AALBORG UNIVERSITY
Student report

Datalogi
Femte studieår
Selma Lagerlöfs Vej 300
9220 Aalborg Øst
http://www.cs.aau.dk

Title:

Typing Reflection in Higher-Order Ψ-Calculi

Theme:

Master's Thesis in Computer Science

Project period:

Spring, 2021

Group name:

pt101

Group members:

Alexander R. Bendixen
Bjarke B. Bojesen
Stian Lybech

Supervisor:

Hans Hüttel

Page count:

131

Date of completion:

June 15, 2021

Abstract:
We study the *Higher-Order Ψ-calculi* (HOΨ) of Parrow et al. [28]: a general framework for representing many variants and extensions of the first- and higher-order π-calculi. Each specific calculus is obtained by setting and varying a small number of parameters; notably, the *terms* (channels) M, N, *conditions* φ, and *assertions* Ψ of the language, which are nominal data types, and an *entailment* relation $\Psi \Vdash \varphi$ for controlling in which environments Ψ a condition φ, such as the equivalence between two channels, M, N, holds.
We review the syntax and labelled semantics, and we furthermore create three different reduction semantics for the HOΨ-calculus: One is fully compositional and uses structural congruence, but does not fully match every τ-labelled transition; the second matches this relation exactly but uses reduction contexts in place of structural rules; and the last is larger than the τ-transition relation and employs an *evaluation relation* rather than structural congruence directly.
Higher-order process mobility appears as a limited form of the more general capability of *reflection;* the generic ability of a program to turn code into data, compute with it, even modify it, and reinstantiate it as running code. We explore this connexion by showing that the HOΨ-framework can even represent the Reflective Higher-Order (RHO or ρ) calculus of Meredith and Radestock [20].
We define a generic type system for HOΨ, using the largest of the three reduction semantics, which thus encompasses all τ-labelled transitions. It includes generic rules for type judgements of processes, but requires type judgements for terms, conditions and assertions to be given as parameters, mirroring the fact that these sets are also parameters in the HOΨ-calculus. We prove a general result of *subject reduction* for the generic type system, but we cannot prove *safety*, since this relies on a notion of *type error* which will depend on the specific choice of parameter setting for the terms, conditions and assertions. Safety must therefore be proved manually for each instantiation.
Lastly, we attempt to *type reflection*, by instantiating the generic type system to the representation of the ρ-calculus. The example hints at the kinds of limitation that must be imposed on reflection to make it typable, such as including type information in the definition of entailment, to create a *typed channel equivalence.*

Summary

Reflection is the generic ability of a program to turn code into data, compute with it, even modify it, and lastly turn it back into code again. This is often a desirable property to have in a programming language, because it affords a greater flexibility. However it comes at the cost of seemingly making static checks of the code, such as type checking, difficult or impossible, when the program itself can generate new code at runtime. For example, we have shown in a previous study [2], that a certain reflective process calculus, the ρ-calculus of Meredith and Radestock [20], cannot be represented within the well-known π-calculus [26], and, not surprisingly, that we were unable to create a π-calculus style type system for the full ρ-calculus, but only for a certain subset of processes that corresponded to π-calculus processes.

In the present thesis, we study the Ψ-*calculi* of Bengtson et al. [3, 4]; a general framework for representing many variants and extensions of the π-calculus, by specifying just a few parameters: notably, the *terms* (channels) M, N, *conditions* φ, and *assertions* Ψ of the language, and an *entailment* relation $\Psi \Vdash \varphi$ for controlling in which environments Ψ a condition φ, such as the equivalence between two channels, $M \leftrightarrow N$, holds. Specifically, we study the *Higher-Order* Ψ-*calculi* of Parrow et al. [28], which is an extension of the 'first-order' Ψ-calculi to enable higher-order capabilities, thereby extending the range of calculi that may be represented within the framework, such as CHOCS [36] and the HOπ-calculus [31; 32].

Besides reviewing the syntax and labelled semantics of the original formulation, we also discuss the formulation of a reduction semantics for the HOΨ-calculus framework, and we create three different variants of a reduction relation $\rightarrow$ with different properties: One is fully compositional and uses structural congruence, but does not fully match every τ-labelled transition; the second matches $\xrightarrow{\tau}$ exactly and is an extension of a reduction semantics by Åman Pohjola [39] for the first-order Ψ-calculus, which uses reduction contexts in place of structural rules; and the last is slightly larger than the $\xrightarrow{\tau}$ relation and makes use of an *evaluation relation* rather than structural congruence directly.

