{-}p}_\alpha(L,N)/\vec{x}\} \text{ else } \mathbf{0}.$$

There are two possibilities, depending on whether $L = \{\vec{M}\}_N$, for some terms $\vec{M}$, or not. If not then trivially we obtain the statement in the theorem. If it is then

$$[\![P]\!] \xrightarrow{\tau} [\![P']\!]\{\mathsf{dec\text{-}p}_\alpha(\mathsf{enc\text{-}p}(\vec{M},N))/\vec{x}\} \equiv [\![P']\!]\{\vec{M}/\vec{x}\} = [\![P'\{\vec{M}/\vec{x}\}]\!],$$

by Lemma 5.3.1. Now for the spi-process P we have $P > P'\{\vec{M}/\vec{x}\}$ as desired.

The second sub case is when N has type Sec and the proof becomes much similar to the previous.

If $P \stackrel{\text{def}}{=} \text{case } M \text{ of } 0 : P' \; suc(x) : Q$ then the encoding of P either has a τ-transition to $[\![P']\!]$, in which case we have $P > P'$ or to

$$R \stackrel{\text{def}}{=} \text{if } M = \mathsf{suc}(\mathsf{pred}(M)) \text{ then } [\![Q]\!]\{\mathsf{pred}(M)/x\} \text{ else } \mathbf{0}.$$

We have that either $R \approx [\![Q]\!]\{\mathsf{pred}(M)/x\}$ or $R \approx \mathbf{0}$. The first case occurs precisely when $M = \mathsf{suc}(N)$ for some term N in which case we get $R \approx [\![Q\{N/x\}]\!]$ and furthermore for the spi-process P we have $P > Q\{N/x\}$. If the second case occurs, then P is deadlocked and therefore $P \approx \mathbf{0}$. ■

Lemma 5.3.8
Let $P \in \mathcal{S}$. If $[\![P]\!] \downarrow_a$ then $P \downarrow_a$.

Proof
By structural in P. We omit the proof. ■

We restate the two previous propositions in the following operational correspondence theorem.

Theorem 5.3.9 (Operational correspondence)
Let $P \in \mathcal{S}$. If $P \xrightarrow{\tau} Q$ then there exists $\widehat{Q}$ such that

$$[\![P]\!] \xrightarrow{\tau}{}^{*} \widehat{Q} \equiv [\![Q]\!].$$

Furthermore, if $[\![P]\!] \xrightarrow{\tau} \widehat{Q}$ then there exists $Q \in \mathcal{S}$ such that $\widehat{Q} \approx [\![Q]\!]$ and either

$$P \xrightarrow{\tau} Q \quad \text{or} \quad P \approx Q.$$

5.4 Abadi's type rules

This section contains a brief review of the type rules for environments, terms and processes in the spi-calculus given in [Aba97]. Abadi uses three types. These are *Sec*, *Pub* and *Any* representing respectively *Secret*, *Public* and *Any* where *Any* is a super-type of *Pub* and *Sec*. As in Chapter 4 the type system consists of type rules for well formed environments, terms and processes. We first give the type rules for well formed environments.

$$(\textsc{Environment Empty})\ \frac{}{\vdash \emptyset \text{ well-formed}}$$

$$(\textsc{Environment Variable})\ \frac{\vdash \Gamma \text{ well-formed} \qquad x \notin \mathrm{dom}(\Gamma)}{\vdash \Gamma \uplus x : T \text{ well-formed}}$$

$$(\textsc{Environment Name})\ \frac{\vdash \Gamma \text{ well-formed} \qquad n \notin \mathrm{dom}(\Gamma) \qquad \Gamma \vdash M_1 : T_1 \quad \cdots \quad \Gamma \vdash M_k : T_k \quad \Gamma \vdash N : S}{\vdash \Gamma \uplus n : T :: \{M_1, \ldots, M_k, n\}_N \text{ well-formed}}.$$

The rule (Environment Name) is quite complicated. It guarentees that no confounder can be used for two different messages, because if a variable occurs in some M_i then it is declared in Γ. We illustrate this by the following example.

Example 5.4.1
Suppose a process can receive a message and use this message in an encrypted message, with a secret key, i.e. a public message. Therefore let $P \stackrel{\text{def}}{=} b(x).Q$ and $Q \stackrel{\text{def}}{=} \overline{a}\langle\{x, M_2, M_3, n\}_N\rangle.\mathbf{0}$. Such a process would possibly use the same confounder for different messages. Let us try to type P, first we type Q.

$$\frac{\Gamma \vdash a : Pub \qquad \dfrac{\dfrac{n : T :: \{x, M_2, M_3, n\}_N \in \Gamma}{\Gamma \vdash x : Sec, M_2 : Any, M_3 : Pub, N : Sec} }{\Gamma \vdash \{x, M_2, M_3, n\}_N : Pub} \qquad \Gamma \vdash \mathbf{0} : \checkmark}{\Gamma \vdash \overline{a}\langle\{x, M_2, M_3, n\}_N\rangle.\mathbf{0}}$$

Since n is not added to Γ in the above derivation it must be the case that n was in Γ from the beginning. Typing the remaining part of P we obtain.

$$\frac{\Gamma \vdash b : T \qquad \Gamma \uplus x : S \vdash Q : \checkmark}{\Gamma \vdash b(x).Q : \checkmark}$$

where we note that both derivations are under the same environment Γ, so this is not possible since x is already in Γ by (E-Nam). *Note that if we modify Q*

in the following way, $Q \stackrel{\text{def}}{=} (\nu n)(\overline{a}\langle\{x, M_2, M_3, n\}_N\rangle.\mathbf{0})$*, then we can type* P *by using the* (Level Restriction) *rule.* △

Abadi assumes that each encrypted message includes a freshly generated name in a fixed position. This name is called a confounder and it is used to minimize implicit flow since a protocol that uses confounders will not generate the same cipher text more than once. The type rule (Environment Name) makes sure that a confounder can be used for no more than one encrypted message as is illustrated by Example 5.4.1 on the facing page.

The rules (Level Encryption Secret) and (Level Encryption Public) uses the idea that if some piece of data is encrypted using a secret key then it can be sent on a public channel and if it encrypted using a public key then the type of the encrypted data is the type of the data itself. The other type rules for terms are straightforward.

$$\text{(Level Subsumption)}\frac{\Gamma \vdash M : T \qquad T \leq R}{\Gamma \vdash M : R}$$

$$\text{(Level Variable)}\frac{\vdash \Gamma \text{ well-formed} \qquad \Gamma(x) = T}{\Gamma \vdash x : T}$$

$$\text{(Level Name)}\frac{\vdash \Gamma \text{ well-formed} \qquad \Gamma(n) = T :: \{M_1, \ldots, M_k, n\}_N}{\Gamma \vdash n : T}$$

$$\text{(Level Zero)}\frac{\vdash \Gamma \text{ well-formed}}{\Gamma \vdash 0 : Pub} \qquad \text{(Level Successor)}\frac{\Gamma \vdash M : T}{\Gamma \vdash suc(M) : T}$$

$$\text{(Level Pair)}\frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : T}{\Gamma \vdash (M, N) : T}$$

$$\text{(Level Encryption Public)}\frac{\Gamma \vdash M_1 : T \qquad \cdots \qquad \Gamma \vdash M_k : T \qquad \Gamma \vdash N : Pub}{\Gamma \vdash \{M_1, \ldots M_k\}_N : T}$$

$$\text{(Level Encryption Secret)}\frac{\begin{array}{c}\Gamma \vdash M_1 : Sec \qquad \Gamma \vdash M_2 : Any \qquad \Gamma \vdash M_3 : Pub \\ \Gamma \vdash N : Sec \qquad \Gamma(n) = T :: \{M_1, M_2, M_3, n\}_N\end{array}}{\Gamma \vdash \{M_1, M_2, M_3, n\}_N : Pub}$$

Another thing worth noticing about Abadi's type system is that it adopts a special format for messages on secret channels or messages which have been encrypted using secret keys. Messages on a secret channel have three components. The first has level Secret, the second level Public and the third level Any. A message encrypted under a secret key has these three components and a fourth which is a confounder. Note that this labelling is only important for messages on secret channels or under secret keys as other messages can only contain Public data. These special formats appear in, among others, the rules (Level Output Secret) and (Level Decryption Secret).

$$(\textsc{Level Output Public})\frac{\Gamma \vdash M : Pub \qquad \Gamma \vdash P : \checkmark \quad \Gamma \vdash M_1 : Pub \quad \cdots \quad \Gamma \vdash M_k : Pub}{\Gamma \vdash \overline{M}\langle M_1, \ldots, M_k\rangle.P : \checkmark}$$

$$(\textsc{Level Output Secret})\frac{\Gamma \vdash M : Sec \qquad \Gamma \vdash M_1 : Sec \quad \Gamma \vdash M_2 : Any \quad \Gamma \vdash M_3 : Pub \quad \Gamma \vdash P : \checkmark}{\Gamma \vdash \overline{M}\langle M_1, M_2, M_3\rangle.P : \checkmark}$$

$$(\textsc{Level Input Public})\frac{\Gamma \vdash M : Pub \quad \Gamma \uplus x_1 : Pub \uplus \cdots \uplus x_k : Pub \vdash P : \checkmark}{\Gamma \vdash M(\vec{x}).P : \checkmark}$$

$$(\textsc{Level Input Secret})\frac{\Gamma \vdash M : Sec \quad \Gamma \uplus x_1 : Sec \uplus x_2 : Any \uplus x_3 : Pub \vdash P : \checkmark}{\Gamma \vdash M(x_1, x_2, x_3).P : \checkmark}$$

$$(\textsc{Level Nil})\frac{\vdash \Gamma \text{ well-formed}}{\Gamma \vdash \mathbf{0} : \checkmark} \qquad (\textsc{Level Restriction})\frac{\Gamma \uplus n : T :: L \vdash P : \checkmark}{\Gamma \vdash (\nu n)P : \checkmark}$$

$$(\textsc{Level Replication})\frac{\Gamma \vdash P : \checkmark}{\Gamma \vdash !P : \checkmark} \qquad (\textsc{Level Parallel})\frac{\Gamma \vdash P : \checkmark \quad Q : \checkmark}{\Gamma \vdash P \mid Q : \checkmark}$$

$$(\textsc{Level Match})\frac{\Gamma \vdash M : T \quad \Gamma \vdash N : R \quad \Gamma \vdash P : \checkmark}{\Gamma \vdash \text{if } M = N \text{ then } P : \checkmark}$$

$$(\textsc{Level Pair Splitting})\frac{\Gamma \vdash M : T \quad \Gamma \uplus x : T \uplus y : T \vdash P : \checkmark}{\Gamma \vdash \text{let } (x, y) = M \text{ in } P : \checkmark}$$

$$(\textsc{Level Integer Case})\frac{\Gamma \vdash M : T \quad \Gamma \vdash P : \checkmark \quad \Gamma \uplus x : T \vdash Q : \checkmark}{\Gamma \vdash \text{case } M \text{ of } 0 : P\ \mathsf{suc}(x) : Q : \checkmark}$$

$$(\textsc{Level Decryption Public})\frac{\Gamma \vdash L : T \qquad \Gamma \vdash N : Pub \quad \Gamma \uplus x_1 : T \uplus \cdots \uplus x_k : T \vdash P : \checkmark}{\Gamma \vdash \text{case } L \text{ of } \{x_1, \ldots, x_k\}_N \text{ in } P : \checkmark}$$

$$(\textsc{Level Decryption Secret})\frac{\Gamma \vdash L : T \qquad \Gamma \vdash N : Sec \quad \Gamma \uplus x_1 : Sec \uplus x_2 : Any \uplus x_3 : Pub \uplus x_4 : Any \vdash P : \checkmark}{\Gamma \vdash \text{case } L \text{ of } \{x_1, x_2, x_3, x_4\}_N \text{ in } P : \checkmark}$$

where, in the last five rules, T and R range over {Pub,Sec}.

5.5 Preservation of types

We have previously seen how the spi-calculus can be represented in Appπ. In this section we will instantiate of the type system for Appπ, defined in Chapter 4, and show how this can be modified to capture the type system Abadi defines in [Aba97]. The modifications are necessary as Abadi defines rules which use a special format. The instantiation is described in the next section.

5.5.1 Instantiation of the type system

We now instantiate the type system from Section 4.2. We need a definition of the types and we need to define the type rules for terms.

Definition 5.5.1 (Types)
The types for Appπ are generated by the grammar

$$S ::= V \mid \checkmark \qquad \text{(Types)}$$
$$V ::= Pub \mid Sec \mid Any. \qquad \text{(Value types)}$$

We only give the type rules for terms involving function names. These are

$$\text{(T-Suc)}\frac{\Gamma \vdash M : T \quad \alpha(\mathsf{suc}) = 1}{\Gamma \vdash \mathsf{suc}(M) : T} \qquad \text{(T-Pred)}\frac{\Gamma \vdash M : T \quad \alpha(\mathsf{pred}) = 1}{\Gamma \vdash \mathsf{pred}(M) : T}$$

$$\text{(T-Fmt)}\frac{\Gamma \vdash M_1 : Sec \quad \Gamma \vdash M_2 : Any \quad \Gamma \vdash M_3 : Pub \quad \Gamma \vdash M_4 : T \quad \alpha(\mathsf{fmt}) = 4}{\Gamma \vdash \mathsf{fmt}(M_1, M_2, M_3, M_4) : Any}$$

$$\text{(T-Encs)}\frac{\Gamma \vdash M_1 : Any \quad \Gamma \vdash M_2 : Sec \quad \alpha(\mathsf{enc\text{-}s}) = 2}{\Gamma \vdash \mathsf{enc\text{-}s}(M_1, M_2) : Pub}$$

$$\text{(T-Encp)}\frac{\Gamma \vdash M_1 : T \quad \Gamma \vdash M_2 : Pub \quad \alpha(\mathsf{enc\text{-}p}) = 2}{\Gamma \vdash \mathsf{enc\text{-}p}(M_1, M_2) : Sec}$$

$$\text{(T-Decs)}\frac{\Gamma \vdash M_1 : Pub \quad \Gamma \vdash M_2 : Sec \quad \alpha(\mathsf{dec\text{-}s}) = 2}{\Gamma \vdash \mathsf{dec\text{-}s}(M_1, M_2) : Any}$$

$$\text{(T-Decp)}\frac{\Gamma \vdash M_1 : Sec \quad M_2 : Pub \quad \alpha(\mathsf{dec\text{-}p}_T) = 2}{\Gamma \vdash \mathsf{dec\text{-}p}_T : T}$$

$$\text{(T-Pair)}\frac{\Gamma \vdash M_1 : T \quad \Gamma \vdash M_2 : T \quad \alpha(\mathsf{pair}) = 2}{\Gamma \vdash \mathsf{pair}(M_1, M_2) : T}$$

$$\text{(T-Casta)}\frac{\Gamma \vdash M : Any \quad \alpha(\mathsf{cast\text{-}a}) = 1}{\Gamma \vdash \mathsf{cast\text{-}a}(M) : Any}$$

$$\text{(T-Fst)}\frac{\Gamma \vdash M : T \quad \alpha(\mathsf{fst}) = 1}{\Gamma \vdash \mathsf{fst}(M) : T} \qquad \text{(T-Casts)}\frac{\Gamma \vdash M : Any \quad \alpha(\mathsf{cast\text{-}s}) = 1}{\Gamma \vdash \mathsf{cast\text{-}s}(M) : Sec}$$

$$\text{(T-Snd)}\frac{\Gamma \vdash M : T \quad \alpha(\mathsf{snd}) = 1}{\Gamma \vdash \mathsf{snd}(M) : T} \qquad \text{(T-Cast)}\frac{\Gamma \vdash M : Any \quad \alpha(\mathsf{cast}_T) = 1}{\Gamma \vdash \mathsf{cast}_T(M) : T}$$

$$\text{(T-Castp)}\frac{\Gamma \vdash M : Any \quad \alpha(\mathsf{cast\text{-}p}) = 1}{\Gamma \vdash \mathsf{cast\text{-}p}(M) : Pub} \qquad \text{(T-Zero)}\frac{\alpha(0) = 0}{\Gamma \vdash 0 : Pub}$$

A term $\mathsf{fmt}(x, y, z, n)$ may be thought of as a dedicated kind of tuple representing the special message format in Abadi [Aba97]. Only terms of this type may be encrypted with a secret key. We could have chosen the result type of fmt arbitrarily. The choice of *Any* is loosely motivated by the type rule for tuples since the type of a tuple is given the type of its constituents. Thus the most precise type judgment one can make on a tuple whose elements have different type, is type *Any*. The cast functions extract the contents of a fmt term while at the same time restoring the type information.

5.5.2 The modified type system

In order to account for the special rules described in Section 5.4 we add or modify some of the rules described in the Chapter 4. First consider the rules for well-formed environments. We split the rule (E-NamVar) into two rules; one for variables and one for names.

$$(\text{E-Var})\ \frac{\vdash \Gamma \text{ well-formed} \qquad x \notin \mathrm{dom}(\Gamma)}{\vdash \Gamma \uplus x : T \text{ well-formed}}$$

$$(\text{E-Nam})\ \frac{\begin{array}{c}\vdash \Gamma \text{ well-formed} \qquad\qquad n \notin \mathrm{dom}(\Gamma) \\ \Gamma \vdash M_1 : T_1 \quad \cdots \quad \Gamma \vdash M_k : T_k \quad \Gamma \vdash N : S\end{array}}{\vdash \Gamma \uplus n : T :: \{M_1, \ldots, M_k, n\}_N \text{ well-formed}}.$$

The rule (E-Nam) is a subtle construct. The essential property is the requirement that all variables in $M_1, \ldots, M_n$ are declared in Γ. This ensures that a variable can not be instantiated in two different ways, hence preventing the same confounder from being used in two different messages. This is illustrated by Example 5.4.1 on page 48.

Regarding the type rules for terms we just split the (T-NamVar) rule in two.

$$(\text{T-Nam})\ \frac{\vdash \Gamma \text{ well-formed} \qquad n : T :: \{M_1, \ldots, M_k, n\}_N \in \Gamma}{\Gamma \vdash n : T}$$

$$(\text{T-Var})\ \frac{\vdash \Gamma \text{ well-formed} \qquad x : T \in \Gamma}{\Gamma \vdash x : T}$$

For Appπ-processes first consider the (T-Ite) rule. This needs to be modified so that the two terms can have different types and the rule (T-Res) should be split into two rules. One for names where the bound name is associated with an arbitrary term and one for variables which is identical to the old rule.

$$(\text{T-ResNam})\ \frac{\Gamma \uplus a : S :: L \vdash P : \checkmark}{\Gamma \vdash (\nu a)P : \checkmark}$$

$$(\text{T-ResVar})\ \frac{\Gamma \uplus x : S \vdash P : \checkmark}{\Gamma \vdash (\nu x)P : \checkmark}$$

$$(\text{T-Ite})\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : T \qquad \Gamma \vdash P : \checkmark \qquad \Gamma \vdash Q : \checkmark}{\Gamma \vdash \text{if } M = N \text{ then } P \text{ else } Q : \checkmark}$$

where in the last rule $S, T \in \{Pub, Sec\}$. Terms of level Any cannot occur in an if-then-else clause in order to prevent implicit flow. Furthermore, the reason for not modifying the (T-ResVar) rule is that a variable under a restriction can only be used to bind a variable in an active substitution.

Lastly we give four rules which replace (T-Out) and (T-Inp).

$$\text{(T-OutputPub)}\frac{\Gamma \vdash u : Pub \quad \Gamma \vdash \vec{M} : \vec{Pub} \quad \Gamma \vdash P : \checkmark}{\Gamma \vdash \overline{u}\langle \vec{M} \rangle.P : \checkmark}$$

$$\text{(T-OutputSec)}\frac{\begin{array}{c}\Gamma \vdash u : Sec \qquad \Gamma \vdash M_1 : Sec \\ \Gamma \vdash M_2 : Any \quad \Gamma \vdash M_3 : Pub \quad \Gamma \vdash P : \checkmark\end{array}}{\Gamma \vdash \overline{u}\langle M_1, M_2, M_3 \rangle.P : \checkmark}$$

$$\text{(T-InputPub)}\frac{\Gamma \vdash u : Pub \quad \Gamma \uplus x_1 : Pub \uplus \cdots \uplus x_k : Pub \vdash P : \checkmark}{\Gamma \vdash u(\vec{x}).P : \checkmark}$$

$$\text{(T-InputSec)}\frac{\Gamma \vdash u : Sec \quad \Gamma \uplus x_1 : Sec \uplus x_2 : Any \uplus x_3 : Pub \vdash P : \checkmark}{\Gamma \vdash u(x_1, x_2, x_3).P : \checkmark}.$$

Before we present the main theorems we need a lemmas. It states that the encoding $[\![\cdot]\!]$ preserves the types of spi-terms.

Lemma 5.5.2
Let $M \in \mathcal{S}$ be a term. Then $\Gamma \vdash M : T$ if and only if $\Gamma \vdash [\![M]\!] : T$.

Proof
We first prove the forward direction. The proof is by induction in the height n of the type derivation of M. Then as induction basis we consider Abadi's (Level Variable), (Level Name) and (Level Zero) rules. These rules are identical to the (T-Var), (T-Nam) and (T-Zero) rules, respectively, thus the induction basis is proved since the encoding is the identity on $\mathbf{0}$, variables and names.

The induction hypothesis is that the theorem holds for terms whose type derivation has height less than n.

In the inductive step there are five rules to consider.

(Level Subsumption) The induction hypothesis gives us that the premises hold for the encoding of a term M. Thus by the (T-Subsumption) rule, we obtain the conclusion.

(Level Successor) Suppose a term $[\![M]\!]$ has type T in Γ. Then by the (T-Suc) rule we get

$$\frac{\Gamma \vdash [\![M]\!] : T \quad \alpha(\mathsf{suc}) = 1}{\Gamma \vdash \mathsf{suc}([\![M]\!]) : T}.$$

(Level Pair) Follows by a similar argument to the (Level Successor) rule, using the (T-Pair) rule instead of (T-Suc).

(Level Encryption Public) Similar to the above, using the (T-Encp) rule.

(Level Encryption Secret) Similar to the above, using the (T-Encs).

The other direction is quite similar and is therefore omitted. ■

In the remainder of this chapter we will sometimes just write M for the encoding of a spi-term, when it is clear that M is an Appπ-term.

Theorem 5.5.3 (Soundness)
Let $P \in \mathcal{S}$ and let Γ be a type environment such that $\Gamma \vdash P : \checkmark$. Then $\Gamma \vdash [\![P]\!] : \checkmark$.

Proof
The proof is by induction in the height n of the type derivation of P. In the induction basis there is only one case to consider. This is the (Level Nil) rule and it follows by the corresponding (T-Nil) rule.

The induction hypothesis is that the theorem holds for terms whose type derivation has height less than n.

In the induction step first consider the input and output rules, (Level Output Public), (Level Output Secret), (Level Input Public) and (Level Input Secret). These rules all follow by the added type rules for processes and by Lemma 5.5.2. Next consider the (Level Parallel) rule. The induction hypothesis gives us that the encoding of the two processes in the premises are well typed. Then by the (T-Par) rule we have that the parallel composition of the encoding of the two processes is well typed. The rules (Level Replication), (Level Restriction) and (Level Match) follow by similar arguments. Lastly we consider the four rules where we use the definition of the encoding of spi-calculus processes.

(Level Pair Splitting) Assume that $\Gamma \vdash M : T$. By the induction hypothesis $\Gamma \uplus x : T \uplus y : T \vdash [\![P]\!] : \checkmark$ and we need to show that

$$\Gamma \vdash [\![\text{let } (x,y) = M \text{ in } P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{pair}(\mathsf{fst}(M), \mathsf{snd}(M)) = M \text{ then } [\![P]\!]\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\} \text{ else } \mathbf{0} : \checkmark.$$

Now by the (T-Fst) and (T-Snd) rules

$$\frac{\Gamma \vdash M : T \qquad \alpha(\mathsf{fst}) = 1}{\Gamma \vdash \mathsf{fst}(M) : T} \text{ and } \frac{\Gamma \vdash M : T \qquad \alpha(\mathsf{snd}) = 1}{\Gamma \vdash \mathsf{snd}(M) : T}.$$

Then by (T-Pair) we then get

$$\frac{\Gamma \vdash \mathsf{fst}(M) : T \qquad \Gamma \vdash \mathsf{snd}(M) : T \qquad \alpha(\mathsf{pair}) = 2}{\Gamma \vdash \mathsf{pair}(\mathsf{fst}(M), \mathsf{snd}(M)) : T}.$$

Now because $\Gamma \uplus x : T \uplus y : T \vdash [\![P]\!] : \checkmark$, $\mathsf{fst}(M) : T$, $\mathsf{snd}(M) : T$ we can apply the corollary of the substitution lemma, Corollary 4.3.7, and we get

$$\Gamma \vdash [\![P]\!]\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\} : \checkmark. \tag{5.3}$$

Lastly we apply (T-Ite)

$$\frac{\begin{array}{cc}\Gamma \vdash \mathsf{pair}(\mathsf{fst}(M), \mathsf{snd}(M)) : T & M : T \\ \Gamma \vdash \mathbf{0} : \checkmark \qquad \Gamma \vdash [\![P]\!]\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\} : \checkmark & \end{array}}{\Gamma \vdash \text{if } \mathsf{pair}(\mathsf{fst}(M), \mathsf{snd}(M)) = M \text{ then } [\![P]\!]\{M/(x,y)\} \text{ else } \mathbf{0} : \checkmark}$$

where $\{M/(x,y)\}$ is the substitution in (5.3). In the remaining three cases we will not give a proof as thorough as for this case. We will just explain how the proofs can be conducted.

(Level Integer Case) By the induction hypothesis we have that $\Gamma \vdash M : T$, $\Gamma \vdash [\![P]\!] : \checkmark$ and that $\Gamma \uplus x : T \vdash [\![Q]\!] : \checkmark$. We need to show that

$$\Gamma \vdash [\![\text{case } M \text{ of } 0 : P \;\; suc(x) : Q]\!] \stackrel{\text{def}}{=} \text{if } M = \mathsf{0} \text{ then } [\![P]\!] \text{ else if } M = \mathsf{suc}(\mathsf{pred}(M)) \text{ then } [\![Q]\!]\{\mathsf{pred}(M)/x\} \text{ else } \mathbf{0} : \checkmark$$

This time the conclusion follows from the (T-Ite), (T-Suc) and (T-Pred) rules and by applying the corollary of the substitution lemma and the induction hypothesis.

(Level Decryption Public) Now, by the induction hypothesis, we have $\Gamma \vdash L : T$, $\Gamma \vdash N : Pub$ and $\Gamma \uplus x_1 : T \uplus \cdots \uplus x_k : T \vdash [\![P]\!] : \checkmark$. We need to show

$$\Gamma \vdash [\![\text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}p}(\mathsf{dec\text{-}p}_\alpha(L,N),N) = L \text{ then } [\![P]\!]\{\mathsf{dec\text{-}p}_\alpha(L,N)/\vec{x}\} \text{ else } \mathbf{0} : \checkmark.$$

Because of the induction hypothesis and Lemma 5.5.2 we can apply the rules (T-Encp), (T-Decp) and (T-Ite) and the corollary of the substitution lemma from which we obtain the conclusion.

(Level Decryption Secret) Now assume that $\Gamma \vdash L : T$, $\Gamma \vdash N : Sec$ and that $\Gamma \uplus x_1 : Sec \uplus x_2 : Any \uplus x_3 : Pub \uplus x_4 : Any \vdash [\![P]\!] : \checkmark$. We need to show that

$$\Gamma \vdash [\![\text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}s}(\mathsf{dec\text{-}s}(L,N),N) = L \text{ then } [\![P]\!]\{\mathsf{cast}(\mathsf{dec\text{-}s}(L,N))/\vec{x}\} \text{ else } \mathbf{0} : \checkmark.$$

Again this follows by Lemma 5.5.2 and the rules (T-Encs), (T-Decs), (T-Casts), (T-Casta), (T-Castp), (T-Cast) and (T-Ite), the corollary of the substitution lemma and the induction hypothesis.

■

In order to prove completeness of the encoding $[\![\cdot]\!]$, with respect to well-typed processes, we need the following lemma which is a somewhat opposite statement of the substitution lemma.

Lemma 5.5.4
Let $P \in \mathcal{A}$ and assume that $\Gamma \vdash M : T$ and $\vdash \Gamma \uplus x : T$ well-formed. If $\Gamma \vdash P\{M/x\} : \checkmark$ then $\Gamma \uplus x : T \vdash P : \checkmark$.

Proof
By Lemma 4.3.5 and because no names which do not occur in dom(Γ) are introduced by M we have that

$$\Gamma \vdash P\{M/x\} : \checkmark \text{ implies } \Gamma \vdash (\nu x)(\{M/x\} \mid P) : \checkmark. \tag{5.4}$$

Now the last rule used to deduce (5.4) can only have been (T-ResVar), therefore

$$\frac{\Gamma \uplus x : T \vdash \{M/x\} \mid P : \checkmark}{\Gamma \vdash (\nu x)(P\{M/x\}) : \checkmark}$$

Again the process $\{M/x\} \mid P$ can only be typed by using (T-Par), thus $\Gamma \uplus x : T \vdash P : \checkmark$. ■

Corollary 5.5.5
Let $P \in \mathcal{A}$ and assume that $\Gamma \vdash \vec{M} : \vec{T}$ and $\vdash \Gamma \uplus \vec{x} : \vec{T}$ well-formed. If $\Gamma \vdash P\{\vec{M}/\vec{x}\} : \checkmark$ then $\Gamma \uplus \vec{x} : \vec{T} \vdash P : \checkmark$.

Theorem 5.5.6 (Completeness)
Let $P \in \mathcal{S}$ and let Γ be a type environment such that $\Gamma \vdash [\![P]\!] : \checkmark$. Then $\Gamma \vdash P : \checkmark$.

Proof
This proof is by structural induction in P. Thus as induction basis we consider the inactive process $\mathbf{0}$ and since $[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$ the induction basis is fulfilled by (T-Nil).

Now suppose that $P \stackrel{\text{def}}{=} a\langle\vec{M}\rangle.P'$. Then $[\![P]\!] \stackrel{\text{def}}{=} (\nu\vec{x})(\{\vec{M}/\vec{x}\} \mid \overline{a}\langle\vec{x}\rangle.[\![P']\!])$. There are two possible ways to type this process, depending on whether $\Gamma \vdash a : Pub$ or $\Gamma \vdash a : Sec$. If $\Gamma \vdash a : Pub$ then by the induction hypothesis and by Lemma 5.5.2 we get that $\Gamma \vdash \vec{M} : \vec{Pub}$ and $\Gamma \vdash P' : \checkmark$ as $\vec{M}$ and $\vec{x}$ have identical types. Then by (Level Output Public)

$$\frac{\Gamma \vdash a : Pub \qquad \Gamma \vdash \vec{M} : \vec{Pub} \qquad \Gamma \vdash P' : \checkmark}{\Gamma \vdash \overline{a}\langle\vec{M}\rangle.P' : \checkmark}.$$

The case where $\Gamma \vdash a : Sec$ is proven similarly. The case where $P \stackrel{\text{def}}{=} a(\vec{x}).P'$ is also similar to the above and is omitted.
Now suppose $P \stackrel{\text{def}}{=} R \mid Q$. Then $[\![P]\!] \stackrel{\text{def}}{=} [\![R]\!] \mid [\![Q]\!]$ and by the induction hypothesis $\Gamma \vdash R : \checkmark$ and $\Gamma \vdash Q : \checkmark$. By (Level Parallel) we have

$$\frac{\Gamma \vdash R : \checkmark \qquad \Gamma \vdash Q : \checkmark}{\Gamma \vdash R \mid Q : \checkmark}.$$

The cases where P is a restriction, replication or a match are all just as easy as this case, because the encoding $[\![\cdot]\!]$ is a homomorphism on these operators.

Next assume $P \stackrel{\text{def}}{=} \text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P'$ and that $\Gamma \vdash N : Sec$. Then by definition of the encoding

$$[\![P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{enc\text{-}s}(\mathsf{dec\text{-}s}(L, N), N) = L \text{ then } [\![P']\!]\{\mathsf{cast}(\mathsf{dec\text{-}s}(L, N))/\vec{x}\} \text{ else } \mathbf{0}.$$

By assumption this process is well-typed in Γ. Therefore we also have $\Gamma \vdash [\![P']\!]\{\mathsf{cast}(\mathsf{dec\text{-}s}(L, N))/\vec{x}\} : \checkmark$ and $\Gamma \vdash L : T$ for $T \in \{Pub, Sec\}$. Then by Corollary 5.5.5

$$\Gamma \uplus x_1 : Sec \uplus x_2 : Any \uplus x_3 : Pub \uplus x_4 : Any \vdash [\![P']\!] : \checkmark.$$

The induction hypothesis and the (Level Decryption Secret) rule then allows us to conclude that

$$\Gamma \vdash \text{case } L \text{ of } \{\vec{x}\}_N \text{ in } P' : \checkmark.$$

The case where $\Gamma \vdash N : Pub$ follows by similar arguments.

Next assume that $P \stackrel{\text{def}}{=} \text{let } (x, y) = M \text{ in } P'$ and that $\Gamma \vdash M : T$. By inspecting the encoding of P

$$[\![P]\!] \stackrel{\text{def}}{=} \text{if } \mathsf{pair}(\mathsf{fst}(M), \mathsf{snd}(M)) = M \text{ then } [\![P']\!]\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\} \text{ else } \mathbf{0},$$

we see that $\Gamma \vdash [\![P]\!]'\{\mathsf{fst}(M)/x, \mathsf{snd}(M)/y\} : \checkmark$. By Corollary 5.5.5 we have $\Gamma \uplus x : T \uplus y : T \vdash [\![P]\!]' : \checkmark$. By the induction hypothesis and (Level Pair Splitting) we then get

$$\Gamma \vdash \text{let } (x, y) = M \text{ in } P' : \checkmark.$$

Lastly consider the case where $P \stackrel{\text{def}}{=} \text{case } M \text{ of } 0 : R \; \mathsf{suc}(x) : Q$. Now let $\Gamma \vdash M : T$. From the encoding of P,

$$[\![P]\!] \stackrel{\text{def}}{=} \text{if } M = \mathsf{0} \text{ then } [\![R]\!] \text{ else if } M = \mathsf{suc}(\mathsf{pred}(M)) \text{ then } [\![Q]\!]\{\mathsf{pred}(M)/x\} \text{ else } \mathbf{0},$$

it is apparent that, by the induction hypothesis and Lemma 5.5.4, we can deduce that $\Gamma \vdash R : \checkmark$ and $\Gamma \uplus x : T \vdash Q : \checkmark$. Then by (Level Integer Case) we obtain the result

$$\Gamma \vdash \text{case } M \text{ of } 0 : R \; \mathsf{suc}(x) : Q : \checkmark.$$

■

Chapter 6

Encoding the Appπ-calculus in the π-calculus

Here we investigate an encoding of the Appπ-calculus in the π-calculus. Our encoding is inspired by the encoding of the spi-calculus into the π-calculus presented in [BPV03]. In this article spi-terms are represented as objects with a number of predefined methods. The main idea in our encoding of Appπ in π is to have an encoding of the rewrite system in parallel with the encoded process. The encoding of the rewrite system will reduce encoded terms until they become the encoding of an irreducible term, which are represented as objects with a number of predefined methods, e.g. a method for retrieving the principal function symbol. Given a term M we say that we reduce the encoded term until it becomes irreducible when it becomes the encoding of an irreducible term. Of course the encoding of an irreducible term does indeed have reductions when considered as an Appπ-process. We prove that the encoding is sound with respect to operational correspondence.

Since there exists a plethora of variants of the π-calculus, we start by recalling the syntax and semantics of the variant we will be using as target language.

6.1 The π-calculus

As our encoding from Appπ to the π-calculus requires operators for match, mismatch and sum, we shall choose a variant of the π-calculus containing these. Also we need to choose between early or late semantics. Furthermore, we briefly look at some equivalences for π-calculus. All definitions are quite standard so we present them without much explanation. Thorough descriptions of the π-calculus can found in [Mil99, Mil93, Par01].

6.1.1 Syntax of the π-calculus

We first define the names of the π-calculus. Let $\mathcal{N}$ be an infinite set of names and let $\overline{\mathcal{N}}$ be the set of co-names, i.e. $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$. We will let $a, b, \ldots$

range over $\mathcal{N}$. We now present the syntax of the π-calculus.

Definition 6.1.1 (π-processes)
The set of prefixes of π-processes is generated by

$$\alpha \quad ::= \quad a(\vec{x}) \quad \mid \quad \overline{a}\langle\vec{b}\rangle \quad \mid \quad \tau.$$

The set $\mathcal{P}$ of π-processes is the set generated by the following grammar

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid P \mid P \mid P + P \mid\ !P \mid (\nu a)P \mid \\ & & [a = b]P \mid [a \neq b]P \mid \alpha.P. \end{array}$$

We will also use the abbreviations listed for Appπ when considering π-processes. E.g. we omit the inactive process in $u(\vec{x}).\mathbf{0}$. Furthermore, we assume that all free and bound names are distinct and that substitutions do not capture names or variables. As usual this can be achieved by α-conversion.

6.1.2 Semantics of the π-calculus

As noted earlier we want to present a version of the π-calculus which is as close to Appπ as possible. Consequently, we give an early semantics of the π-calculus and we also have a (Struct) rule in the transition relation as opposed to incorporating these axioms in the transition relation. The definition of the structural congruence relation now follows.

Definition 6.1.2 (Structural congruence on π)
The structural congruence relation on π is the smallest congruence which satisfies the following axioms.

$$\begin{array}{rcll} P & \equiv & Q & \text{if } P \equiv_\alpha Q \qquad \text{(SC-EquivAlpha)} \\ P \mid Q & \equiv & Q \mid P & \qquad \text{(SC-ParCommute)} \\ P \mid \mathbf{0} & \equiv & P & \qquad \text{(SC-ParInact)} \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \qquad \text{(SC-ParAssoc)} \\ (\nu a)(P \mid Q) & \equiv & P \mid (\nu a)Q & \text{if } a \notin \mathrm{fn}(P) \qquad \text{(SC-ResPar)} \\ (\nu a)\mathbf{0} & \equiv & \mathbf{0} & \qquad \text{(SC-ResInact)} \\ (\nu a)(\nu b)P & \equiv & (\nu b)(\nu a)P & \qquad \text{(SC-ResCommute)} \\ !P & \equiv & P \mid !P & \qquad \text{(SC-Rep)} \end{array}$$

We can now define the transition relation for π-processes. The set Act is the usual set of early actions.

Definition 6.1.3 (Transition relation on π)
The transition relation of $\mathcal{P}$ is the relation on $\mathcal{P} \times Act \times \mathcal{P}$ generated by the

following rules

$$(\textsc{Input})\ \frac{}{a(\vec{x}).P \xrightarrow{a\langle\vec{b}\rangle} P\{\vec{b}/\vec{x}\}} \qquad (\textsc{Output})\ \frac{}{\overline{a}\langle\vec{b}\rangle.P \xrightarrow{\overline{a}\langle\vec{b}\rangle} P}$$

$$(\textsc{Comm})\ \frac{}{\overline{a}\langle\vec{b}\rangle.P \mid a(\vec{x}).Q \xrightarrow{\tau} P \mid Q\{\vec{b}/\vec{x}\}} \quad \text{if } |\vec{b}| = |\vec{x}|$$

$$(\textsc{Match})\ \frac{}{[a=b]P \xrightarrow{\tau} P} \quad \text{if } a = b$$

$$(\textsc{Mismatch})\ \frac{}{[a\neq b]P \xrightarrow{\tau} P} \quad \text{if } a \neq b$$

$$(\textsc{Open})\ \frac{P \xrightarrow{(\nu\vec{b})\overline{a}\langle\vec{c}\rangle} P'}{(\nu b')A \xrightarrow{(\nu b',\vec{b})\overline{a}\langle\vec{c}\rangle} A'} \quad \text{if } b' \neq a \text{ and } b' \in \vec{c}$$

$$(\textsc{Res})\ \frac{P \xrightarrow{\alpha} P' \qquad b \notin \mathrm{n}(\alpha)}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'} \qquad (\textsc{Par})\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$(\textsc{Sum})\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad (\textsc{Struct})\ \frac{P \equiv Q \xrightarrow{\alpha} Q' \equiv P'}{P \xrightarrow{\alpha} P'}$$

6.1.3 Barbed bisimulation

In this section we define bisimulation for π. We rely on the notion of barbed bisimulation [MS92].

The observation predicate $P \downarrow_a$ holds if $P \xrightarrow{\alpha}$ and $\mathrm{subj}(\alpha) = a$.

Definition 6.1.4 (Strong barbed bisimulation)
A strong barbed bisimulation is a symmetric relation $\mathcal{R} \subseteq \mathcal{P}^2$ such that if $P\mathcal{R}Q$ then

(i) if $P \xrightarrow{\tau} P'$ then $\exists Q'.\, Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$ and

(ii) if $P \downarrow_a$ then $Q \downarrow_a$.

We write $P \dot{\sim} Q$ if P and Q are related by a strong barbed bisimulation $\mathcal{R}$ and we let $\sim$ denote the congruence induced by $\dot{\sim}$. Weak barbed bisimulation and congruence are defined by modifying Definition 6.1.4 in the usual way.

6.2 The Encoding

The basic idea is to encode processes and place them in parallel with an encoding of the rewrite system $\mathscr{R}$ (see Appendix B). The encoding of the rewrite system

will be a process capable of rewriting terms until they become irreducible. So, if we encode an Appπ-process P we should end up with $[\![P]\!] \mid [\![\mathscr{R}]\!]$. The encoding of the rewrite system consists of a parallel composition of processes that we will call term constructors.

We encode only processes which are on substitution normal form, which means that all names, terms and subterms are factored out into active substitutions. This will be very helpful as we can consider an active substitution $\{M/x\}$ as the term M located at x and the subterms of M are only variables referencing the actual subterms.

The encoding of an active substitution $\{M/x\}$ depends on whether M is a name, a variable or a term with a principal function name. If M is a name, the active substitution is encoded as a process which will have certain methods that can be invoked by sending a request on the location name x. Methods for retrieving the identity of the name, and sending and receiving on the name in question are required. Since subjects of prefixes are also factored out they can only be used indirectly; the subject is only a reference to a process holding the actual subject. A prefix is then encoded as a process that requests its object to be sent. If M is a term with a principal function name, the active substitution is encoded as a process that simply sends its arguments to the term constructors, which in turn reduces the term until it becomes irreducible. If M is a variable y, the active substitution is encoded as a link $x \rhd y$ that redirects requests. In the literature a link is also called a forwarder or a wire [MS98, SW01]. For instance links are used in by Sangiorgi to obtain a characterisation of barbed congruence in the $L\pi$-calculus [MS98].

In order to facilitate rewriting we need to be able to create new terms on the fly. This is the task of the encoded rewrite system. The encoding of the rewrite system will be responsible for reducing terms by creating new terms from old ones according to the rewrite rules. The encoding of the rewrite system consists of a parallel process for each function name; these processes are the term constructors mentioned above. To instantiate a term at some location, a location name and references to the arguments are sent to one of these term constructors which in turn reduces the term until it becomes irreducible. This may include calling other term constructors.

The names and variables of the Appπ-calculus will all be the names of the π-calculus. Furthermore, we assume that we have reserved distinct names for each function symbol of the signature and for each method name along with the names $\top$ and $\bot$ representing boolean values true and false.

We present the encoding in small fragments and explain how each fragment is supposed to interact with the rest of the encoded process.

6.3 Processes encoded

For simplicity we show the encoding in a monadic form but extending it to polyadic form is straightforward. We will use $\overline{a}\langle\nu r\rangle.P$ as an abbreviation for $(\nu r)(\overline{a}\langle r\rangle.P)$.

We present the simplest part of the encoding first. This consists of the encoding

of **0** which is just the identity mapping and of composition, restriction and replication which are homomorphisms.

$$\begin{aligned}
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \mathbf{0} \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu a)P]\!] &\stackrel{\text{def}}{=} (\nu a)[\![P]\!] \\
[\![!P]\!] &\stackrel{\text{def}}{=} ![\![P]\!]
\end{aligned}$$

Names in active substitutions are encoded as objects having methods for retrieving its identity, sending and receiving and matching against another term.

$$\begin{aligned}
[\![\{a/x\}]\!] \stackrel{\text{def}}{=}\ & !x(c,m,r). \\
& [c = \mathbf{id}]\overline{r}\langle a\rangle + \\
& [c = \mathbf{send}]\overline{a}\langle m\rangle.\overline{r} \\
& [c = \mathbf{receive}]a(y).\overline{r}\langle y\rangle \\
& [c = \mathbf{match}]\overline{m}\langle \mathbf{id}, \bot, \nu s\rangle.s(b). \\
& \quad ([b = a]\overline{r}\langle \top\rangle + [b \neq a]\overline{r}\langle \bot\rangle)
\end{aligned}$$

As mentioned in the introduction, subjects of prefixes must be used indirectly. The encoding of a prefixed process $x(y).P$ must relay the task of receiving to the process located at x. This is done by invoking the **receive** method of the process located at x. If the term is a name, it will receive on its identity name and return the received name. If the term is not a name, the process will deadlock since only names implement the **receive** (and **send**) method.

$$\begin{aligned}
[\![\overline{x}\langle y\rangle.P]\!] &\stackrel{\text{def}}{=} \overline{x}\langle \mathbf{send}, y, \nu r\rangle.r.[\![P]\!] \\
[\![x(y).P]\!] &\stackrel{\text{def}}{=} \overline{x}\langle \mathbf{receive}, \bot, \nu r\rangle.r(y).[\![P]\!]
\end{aligned}$$

The encoding of a match is not difficult, since it only requires us to invoke the **match** method which is implemented by all terms. The result of the match is then used to decide which branch to take.

$$\begin{aligned}
[\![\text{if } x = y \text{ then } P \text{ else } Q]\!] \stackrel{\text{def}}{=}\ & \overline{x}\langle \mathbf{match}, y, \nu r\rangle.r(m). \\
& \quad ([m = \top][\![P]\!] + [m = \bot][\![Q]\!])
\end{aligned}$$

Terms in active substitutions are encoded simply as a process which sends its desired location, say x, and arguments to the appropriate term constructor process. The term constructor process will then reduce the term until it becomes irreducible and locate it at x.

$$[\![\{f(\vec{y})/x\}]\!] \stackrel{\text{def}}{=} \overline{f}\langle x, \vec{y}\rangle.$$

Active substitutions which contains a variable are just an extra indirection and are encoded as a link.

$$[\![\{y/x\}]\!] \stackrel{\text{def}}{=} x \triangleright y.$$

The link $x \triangleright y$ is the process $!x(c,m,r).\overline{y}\langle c,m,r\rangle$ which simply receives on x and resends on y any number of times.

6.4 The rewrite system encoded

A rewrite system $\mathscr{R}$ over a signature $\Sigma = (\Omega, \alpha)$ is encoded as a parallel composition of an encoding of each function symbol, that is

$$[\![\mathscr{R}]\!] \stackrel{\text{def}}{=} \prod_{f \in \Omega} [\![f]\!].$$

Here the processes $[\![f]\!]$ are the aforementioned term constructors. The encoding of function symbols is given by

$$[\![f]\!] \stackrel{\text{def}}{=} !f(x, \vec{y}).(\nu s, e)\left(\prod_{r \in \mathscr{R}_f} [\![r]\!]_{x,\vec{y},s,e} \mid s \mid \underbrace{e.\ldots.e}_{|\mathscr{R}_f|}.[\![f(\vec{y})]\!]_x\right)$$

where $\mathscr{R}_f$ is the set of rules in the rewrite system with f as principal function symbol on the left hand side and $|\vec{y}| = \alpha(f)$. The encoding of irreducible terms is

$$\begin{aligned}
[\![f(\vec{y})]\!]_x \stackrel{\text{def}}{=}\ & !x(c, m, r). \\
& [c = \mathbf{id}].\overline{r}\langle f\rangle + \\
& [c = \mathbf{subterms}].\overline{r}\langle \vec{y}\rangle + \\
& [c = \mathbf{match}]\overline{m}\langle \mathbf{id}, \bot, \nu t\rangle.t(h). \\
& \quad [h \neq f]\overline{r}\langle\bot\rangle + \\
& \quad [h = f]\overline{m}\langle \mathbf{subterms}, \bot, t\rangle.t(\vec{z}). \\
& \qquad (\nu s, e)\left(\prod_{i=1}^{|\vec{y}|}\left(\overline{y}_i\langle \mathbf{match}, z_i, \nu p\rangle.p(b).([b = \top].\overline{s} + [b = \bot].\overline{e})\right)\right. \\
& \qquad\qquad \left. \mid e.\overline{r}\langle\bot\rangle \mid \underbrace{s.\ldots.s}_{|\vec{y}|}.\overline{r}\langle\top\rangle\right).
\end{aligned}$$

For proof technical reasons only we need to extend the encoding of irreducible terms to arbitrary terms. We do this by defining

$$\begin{aligned}
[\![f(\vec{z}, N, \vec{M})]\!]_x &\stackrel{\text{def}}{=} (\nu p)([\![f(\vec{z}, p, \vec{M})]\!]_x \mid [\![N]\!]_p) \\
[\![a]\!]_x &\stackrel{\text{def}}{=} [\![\{a/x\}]\!],
\end{aligned}$$

where $\vec{z}$ and $\vec{M}$ are possibly empty sequences of names and terms respectively and N is not a variable.

6.5 The rules encoded

The encoding of the rules—although not very difficult to do by hand—are by far the most complicated and we break it into several parts.

Given a rule $(f(g(z_1, z_2), z_3) \longrightarrow h(z_1, z_2), z_1, z_2 z_3)$ we want to be able to match a term such as $f(g(a, h(b)), h(b))$ against the left hand side of the rule and—if there is a match—instantiate the right hand side of the rule. The left hand

side of the rule tells us were the sub-terms are located in the parse tree and the sequence $z_1, z_2 z_3$ tells us which sub-terms are required to be equal. In this example there is a match since the two occurrences of $h(b)$ appear in place of the variables z_2 and z_3 and the term a does not need to match any other sub-term. In our encoding we must obtain references to the two occurrences of the sub-terms $h(b)$ in order to match them.

We identify the following three steps which must be carried out by the encoding of the rewrite rules:

(i) Fetch and bind appropriate sub-terms,

(ii) check for equality on terms and

(iii) instantiate new term.

If (i) and (ii) a failure should be indicated by synchronizing on a dedicated channel.

We describe each step separately starting from the bottom of the list.

6.5.1 Instantiating new terms

Instantiating new terms from existing ones is easy in our encoding. Suppose we have the encoding of terms M_1, M_2 and M_3 located at y_1, y_2 and y_3 and we want to instantiate the term $f(M_1, M_2, M_3)$ then we simply send the references to the existing sub-terms on f, that is $\overline{f}\langle x, y_1, y_2, y_3\rangle$ where x is the location of the new term.

If the known terms are nested deeper in the new term as in $f(g(M_1, M_2), M_3)$ we must instantiate the sub-term $g(M_1, M_2)$ first and locate it at some new name. We get $(\nu p)(\overline{g}\langle p, y_1, y_2\rangle \mid \overline{f}\langle x, p, y_3\rangle)$.

The general encoding for instantiating terms is easy to write out, since it can be defined by the composition of two previously defined encodings.

$$\mathcal{I}[\![M]\!]_x \stackrel{\text{def}}{=} [\![\{\!\{M/x\}\!\}]\!]$$

where x is the location of the new term and $\{\!\{\cdot\}\!\}$ is the substitution normal form encoding form Section 3.4 on page 21. Note that we omit the parameter in the rest of this chapter.

6.5.2 Checking for equality

Given a sequence ζ of sequences of variables, it is not too difficult to encode a process which matches each element in ξ for each $\xi \in \zeta$. Take for example the sequence xy, zuv where x and y must be matched and z, u and v likewise. We add two extra parameters to the encoding, namely a process which must be started if the all matches are successful and a name e which should be used to signal failure.

A general encoding looks like this

$$\begin{aligned}
\mathcal{E}[\![P]\!]_e &\stackrel{\text{def}}{=} P \\
\mathcal{E}[\![z, \zeta, P]\!]_e &\stackrel{\text{def}}{=} \mathcal{E}[\![\zeta, P]\!]_e \\
\mathcal{E}[\![z_1 z_2 \xi, \zeta, P]\!]_e &\stackrel{\text{def}}{=} \overline{z}_1\langle \mathbf{match}, z_2, \nu t\rangle.t(b). \\
&\qquad [b = \bot]\overline{e} + \\
&\qquad [b = \top]\mathcal{E}[\![z_2\xi, \zeta, P]\!]_e
\end{aligned}$$

where the z_i's are variable and ζ is a possibly empty sequence of sequences of variables and ξ is possibly empty sequence of variables.

6.5.3 Fetch and bind

Given a location of an encoded term we wish to be able to determine whether the term is of a certain syntactic form and bind sub-terms to certain variables. If for example want to match the term $f(g(M_1, M_2), M_3)$ against the left hand side $f(g(y_1, y_2), y_3)$ of some rule, we want to check if the term has the syntactic form and bind the locations of the sub-terms M_1, M_2 and M_3 to variables y_1, y_2 and y_3. We will write $x \succ L$ to mean "check whether the term at location x has the syntactic form given by L and bind actual sub-terms to the corresponding variables in L."

The encoding is given by

$$\begin{aligned}
\mathcal{F}[\![P]\!]_e &\stackrel{\text{def}}{=} P \\
\mathcal{F}[\![x \succ f(\vec{z}), \psi, P]\!]_e &\stackrel{\text{def}}{=} \overline{x}\langle \mathbf{id}, \bot, \nu s\rangle.s(h). \\
&\qquad [h \neq f]\overline{e} + \\
&\qquad [h = f]\overline{x}\langle \mathbf{subterms}, \bot, s\rangle.s(\vec{z}). \\
&\qquad \mathcal{F}[\![\psi, P]\!]_e \\
\mathcal{F}[\![z_1 \succ z_2, \psi, P]\!]_e &\stackrel{\text{def}}{=} \mathcal{F}[\![\psi, P]\!]_e\{z_1/z_2\} \\
\mathcal{F}[\![x \succ f(M_1, \ldots, M_m, \vec{z}), \psi, P]\!]_e &\stackrel{\text{def}}{=} \mathcal{F}[\![x \succ f(M_1, \ldots, M_{m-1}, p, \vec{z}), p \succ M_m, \psi, P]\!]_e
\end{aligned}$$

where z_1 and z_2 are variables, $\vec{z}$ is a possibly empty sequence of variables and M_m is a term but not a variable and $M_1, \ldots, M_{m-1}$ are arbitrary terms (including variables). The new term will be located at x. The name e is for signalling failure.

6.5.4 Assembling the encoding

We can now compose the three components describe above into an encoding of the rules of rewrite system.

Let $(f(M_1, \ldots, M_n) \longrightarrow R, \zeta)$ be a left-linear rule where $f(M_1, \ldots, M_n)$ then the encoding is given by

$$[\![(f(M_1, \ldots, M_n) \longrightarrow R, \zeta)]\!]_{x,\vec{y},s,e} \stackrel{\text{def}}{=} \mathcal{F}[\![y_1 \succ M_1, \ldots, y_n \succ M_n, \mathcal{E}[\![\zeta, \overline{s}.\mathcal{I}[\![R]\!]_x]\!]_e]\!]_e.$$

Relating this to the encoding of the function symbols we see that the match on the principal function symbol is done by sending on the name corresponding to the function symbol and thus it is not necessary to check the that the principal function symbol is correct; this has already been done implicitly.

6.5.5 Examples of encoded rules

Consider the rule $\mathsf{dec}(\mathsf{enc}(x,y),y) \longrightarrow x$ which we convert to

$$(\mathsf{dec}(\mathsf{enc}(z_1,z_2),z_3) \longrightarrow z_1,(z_1,z_2z_3)).$$

We then have

$$\begin{aligned}
& [\![(\mathsf{dec}(\mathsf{enc}(z_1,z_2),z_3) \longrightarrow z_1,(z_1,z_2z_3))]\!]_{x,y_1,y_2,s,e} \\
\stackrel{\text{def}}{=}\ & \mathcal{F}[\![y_1 \succ \mathsf{enc}(z_1,z_2), y_2 \succ z_3, \mathcal{E}[\![z_1,z_2z_3,\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \mathcal{F}[\![y_2 \succ z_3, \mathcal{E}[\![z_1,z_2z_3,\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \mathcal{F}[\![\mathcal{E}[\![z_1,z_2y_2,\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \mathcal{E}[\![z_1,z_2y_2,\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \mathcal{E}[\![z_2y_2,\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \overline{z}_2\langle \mathbf{match},y_2,\nu t\rangle.t(b) \\
& \qquad\quad [b = \bot]\overline{e} + \\
& \qquad\quad [b = \top]\mathcal{E}[\![\overline{s}.\mathcal{I}[\![z_1]\!]_x]\!]_e \\
=\ & \overline{y}_1\langle \mathbf{id},\bot,\nu t\rangle.t(h). \\
& \quad [h \neq \mathsf{enc}]\overline{e} + \\
& \quad [h = \mathsf{enc}]\overline{y}_1\langle \mathbf{subterms},\bot,t\rangle.t(z_1,z_2). \\
& \qquad \overline{z}_2\langle \mathbf{match},y_2,\nu t\rangle.t(b) \\
& \qquad\quad [b = \bot]\overline{e} + \\
& \qquad\quad [b = \top]\overline{s}.x \triangleright z_1
\end{aligned}$$

6.6 A rewrite system

In the previous section we gave an encoding of Appπ in the π-calculus. We assumed that the set of equations could be transformed to a terminating and confluent rewrite system such that equality is decidable, see Appendix B for background on rewrite systems. In this section we define a rewrite system from the set of equations Φ used for instantiating spi in Appπ. These equations are defined in Section 5.1. The rewrite system is created by application of the Knuth-Bendix procedure. We will also prove that this rewrite system is terminating and confluent by applying the Knuth-Bendix test.

6.6.1 Definition of $\mathscr{R}_S$

As mentioned in Appendix B it is often simple to create a rewrite system from a set of equations. This is indeed the case for the set of equations Φ. The well founded ordering $>$ is defined on terms such that $L > R$ if $|L| > |R|$ where $|L|$ is the length of the string L. From this we get the following rewrite system.

Definition 6.6.1
The rewrite system $\mathscr{R}_S$ is the following set of rules.

$$\begin{aligned}
\mathsf{pred}(\mathsf{suc}(x)) &\longrightarrow x \\
\mathsf{dec\text{-}p}_\alpha(\mathsf{enc\text{-}p}(x,k),k) &\longrightarrow x \\
\mathsf{dec\text{-}s}(\mathsf{enc\text{-}s}(x,k),k) &\longrightarrow x \\
\mathsf{fst}(\mathsf{pair}(x,y)) &\longrightarrow x \\
\mathsf{snd}(\mathsf{pair}(x,y)) &\longrightarrow y \\
\mathsf{cast\text{-}s}(\mathsf{fmt}(x,y,z,n)) &\longrightarrow x \\
\mathsf{cast\text{-}a}(\mathsf{fmt}(x,y,z,n)) &\longrightarrow y \\
\mathsf{cast\text{-}p}(\mathsf{fmt}(x,y,z,n)) &\longrightarrow z \\
\mathsf{cast}_\alpha(\mathsf{fmt}(x,y,z,n)) &\longrightarrow n
\end{aligned}$$

Lemma 6.6.2
The rewrite system $\mathscr{R}_S$ is terminating.

Proof
Since each rule in $\mathscr{R}_S$ strictly reduces the length of the string of a term, by Theorem B.3.3, we have that $\mathscr{R}_S$ is terminating. ■

Next we will prove that $\mathscr{R}_S$ is confluent. We do this by applying the Knuth-Bendix test.

Lemma 6.6.3
The rewrite system $\mathscr{R}_S$ is confluent.

Proof
Critical pairs may only occur between rules $L \longrightarrow R$ and $L' \longrightarrow R'$ where the principal function of L' occurs in L. Therefore we only have trivial critical pairs.

Then by Theorem B.3.7 we get that $\mathscr{R}_S$ is locally confluent and by Theorem B.3.4 $\mathscr{R}_S$ is also confluent. ■

6.7 Soundness

This section contains the lemmas, propositions and theorems needed to prove operational soundness of the encoding of Appπ in the π-calculus. Most of them regards the encoding of the rewrite system. The first lemma shows that matching works as expected on syntactically equal terms which are irreducible.

Lemma 6.7.1
Let $\mathscr{R}$ be a rewrite system over a signature Σ and let X be the set of Appπ-names and let $N \in T_\Sigma(X)$ be an irreducible term. Then

$$\overline{x}\langle \mathbf{match}, x, r\rangle.P \mid [\![N]\!]_x \quad \xrightarrow{\tau}{}^* \approx \quad P \mid \overline{r}\langle\top\rangle \mid [\![N]\!]_x$$

and

$$\overline{x}\langle \mathbf{match}, y, r\rangle.P \mid [\![N]\!]_x \mid [\![N]\!]_y \quad \xrightarrow{\tau}{}^* \approx \quad P \mid \overline{r}\langle\top\rangle \mid [\![N]\!]_x \mid [\![N]\!]_y.$$

Proof
The proof is by induction in the height of the parse tree of the term N. The induction basis is the case when $N = a$ and the proof is a simple trace of the "execution" of the process:

$$\begin{aligned}
& \overline{x}\langle \mathbf{match}, x, r\rangle.P \mid [\![N]\!]_x \\
\xrightarrow{\tau}{}^2 \quad & P \mid \overline{x}\langle \mathbf{id}, \bot, \nu s\rangle.s(b).([b = a]\overline{r}\langle\top\rangle + [b \neq a]\overline{r}\langle\bot\rangle) \mid [\![N]\!]_x \\
\xrightarrow{\tau}{}^2 \quad & P \mid (\nu s)(s(b).([b = a]\overline{r}\langle\top\rangle + [b \neq a]\overline{r}\langle\bot\rangle) \mid \overline{s}\langle a\rangle) \mid [\![N]\!]_x \\
\xrightarrow{\tau}{}^2 \quad & P \mid (\nu s)\overline{r}\langle\top\rangle \mid [\![N]\!]_x \\
\approx \quad & P \mid \overline{r}\langle\top\rangle \mid [\![N]\!]_x
\end{aligned}$$

Suppose $N = f(M_1, \ldots, M_n)$ and observe that

$$[\![f(M_1, \ldots, M_n)]\!]_x \equiv (\nu y_1, \ldots, y_n)([\![f(y_1, \ldots, y_n)]\!]_x \mid [\![M_1]\!]_{y_1} \mid \cdots \mid [\![M_n]\!]_{y_n}).$$

Then we have

$$\begin{aligned}
& \overline{x}\langle \mathbf{match}, x, r\rangle.P \mid [\![N]\!]_x \\
\xrightarrow{\tau}{}^2 \quad & P \mid (\nu y_1, \ldots, y_n)\Bigg(\overline{x}\langle \mathbf{id}, \bot, \nu t\rangle.t(h). \\
& \quad [h \neq f]\overline{r}\langle\bot\rangle + \\
& \quad [h = f]\overline{x}\langle \mathbf{subterms}, \bot, t\rangle.t(\vec{z}). \\
& \qquad (\nu s, e)\Bigg(\prod_{i=1}^{|\vec{y}|}\Bigg(\overline{y}_i\langle \mathbf{match}, z_i, \nu p\rangle.p(b).([b = \top].\overline{s} + [b = \bot].\overline{e})\Bigg) \\
& \qquad\quad \mid e.\overline{r}\langle\bot\rangle \mid \underbrace{s.\ldots.s}_{|\vec{y}|}.\overline{r}\langle\top\rangle\Bigg) \mid \\
& \qquad\quad [\![f(y_1, \ldots, y_n)]\!]_x \mid [\![M_1]\!]_{y_1} \mid \cdots \mid [\![M_n]\!]_{y_n}\Bigg)
\end{aligned}$$

$$
\begin{aligned}
\xrightarrow{\tau}{}^{4} \quad & P \mid (\nu y_1, \ldots, y_n, t)\Bigg(\overline{x}\langle \mathbf{subterms}, \bot, t\rangle.t(\vec{z}). \\
& \qquad (\nu s, e)\Bigg(\prod_{i=1}^{|\vec{y}|}\Big(\overline{y}_i\langle \mathbf{match}, z_i, \nu p\rangle.p(b).([b=\top].\overline{s} + [b=\bot].\overline{e})\Big) \\
& \qquad\qquad \mid e.\overline{r}\langle\bot\rangle \mid \underbrace{s.\ldots.s}_{|\vec{y}|}.\overline{r}\langle\top\rangle\Bigg) \mid \\
& \qquad\qquad [\![f(y_1,\ldots,y_n)]\!]_x \mid [\![M_1]\!]_{y_1} \mid \cdots \mid [\![M_n]\!]_{y_n}\Bigg) \\
\xrightarrow{\tau}{}^{3} \quad & P \mid (\nu y_1, \ldots, y_n, t)\Bigg((\nu s, e)\Bigg(\prod_{i=1}^{|\vec{y}|}\Big(\overline{y}_i\langle \mathbf{match}, y_i, \nu p\rangle.p(b).([b=\top].\overline{s} + \\
& \qquad [b=\bot].\overline{e})\Big) \mid e.\overline{r}\langle\bot\rangle \mid \underbrace{s.\ldots.s}_{|\vec{y}|}.\overline{r}\langle\top\rangle\Bigg) \mid \\
& \qquad\qquad [\![f(y_1,\ldots,y_n)]\!]_x \mid [\![M_1]\!]_{y_1} \mid \cdots \mid [\![M_n]\!]_{y_n}\Bigg) \\
\xrightarrow{\tau}{}^{*} \quad & P \mid (\nu y_1, \ldots, y_n, t)((\nu s, e)\Big(e.\overline{r}\langle\bot\rangle \mid \overline{r}\langle\top\rangle) \mid [\![f(y_1,\ldots,y_n)]\!]_x \mid \\
& \qquad\qquad [\![M_1]\!]_{y_1} \mid \cdots \mid [\![M_n]\!]_{y_n}\Big) \\
\approx_b \quad & P \mid \overline{r}\langle\top\rangle) \mid [\![N]\!]_x
\end{aligned}
$$

■

The next lemma shows that match works as expected on distinct irreducible terms.

Lemma 6.7.2
Let $\mathscr{R}$ be a rewrite system over a signature Σ and let X be the set of Appπ-calculus names and $N, N' \in T_\Sigma(X)$ be irreducible terms such that $N \neq N'$. Then

$$\overline{x}\langle \mathbf{match}, y, r\rangle.P \mid [\![N]\!]_x \mid [\![N']\!]_y \xrightarrow{\tau}{}^{*} \approx P \mid \overline{r}\langle\bot\rangle \mid [\![N]\!]_x \mid [\![N']\!]_y$$

Proof
The proof is similar (in structure and length) to the proof of the previous lemma and we omit the details. There are a few more cases to check than in the proof of the previous lemma. The induction basis includes checking that the lemma holds

- if N and N' are distinct names,
- if one of N and N' is a name and the other is a function application and
- if both N and N' are function applications but with distinct principal function names.

The induction step then consists of checking that the lemma holds if N and N' are both function applications with the same principal function names. ■

The next two lemmas states that the encoding $\mathcal{E}$ works as intended.

Lemma 6.7.3
Let $\mathscr{R}$ be a rewrite system over a signature Σ and let X be the set of Appπ-names. Let η be a function from the set of Appπ-variables to the set of irreducible terms in $T_\Sigma(X)$. Let ξ be a finite, possibly empty, sequence of not necessarily distinct variables, let ζ be a possibly empty sequence of sequences of variables and let P be a π-calculus process. If there exist $z, z' \in \xi$ such that $\eta(z) \neq \eta(z')$, then

$$\mathcal{E}[\![\xi, \zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x \quad \xrightarrow{\tau}{}^* \approx_b \quad \overline{e} \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x, \tag{6.1}$$

otherwise

$$\mathcal{E}[\![\xi, \zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x \quad \xrightarrow{\tau}{}^* \approx_b \quad \mathcal{E}[\![\zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x. \tag{6.2}$$

Proof
We prove (6.2) by induction in the length of ξ. Initially, we note that if ξ has length 0 or 1 the result holds trivially, since in this case $\mathcal{E}[\![\xi, \zeta, P]\!] = [\![\zeta, P]\!]$. Suppose that (6.2) holds for sequences ξ of length at least two and let $\xi = z_1 z_2 \xi'$. By assumption we have that $\eta(z_1) = \eta(z_2)$, so by applying Lemma 6.7.1 and the induction hypothesis we can obtain

$$\begin{aligned}
& \mathcal{E}[\![z_1 z_2 \xi', \zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x \\
= \quad & \overline{z}_1\langle \mathbf{match}, z_2, \nu t\rangle.t(b).\big([b = \bot]\overline{e} + [b = \top]\mathcal{E}[\![z_2 \xi', \zeta, P]\!]_e\big) \mid \prod_{x \in \xi'} [\![\eta(x)]\!]_x \\
\xrightarrow{\tau}{}^* \approx_b \quad & \mathcal{E}[\![z_2 \xi', \zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x \\
\xrightarrow{\tau}{}^* \approx_b \quad & \mathcal{E}[\![\zeta, P]\!]_e \mid \prod_{x \in \xi} [\![\eta(x)]\!]_x.
\end{aligned}$$

For the proof of (6.1), we define $\delta = \min\{i \mid \eta(\xi_i) \neq \eta(\xi_{i+1})\}$. The proof is by induction in δ, so first assume that $\delta = 1$. Then it is not difficult to use Lemma 6.7.2 to establish the result. The induction step is equally straightforward ■

Lemma 6.7.4
Let $\mathscr{R}$ be a rewrite system over a signature Σ and let X be the set of Appπ-names. Let η be a function from the set of Appπ-variables to the set of irreducible terms in $T_\Sigma(X)$. Let ζ be a finite, possibly empty, sequence of sequences of variables and let S be the set of variables occurring in the elements of ζ. Let P be a π-calculus process. If there exist $\xi \in \zeta$ and $z, z' \in \xi$ such that $\eta(z) \neq \eta(z')$ then

$$\mathcal{E}[\![\zeta, P]\!]_e \mid \prod_{x \in S} [\![\eta(x)]\!]_x \quad \xrightarrow{\tau}{}^* \approx \quad \overline{e} \mid \prod_{x \in S} [\![\eta(x)]\!]_x, \tag{6.3}$$

otherwise

$$\mathcal{E}[\![\zeta, P]\!]_e \mid \prod_{x\in S}[\![\eta(x)]\!]_x \quad \xrightarrow{\tau}{}^{*} \approx \quad P \mid \prod_{x\in S}[\![\eta(x)]\!]_x. \tag{6.4}$$

Proof
Yet another proof by induction; this time by induction in the length of ζ. As in the proof of the previous lemma, each of the cases are proven separately. We omit the proof. ■

The next lemma states that fetch and bind works correctly on terms that are substitution instances of one another. We prove that when matching a rule with a substitution instance of the same rule, $\mathcal{F}$ successfully executes resulting in a process where each variable in the rule is bound to the correct sub-term of its substitution instance.

Lemma 6.7.5
Let N be a term with variables $z_1, \ldots, z_k$. Let $\sigma = \{\vec{N}/\vec{z}\}$ be a substitution, where $|\vec{N}| = k = |\vec{z}|$, and the terms N_i are irreducible, $1 \leq i \leq k$, and $\sigma(N)$ is irreducible. Then

$$(\nu\vec{y})\big([\![N\{\vec{y}/\vec{z}\}]\!]_x \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![x \succ N, \psi, P]\!]\big)$$
$$\longrightarrow^{*} (\nu\vec{y})\big([\![N\{\vec{y}/\vec{z}\}]\!]_x \mid \prod_{i=1}^{n}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![\psi, P]\!]\{\vec{y}/\vec{z}\}\big).$$

Proof
The proof is carried out by induction in N.

Induction basis: Suppose $N = f(\vec{z})$, $|\vec{z}| = k$. Then expanding the encoding and reducing seven times immediately yields.

$$(\nu\vec{y})\big([\![f(\vec{y})]\!]_x \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![x \succ f(\vec{z}), \psi, P]\!]\big)$$
$$\longrightarrow^{7} (\nu\vec{y})\big([\![f(\vec{y})]\!]_x \mid \prod_{i=1}^{n}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![\psi, P]\!]\{\vec{y}/\vec{z}\}\big),$$

where $|\vec{y}| = k$.

Induction hypothesis: Given a term $f(\vec{t})$, where $|\vec{t}| = n$ assume the lemma holds for all sub-terms t_i, $1 \leq i \leq n$.

Induction step: Suppose $N = f(\vec{t})$, where $\mathrm{v}(f(\vec{t})) = \{\vec{z}\}$ and $|\vec{t}| = n$, and $|\vec{z}| = k$. We have $f(\vec{t})\{\vec{N}/\vec{z}\} = f(M_1, \ldots, M_n)$. Note that since $f(\vec{t})\{\vec{N}/\vec{z}\} = f(t_1\{\vec{N}/\vec{z}\}, \ldots, t_n\{\vec{N}/\vec{z}\})$, we have

$$[\![f(t_1, \ldots, t_n)\{\vec{N}/\vec{z}\}]\!]_x \equiv (\nu\vec{v})\big(\prod_{i=1}^{n}[\![t_i\{\vec{N}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big).$$

We now get:

$$
\begin{aligned}
&(\nu\vec{v},\vec{y})\big(\prod_{i=1}^{n}[\![t_i\{\vec{y}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big) \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![x \succ f(\vec{t}),\psi,P]\!]\big)\\
&= (\nu\vec{v},\vec{y})\big(\prod_{i=1}^{n}[\![t_i\{\vec{y}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big) \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![x \succ f(\vec{p}),\vec{p}\succ \vec{t},\psi,P]\!]\big)\\
&\longrightarrow^* (\nu\vec{v},\vec{y})\big(\prod_{i=1}^{n}[\![t_i\{\vec{y}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big) \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![\vec{p}\succ \vec{t},\psi,P]\!]\{\vec{v}/\vec{p}\}\big)\\
&\quad = (\nu\vec{v},\vec{y})\big(\prod_{i=1}^{n}[\![t_i\{\vec{y}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big) \mid \prod_{i=1}^{n}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![\vec{v}\succ \vec{t},\psi,P]\!]\big).
\end{aligned}
$$

Since the induction hypothesis applies to all t_i repeated application of the induction hypothesis yields

$$
\longrightarrow^* (\nu\vec{v},\vec{y})\big(\prod_{i=1}^{n}[\![t_i\{\vec{y}/\vec{z}\}]\!]_{v_i} \mid [\![f(\vec{v})]\!]_x\big) \mid \mid \prod_{i=1}^{n}[\![N_i]\!]_{y_i} \mid \mathcal{F}[\![\psi,P]\!]\{\vec{y}/\vec{z}\}\big).
$$

This concludes the proof of the lemma.

■

We need a notation for the location of a subterm which occurs in the encoding of an irreducible term. This is the purpose of the following section.

Definition 6.7.6 (References)
Let $[\![f(M_1,\ldots,M_n)]\!]_x$ be an encoded irreducible term. Then inductively define the variable which refers to the term at position ω as follows.

- $\mathrm{ref}([\![\{a/x\}]\!]|_\omega) = x \;\; \textit{if} \;\; \omega = \varepsilon$
- $\mathrm{ref}([\![f(M_1,\ldots,M_n)]\!]_x|_\omega) = \begin{cases} x & \textit{if } \omega = \varepsilon \\ \mathrm{ref}([\![M_i]\!]_{y_i}|_{\omega'}) & \textit{if } \omega = i\omega' \end{cases}$

If $\mathrm{ref}([\![M]\!]_x|_\omega) = y$ we say that the sub-term of $[\![M]\!]_x$ at position ω is referenced by y.

Using references Lemma 6.7.5 has the following corollary.

Corollary 6.7.7
Let N be a term with $\mathrm{v}(N) = \{z_1,\ldots,z_k\}$, distinct, occurring at positions $\omega_1,\ldots,\omega_k$. Let X be the set of Appπ-names and let $N_1,\ldots,N_k \in T_\Sigma(X)$ be irreducible terms such that $N\{\vec{N}/\vec{z}\}$ is irreducible. Assume that $\mathrm{ref}\left([\![N\{\vec{N}/\vec{z}\}]\!]_x|_{\omega_i}\right) = y_i$ for all $1 \le i \le k$. Then

$$
(\nu\vec{y})\big([\![N\{\vec{N}/\vec{z}\}]\!]_x \mid \mathcal{F}[\![x\succ N,\psi,P]\!]\big) \longrightarrow^* \approx (\nu\vec{y})\big([\![N\{\vec{N}/\vec{z}\}]\!]_x \mid \mathcal{F}[\![\psi,P]\!]\{\vec{y}/\vec{z}\}\big),
$$

where

$$
(\nu\vec{y})\big([\![N\{\vec{N}/\vec{z}\}]\!]_x \mid \mathcal{F}[\![x\succ N,\psi,P]\!]\big) \equiv [\![N\{\vec{N}/\vec{z}\}]\!]_x \mid \mathcal{F}[\![x\succ N,\psi,P]\!].
$$

Proof
We prove this corollary by showing that

$$[\![N\{\vec{N}/\vec{z}\}]\!]_x \equiv (\nu\vec{y})\big([\![N\{\vec{y}/\vec{z}\}]\!]_x \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i}\big).$$

The corollary then follows from Lemma 6.7.5. This is shown by induction in the structure of N.

Induction basis: Suppose $N = f(z_1, \ldots, z_k)$. Since $\mathrm{ref}\big([\![f(\vec{N})]\!]_x|_{\omega_i}\big) = y_i$ for $\omega_i \in \{1, \ldots, k\}$, we get

$$(\nu\vec{y})\big([\![f(\vec{z})\{\vec{N}/\vec{z}\}]\!]_x\big) = (\nu\vec{y})\big([\![f(\vec{y})]\!]_x \mid \prod_{i=1}^{k}[\![N_i]\!]_{y_i}\big).$$

Induction hypothesis: Let $f(t_1, \ldots, t_n)$ and assume the corollary holds for all t_i, $1 \leq i \leq n$.

Induction step: Suppose $N = f(t_1, \ldots, t_n)$. By the definition of the rewrite rules we know that for some sets Z_i, $\mathrm{v}(N) = \cup_{i=1}^{k} Z_i$ and $Z_i \cap Z_j = \emptyset$, when $i \neq j$. Let $\vec{z}_i$ denote the vector of variables in Z_i and let $\vec{N}_i$ denote the set of terms which is being substituted for $\vec{z}_i$. Similarly $\vec{y}_i$ denotes the vector of variables referencing $[\![\vec{N}_i]\!]$ Suppose furthermore that the n arguments of $f(t_1, \ldots, t_n)$ are factored out on the variables $k_1, \ldots, k_n$, i.e. $\mathrm{ref}\big([\![f(\vec{t})]\!]_x|_{\omega'_i}\big) = k_i$, for $\omega'_i \in \{1, \ldots, n\}$. We then have

$$\begin{aligned}
(\nu\vec{y})\big([\![f(\vec{t})\{\vec{N}/\vec{z}\}]\!]_x\big) &= (\nu\vec{y}, \vec{k})\big([\![f(\vec{k})]\!]_x \mid \prod_{i=1}^{n}[\![t_i\{\vec{N}/\vec{z}\}]\!]_{k_i}\big) \\
&\equiv (\nu\vec{k})\big([\![f(\vec{k})]\!]_x \mid \prod_{i=1}^{n}(\nu\vec{y}_i)\big([\![t_i\{\vec{y}_i/\vec{z}_i\}]\!]_{k_i} \mid \prod_{j=1}^{|Z_i|}[\![\vec{N}_{ij}]\!]_{y_{i_j}}\big)\big) \\
&\equiv (\nu\vec{y}_1, \ldots, \vec{y}_n)\big([\![f(\vec{t})\{\vec{y}_1, \ldots, \vec{y}_n/\vec{z}_1, \ldots, \vec{z}_n\}]\!]_x \mid \prod_{i=1}^{n}\prod_{j=1}^{|Z_i|}[\![\vec{N}_{ij}]\!]_{y_{j_i}}\big),
\end{aligned}$$

where the second equivalence follows from the induction hypothesis.

■

The next lemma also concerns the $\mathcal{F}$ encoding. It applies when a term does not have the same syntactic form as the left hand side of the rule it is being matched against.

Lemma 6.7.8
Let $\mathscr{R}$ be a rewrite system and let $(f(\vec{t}) \longrightarrow R, \zeta) = r' \in \mathscr{R}$ and $f(\vec{M})$ be a term. Assume that there exists a smallest ω with respect to the lexicographic ordering such that $f(\vec{t})|_\omega = g(\vec{s})$ and $f(\vec{M})|_\omega = h(\vec{M'})$ and $g \neq h$. Then

$$(\nu\vec{y})\Big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{F}[\![y_1 \succ t_1, \ldots, y_n \succ t_n, P]\!]_e\Big) \xrightarrow{\tau}{}^{*} \overline{e} \mid (\nu\vec{y})\Big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i}\Big).$$

Proof
The proof is by induction in the height of the parse tree of $f(\vec{t})$. Assume that the height of $f(\vec{t})$ is two and that $\omega = j$. Assume that the terms t_i, $i = 1, \ldots, j-1$, have the same principal function symbol as M_i. Then we have

$$(\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{F}[\![y_1 \succ t_1, \ldots, y_n \succ t_n, P]\!]_e\big) \tag{6.5}$$

$$\xrightarrow{\tau}{}^{*}\ (\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{F}[\![y_j \succ t_j, \ldots, y_n \succ t_n, P]\!]_e\{\cdot/\cdot\}\big), \tag{6.6}$$

where $\{\cdot/\cdot\}$ denotes some substitution of variables. However, as none of the variables in the substitution can occur in t_j we can ignore this substitution. We use the definition of the encoding $\mathcal{F}$ to get that the process in (6.6) reduces in three steps to

$$(\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i}\big) \mid [h \neq g]\overline{e} + [h = g]Q \xrightarrow{\tau} (\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i}\big) \mid \overline{e},$$

where Q is defined as in the $\mathcal{F}$ encoding.

For the inductive step assume that $\omega = i_1 i_2 \cdots i_l$. and consider again the process in (6.5). We can apply lemma 6.7.5 and the process reduces to

$$(\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{F}[\![y_2 \succ t_2, \ldots, y_n \succ t_n, P]\!]_e\{\cdot/\cdot\}\big),$$

where $\{\cdot/\cdot\}$ again is some unimportant substitution. We continue in this way until we reach

$$(\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{F}[\![y_{i_1} \succ t_{i_1}, \ldots, y_n \succ t_n, P]\!]_e\{\cdot/\cdot\}\big), \tag{6.7}$$

Now let $t_{i_1} = \widehat{g}(\vec{t'})$ and $M_{i_1} = \widehat{g}(\vec{N})$. Furthermore, let

$$(\nu\vec{y})\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i}\big) = (\nu\vec{y}, \vec{z})\big(\prod_{j\in\{1,\ldots,n\}\setminus i_1}[\![M_j]\!]_{y_j} \mid [\![\widehat{g}(\vec{z})]\!]_{y_{i_1}} \mid \prod_{j=1}^{m}[\![N_j]\!]_{z_j}\big). \tag{6.8}$$

Then (6.7) reduces to

$$(6.8) \mid \mathcal{F}[\![z_1 \succ t'_1, \ldots, z_m \succ t'_m, y_{i_1+1} \succ t_{i_1+1}, \ldots y_n \succ t_n, P]\!]_e\{\cdot/\cdot\},$$

where the scope of the new variables is all of the process. We now apply Corollary 6.7.7 $i_2 - 1$ times and we get

$$(6.8) \mid \mathcal{F}[\![z_{i_2} \succ t'_{i_2}, \ldots, z_m \succ t'_m, y_{i_1+1} \succ t_{i_1+1}, \ldots y_n \succ t_n, P]\!]_e\{\cdot/\cdot\},$$

where again the scope of $\vec{z}$ is all of the process. Applying the induction hypothesis then yields

$$\xrightarrow{\tau}{}^{*}\ (6.8) \mid \overline{e}$$

which completes the proof. ■

The next lemma states that if a term is irreducible, then the encoding of the term in parallel with the encoding of the rewrite system reduces to the encoding of irreducible terms. The proof uses that, when checking if a term M can be reduced by a rule $(L \longrightarrow R, \zeta)$, then it suffices to check

- if M and L have the same syntactical form, and
- if for all sequences $\xi \in \zeta$, where $\xi = x_1 \cdots x_n$ and $\omega_1, \ldots, \omega_n$ are the positions of these variables in L, we have $M|_{\omega_1} = \cdots = M|_{\omega_n}$.

Lemma 6.7.9
Let $f(M_1, \ldots, M_n)$ be an Appπ-term which is irreducible. Then

$$[\![\{\!\{\{f(\vec{M})/x\}\}\!\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow}^* \approx [\![f(\vec{M})]\!]_x \mid [\![\mathscr{R}]\!].$$

Proof
The proof is by induction in the height n of the parse tree of $f(\vec{M})$. We will denote the height of the parse tree by $\Upsilon(f(\vec{M}))$. As induction basis we consider the following case.

$f(\vec{M}) = f()$**:** Because $f()$ is irreducible and is a constant we get that $\mathscr{R}_f = \emptyset$. Therefore the encoding gives us

$$[\![\{f()/x\}]\!] \mid \prod_{g \in \Omega \setminus \{f\}} [\![g]\!] \mid !f(y).(\nu s)(s \mid [\![f()]\!]_y) \stackrel{\tau}{\longrightarrow}^* \approx [\![f()]\!]_x \mid [\![\mathscr{R}]\!].$$

The induction hypothesis is that the lemma holds for all terms $f(\vec{M'})$, where $\Upsilon(f(\vec{M'})) < n$.

Now assume that

$$[\![\{\!\{\{f(\vec{M})/x\}\}\!\}]\!] = (\nu \vec{y})(\prod_{i=1}^{n} [\![\{\!\{\{M_i/y_i\}\}\!\}]\!] \mid [\![\{f(\vec{y})/x\}]\!]).$$

Then by definition of the encoding we get

$$[\![\{\!\{\{f(\vec{M})/x\}\}\!\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow} (\nu \vec{y})([\![\mathscr{R}]\!] \mid \prod_{i=1}^{n} [\![\{\!\{\{M_i/y_i\}\}\!\}]\!] \mid (\nu s, e)(\prod_{r \in \mathscr{R}_f} [\![r]\!]_{x,\vec{y},s,e} \mid s \mid e.\cdots.e.[\![f(\vec{y})]\!]_x)).$$

We can use the induction hypothesis on $\prod_{i=1}^{n} [\![\{\!\{\{M_i/y_i\}\}\!\}]\!] \mid [\![\mathscr{R}]\!]$, we get

$$[\![\{f(\vec{M})/x\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow}^* \approx (\nu \vec{y})([\![\mathscr{R}]\!] \mid \prod_{i=1}^{n} [\![M_i]\!]_{y_i} \mid (\nu s, e)(\prod_{r \in \mathscr{R}_f} [\![r]\!]_{x,\vec{y},s,e} \mid s \mid e.\cdots.e.[\![f(\vec{y})]\!]_x)). \quad (6.9)$$

Consider a rule $r = (L \longrightarrow R, \zeta) \in \mathscr{R}_f$. Now there are two possibilities. If L and $f(\vec{M})$ have different syntactical forms, then we apply Lemma 6.7.8 and we can then infer a reduction on e. If the syntactical forms are identical we apply Corollary 6.7.7 k times.

$$\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![L \longrightarrow R, \zeta]\!]_{x,\vec{y},s,e} \xrightarrow{\tau}{}^{*} \approx \prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid \mathcal{E}[\![\zeta, \overline{s}.\mathcal{I}[\![R]\!]_x]\!]_e\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_1})/z_1\}\cdots\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_k})/z_k\},$$

where $\vec{z}$ are the k variables in L, occurring at positions $\omega_1, \ldots, \omega_k$. We abbreviate this substitution as $\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}$.

We know that there exists $z_i, z_j \in \xi$, where $\xi \in \zeta$, such that for the terms which are referenced by $\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_i})$ and $\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_j})$, say N_1 and N_2, we have $N_1 \neq N_2$. Therefore we can apply Lemma 6.7.4 and infer a $\xrightarrow{\overline{e}}{}^{*}$ transition.

From this we see that for each rule $r \in \mathscr{R}$ we get a reduction on e. Therefore (6.9) reduces to something which is weakly barbed congruent to

$$(\nu y)([\![f(\vec{y})]\!]_x \mid \prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!]).$$

■

Next we look at how the encoding of a term behaves when the term can be reduced.

Lemma 6.7.10
Let $\mathscr{R}$ be a rewrite system and let $(L \longrightarrow R, \zeta) \in \mathscr{R}$. Furthermore, let σ be a substitution and $f(\vec{M})$ a term where $\vec{M}$ are irreducible such that $\sigma(L) = f(\vec{M})$. Furthermore, assume that for all $\xi \in \zeta$, and for all pairs of variables $x_1, x_2 \in \xi$ we have $\sigma(x_1) = \sigma(x_2)$. Then

$$(\nu\vec{y}, \vec{u})\left(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\{f(\vec{y})/x\}]\!]\right) \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx (\nu\vec{y}, \vec{u})\left(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid \mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}\right), \quad (6.10)$$

where $\mathrm{v}(L) = \vec{z}$, $L|_{\omega_i} = z_i,\ i = 1, \ldots, k.$, *and* $\vec{u} = \mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})$

Proof
We simply use the definition of the encoding of active substitutions and the rewrite system to infer that

$$(\nu\vec{y}, \vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\{f(\vec{y})/x\}]\!]) \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} (\nu\vec{y}, \vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid (\nu s, e)(\prod_{r\in\mathscr{R}_f}[\![r]\!]_{x,\vec{y},s,e} \mid s \mid e.\cdots.e.[\![f(\vec{y})]\!]_x)) \quad (6.11)$$

Now let $L = f(\vec{t})$. Obviously $(L \longrightarrow R, \zeta) = r' \in \mathscr{R}_f$ and we consider its encoding

$$\mathcal{F}[\![y_1 \succ t_1, \ldots, y_n \succ t_n, \mathcal{E}[\![\zeta, \overline{s}.\mathcal{I}[\![R]\!]_x]\!]_e]\!]_e.$$

From this encoding and Corollary 6.7.7 we get that the process in (6.11) reduces to

$$(\nu\vec{y},\vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid (\nu s,e)(\prod_{r\in\mathscr{R}_f\setminus\{r'\}}[\![r]\!]_{x,\vec{y},s,e} \mid s \mid e.\cdots.e.[\![f(\vec{y})]\!]_x \mid \mathcal{E}[\![\zeta,\overline{s}.\mathcal{I}[\![R]\!]_x]\!]_e\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}))$$

Now if $x_1, x_2 \in \xi$, where $\xi \in \zeta$ and $L|_{\omega_{x_1}} = x_1$ and $L|_{\omega_{x_2}} = x_2$, then the two terms referenced by $\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_{x_1}})$ and $\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_{x_2}})$ are equal as $\sigma(x_1) = \sigma(x_2)$. Therefore by Lemma 6.7.4 the process again reduces to

$$\begin{aligned}(\nu\vec{y},\vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid (\nu s,e)(\prod_{r\in\mathscr{R}_f\setminus\{r'\}}[\![r]\!]_{x,\vec{y},s,e} \mid s \mid e.\cdots.e.[\![f(\vec{y})]\!]_x \mid \\ \overline{s}.\mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\})) \xrightarrow{\tau} \\ (\nu\vec{y},\vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid (\nu s,e)(\prod_{r\in\mathscr{R}_f\setminus\{r'\}}[\![r]\!]_{x,\vec{y},s,e} \mid e.\cdots.e.[\![f(\vec{y})]\!]_x \mid \\ \mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\})) \quad (6.12)\end{aligned}$$

Now we only need to show that

$$\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid (\nu s,e)(\prod_{r\in\mathscr{R}_f\setminus\{r'\}}[\![r]\!]_{x,\vec{y},s,e} \mid \underbrace{e.\cdots.e}_{|\mathscr{R}_f|}.[\![f(\vec{y})]\!]_x) \xrightarrow{\tau}{}^{*} \approx \prod_{i=1}^{n}[\![M_i]\!]_{y_i}.$$

This follows as for each rule $r \in \mathscr{R} \setminus \{r'\}$ we have that either the syntactic forms of $f(\vec{M})$ and the left hand side of r are different, in which case we apply Lemma 6.7.8 and we get a reduction on e, or the syntactic forms are identical. Also here we have two subcases. Either checking for equality fails. The proof of this is identical to that given in the proof of Lemma 6.7.9, and also here we get a reduction on e. Otherwise we repeat the argument from this proof and we see that the process reduces to

$$\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid (\nu s,e)(\overline{s}.\mathcal{I}[\![R]\!]_x\{\cdot/\cdot\}) \approx \prod_{i=1}^{n}[\![M_i]\!]_{y_i}.$$

Therefore after reducing all the rules in $\mathscr{R} \setminus \{r'\}$ we have that the encoding of irreducible terms $[\![f(\vec{y})]\!]_x$ is at least prefixed by one e. Completing the picture we get

$$\begin{aligned}(6.12) \xrightarrow{\tau}{}^{*} &\approx (\nu\vec{y},\vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid (\nu e)(e.\cdots.e.[\![f(\vec{y})]\!]_x) \mid \\ &\qquad \mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}) \\ &\approx (\nu\vec{y},\vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid \mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\})\end{aligned}$$

■

As a consequence of Lemma 6.7.10 we have the following lemma.

Lemma 6.7.11
Let X be the set of Appπ-names and let $f(\vec{M}) \in T_\Sigma(X)$, where $\vec{M}$ are irreducible and $|\vec{M}| = n$. Then if $f(\vec{M}) \longrightarrow_{\mathscr{R}} N$, by applying the rule $(L \longrightarrow_{\mathscr{R}} R, \zeta) \in \mathscr{R}$, and N is irreducible, it holds that

$$[\![\{\!\{f(\vec{M})/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow}^* \approx [\![N]\!]_x \mid [\![\mathscr{R}]\!].$$

If N is not irreducible, then

$$[\![\{\!\{f(\vec{M})/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow}^* \approx K \qquad \text{and} \qquad [\![\{\!\{N/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \stackrel{\tau}{\longrightarrow}^* \approx K,$$

where K is

$$(\nu \vec{y}, \vec{u})(\prod_{i=1}^{n} [\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid \mathcal{I}[\![R]\!]_x\{\vec{u}/\vec{z}\}),$$

where $\vec{u} = \mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})$ and $\mathrm{v}(L) = \vec{z}$.

Proof
Note that

$$A \stackrel{\mathrm{def}}{=} [\![\{\!\{f(\vec{M})/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] = \prod_{i=1}^{n} [\![\{\!\{M_i/y_i\}\!\}]\!] \mid [\![\{f(\vec{y})/x\}]\!] \mid [\![\mathscr{R}]\!].$$

By Lemma 6.7.9 the right-hand process reduces to something which is barbed congruent to

$$B \stackrel{\mathrm{def}}{=} (\nu \vec{y})(\prod_{i=1}^{n} [\![M_i]\!]_{y_i} \mid [\![\{f(\vec{y})/x\}]\!]) \mid [\![\mathscr{R}]\!].$$

By Lemma 6.7.10 this process again reduces to a process which is barbed congruent to

$$C \stackrel{\mathrm{def}}{=} (\nu \vec{y}, \vec{u})(\prod_{i=1}^{n} [\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid \mathcal{I}[\![R]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}), \tag{6.13}$$

where $\vec{u} = \mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})$. Now since for some A' and B' we have that $A \stackrel{\tau}{\longrightarrow}^* A' \approx B$ and $B \stackrel{\tau}{\longrightarrow}^* B' \approx C$, we conclude that there must exist some A'' such that $A' \stackrel{\tau}{\longrightarrow}^* A''$ and $A'' \approx C$, hence $A \stackrel{\tau}{\longrightarrow}^* \approx C$.

Since $\vec{M}$ are irreducible there must exist some rule $(L \longrightarrow R, \zeta) \in \mathscr{R}$ such that $\sigma(L) = f(\vec{M})$. This implies that $N = \sigma(R)$.

Suppose $\mathrm{v}(L) = \{z_1, \ldots, z_k\}$ at positions $\omega_1, \ldots, \omega_k$. We then have:

$$\sigma(z_1) = f(\vec{M})|_{\omega_1}, \ldots, \sigma(z_k) = f(\vec{M})|_{\omega_k}.$$

Assume that $\mathrm{v}(R) = \{z_{m_1}, \ldots, z_{m_l}\}$, with multiplicities, where $\{m_1, \ldots, m_l\} \subseteq \{1, \ldots, k\}$. Then $\sigma(R) = R\{f(\vec{M})|_{\omega_{m_1}}, \ldots, f(\vec{M})|_{\omega_{m_l}}/z_{m_1}, \ldots, z_{m_l}\}$. First consider the case when N is irreducible. We show that $C \stackrel{\tau}{\longrightarrow}^* \approx C'$ for some C' where

$$[\![R\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}]\!]_x \mid [\![\mathscr{R}]\!] \approx C' \tag{6.14}$$

by induction in the structure of R.

Induction basis: Suppose $R = z$. The left-hand side of Equation (6.14) and the right-hand side of Equation 6.13 now equals respectively

$$[\![f(\vec{M})|_\omega]\!]_x \mid [\![\mathscr{R}]\!] \qquad \text{and} \qquad (\nu\vec{y}, u)\big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid x \triangleright u\big),$$

where $u = \mathrm{ref}([\![f(\vec{M})]\!]_x|_\omega)$. These processes have the same barbs since all names except x are restricted. The only possible way for these process to interact with the environment is by a communication on x. If the second process receives on x it can only perform a synchronization on u, and becomes the encoding of the irreducible term $f(\vec{M})|_\omega$, which exactly equals the left-hand side. Also note that all M_i are irreducible, and that all names are restricted. Therefore all subterms except $[\![f(\vec{M})|_\omega]\!]_u$ are barbed congruent to $\mathbf{0}$. Hence they are barbed congruent, so in this case equivalence 6.14 follows by taking $C = C'$.

The case when $R = f(\vec{z})$ is similar to the above case noting that when $R = f(\vec{z})$ we have to reduce C in order to arrive at the irreducible form encoding of f. This encoding is immediately obtained from the left-hand side of Equation (6.14).

Induction hypothesis: Let $R = g(s_1, \ldots, s_r)$, and assume

$$(\nu\vec{y}, \vec{u})(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid [\![\mathscr{R}]\!] \mid \mathcal{I}[\![s_i]\!]_x\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\vec{\omega}})/\vec{z}\}) \xrightarrow{\tau}{}^{*} \approx [\![s_i\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}]\!]_x \mid [\![\mathscr{R}]\!]$$

holds for all subterms s_i of R.

Induction step: Consider the left-hand side of (6.14).

$$(\nu\vec{k})\big([\![g(\vec{k})]\!]_x \mid \prod_{i=1}^{r}[\![s_i\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}]\!]_{k_i}\big) \mid [\![\mathscr{R}]\!].$$

The right-hand side of (6.13) now equals.

$$(\nu\vec{y}, \vec{u})\Big(\prod_{i=1}^{n}[\![M_i]\!]_{y_i} \mid (\nu\vec{l})\big(([\![\{g(\vec{l})/x\}]\!] \mid \prod_{i=1}^{r}[\![\{\!\{s_i/l_i\}\!\}]\!]) \{\mathrm{ref}(f(\vec{M})|_{\omega_{\vec{m}}})/z_{\vec{m}}\}\big) \mid [\![\mathscr{R}]\!]\Big), \quad (6.15)$$

Now apply the induction hypothesis to each $[\![\{\!\{s_i/l_i\}\!\}]\!])\{\mathrm{ref}(f(\vec{M})|_{\omega_{\vec{m}}})/z_{\vec{m}}\}$. Then (6.15) reduces to a process which is weakly barbed congruent to

$$(\nu\vec{u})\Big((\nu\vec{l})([\![\{g(\vec{l})/x\}]\!] \mid \prod_{i=1}^{r}[\![s_i\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}]\!]_{l_i}) \mid [\![\mathscr{R}]\!]\Big)$$
$$\xrightarrow{\tau}{}^{*} \approx (\nu\vec{u})\Big((\nu\vec{l})[\![g(\vec{s})\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}]\!]_x \mid [\![\mathscr{R}]\!]\Big),$$

since N is irreducible. Furthermore, all subterms of M_i that are not being referenced by variables in $f(\vec{M})|_{\omega_{\vec{m}}}$ are barbed congruent to $\mathbf{0}$. Hence the right hand side of (6.13) weakly reduces to something which is weakly barbed congruent to the left hand side of (6.14).

The second case when N is not irreducible is similar to the first case, so we only give a rough sketch of the proof. Instead of Equivalence (6.14), we show that

$$[\![\{\!\{\sigma(R)/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx C.$$

This is enough by Lemma 6.7.10. Again we may assume that

$$\sigma(R) = R\{f(\vec{M})|_{\omega_{m_1}}, \ldots, f(\vec{M})|_{\omega_{m_l}}/z_{m_1}, \ldots, z_{m_l}\}.$$

We proceed by induction in R.

Induction basis: Suppose $R = g(\vec{y})$. Then

$$\begin{aligned}[\![\{\!\{\sigma(R)/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] &= [\![\{\!\{g(\vec{y})\{f(\vec{M})|_{\vec{\omega}}/\vec{y}\}/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \\ \xrightarrow{\tau}{}^{*} &\approx (\nu\vec{k})([\![\{g(\vec{k})/x\}]\!] \mid \prod_{i=1}^{m_l} [\![f(\vec{M})|_{\omega_i}]\!]_{k_i} \mid [\![\mathscr{R}]\!]),\end{aligned}$$

as $\vec{M}$ are irreducible. Now as the process C equals

$$(\nu\vec{y},\vec{u})(\prod_{i=1}^{n} [\![M_i]\!]_{y_i} \mid [\![\{g(\vec{y})/x\}]\!]\{\mathrm{ref}(f(\vec{M})|_{\vec{\omega}}/\vec{y})\}),$$

the conclusion again follows as all subterms not referenced by $\vec{u}$ are weakly barbed congruent to $\mathbf{0}$.

Induction hypothesis: Let $R = g(s_1, \ldots, s_r)$, and suppose the claim holds for $s_1, \ldots, s_r$.

Induction step: We get

$$\begin{aligned}&[\![\{\!\{\sigma(R)/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \\ &\qquad = (\nu\vec{k})\big([\![\{g(\vec{k})/x\}]\!] \mid \prod_{i=1}^{r} [\![\{\!\{s_i\{f(\vec{M})|_{\omega_{\vec{m}}}/z_{\vec{m}}\}/k_i\}\!\}]\!] \mid [\![\mathscr{R}]\!]\big),\end{aligned}$$

which can be seen to be weakly barbed congruent to C by repeatedly using the induction hypothesis since

$$\begin{aligned}&\mathcal{I}[\![R]\!]\{\mathrm{ref}([\![f(\vec{M})|_{\omega_{\vec{m}}}]\!]_x)/\vec{z}\} \\ &\qquad = (\nu\vec{k})\big([\![\{g(\vec{k})/x\}]\!] \mid \prod_{i=1}^{r} [\![\{\!\{s_i/k_i\}\!\}]\!]\{\mathrm{ref}([\![f(\vec{M})]\!]_x|_{\omega_{\vec{m}}})/\vec{z}_{\vec{m}}\}.\end{aligned}$$

■

At last we have the following proposition which states how reductions in the rewrite system can be matched.

Proposition 6.7.12
Let $\mathscr{R}$ be a rewrite system and let $M \in T_\Sigma(X)$, where X is the set of Appπ-names. If $M \rightsquigarrow^*_{\mathscr{R}} N$, and N is irreducible, then

$$[\![\{\!\{M/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx [\![N]\!]_x \mid [\![\mathscr{R}]\!].$$

Proof
Suppose the reduction has length l, i.e.

$$M = K_0 \rightsquigarrow^*_{\mathscr{R}} \cdots \rightsquigarrow^*_{\mathscr{R}} K_l = N.$$

If $l = 1$, so $K_1 = N$ we consider two cases. The first case applies when $\omega = 0$ and the second case applies when $\omega = i\omega'$ for some ω' and $i > 0$.

If $\omega = 0$ then L must have the syntactic form $f(\vec{t})$, $|\vec{t}| = n$ and for some σ and $(L \longrightarrow R, \zeta) \in \mathscr{R}$ we have $\sigma(L) = f(\vec{M})$. Since $\vec{M}$ is irreducible, the theorem follows by Lemma 6.7.11.

Suppose now that $\omega = i\omega'$ for some $\omega' \neq \varepsilon$. Now $M_i|_\omega$ must be of the form $\widehat{f}(\vec{N})$ for some $\vec{N}$ which is irreducible. Suppose $\mathrm{ref}([\![M]\!]_x|_\omega) = y$, then the process $[\![\{\!\{\widehat{f}(\vec{N})/y\}\!\}]\!]$ must occur in $[\![\{\!\{M/x\}\!\}]\!]$, i.e.

$$[\![\{\!\{M/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] = \cdots \mid [\![\{\!\{\widehat{f}(\vec{N})/y\}\!\}]\!] \mid \cdots \mid [\![\mathscr{R}]\!].$$

Now let $(L' \longrightarrow R', \zeta) \in \mathscr{R}$ and σ be such that $\widehat{f}(\vec{N}) = \sigma(L')$, and let $S = \sigma(R')$. Since $\vec{N}$ are irreducible we can apply Lemma 6.7.11 to infer the transition.

$$[\![\{\!\{\widehat{f}(\vec{N})/y\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx [\![S]\!]_y \mid [\![\mathscr{R}]\!], \tag{6.16}$$

By using (Res) and (Par) we can from the right-hand side reconstruct the process $\cdots \mid [\![\{\!\{\widehat{f}(\vec{N})/y\}\!\}]\!] \mid \cdots \mid [\![\mathscr{R}]\!]$. Applying the same constructs to the left-hand side and using the congruence property of $\approx$ we get

$$\cdots \mid [\![\{\!\{\widehat{f}(\vec{N})/y\}\!\}]\!] \mid \cdots \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx \cdots \mid [\![S]\!]_y \cdots \mid [\![\mathscr{R}]\!].$$

Since all encoded terms on the right-hand side are now irreducible, and $K_1 = K_0\{\sigma(R')/K_0|_\omega\}$ we know by Lemma 6.7.9 that the right hand side reduces to something which is weakly barbed congruent to $[\![N]\!]_x$.

If $l > 1$ we argue as follows. Instead of the reduction in Equation (6.16), we get by Lemma 6.7.11, the reduction.

$$[\![\{\!\{f(\vec{M})/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx K \quad \text{and} \quad [\![\{\!\{M_1/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx K. \tag{6.17}$$

where K is as in Lemma 6.7.11. Suppose M_2 is irreducible. We can repeat the argument for $l = 1$ and this together with Lemma 6.7.11 yields the following reduction:

$$[\![\{\!\{M_1/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* K \xrightarrow{\tau}{}^* \approx [\![M_2]\!]_x \mid [\![\mathscr{R}]\!].$$

This implies that

$$[\![\{\!\{f(\vec{M})/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^* \approx [\![M_2]\!]_x \mid [\![\mathscr{R}]\!],$$

which is what we wanted. If the first reduction occurs at some sub-term of $f(\vec{M})$ we can argue similarly to the case when $l = 1$ to establish (6.17). The remaining parts of the proof then follows by exactly the same reasoning as above. ■

The next four lemmas shows that the encoding is sound with respect to each axiom in Definition 3.2.2 of the structural equivalence relation for Appπ.

Lemma 6.7.13
Let $A, B \in \mathcal{A}$ If $A \equiv B$ without use of (SC-Alias), (SC-Subst) *and* (SC-Rewrite) *then $[\![\{\!|A|\!\}]\!] \equiv [\![\{\!|B|\!\}]\!]$.*

Proof
We go through each axiom of Definition 3.2.2. However, as the axioms we consider all have a counterpart in the π-calculus and because the encoding is homomorphic on parallel composition, restriction and replication all cases are relatively easy to prove. Therefore we only go through two of these axioms.

(SC-ParCommute) Let $A = A_1 \mid A_2$. Then $B = A_2 \mid A_1$ and for the encoding we get

$$[\![\{\!|A_1 \mid A_2|\!\}]\!] = [\![\{\!|A_1|\!\}]\!] \mid [\![\{\!|A_2|\!\}]\!] \equiv [\![\{\!|A_2|\!\}]\!] \mid [\![\{\!|A_1|\!\}]\!] = [\![\{\!|A_2 \mid A_1|\!\}]\!].$$

(SC-ParInact) Let $A = A_1 \mid \mathbf{0}$. Then $B = A_1$ and from the encoding we get

$$[\![\{\!|A_1 \mid \mathbf{0}|\!\}]\!] = [\![\{\!|A_1|\!\}]\!] \mid [\![\{\!|\mathbf{0}|\!\}]\!] = [\![\{\!|A_1|\!\}]\!] \mid \mathbf{0} \equiv [\![\{\!|A_1|\!\}]\!] = [\![\{\!|B|\!\}]\!].$$

■

Lemma 6.7.14
Let $A \in \mathcal{A}$ and let M be an Appπ-term. Then

$$[\![\{\!|A \mid \{M/x\}|\!\}]\!] = [\![\{\!|A\{M/x\} \mid \{M/x\}|\!\}]\!].$$

Proof
Simply use the definition of the $\{\!|\,|\!\}$ encoding. The right hand side of the equation in the lemma equals

$$\begin{aligned}[\![\{\!|A\{M/x\} \mid \{M/x\}|\!\}]\!] &= [\![\{\!|A\{x/x\} \mid \{M/x\}|\!\}]\!] \\ &= [\![\{\!|A \mid \{M/x\}|\!\}]\!].\end{aligned}$$

■

Lemma 6.7.15
The encoding is sound with respect to (SC-Alias). *I.e.*

$$[\![\{\!|(\nu x)\{M/x\}|\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx (\nu x)([\![N]\!]_x) \mid [\![\mathscr{R}]\!] \approx [\![\mathscr{R}]\!],$$

where N is the irreducible term obtained by reducing M.

Proof
The lemma follows almost immediately from Proposition 6.7.12. We simply use that the encoding is a homomorphism on restriction and then by the (Res) rule we can infer the same transition as stated in Proposition 6.7.12. From the encoding of $[\![N]\!]_x$ it is also immediate that $(\nu x)[\![N]\!]_x \approx \mathbf{0}$. ■

Lemma 6.7.16
The encoding is sound with respect to (SC-Rewrite). *I.e. if $\mathscr{R}$ is a rewrite system and M_1 and M_2 two terms such that $\Phi \vdash_\Sigma M_1 = M_2$ then*

$$[\![\{\!\{M_1/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx [\![N]\!]_x \mid [\![\mathscr{R}]\!] \quad \textit{and}$$
$$[\![\{\!\{M_2/x\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx [\![N]\!]_x \mid [\![\mathscr{R}]\!],$$

where N is the irreducible term obtained by reducing M_1 and M_2.

Proof
If $\Phi \vdash_\Sigma M_1 = M_2$ then M_1 and M_2 have the same irreducible form and we simply apply Proposition 6.7.12 to obtain the statement in the lemma. ■

We are now able to state and prove the main theorem of this section, namely that the encoding is operationally sound.

Theorem 6.7.17
Let P be an Appπ-process. If there exists Q such that $P \xrightarrow{\tau} Q$, then

$$[\![\{\!\{P\}\!\}]\!] \mid [\![\mathscr{R}]\!] \xrightarrow{\tau}{}^{*} \approx [\![\{\!\{Q\}\!\}]\!] \mid [\![\mathscr{R}]\!].$$

Proof
Since $P \xrightarrow{\tau} Q$ there is a derivation tree with this transition at the root, and containing one of the rules (Comm), (Then) or (Else) at distance d from the root. The proof is by induction in d and for the induction basis we examine the rules (Comm), (Then) or (Else).

Comm: If it is the case that the (Comm) rule is at the root of the derivation tree then

$$(\textsc{Comm})\frac{}{\overline{a}\langle y\rangle.P \mid a(x).Q \xrightarrow{\tau} P \mid Q\{y/x\}},$$

and then we have

$$\begin{aligned}
& [\![\{\!\{\overline{a}\langle y\rangle.P \mid a(x).Q\}\!\}]\!] \mid [\![\mathscr{R}]\!] \\
\equiv\ & (\nu z_1, z_2)\big([\![\{a/z_1\}]\!] \mid \overline{z}_1\langle \mathbf{send}, y, \nu r\rangle.r.[\![\{\!\{P\}\!\}]\!] \mid \\
& \quad [\![\{a/z_2\}]\!] \mid \overline{z}_2\langle \mathbf{receive}, \bot, \nu r'\rangle.r(x).[\![\{\!\{Q\}\!\}]\!]\big) \mid [\![\mathscr{R}]\!] \\
\xrightarrow{\tau}{}^{*}\ & (\nu z_1, z_2, r, r')\big([\![\{a/z_1\}]\!] \mid [\![\{\!\{P\}\!\}]\!] \mid \\
& \quad [\![\{a/z_2\}]\!] \mid [\![\{\!\{Q\}\!\}]\!]\{y/x\}\big) \mid [\![\mathscr{R}]\!] \\
\equiv\ & [\![\{\!\{P \mid Q\{y/x\}\}\!\}]\!] \mid [\![\mathscr{R}]\!]
\end{aligned}$$

Then: If

$$(\textsc{Then})\frac{}{\text{if } M = M \text{ then } P \text{ else } Q \xrightarrow{\tau} P}$$

is the root of the derivation tree then

$$\begin{aligned}
& [\![\{\!|\text{if } M = M \text{ then } P \text{ else } Q|\!\}]\!] \mid [\![\mathscr{R}]\!] \\
\equiv\ & (\nu z_1, z_2)\big([\![\{\!|\{M/z_1\}|\!\}]\!] \mid [\![\{\!|\{M/z_2\}|\!\}]\!] \mid \overline{z}_1\langle \mathbf{match}, z_2, \nu r\rangle.r(m). \\
& \quad ([m = \top][\![\{\!|P|\!\}]\!] + [m = \bot][\![\{\!|Q|\!\}]\!])\big) \mid [\![\mathscr{R}]\!] \\
\xrightarrow{\tau}{}^{*} \approx\ & (\nu z_1, z_2)\big([\![N]\!]_{z_1} \mid [\![N]\!]_{z_2} \mid \overline{z}_1\langle \mathbf{match}, z_2, \nu r\rangle.r(m). \\
& \quad ([m = \top][\![\{\!|P|\!\}]\!] + [m = \bot][\![\{\!|Q|\!\}]\!])\big) \mid [\![\mathscr{R}]\!] \\
\xrightarrow{\tau}{}^{*}\ & (\nu z_1, z_2)\big([\![N]\!]_{z_1} \mid [\![N]\!]_{z_2} \mid [\![\{\!|P|\!\}]\!]\big) \mid [\![\mathscr{R}]\!] \\
\approx\ & [\![\{\!|P|\!\}]\!] \mid [\![\mathscr{R}]\!]
\end{aligned}$$

where N is the irreducible form of M and Proposition 6.7.12 and Lemma 6.7.1 were used.

Else: This case is almost identical to the previous case except Lemma 6.7.2 is used instead of Lemma 6.7.1.

For the induction step we are required to check the rules (Res) and (Par) which is just about trivial as the encoding is homomorphic with respect to both restriction and composition and the rules for restriction and composition are similar in the π-calculus. The (Struct) rule follows by the Lemmas 6.7.13, 6.7.14, 6.7.15 and 6.7.16 as we get that the encoding of structurally equivalent processes either are structurally congruent, weakly barbed congruent or reduce to processes which are weakly barbed congruent. ■

The last theorem states that the encoding preserves barbs.

Theorem 6.7.18
Let $A \in \mathcal{A}$ and assume that $A \downarrow_a$. Then $[\![\{\!|A|\!\}]\!] \Downarrow_a$.

Proof
By structural induction in A. Assume $A = \overline{a}\langle M\rangle.P$.

$$\begin{aligned}
[\![\{\!|\overline{a}\langle M\rangle.P|\!\}]\!] &= (\nu x, y)([\![\{a/y\}]\!] \mid [\![\{M/x\}]\!] \mid [\![\{\!|\overline{y}\langle x\rangle.P|\!\}]\!]) \\
&= (\nu x, y)([\![\{a/y\}]\!] \mid [\![\{M/x\}]\!] \mid \overline{y}\langle \mathbf{send}, x, \nu r\rangle.r.[\![\{\!|P|\!\}]\!] \\
&\xrightarrow{\tau}{}^{*} (\nu x, y)([\![\{a/y\}]\!] \mid [\![\{M/x\}]\!] \mid \overline{a}\langle x\rangle.\overline{r} \mid \overline{r}.[\![\{\!|P|\!\}]\!]) \downarrow_{\overline{a}} .
\end{aligned}$$

Next assume $A = (\nu u)A'$. Now if $A' \xrightarrow{\alpha}$ where $\mathrm{subj}(\alpha) = a$ and $u \notin \mathrm{v}(\alpha) \cup \mathrm{n}(\alpha)$, we apply the induction hypothesis and the (Res) rule. Otherwise, if $u \in \mathrm{v}(\alpha) \cup \mathrm{n}(\alpha)$ and $u \neq a$ we apply the induction hypothesis and the (Open) rule to obtain the conclusion.

The cases where A is a parallel composition, replication or a conditional are trivial and are omitted. ■

Chapter 7

Conclusion

In this master thesis various aspects pertaining to Appπ have investigated. We first recapitulate and discuss some of the most important results obtained and finally discuss possible future work.

7.1 Results

Appπ was introduced by Abadi and Fournet in [AF01]. The original article leaves a number of aspects open to further investigation. We have chosen to concentrate on a few of these aspects. The major contributions of this master thesis are the following

- Definition of type systems for Appπ.
- A representation of the spi-calculus in Appπ.
- An encoding of Appπ in the π-calculus.

7.1.1 Type systems and representation of the spi-calculus.

A type system for the π-calculus has been defined by Sangiorgi and Walker in [SW01], but many others exist. We have established a basic type system for Appπ similar to the one in [SW01], capable of capturing runtime errors such as arity mismatch and erroneous use of names. A subject-reduction theorem which states that transitions preserves typing has been established for this type system.

As suggested in [AF01] the development of Appπ has been inspired by the spi-calculus [AG97]. We have formalised this correspondence by establishing a representation of the spi-calculus in an instantiation of Appπ. That is, an instance of Appπ has been created by defining a presentation Π, and an encoding of spi-processes in this language has been defined.

This encoding is shown to be fully abstract with respect to operational correspondence. In order to facilitate the proof of this result, the semantics of the

spi-calculus has been tailored to be conceptually as close to the semantics of Appπ as possible. Thus minor changes have been made compared to the semantics originally given in [AG97]. Specifically, we use an early labelled transition relation with action labels in the style of $a\langle M\rangle$ for input and $(\nu q)\overline{a}\langle M\rangle$ for output. This contrasts to the semantics given in [AG97] which uses the notion of abstractions and concretions. Furthermore, scope extrusion is handled differently in the original presentation. In order to ensure that our results holds also for the original spi-calculus, we have proved that the modified semantics is strongly equivalent to that of the original.

Considerable effort has been put into investigating the use of type systems for verification of properties related to securety issues in the spi-calculus. In [Aba97] a type system which ensures secrecy in the spi-calculus is given. In light of our encoding of the spi-calculus in Appπ we have established a type system for Appπ similar to the one in [Aba97]. We have proven that the encoding of the spi-calculus in Appπ is sound and complete with respect to well-typedness. Thus our encoding captures the spi-calculus in a quite strong sense since the encoding preserves transitions as well as types.

7.1.2 Encoding of Appπ in the π-calculus

The final major and most innovative contribution of this master thesis is an encoding of Appπ in the π-calculus. Such an encoding is appealing for several reasons. It opens the doors for the possibility of applying the vast amount of theoretical tools developed for the π-calculus when reasoning in Appπ. Furthermore a number of tools for automated analysis of π-processes have been developed. By using the encoding form Appπ to the π-calculus these can also be utilised for analysis of Appπ-processes.

Although the π-calculus is well-known to be Turing-complete, the existence of such an encoding is not immediate, and indeed the encoding is non-trivial. The main difficulty lies in the encoding of the equational theory which is used for comparing terms in Appπ. Our encoding is inspired by a recent proposal for an encoding of the spi-calculus in the π-calculus [BPV03], but differs considerably in the way that terms are encoded. We rely on well-known results for transforming an equational theory into a terminating and confluent rewrite system. In fact this transformation can be done by using the Knuth-Bendix procedure [KB70]. Moreover the encoding requires the rules of the rewrite system to be left linear.

The general idea is that all terms are forced to reduce to their irreducible form before they can be accessed by the remaining parts of the system. This is done by initially encoding the terms in paralell with the encoding of the rewrite system which reduces the terms until they become irreducible. When on irreducible form the term becomes an object-like process with a number of predefined methods, including methods for comparing syntactically against another term. Since the rewrite system is terminating and confluent checking for equality then becomes a matter of checking for syntactic identity. The encoding also imposes a bottom-up evaluation strategy of terms. The choice of evaluation strategy is inessential for confluent rewrite systems. We prove soundness of this encoding with respect to operational correspondence.

7.2 Future work

As is often the case the amount of work that needs to be done has exceeded the amount of work that we were able perform within the given period of time. Thus there are a number of open issues which could be further investigated. Some of these issues are outlined in this section.

- Further investigation of different type systems for Appπ.
- Full abstraction for the encoding of the spi-calculus in Appπ with respect to some suitable equivalence.
- Further development of the theory of the encoding of Appπ in the π-calculus.

7.2.1 Type systems

The general type system we have defined for Appπ is quite basic. More advanced type systems for the π-calculus have been developed, and we believe that such systems can also be defined for Appπ. In the style of [SW01], a possible extension is to introduce recursive types which make it possible to type processes such as $\overline{a}\langle a\rangle.\mathbf{0}$. More generally recursive types are essential in order to avoid loosing expressive power in the typed version of Appπ compared to the untyped. Another possible extension is to introduce input/output types which separate names that can be used for input from those that can be used for output. Such a refinement of the channel types is useful for several reasons. One example of its use is in the programming language Pict [PT97], which is a typed programming language based on the π-calculus. The argument for including input/output types in Pict is that in practice it turns out that it is rare that a channel is used for both input and output. Hence if a channel is being used for both purposes, it is fair to assume that this is due to a programming error, which the type system should be able to catch.

Input/output types are certainly also useful also from a theoretical point of view. The arch example of this in the setting of the π-calculus is maybe to be found in Pierce and Sangiorgis treatment [PS93] of the more efficient of Milners two encodings of the call-by-value λ-calculus in the π-calculus [Mil90]. In this encoding, $\mathcal{V}[\![\cdot]\!]$, β-reduction is not valid, i.e. $\mathcal{V}[\![(\lambda x.M)(\lambda y.N)]\!]$ is not equivalent to $\mathcal{V}[\![M\{\lambda y.N/x\}]\!]$. Pierce and Sangiorgi observes in [PS93] that the reason for this is that certain names can be wrongly used relative to the intention of the encoding. By utilizing the extra refinement obtained by the separation of input and output names, validity of β-reduction is obtained.

Gordon and Jeffrey have implemented a type checker for cryptographic protocols [GJ] based on the type system for the spi-calculus presented in [GJ01a]. They are currently not able to handle protocols based on hashing. Since hash-functions can easily be represented in Appπ by simply defining a function h with no equations, is should be possible to capture such protocols in Appπ. In order to perform the analysis suitable type systems for Appπ will have to be developed.

7.2.2 Full abstraction with respect to equivalence

As mentioned in the previous section full abstraction with respect to operational correspondence has been established for the encoding of the spi-calculus in Appπ. In the light of this we believe that this result can be extended to obtain full abstraction with respect to weak barbed congruence.

7.2.3 Further development of the encoding of Appπ.

The encoding of Appπ in the π-calculus leaves a number of issues open to further development. We have only established operational soundness. Whether the encoding is also complete with respect to operational correspondence is left as an open question to which the answer is far from trivial. In [BPV03] Baldamus, Parrow, and Victor prove that their encoding of the spi-calculus in the π-calculus preserves may-testing. Certainly the most difficult part in the proof of this result is to relate operational steps of an encoded process with that of the original spi-process. As is the case in [BPV03], we anticipate that also for the present encoding, completeness is significantly harder to obtain. The reasons for this are manifold. In general an encoded Appπ-processes will have several reductions where there are only zero or one reduction in the original Appπ-process. These reductions are mainly introduced by the encoding of terms, but also the manner in which input and output are encoded causes reductions which are not directly related to reductions in the source process.

The encoding of terms, in particular functions, is probably the most difficult part to handle. The encoding of terms ensures that an encoded term does not become available to other parts of the system before it has reduced to its irreducible form. However, even for terms that are irreducible the encoding of the term is not equivalent to the irreducible form encoding since the initial coding has a barb cf. Lemma 6.7.9 on page 76, namely the principal function name of the term. This issue could possibly be circumvented by imposing a stricter form of control of the scope of the function names introduced by the encoding. That is, given a term M with principal function name f, the scope of f in the encoded process need only include the term constructors containing its co-name. Another possibility could be to modify the notion of completeness so completeness is only required for the irreducible form of terms. Clearly this can not be done in general since one initially may not know the argument values of a given term. However, since the rewrite system is terminating whenever a term is being used its normal form can be found. Hence such an approach could also be feasible.

Finally we mention an issue related to the discussion in the previous section. The encoding makes the substantial amount of theoretical and practical tools developed for the π-calculus available to Appπ. However, before this can be utilised to its fullest potential, the encoding, and in particular its theoretical foundation needs to be further matured. In particular it would be useful to know that the encoding preserves some equivalence. Assuming that this can be obtained, either as discussed in the previous paragraph, or in some other way, an interesting project could be to implement a compiler based on the encoding which translates Appπ-processes written in some specification language into for example Mobility Workbench readable input. This has been done with the

translation of spi-processes into π-processes.

Appendix A

Universal Algebra

In this chapter we present the fragment of the theory of universal algebra that is relevant for our investigations of the Appπ-calculus. The presentation is based on [ST99] and [Joh87] but with slight alterations.

A.1 Signatures

A signature (often called an operational type in the literature) is a structure containing two elements: A set of function names and a function assigning arities to the functions names. The function names have no structure and are *not* actual functions. The role of the arity function is to assign an integral arity to each of the function names. The signature is a syntactic structure which can be given a semantics in the form of an algebra. We shall, however, not need this kind of semantics.

The formal definition of a signature is as follows.

Definition A.1.1 (Signature)
A signature Σ is a pair (Ω, α), where Ω is a set of function names and $\alpha : \Omega \to \mathbb{N}_0$ is an arity function.

Let $\Sigma = (\Omega, \alpha)$ be a signature. The concrete operation names of Ω are written in sans serif font, for example $\mathsf{multiply}$. If $f \in \Omega$ and $\alpha(f) = 0$ we say that f is a constant. Overloading is not formally permitted, since a function name has exactly one arity. This is not a restriction in our setting as we can consider an overloaded function name as several distinct function names generated by decorating the original function name with each of its arities.

A.2 Terms

Given a signature $\Sigma = (\Omega, \alpha)$ we can construct terms from the function symbols and some set of variables.

Definition A.2.1 (Terms and equations)
Let $\Sigma = (\Omega, \alpha)$ be a signature and let X be an set of variables disjoint from Ω. Then the set of terms over Σ with variables in X, denoted by $T_\Sigma(X)$, is the least set satisfying

(i) if $x \in X$ then $x \in T_\Sigma(X)$, and

(ii) if $f \in \Omega$ and $t_1, \ldots, t_{\alpha(f)} \in T_\Sigma(X)$ then $f(t_1, \ldots, t_{\alpha(f)}) \in T_\Sigma(X)$.

For some $t \in T_\Sigma(X)$ we let $\mathrm{v}(t)$ denote the set of variables occurring in t. A Σ-equation with variables in X is an equation of the form $t = t'$ where $t, t' \in T_\Sigma(X)$.

It is necessary that the set of variables X be disjoint from Σ to avoid confusing variables with function names. The set of variables can be empty, finite of infinite. Note that if Σ has no constants and $X = \emptyset$ then $T_\Sigma(X) = \emptyset$. A term which contains no variables is called a ground term.

A.3 Provable equations

We define an axiomatic system in which equations of a signature are either provable or not.

Definition A.3.1
Let $\Sigma = (\Omega, \alpha)$ be a signature, X a set of variables and let Φ be a set of Σ-equations with variables in X. A Σ-equation φ with variables in X is a provable consequence of Φ, written $\Phi \vdash_\Sigma \varphi$ if φ can be proven from the following rules and axioms.

$$(\textsc{Axiom})\ \frac{}{\Phi \vdash_\Sigma \eta} \quad \eta \in \Phi$$

$$(\textsc{Reflexivity})\ \frac{}{\Phi \vdash_\Sigma t = t} \quad t \in T_\Sigma(X)$$

$$(\textsc{Symmetry})\ \frac{\Phi \vdash_\Sigma t = t'}{\Phi \vdash_\Sigma t' = t}$$

$$(\textsc{Transitivity})\ \frac{\Phi \vdash_\Sigma t = \widehat{t} \qquad \Phi \vdash_\Sigma \widehat{t} = t'}{\Phi \vdash_\Sigma t = t'}$$

$$(\textsc{Congruence})\ \frac{\Phi \vdash_\Sigma t = t'}{\Phi \vdash_\Sigma s\{t/x\} = s\{t'/x\}} \quad x \in X \text{ and } s \in T_\Sigma(X)$$

$$(\textsc{Instantiation})\ \frac{\Phi \vdash_\Sigma t = t'}{\Phi \vdash_\Sigma t\sigma = t'\sigma} \quad \sigma : X \to T_\Sigma(X)$$

Given a signature and a set of equations over this signature we define an equational theory, or presentation as follows.

Definition A.3.2 (Presentation)
A presentation is a pair (Σ, Φ) *where* Σ *is a signature and* Φ *is a set of* Σ*-equations.*

A.4 Examples

A.4.1 Pairs

Define a signature $\Sigma = (\Omega, \alpha)$ where

$$\begin{aligned} \Omega &= \{\mathsf{pair}, \mathsf{first}, \mathsf{second}\} \\ \alpha(\mathsf{pair}) &= 2 \\ \alpha(\mathsf{first}) &= \alpha(\mathsf{second}) = 1. \end{aligned}$$

With the the set of equations

$$\begin{aligned} \Phi = \{&\mathsf{first}(\mathsf{pair}(x, y)) = x, \\ &\mathsf{second}(\mathsf{pair}(x, y)) = y\}, \end{aligned}$$

we can prove

$$\begin{aligned} \Phi &\vdash_\Sigma \mathsf{first}(\mathsf{first}(\mathsf{pair}(\mathsf{pair}(x, y), z))) = x \\ \Phi &\vdash_\Sigma \mathsf{pair}(\mathsf{second}(\mathsf{pair}(x, y)), z) = \mathsf{pair}(y, z), \end{aligned}$$

and we expect that

$$\Phi \nvdash_\Sigma \mathsf{first}(x) = \mathsf{second}(x)$$

though proving that some equation is unprovable is usually not straightforward. It may involve defining an algebra or an equivalent rewrite system.

A.4.2 Cryptography

Abstract cryptographic functionality can be described by the signature $\Sigma = (\Omega, \alpha)$ where

$$\begin{aligned} \Omega &= \{\mathsf{enc}, \mathsf{dec}, \mathsf{aenc}, \mathsf{adec}, \mathsf{pkey}, \mathsf{skey}, \mathsf{hash}\} \\ \alpha(\mathsf{enc}) = \alpha(\mathsf{dec}) &= \alpha(\mathsf{aenc}) = \alpha(\mathsf{adec}) = 2 \\ \alpha(\mathsf{pkey}) = \alpha(\mathsf{skey}) &= \alpha(\mathsf{hash}) = 1. \end{aligned}$$

Now, with the set of equations

$$\begin{aligned} \Phi = \{&\mathsf{dec}(\mathsf{enc}(x, y), y) = x, \\ &\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pkey}(y)), \mathsf{skey}(y)) = x, \\ &\mathsf{adec}(\mathsf{aenc}(x, \mathsf{skey}(y)), \mathsf{pkey}(y)) = x\} \end{aligned}$$

we have an abstract description of the cryptographic primitives for both symmetric and asymmetric encryption and decryption as well as hashing. There are no equations involving hash, which indicates that the hash of a message cannot be equal to the hash of some other message and that the message cannot be obtained from the hash. Similarly, pkey and skey have no equations associated with them, so it is not possible to create the same key from two distinct seeds and the seed cannot be extracted from the key.

Appendix B

Rewrite Systems

An operational approach to deciding whether two terms M and N are provable from a set Φ of equations is given by the concept of rewrite systems. The idea is basically to interpret each equation $R = L \in \Phi$ as a directed rule $R \longrightarrow L$ and from these rules define a relation $\longrightarrow$ on the set of terms. If the rewrite system is confluent, two terms M and N are equal if they can be reduced to the same term. If in addition the rewrite system is terminating one can prove inequality. These ideas are formalised in the next sections. We only include material necessary for our purpose. More complete explanations can be found in [Mit96] and [KK01] in which further references can also be found.

B.1 Basic definitions

Before we define rewrite systems and look at some of their properties we need some tools to work with. The first definition formalizes the idea of a substitution. The elements from universal algebra used in this appendix can be found in Appendix A.

Definition B.1.1 (Substitution)
A substitution is a mapping from variables to terms, $\sigma : X \longrightarrow T_\Sigma(X)$ which is the identity for all but a finite number of variables. The domain of σ is extended to $T_\Sigma(X)$ by defining $\sigma(f(M_1, \ldots, M_n)) = f(\sigma(M_1), \ldots, \sigma(M_n))$.

Given a term M we need to be able to refer to a specific subterm of M. This is achieved by considering the parse tree for the term. Given a term M the parse tree for M is a finite labelled tree with leaves labelled by variables and constants and the internal nodes labelled with function names with arity greater than zero. The outgoing edges of each nodes are numbered from left to right.

Definition B.1.2 (Position)
Let M be a term and T the parse tree for M. A position T is a finite sequence $\omega = n_1 \ldots n_k$, where $n_i \in \mathbb{N}$. The position ω identifies the node found by traversing the tree from the root node and for each node at level i following the edge

numbered n_i. The subterm of M whose parse tree is rooted at ω is denoted $M|_\omega$.

If M is a term then we denote the set of positions of M by P_M.

As usual if $>$ is a binary relation, $>^*$ denotes the reflexive transitive closure of $>$.

Definition B.1.3 (Termination, confluence and irreducible terms)
Let $>$ be a binary relation on some set T.

- *The relation $>$ is terminating if there is no infinite sequence s such that $s_i > s_{i+1}$ for all $i \in \mathbb{N}$.*
- *An element t is said to be irreducible if for all t', it holds that $t \not> t'$.*
- *The relation $>^*$ is confluent if whenever $M >^* K$ and $M >^* L$, then there exists N such that $K >^* N$ and $L >^* N$.*
- *The relation $>$ is locally confluent if whenever $M > K$ and $M > L$, then there exists N such that $K >^* N$ and $L >^* N$.*

B.2 Rewrite system

A rewrite system consists of a set of rewrite rules. Syntactically these rules are formed from elements of the set $T_\Sigma(X)$.

Definition B.2.1 (Rewrite system)
A rewrite system $\mathscr{R}$ over Σ is an indexed family $\{L_i \longrightarrow R_i\}_{i\in I}$ where for all $i \in I$, $L_i, R_i \in T_\Sigma(X)$ and each $L_i \longrightarrow R_i$ fulfills the following condition:

(i) $L_i \notin X$, *and*

(ii) $\mathrm{v}(R_i) \subseteq \mathrm{v}(L_i)$.

The elements of $\mathscr{R}$ are called rewrite rules.

In general condition (i) and (ii) are not strictly necessary. However, in this chapter we wish to restrict out attention to terminating rewrite systems. The purpose of item (i) and (ii) is illustrated after we define the reduction relation induced by $\mathscr{R}$.

The idea is that the rule $L \longrightarrow R$ allows us to replace any substitution instance of L with the R under the same substitution. Given a rewrite system a reduction relation on terms is defined as follows.

Definition B.2.2 (Reduction relation)
Let $\mathscr{R}$ be a rewrite system, and let $M, N \in T_\Sigma(X)$. Then M can be rewritten to N if

- *there exists a rule $(L \longrightarrow R) \in \mathscr{R}$ and*

- *a position ω, and*
- *a substitution σ satisfying $M|_\omega = \sigma(L)$,*

such that $N = M\{\sigma(R)/M|_\omega\}$. If M can be rewritten to N we write $M \longrightarrow_{\mathscr{R}} N$.

We also need a definition of a bottom-up reduction relation. This is a sub-relation of $\longrightarrow_{\mathscr{R}}$.

Definition B.2.3 (Bottom-up reduction relation)
Let $\mathscr{R}$ be a rewrite system, and let $M, N \in T_\Sigma(X)$. Then $M \rightsquigarrow_{\mathscr{R}} N$ if

- *there exists a rule $(L \longrightarrow R) \in \mathscr{R}$ and*
- *a position ω, and*
- *a substitution σ satisfying $M|_\omega = \sigma(L)$, and*
- *$\forall \omega'$ such that $\omega\omega' \in P_M$ we have $M_{\omega\omega'} \not\longrightarrow_{\mathscr{R}}$,*

such that $N = M\{\sigma(R)/M|_\omega\}$.

Basically, the bottom-up reduction relation specifies that a term M cannot be reduced unless all subterms of M have been reduced.

Definition B.2.4
Let $(L \longrightarrow R)$ be a rule in some rewrite system $\mathscr{R}$ and M a term. Then we say that L and M have the same syntactical form if M is a substitution instance of L. I.e. if for all positions $\omega \in P_L$ such that $L|_\omega$ is not a variable we have that the principal function symbols of $L|_\omega$ and $M|_\omega$ are identical.

Definition B.2.5 (Terminating and confluent rewrite system)
A rewrite system $\mathscr{R}$ is terminating if $\longrightarrow_{\mathscr{R}}$ is terminating. A rewrite system is confluent if $\longrightarrow^*_{\mathscr{R}}$ is confluent. A rewrite system is locally confluent if $\longrightarrow_{\mathscr{R}}$ is locally confluent. A term M is irreducible if there is no M' such that $M \longrightarrow_{\mathscr{R}} M'$.

Next we prove some properties of the bottom-up reduction relation.

Lemma B.2.6
Assume that $\longrightarrow_{\mathscr{R}}$ is terminating and confluent. Then $\rightsquigarrow_{\mathscr{R}}$ is terminating and if $M \longrightarrow^*_{\mathscr{R}} N$, where N is irreducible, then also $M \rightsquigarrow^*_{\mathscr{R}} N$.

Proof
The relation $\rightsquigarrow_{\mathscr{R}}$ is obviously terminating as it is a sub-relation of a terminating relation. Now assume that $M \longrightarrow^*_{\mathscr{R}} N$ and that $M \rightsquigarrow^*_{\mathscr{R}} M' \not\rightsquigarrow_{\mathscr{R}}$ and $M' \neq N$. We have that $M' \longrightarrow^*_{\mathscr{R}} N$. Now take the sequence ω of longest length such that there exists a rule $(L \longrightarrow R) \in \mathscr{R}$ and a substitution σ such that $M_\omega = \sigma(L)$. Then $M' \rightsquigarrow_{\mathscr{R}} M'\{\sigma(R)/M'|_\omega\}$, a contradiction. This proves the lemma. ■

Corollary B.2.7
$\rightsquigarrow_{\mathscr{R}}$ is confluent.

We now briefly return to item (i) and (ii) in the definition of rewrite systems. As mentioned in the introduction we are only interested in terminating rewrite systems. Indeed, if the two conditions where not included we could not hope for this property to hold. If for instance the rule $x \longrightarrow f(x)$ where allowed, we could rewrite $x \longrightarrow_{\mathscr{R}} \longrightarrow f(x) \longrightarrow_{\mathscr{R}} f(f(x)) \longrightarrow_{\mathcal{R}} \longrightarrow_{\mathscr{R}} \cdots$. Furthermore, if we allow the rule $f(x) \longrightarrow g(y)$ we have no control over y. Specifically, we can let $\sigma(y) = f(x)$, hence $f(x) \longrightarrow_{\mathscr{R}} g(f(x)) \longrightarrow_{\mathcal{R}} g(g(f(x))) \longrightarrow_{\mathscr{R}} \cdots$. But note that this is a necessary but not sufficient condition for termination.

The main result for confluent rewrite systems is that equality between terms M and N holds exactly when they both reduce to the same term. Let $\mathscr{R}$ be a rewrite system, and $\Phi_{\mathscr{R}}$ the corresponding set of undirected equations.

Theorem B.2.8
Let $\mathscr{R}$ be a confluent rewrite system. Then $\Phi_{\mathscr{R}} \vdash_{\Sigma} M = N$ if and only if there exists a term L such that $M \longrightarrow^*_{\mathscr{R}} L$ and $N \longrightarrow^*_{\mathscr{R}} L$.

Note that if one, as in our case, wishes to use the rewrite system for proving equality in a given presentation, this can be done by simply forming the rewrite rules from the set of equations by selecting a direction for each, keeping the condition on the variables in mind. It is in general undecidable whether a rewrite system is terminating and confluent, however, there exists a number of methods for establishing these. We briefly mention some simple ways of ensuring this in the next section.

B.3 Termination and Confluence

The basic idea is to impose on the set of terms a well-founded ordering $>$ such that for a sequence of reductions $M_1 \longrightarrow_{\mathscr{R}} M_2 \longrightarrow_{\mathscr{R}} \cdots$, we have $M_1 > M_2 > \cdots$.

Definition B.3.1 (Well-founded relation)
Let $>$ be a relation on a set A. The relation $>$ is a well-founded ordering if it is reflexive, transitive, and there exists no infinite decreasing sequence $a_1 > a_2 > \cdots$ of elements of A.

Definition B.3.2 (Reduction ordering)
A reduction ordering $>$ is a well-founded ordering closed under substitution and contexts.

The main result for well founded reduction orderings is given below.

Theorem B.3.3
A reduction ordering $\mathscr{R}$ is terminating if there is a well-founded ordering such that for all $(L_i \longrightarrow R_i) \in \mathscr{R}$, we have $L_i > R_i$.

For confluence we have the following result.

Theorem B.3.4
Let $\mathscr{R}$ be a terminating rewrite system. Then $\mathscr{R}$ is confluent if an only if $\mathscr{R}$ is locally confluent.

Local confluence can also be characterised by using critical pairs. We need the following two definitions for Theorem B.3.7.

Definition B.3.5 (Overlap)
Let $\mathscr{R}$ be a rewrite system and let $(L \longrightarrow R) \in \mathscr{R}$. Furthermore, let σ be a substitution and ω a position in L. Then if $\sigma(L)|_\omega = \sigma(L')$, where $L|_\omega$ is not a variable, we say that the triple $\langle \sigma(L), \sigma(L'), \omega \rangle$ is an overlap.

Definition B.3.6 (Critical pair)
Let $\mathscr{R}$ be a rewrite system and let $(L \longrightarrow R), (L' \longrightarrow R') \in \mathscr{R}$. Let ω be a position in L and let σ be the minimal substitution such that $\langle \sigma(L), \sigma(L'), \omega \rangle$ is an overlap. Then the pair $\langle \sigma(R), \sigma(L)\{\sigma(R')/\sigma(L)|_\omega\} \rangle$ is called a critical pair.

Next we consider the problem of determining whether a rewrite system $\mathscr{R}$ is locally confluent.

Consider a term M that can be reduced in two different ways, i.e. there are two rules $L \longrightarrow R$, $L' \longrightarrow R' \in \mathscr{R}$ and a substitution σ such that $M|_\omega = \sigma(L)$ and $M|_{\omega'} = \sigma(L')$. Note that we may assume that only a single substitution is necessary as we can rename the variables of L and L' such that they have no in common. In the following we look at the possible relationships between ω and ω'.

- If neither ω or ω' is a subsequence of the other we obviously have local confluence.
- The second case occurs when either ω' is a subsequence of ω or if ω is a subsequence of ω'. Assume that ω' is a subsequence of ω. For simplicity we will also assume that $M = \sigma(L)$. This case can again be sub-divided into two depending on whether the term at position ω' in $\sigma(L)$ only contains symbols introduced by σ. If $\sigma(L)|_{\omega'}$ only contains symbols introduced by σ then we have local confluence as the reduction $\sigma(L) \longrightarrow_{\mathscr{R}_S} \sigma(R)$ either eliminates the subterm $\sigma(L)$ or we get a subterm containing one or more occurrences of $\sigma(L')$. Each of these can be reduced to $\sigma(R')$.

  In the last subcase we again have that $\sigma(L)|_{\omega'} = \sigma(L')$, but we also know that the term at $L|_{\omega'}$ is not a variable. Obviously this cannot happen unless the principal function of L' is the principal function of $L|_{\omega'}$. In this case we must examine critical pairs as the following theorem specifies.

Theorem B.3.7
A rewrite system $\mathscr{R}$ is locally confluent if and only if for all critical pairs $\langle M, M' \rangle$ there exists a term N such that $M \longrightarrow^*_{\mathscr{R}} N$ and $M' \longrightarrow^*_{\mathscr{R}} N$.

This theorem is often called the Knuth-Bendix test [Mit96, Sec. 3.7]. It is a part of an algorithm, the Knuth-Bendix completion procedure, first presented in [KB70], which transforms a finite set of identities into a terminating confluent rewrite system, if one exists. The procedure uses a well founded ordering $>$ which is used to test whether $L > R$ for each rule. If so Theorem B.3.7 is applied. However, if for a critical pair $\langle M, M' \rangle$, there do not exist N such that $M \longrightarrow^*_{\mathscr{R}} N$ and $M' \longrightarrow^*_{\mathscr{R}} N$ then either $M \longrightarrow M'$ or $M' \longrightarrow M$ is added to the rewrite system which makes sense as there is a term which reduces to both M

and M'. After this we again follow the just described procedure. The algorithm may terminate with success or failure or loop without terminating.

B.4 Left linear rewrite systems

In this section we define left linear rewrite systems and we show how a rewrite system can be transformed into an equivalent left linear rewrite system.

Definition B.4.1 (Left linear rewrite system)
A rewrite system $\mathscr{R}$ is left linear if for all rules $(L \longrightarrow R) \in \mathscr{R}$ no variable occurs twice in L.

Let $\mathscr{R}$ be a rewrite system and let $(L \longrightarrow R) \in \mathscr{R}$. Furthermore, let $\omega_i \in P_L$ and let $<$ be the lexicographical ordering. Let $z_1, \ldots, z_k$ be distinct fresh variables. Then define the substitution σ_l as $\sigma_l(L|_{\omega_i}) = z_i$.

Now define the function $\delta(x) = \min\{i \mid L|_{\omega_i} = x\}$, then $\omega_{\delta(x)}$ is the first occurrence of the variable x in L. Now define a substitution by $\sigma_r(x) = z_{\delta(x)}$.

The rules are then transformed by the mapping

$$(L \longrightarrow R) \quad \mapsto \quad (\sigma_l(L) \longrightarrow \sigma_r(R), \zeta),$$

where ζ is a sequence of sequences of variables such that for each variable $x \in L$ we have a sequence $\xi_x \in \zeta$, where ξ_x is the sequence of variables in the set $\{z_i \mid L|_{\omega_i} = x\}$.

Suppose $\mathscr{R}_A$ is a rewrite system where all $r \in \mathscr{R}_A$ are obtained from the rules in $\mathscr{R}$ by the transformation just described.

Definition B.4.2 (Alternate Reduction relation)
Let $\mathscr{R}_A$ be an alternate rewrite system, and let $M, N \in T_\Sigma(X)$. Then M can be rewritten to N if

- *there exists a rule $(L \longrightarrow R, \zeta) \in \mathscr{R}_A$ and*
- *a position ω, and*
- *a substitution σ satisfying the following:*
  - *$M|_\omega = \sigma(L)$, and*
  - *for all sequences $z_{i_1}, \ldots, z_{i_n} \in \zeta$ it holds that $\sigma(z_{i_k}) = \sigma(z_{i_l})$ for all $1 \leq k, l \leq n$.*

such that $N = M\{\sigma(R)/M|_\omega\}$. If M can be rewritten to N we write $M \longrightarrow_{\mathscr{R}_A} N$.

Appendix C

Abadi's commitment relation

We now define the commitment relation given in [Aba97]. It relies on the syntactic forms abstractions and concretions. An abstraction is an expression of the form $(x_1, \ldots, x_k).P$ where $k \geq 0$ and $x_1, \ldots, x_k$ are bound variables in P. A concretion is an expression of the form $(\nu y_1, \ldots, y_l)\langle M_1, \ldots, M_k\rangle P$, where $l, k \geq 0$ and $M_1, \ldots, M_k$ are terms and $y_1, \ldots, y_l$ are bound in $M_1, \ldots, M_k$ and P. An agent is an abstraction, concretion or process and we let $A, B, \ldots$ range over agents.

Furthermore, we define

$$\begin{aligned}(\nu y)(x_1, \ldots, x_k)P &\stackrel{\text{def}}{=} (x_1, \ldots, x_k)(\nu y)P \\ Q \mid (x_1, \ldots, x_k)P &\stackrel{\text{def}}{=} (x_1, \ldots, x_k)(Q \mid P) \\ (\nu m)(\nu \vec{n})\langle \vec{M}\rangle P &\stackrel{\text{def}}{=} (\nu m, \vec{n})\langle \vec{M}\rangle P \\ Q \mid (\nu \vec{n})\langle \vec{M}\rangle P &\stackrel{\text{def}}{=} (\nu \vec{n})\langle \vec{M}\rangle (Q \mid P),\end{aligned}$$

and the dual composition $A \mid Q$ is defined symmetrically. Now let

$$\begin{aligned}A &= (x_1, \ldots, x_k)P \\ B &= (\nu \vec{n})\langle \vec{M}\rangle Q\end{aligned}$$

Then $A@B$ and $B@A$ are defined as

$$\begin{aligned}A@B &= (\nu \vec{n})(P\{\vec{M}/\vec{x}\} \mid Q) \\ B@A &= (\nu \vec{n})(Q \mid P\{\vec{M}/\vec{x}\}).\end{aligned}$$

The reduction relation is given in Definition 2.2.1. The commitment relation is then defined as follows.

Definition C.0.3
Let P be a closed process, A a closed agent and α an action. The commitment

relation $\xrightarrow{\alpha}$ *is the relation on spi-processes that satisfies the rules*

$$\text{(Comm Out)}\ \frac{}{\overline{m}\langle\vec{M}\rangle.P \xrightarrow{\overline{m}} (\nu)\langle\vec{M}\rangle.P}$$

$$\text{(Comm In)}\ \frac{}{m(\vec{x}).P \xrightarrow{m} (\vec{x}).P}$$

$$\text{(Comm Inter 1)}\ \frac{P \xrightarrow{m} A \qquad Q \xrightarrow{\overline{m}} B}{P \mid Q \xrightarrow{\tau} A@B}$$

$$\text{(Comm Inter 2)}\ \frac{P \xrightarrow{\overline{m}} B \qquad Q \xrightarrow{m} A}{P \mid Q \xrightarrow{\tau} B@A}$$

$$\text{(Comm Par 1)}\ \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}$$

$$\text{(Comm Par 2)}\ \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}$$

$$\text{(Comm Res)}\ \frac{P \xrightarrow{\alpha} A \qquad \mathrm{n}(\alpha) \cup \mathrm{v}(\alpha) \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A}$$

$$\text{(Comm Red)}\ \frac{P > Q \qquad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

Appendix D

Proofs

The following is a proof of Lemma 2.3.1.

Proof
Proofs of cases (i) and (ii) are by induction in the height of the derivation tree of the transition.

We first prove (i) by examining each possible root of the derivation tree. We start by noting that (Input) and (Communication) are not possible roots and that proving rule (Output) is the induction basis.

Output: Assume that

$$(\textsc{Output})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}\langle\vec{M}\rangle} P'},$$

then we obtain immediately

$$(\textsc{Comm Out})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}} (\nu)\langle\vec{M}\rangle P'},$$

which satisfies the condition $P' \stackrel{\nu}{=} P'$.

Restriction: Assume that

$$(\textsc{Restriction})\frac{Q \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q'}{P = (\nu s)Q \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} (\nu s)Q' = P'} \quad s \notin \mathrm{n}((\nu\vec{p})\overline{a}\langle\vec{M}\rangle).$$

From the induction hypothesis we get that there exist $\vec{r}$ and Q'' such that $Q \xrightarrow{\overline{a}} (\nu\vec{r})\langle\vec{M}\rangle Q''$ and $Q' \stackrel{\nu}{=} (\nu\vec{r} \setminus \vec{p})Q''$. Then we can derive

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\overline{a}} (\nu\vec{r})\langle\vec{M}\rangle Q''}{P = (\nu s)Q \xrightarrow{\overline{a}} (\nu s, \vec{r})\langle\vec{M}\rangle Q''},$$

and since we have $Q' \stackrel{\nu}{=} (\nu\vec{r} \setminus \vec{p})Q''$ then we also have $P' = (\nu s)Q' \stackrel{\nu}{=} (\nu s)(\nu\vec{r} \setminus \vec{p})Q'' = (\nu s, \vec{r} \setminus \vec{p})Q''$ since $s \notin \vec{p}$.

Open: Assume that

$$(\textsc{Open})\frac{Q \xrightarrow{(\nu\vec{r})\overline{a}\langle\vec{M}\rangle} P'}{P = (\nu s)Q \xrightarrow{(\nu s,\vec{r})\overline{a}\langle\vec{M}\rangle} P'} \quad a \neq s \text{ and } s \in \mathrm{fn}(\vec{M}).$$

From the induction hypothesis we get that the premise implies that there exists $\vec{t}$ and Q'' such that $Q \xrightarrow{\overline{a}} (\nu\vec{t})\langle\vec{M}\rangle Q''$ and $P' \stackrel{\nu}{=} (\nu\vec{t} \setminus \vec{r})Q''$. Then we can derive

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\overline{a}} (\nu\vec{t})\langle\vec{M}\rangle Q''}{P = (\nu s)Q \xrightarrow{\overline{a}} (\nu s,\vec{t})\langle\vec{M}\rangle Q''},$$

and since we have $P' \stackrel{\nu}{=} (\nu\vec{t} \setminus \vec{r})Q'' = (\nu s\vec{t} \setminus s\vec{r})Q''$ we are done.

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1' \mid Q_2 = P'},$$

then by the induction hypothesis there exists $\vec{q}$ and Q_1'' such that $Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''$ and $Q_1' \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})Q_1''$. We can then derive

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle(Q_1'' \mid Q_2)},$$

and since $Q_1' \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})Q_1''$ then also $P' = Q_1' \mid Q_2 \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})Q_1'' \mid Q_2 \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})(Q_1'' \mid Q_2)$.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > P'' \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} P'}{P \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} P'},$$

then by the induction hypothesis there exists $\vec{q}$ and Q such that $P'' \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q$ and $P' \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})Q$. We can then derive

$$(\textsc{Comm Red})\frac{P > P'' \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q}{P \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q},$$

where $P' \stackrel{\nu}{=} (\vec{q} \setminus \vec{p})Q$ as required.

Next up is the proof of (ii): The induction basis is the case when the transition was derived using (Input). Furthermore, we need not consider the rules (Output), (Open), (Communication)

Input: Assume that

$$(\textsc{Input})\frac{}{P = a\langle\vec{x}\rangle.Q \xrightarrow{a\langle\vec{M}\rangle} Q\{\vec{M}/\vec{x}\} = P'}.$$

Then we have

$$(\textsc{Comm In})\frac{}{P = a\langle\vec{x}\rangle.Q \xrightarrow{a} (\vec{x})Q}$$

where $Q\{\vec{M}/\vec{x}\} = P'$ as required.

Restriction: Assume that

$$(\textsc{Restriction})\frac{Q \xrightarrow{a\langle\vec{M}\rangle} Q'}{P = (\nu s)Q \xrightarrow{a\langle\vec{M}\rangle} (\nu s)Q' = P'} \quad s \notin \mathrm{n}(a\langle\vec{M}\rangle)$$

then by the induction hypothesis there exists Q'' such that $Q \xrightarrow{a} (\vec{x})Q''$ and $Q' = Q''\{\vec{M}/\vec{x}\}$. Then

$$(\textsc{Comm Res})\frac{Q \xrightarrow{a} (\vec{x})Q''}{P = (\nu s)Q \xrightarrow{a} (\vec{x})(\nu s)Q''},$$

where $P' = (\nu s)Q' = (\nu s)(Q''\{\vec{M}/\vec{x}\}) = ((\nu s)Q'')\{\vec{M}/\vec{x}\}$.

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{\alpha} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\alpha} Q_1' \mid Q_2 = P'}$$

then from the induction hypothesis there exists Q_1'' such that $Q_1 \xrightarrow{a} (\vec{x})Q_1''$ and $Q_1' = Q_1''\{\vec{M}/\vec{x}\}$. Then, since $\vec{x} \notin \mathrm{n}(Q_2)$,

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{a} (\vec{x})Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\alpha} (\vec{x})(Q_1'' \mid Q_2)}$$

and $P' = Q_1''\{\vec{M}/\vec{x}\} \mid Q_2 = (Q_1'' \mid Q_2)\{\vec{M}/\vec{x}\}$ which proves this case.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{a\langle\vec{M}\rangle} P'}{P \xrightarrow{a\langle\vec{M}\rangle} P'}$$

then by the induction hypothesis there exists Q such that $\widetilde{P} \xrightarrow{a} (\vec{x})Q$ and $P' = Q\{\vec{M}/\vec{x}\}$ so we can reconstruct the derivation tree

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{a} (\vec{x})Q}{P \xrightarrow{a} (\vec{x})Q}.$$

In (iii) there are only four possible roots in the derivation tree, namely (Reduction),(Restriction), (Parallel) and (Communication). A τ-transition always stems from an application of (Communication) and we use induction in the depth at which this rule occurs in the derivation tree. The induction basis is thus the case when (Communication) is the root of the derivation tree.

Communication: Assume that

$$(\textsc{Communication})\frac{Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1' \qquad Q_2 \xrightarrow{a\langle\vec{M}\rangle} Q_2'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} (\nu\vec{p})(Q_1' \mid Q_2') = P'}$$

which allows us to use (i) and (ii) to deduce that there exists $\vec{q}$ and Q_1'' such that $Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''$ and $Q_1' \stackrel{\nu}{=} (\nu\vec{q} \setminus \vec{p})Q_1''$ and that there exists Q_2'' such that $Q_2 \xrightarrow{a} (\vec{x})Q_2''$ and $Q_2' \stackrel{\nu}{=} Q_2''\{\vec{M}/\vec{x}\}$. This allows us to construct the derivation

$$(\textsc{Comm Inter})\frac{Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1'' \qquad Q_2 \xrightarrow{a} (\vec{x})Q_2''}{P = \xrightarrow{\tau} (\nu\vec{q})\langle\vec{M}\rangle Q_1'' @ (\vec{x})\, Q_2'' = (\nu\vec{q})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\})},$$

and we get

$$\begin{aligned} P' &= (\nu\vec{p})(Q_1' \mid Q_2') \\ &\stackrel{\nu}{=} (\nu\vec{p})((\nu\vec{q} \setminus \vec{p})Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \\ &\stackrel{\nu}{=} (\nu\vec{p})(\nu\vec{q} \setminus \vec{p})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \\ &\stackrel{\nu}{=} (\nu\vec{q})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \end{aligned}$$

as required

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{\tau} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1' \mid Q_2 = P'},$$

then $Q_1 \xrightarrow{\tau} Q_1'' \stackrel{\nu}{=} Q_1'$ and

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\tau} Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1'' \mid Q_2},$$

and since $Q_1'' \stackrel{\nu}{=} Q_1'$ then $Q_1'' \mid Q_2 \stackrel{\nu}{=} Q_1' \mid Q_2$ as required.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{\tau} P'}{P \xrightarrow{\tau} P'}$$

then $\widetilde{P} \xrightarrow{\tau} P''$ such that $P'' \stackrel{\nu}{=} P'$ and

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{\tau} P''}{P \xrightarrow{\tau} P''}$$

which is as required.

Restriction: Assume that

$$(\textsc{Restriction})\frac{Q \xrightarrow{\tau} Q'}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q' = P'}$$

then $Q \xrightarrow{\tau} Q'' \stackrel{\nu}{=} Q'$ and

$$(\text{Comm Res})\frac{Q \xrightarrow{\tau} Q''}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q''}$$

where $P' = (\nu q)Q' \stackrel{\nu}{=} (\nu q)Q''$.

■

The following is a proof of Lemma 2.3.2.

Proof
Proofs of cases (i) and (ii) are by induction in the height of the derivation tree of the transition.

We first prove (i) by examining each possible root of the derivation tree. We start by noting that (Input) and (Communication) are not possible roots and that proving rule (Output) is the induction basis.

Output: Assume that

$$(\text{Output})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}\langle\vec{M}\rangle} P'},$$

then we obtain immediately

$$(\text{Comm Out})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}} (\nu)\langle\vec{M}\rangle P'},$$

which satisfies the condition $P' \stackrel{\nu}{=} P'$.

Restriction: Assume that

$$(\text{Restriction})\frac{Q \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q'}{P = (\nu s)Q \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} (\nu s)Q' = P'} \quad s \notin \text{n}((\nu\vec{p})\overline{a}\langle\vec{M}\rangle).$$

From the induction hypothesis we get that there exists $\vec{r}$ and Q'' such that $Q \xrightarrow{\overline{a}} (\nu\vec{r})\langle\vec{M}\rangle Q''$ and $Q' \stackrel{\nu}{=} (\nu\vec{r} \setminus \vec{p})Q''$. Then we can derive

$$(\text{Comm Res})\frac{Q \xrightarrow{\overline{a}} (\nu\vec{r})\langle\vec{M}\rangle Q''}{P = (\nu s)Q \xrightarrow{\overline{a}} (\nu s, \vec{r})\langle\vec{M}\rangle Q''},$$

and since we have $Q' \stackrel{\nu}{=} (\nu\vec{r} \setminus \vec{p})Q''$ then we also have $P' = (\nu s)Q' \stackrel{\nu}{=} (\nu s)(\nu\vec{r} \setminus \vec{p})Q'' = (\nu s, \vec{r} \setminus \vec{p})Q''$ since $s \notin \vec{p}$.

Open: Assume that

$$(\text{Open})\frac{Q \xrightarrow{(\nu\vec{r})\overline{a}\langle\vec{M}\rangle} P'}{P = (\nu s)Q \xrightarrow{(\nu s,\vec{r})\overline{a}\langle\vec{M}\rangle} P'} \quad a \neq s \text{ and } s \in \text{fn}(\vec{M}).$$

From the induction hypothesis we get that the premise implies that there exists $\vec{t}$ and Q'' such that $Q \xrightarrow{\overline{a}} (\nu\vec{t})\langle\vec{M}\rangle Q''$ and $P' \overset{\nu}{=} (\nu\vec{t} \setminus \vec{r})Q''$. Then we can derive

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\overline{a}} (\nu\vec{t})\langle\vec{M}\rangle Q''}{P = (\nu s)Q \xrightarrow{\overline{a}} (\nu s, \vec{t})\langle\vec{M}\rangle Q''},$$

and since we have $P' \overset{\nu}{=} (\nu\vec{t} \setminus \vec{r})Q'' = (\nu s\vec{t} \setminus s\vec{r})Q''$ we are done.

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1' \mid Q_2 = P'},$$

then by the induction hypothesis there exists $\vec{q}$ and Q_1'' such that $Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''$ and $Q_1' \overset{\nu}{=} (\vec{q} \setminus \vec{p})Q_1''$. We can then derive

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle(Q_1'' \mid Q_2)},$$

and since $Q_1' \overset{\nu}{=} (\vec{q} \setminus \vec{p})Q_1''$ then also $P' = Q_1' \mid Q_2 \overset{\nu}{=} (\vec{q} \setminus \vec{p})Q_1'' \mid Q_2 \overset{\nu}{=} (\vec{q} \setminus \vec{p})(Q_1'' \mid Q_2)$.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > P'' \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} P'}{P \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} P'},$$

then by the induction hypothesis there exists $\vec{q}$ and Q such that $P'' \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q$ and $P' \overset{\nu}{=} (\vec{q} \setminus \vec{p})Q$. We can then derive

$$(\textsc{Comm Red})\frac{P > P'' \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q}{P \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q},$$

where $P' \overset{\nu}{=} (\vec{q} \setminus \vec{p})Q$ as required.

Next up is the proof of (ii): The induction basis is the case when the transition was derived using (Input). Furthermore, we need not consider the rules (Output), (Open), (Communication)

Input: Assume that

$$(\textsc{Input})\frac{}{P = a\langle\vec{x}\rangle.Q \xrightarrow{a\langle\vec{M}\rangle} Q\{\vec{M}/\vec{x}\} = P'}.$$

Then we have

$$(\textsc{Comm In})\frac{}{P = a\langle\vec{x}\rangle.Q \xrightarrow{a} (\vec{x})Q}$$

where $Q\{\vec{M}/\vec{x}\} = P'$ as required.

Restriction: Assume that

$$(\textsc{Restriction})\frac{Q \xrightarrow{a\langle\vec{M}\rangle} Q'}{P = (\nu s)Q \xrightarrow{a\langle\vec{M}\rangle} (\nu s)Q' = P'} \quad s \notin \mathrm{n}(a\langle\vec{M}\rangle)$$

then by the induction hypothesis there exists Q'' such that $Q \xrightarrow{a} (\vec{x})Q''$ and $Q' = Q''\{\vec{M}/\vec{x}\}$. Then

$$(\textsc{Comm Res})\frac{Q \xrightarrow{a} (\vec{x})Q''}{P = (\nu s)Q \xrightarrow{a} (\vec{x})(\nu s)Q''},$$

where $P' = (\nu s)Q' = (\nu s)(Q''\{\vec{M}/\vec{x}\}) = ((\nu s)Q'')\{\vec{M}/\vec{x}\}$.

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{\alpha} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\alpha} Q_1' \mid Q_2 = P'}$$

then from the induction hypothesis there exists Q_1'' such that $Q_1 \xrightarrow{a} (\vec{x})Q_1''$ and $Q_1' = Q_1''\{\vec{M}/\vec{x}\}$. Then, since $\vec{x} \notin \mathrm{n}(Q_2)$,

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{a} (\vec{x})Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\alpha} (\vec{x})(Q_1'' \mid Q_2)}$$

and $P' = Q_1''\{\vec{M}/\vec{x}\} \mid Q_2 = (Q_1'' \mid Q_2)\{\vec{M}/\vec{x}\}$ which proves this case.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{a\langle\vec{M}\rangle} P'}{P \xrightarrow{a\langle\vec{M}\rangle} P'}$$

then by the induction hypothesis there exists Q such that $\widetilde{P} \xrightarrow{a} (\vec{x})Q$ and $P' = Q\{\vec{M}/\vec{x}\}$ so we can reconstruct the derivation tree

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{a} (\vec{x})Q}{P \xrightarrow{a} (\vec{x})Q}.$$

In (iii) there are only four possible roots in the derivation tree, namely (Reduction),(Restriction), (Parallel) and (Communication). A τ-transition always stems from an application of (Communication) and we use induction in the depth at which this rule occurs in the derivation tree. The induction basis is thus the case when (Communication) is the root of the derivation tree.

Communication: Assume that

$$(\textsc{Communication})\frac{Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1' \qquad Q_2 \xrightarrow{a\langle\vec{M}\rangle} Q_2'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} (\nu\vec{p})(Q_1' \mid Q_2') = P'}$$

which allows us to use (i) and (ii) to deduce that there exists $\vec{q}$ and Q_1'' such that $Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1''$ and $Q_1' \overset{\nu}{=} (\nu\vec{q} \setminus \vec{p})Q_1''$ and that there exists Q_2''

such that $Q_2 \xrightarrow{a} (\vec{x})Q_2''$ and $Q_2' \stackrel{\nu}{=} Q_2''\{\vec{M}/\vec{x}\}$. This allows us to construct the derivation

$$(\textsc{Comm Inter})\frac{Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1'' \qquad Q_2 \xrightarrow{a} (\vec{x})Q_2''}{P = \xrightarrow{\tau} (\nu\vec{q})\langle\vec{M}\rangle Q_1''@(\vec{x})\,Q_2'' = (\nu\vec{q})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\})},$$

and we get

$$\begin{aligned} P' &= (\nu\vec{p})(Q_1' \mid Q_2') \\ &\stackrel{\nu}{=} (\nu\vec{p})((\nu\vec{q} \setminus \vec{p})Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \\ &\stackrel{\nu}{=} (\nu\vec{p})(\nu\vec{q} \setminus \vec{p})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \\ &\stackrel{\nu}{=} (\nu\vec{q})(Q_1'' \mid Q_2''\{\vec{M}/\vec{x}\}) \end{aligned}$$

as required

Parallel: Assume that

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{\tau} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1' \mid Q_2 = P'},$$

then $Q_1 \xrightarrow{\tau} Q_1'' \stackrel{\nu}{=} Q_1'$ and

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\tau} Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1'' \mid Q_2},$$

and since $Q_1'' \stackrel{\nu}{=} Q_1'$ then $Q_1'' \mid Q_2 \stackrel{\nu}{=} Q_1' \mid Q_2$ as required.

Reduction: Assume that

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{\tau} P'}{P \xrightarrow{\tau} P'}$$

then $\widetilde{P} \xrightarrow{\tau} P''$ such that $P'' \stackrel{\nu}{=} P'$ and

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{\tau} P''}{P \xrightarrow{\tau} P''}$$

which is as required.

Restriction: Assume that

$$(\textsc{Restriction})\frac{Q \xrightarrow{\tau} Q'}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q' = P'}$$

then $Q \xrightarrow{\tau} Q'' \stackrel{\nu}{=} Q'$ and

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\tau} Q''}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q''}$$

where $P' = (\nu q)Q' \stackrel{\nu}{=} (\nu q)Q''$.

■

The following is a proof of Lemma 2.3.2.

Proof
The proof is by transition induction in the cases (i) and (ii). We start by proving (i).

Comm Out: Assume that

$$(\textsc{Comm Out})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}} (\nu)\langle\vec{M}\rangle P'}$$

and we have immediately

$$(\textsc{Output})\frac{}{P = \overline{a}\langle\vec{M}\rangle.P' \xrightarrow{\overline{a}\langle\vec{M}\rangle} P'},$$

which of course satisfies the condition $P' \stackrel{\nu}{=} P'$.

Comm Par: Assume that

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\overline{a}} (\nu p)\langle\vec{M}\rangle Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\overline{a}} (\nu p)\langle\vec{M}\rangle(Q_1' \mid Q_2)}$$

then there exists Q_1'' and $\vec{q}$ such that $Q_1 \xrightarrow{(\nu q)\overline{a}\langle\vec{M}\rangle} Q_1''$ and $Q_1'' \stackrel{\nu}{=} (\nu\vec{p}\setminus\vec{q})Q_1'$, so

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{(\nu q)\overline{a}\langle\vec{M}\rangle} Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{(\nu q)\overline{a}\langle\vec{M}\rangle} Q_1'' \mid Q_2}$$

where $Q_1'' \mid Q_2 \stackrel{\nu}{=} (\nu\vec{p}\setminus\vec{q})Q_1' \mid Q_2 \stackrel{\nu}{=} (\nu\vec{p}\setminus\vec{q})(Q_1' \mid Q_2)$.

Comm Res: Assume that

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\overline{a}} (\nu\vec{p})\langle\vec{M}\rangle Q'}{P = (\nu r)Q \xrightarrow{\overline{a}} (\nu r\vec{p})\langle\vec{M}\rangle Q'} \quad \overline{a} \notin \{r, \overline{r}\}$$

then by the induction hypothesis there exists Q'' and $\vec{q}$ such that $Q \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} Q''$ and $Q'' \stackrel{\nu}{=} (\nu\vec{p}\setminus\vec{q})Q'$. It is either the case that $r \in \mathrm{n}((\nu\vec{q})\overline{a}\langle\vec{M}\rangle)$ or not. If $r \in \mathrm{n}((\nu\vec{q})\overline{a}\langle\vec{M}\rangle)$ then

$$(\textsc{Open})\frac{Q \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} Q''}{P = (\nu r)Q \xrightarrow{(\nu r,\vec{q})\overline{a}\langle\vec{M}\rangle} Q''} \quad r \neq a \text{ and } r \in \mathrm{fn}(\vec{M})$$

where $Q'' \stackrel{\nu}{=} (\nu r, \vec{p}\setminus r\vec{q})Q'$. On the other hand, if $r \notin \mathrm{n}((\nu q)\overline{a}\langle\vec{M}\rangle)$ then

$$(\textsc{Restriction})\frac{Q \xrightarrow{(\nu q)\overline{a}\langle\vec{M}\rangle} Q''}{P = (\nu r)Q \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} (\nu r)Q''} \quad r \notin \mathrm{n}((\nu\vec{q})\overline{a}\langle\vec{M}\rangle)$$

where $(\nu r)Q'' \stackrel{\nu}{=} (\nu r)(\nu\vec{p}\setminus q)Q' \stackrel{\nu}{=} (\nu r\vec{p}\setminus q)Q'$ because $r \notin \vec{q}$.

Comm Red: Assume that

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle P'}{P \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle P'}$$

then by the induction hypothesis there exists P'' and $\vec{q}$ such that $\widetilde{P} \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} P''$ and $P'' \stackrel{\nu}{=} (\nu\vec{q} \setminus \vec{p})P'$. So we have

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} P''}{P \xrightarrow{(\nu\vec{q})\overline{a}\langle\vec{M}\rangle} P''}$$

where $P'' \stackrel{\nu}{=} (\nu\vec{q} \setminus \vec{p})P'$.

Now we turn to the proof of (ii):

Comm In: Assume that

$$(\textsc{Comm In})\frac{}{P = a(\vec{x}).Q \xrightarrow{a} (\vec{x})Q}$$

then

$$(\textsc{Input})\frac{}{P = a(\vec{x}).Q \xrightarrow{a\langle\vec{M}\rangle} Q\{\vec{M}/\vec{x}\}}$$

for any $\vec{M}$ such that $|\vec{M}| = |\vec{x}|$.

Comm Par: Assume that

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{a} (\vec{x})Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{a} (\vec{x})(Q_1' \mid Q_2)}$$

then $Q_1 \xrightarrow{a\langle\vec{M}\rangle} Q_1'\{\vec{M}/\vec{x}\}$ and

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{a\langle\vec{M}\rangle} Q_1'\{\vec{M}/\vec{x}\}}{P = Q_1 \mid Q_2 \xrightarrow{a\langle\vec{M}\rangle} Q_1'\{\vec{M}/\vec{x}\} \mid Q_2 = (Q_1' \mid Q_2)\{\vec{M}/\vec{x}\}}$$

for all $\vec{M}$ such that $|\vec{M}| = |\vec{x}|$.

Comm Res: Assume that

$$(\textsc{Comm Res})\frac{Q \xrightarrow{a} (\vec{x})Q'}{P = (\nu r)Q \xrightarrow{a} (\vec{x})(\nu r)Q'}$$

then $Q \xrightarrow{a(\vec{M})} Q'\{\vec{M}/\vec{x}\}$ and

$$(\textsc{Restriction})\frac{Q \xrightarrow{a\langle\vec{M}\rangle} Q'\{\vec{M}/\vec{x}\}}{P = (\nu r)Q \xrightarrow{a\langle\vec{M}\rangle} (\nu r)Q'\{\vec{M}/\vec{x}\}}$$

for all $\vec{M}$ such that $|\vec{M}| = |\vec{x}|$.

Comm Red: Assume that

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{a} (\vec{x})P'}{P \xrightarrow{a} (\vec{x})P'}$$

then $\widetilde{P} \xrightarrow{a\langle\vec{M}\rangle} P'\{\vec{M}/\vec{x}\}$ and

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{a\langle\vec{M}\rangle} P'\{\vec{M}/\vec{x}\}}{P \xrightarrow{a\langle\vec{M}\rangle} P'\{\vec{M}/\vec{x}\}}$$

for all $\vec{M}$ such that $|\vec{M}| = |\vec{x}|$.

Proof of case (iii) is by induction in the depth of the application of (Comm Inter) in the derivation tree.

Comm Inter: Assume that

$$(\textsc{Comm Inter})\frac{Q_1 \xrightarrow{\overline{a}} (\nu\vec{q})\langle\vec{M}\rangle Q_1' \qquad Q_2 \xrightarrow{a} (\vec{x})Q_2'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} (\nu\vec{q})\langle\vec{M}\rangle Q_1' @ (\vec{x})Q_2' = (\nu\vec{q})(Q_1' \mid Q_2\{\vec{M}/\vec{x}\})}$$

then from (i) we have that $Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1''$ where $Q_1'' \stackrel{\nu}{=} (\nu\vec{q} \setminus \vec{p})Q_1'$ and from (ii) we have that $Q_2 \xrightarrow{a\langle\vec{M}\rangle} Q_2'\{\vec{M}/\vec{x}\}$. So

$$(\textsc{Communication})\frac{Q_1 \xrightarrow{(\nu\vec{p})\overline{a}\langle\vec{M}\rangle} Q_1'' \qquad Q_2 \xrightarrow{a\langle\vec{M}\rangle} Q_2'\{\vec{M}/\vec{x}\}}{P = Q_1 \mid Q_2 \xrightarrow{\tau} (\nu\vec{p})(Q_1'' \mid Q_2'\{\vec{M}/\vec{x}\})}$$

where

$$\begin{aligned}(\nu\vec{p})(Q_1'' \mid Q_2'\{\vec{M}/\vec{x}\}) &\stackrel{\nu}{=} (\nu\vec{p})((\nu\vec{q} \setminus \vec{p})Q_1' \mid Q_2'\{\vec{M}/\vec{x}\}) \\ &\stackrel{\nu}{=} (\nu\vec{q})(Q_1' \mid Q_2'\{\vec{M}/\vec{x}\}).\end{aligned}$$

Comm Par: Assume that

$$(\textsc{Comm Par})\frac{Q_1 \xrightarrow{\tau} Q_1'}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1' \mid Q_2 = P'}$$

then from the induction hypothesis $Q_1 \xrightarrow{\tau} Q_1'' \stackrel{\nu}{=} Q_1'$ and thus

$$(\textsc{Parallel})\frac{Q_1 \xrightarrow{\tau} Q_1''}{P = Q_1 \mid Q_2 \xrightarrow{\tau} Q_1'' \mid Q_2 \stackrel{\nu}{=} Q_1' \mid Q_2}.$$

Comm Res: Assume that

$$(\textsc{Comm Res})\frac{Q \xrightarrow{\tau} Q'}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q' = P'}$$

then by the induction hypothesis we have $Q \xrightarrow{\tau} Q'' \stackrel{\nu}{=} Q'$ and so

$$(\textsc{Restriction})\frac{Q \xrightarrow{\tau} Q''}{P = (\nu q)Q \xrightarrow{\tau} (\nu q)Q'' = (\nu q)Q'}.$$

Comm Red: Assume that

$$(\textsc{Comm Red})\frac{P > \widetilde{P} \xrightarrow{\tau} P'}{P \xrightarrow{\tau} P'}$$

then by the induction hypothesis $\widetilde{P} \xrightarrow{\tau} P'' \stackrel{\nu}{=} P'$ so

$$(\textsc{Reduction})\frac{P > \widetilde{P} \xrightarrow{\tau} P''}{P \xrightarrow{\tau} P'' \stackrel{\nu}{=} P'}.$$

■

Bibliography

[Aba97] M. Abadi. Secrecy by typing in security protocols. January 1997. *Proc. 3rd Int'l. Symp. on Theoretical Aspects of Computer Software (TACS '97)*, Springer-Verlag, Berlin, Germany, 1997, pp. 611-638.

[AF01] Martín Abadi and Cédric Fournet. Mobile Values, New Names, and Secure Communication. *ACM SIGPLAN Notices*, 36(3):104–115, March 2001.

[AF03] Martín Abadi and Cedric Fournet. Hiding names: Private authentication in the applied pi calculus. March 02 2003.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[BDP02] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[BPV03] Michael Baldamus, Joachim Parrow, and Björn Victor. Spi Calculus Translated to π-calculus Preserving May-Testing. Technical report, Department of Information Technology, Uppsala University, Sweden, December 2003.

[Gay00] Simon Gay. Some type systems for the pi calculus. May 18 2000.

[GJ] Andrew D. Gordon and Allan Jeffrey. Cryptyp - cryptographic protocol type checker. http://cryptyc.cs.depaul.edu/.

[GJ01a] Andrew D. Gordon and Alan Jeffrey. Authenticity by typing for security protocols. Technical Report MSR-TR-2001-49, Microsoft Research (MSR), May 2001.

[GJ01b] Andrew D. Gordon and Alan Jeffrey. Typing correspondence assertions for communication protocols. Technical Report MSR-TR-2001-48, Microsoft Research (MSR), May 2001.

[GJ02] Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. Technical Report MSR-TR-2002-31, Microsoft Research (MSR), August 2002. a portion of this work appears in the proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW 15), Cape Breton, June 24–26, 2002.

[Joh87] Peter T. Johnstone. *Notes on logic and set theory*. Cambridge University Press, 1987.

[KB70] D. E. Knuth and P. B. Bendix, editors. *Simple word problems in universal algebra*, pages 263–297. Pergamon Press, 1970.

[KK01] Claude Kirchner and Hélène Kirchner. Rewriting solving proving. Unpublished book, December 2001.

[Mil80] R. Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[Mil90] Robin Milner. Functions as processes. *Lecture Notes in Computer Science*, (443):167–180, 1990.

[Mil93] Robin Milner. The Polyadic π-calculus: A Tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktoberdorf, Germany, 1991)*, volume 94 of *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[Mit96] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, MA, 1996.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Proceedings ICALP '92*, volume 623, pages 685–695, Vienna, 1992. Springer-Verlag.

[MS98] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *25th Colloquium on Automata, Languages and Programming (ICALP) (Aalborg, Denmark)*, volume 1443 of *LNCS*, pages 856–867. Springer, July 1998.

[Par01] Joachim Parrow. An introduction to the pi-calculus. In Jan Bergstra, Alban Ponse, and Scott Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

[PS93] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings 8th IEEE Logics in Computer Science*, pages 376–385, Montreal, Canada, June 1993.

[PT97] Benjamin Pierce and David Turner. Pict: A Programming Language Based on the PI-Calculus. Technical Report 476, Indiana University, 1997.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[ST99] Donald Sanella and Andrzej Tarlecki. Algebraic Preliminaries. In Egidio Astensiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, IFIP State-of-the-Art Reports, chapter 2, pages 13–30. Springer, 1999.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus - A Theory of Mobile Processes.* Cambridge University Press, 2001.

[VM94] Björn Victor and Faron Moller. The Mobility Workbench — A tool for the π-calculus. In David Dill, editor, *Computer Aided Verification (Proc. of CAV'94)*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.