It may be noted that the higher-order capabilities of calculi such as HOπ and CHOCS are just a special, limited form of reflection, since they allow processes to be communicated as data terms, and later reinstantiated as processes, but without the ability to compute with, or modify, the process code. As mentioned above, these calculi are representable within the HOΨ-framework; but in the present thesis we also show that HOΨ is also capable of representing the full ρ-calculus. This is first

and foremost made possible by the way in which one may parametrise the HOΨ-framework with (almost) arbitrarily complex definitions of terms and a channel equivalence relation $\dot{\leftrightarrow}$ for concluding the equivalence between these structured terms. This allows us to build the equivalent of the name equivalence relation $\equiv_{\mathcal{N}}$ of the ρ-calculus directly into the specification of this parameter. Thus, the HOΨ-framework can even be used to represent reflection.

A second strand in our work concerns the development of a generic type system for the HOΨ-calculus framework: Hüttel [17] creates a generic type system for the *first-order* Ψ-calculus, which may likewise be instantiated through parameter setting to yield type systems for the calculi representable within the Ψ-calculus framework, and we build upon this work to create a similarly generic type system for HOΨ, thereby allowing us to instantiate type systems for higher-order calculi.

Notably, we give generic rules for type judgements of processes, but require type judgements for terms, conditions and assertions to be given as parameters, mirroring the fact that these sets are also parameters in the HOΨ-calculus. Likewise, we prove a general result of *subject reduction* in the style of Wright and Felleisen [38] for the generic type system, but we cannot prove *safety*, since this relies on a notion of *type error* (an error predicate) or, conversely, of *well-behaved* processes, which, in either case, will depend on the specific choice of parameter setting for the terms, conditions and assertions. Safety must therefore be proved manually for each instantiation.

We demonstrate how the generic type system may be used by creating an instantiation for a simplified HOπ-like calculus, and lastly, we return to the topic of typing reflection, by instantiating the generic type system to our instantiation of the ρ-calculus, thereby again attempting to create a type system for this calculus, which had previously eluded us [2].

However, surprisingly, we find that even though we can *represent* the ρ-calculus within the HOΨ-framework, we cannot make the corresponding type system instantiation sound, without including the type environment Γ in the definition of entailment for channel equivalence. This is necessary, because the *type* of a ρ-calculus name is not derived from the *structure* of the name, which thus makes it possible to build two names that are channel equivalent, but nevertheless have different types. Including type information in the definition of entailment would allow us to restrict channel equivalence to only hold between terms of the *same* type, i.e. yielding a *typed channel equivalence*; but thereby also *either* restricting the flexibility afforded by the reflective capability of the ρ-calculus, *or* requiring us to know in advance the type of every name that will be generated at runtime. Neither option seems attractive, or tractable.

Thus in conclusion, we have shown that the HOΨ-framework can represent reflection, and we have created a generic type system for HOΨ, which to our knowledge is the first of its kind. However, as our examples show, the type system cannot be instantiated to *every* calculus that is representable within the HOΨ-framework, including, unfortunately, the elusive ρ-calculus.

Contents

1 Introduction

Reflection is the ability of a program to inspect its own code at runtime, turn it into data, compute with it, even modify it, and then re-instantiate it as running code.[1] This affords a greater flexibility for the programmer, but simultaneously presents a problem: We often want to reason about properties of our programs *before* the code is executed, e.g. by using *type systems* to ensure a form of safety or correctness; but with reflective capabilities in the language, new code may be generated at *runtime* which therefore cannot be available for static inspection. Hence, our purpose in the present thesis is to develop a type system for a reflective language, to investigate whether or to what extent we may be able to resolve this apparent conflict between safety and flexibility.

1.1 Computational model

Process calculi are a well-known family of formalisms for modelling and reasoning about concurrent programming, encompassing both parallel computations within a single program, and distributed programs executing at multiple sites. Among the most prominent examples is undoubtedly the π-calculus of Milner [26] and Sangiorgi and Walker [33], that has proved to be a highly versatile computational model, capable of capturing a wide range of the aforementioned phenomena. It has a well-developed theory with several formulations of operational semantics (i.e. labelled, symbolic, reduction), bisimulation and characterisation with modal logic and so forth, most of which is summarised in [26; 27].

Furthermore, a wealth of type systems have also been developed to capture a range of safety features: for example Milner's original sorting system for correct channel usage [26]; a type system with subtyping by Pierce and Sangiorgi [30]; and type systems with input/output capabilities for channel usage and for ensuring termination [9; 8], to name but a few.

Briefly, the π-calculus assumes an infinite set of names $\mathcal{N}$ ranged over by x, y, z, that can be used as both *channels, data* and *variables*. In an *output* construct $\overline{x}\!<\!z\!>\!.P$

[1] Actually, there is, at least to our knowledge, no universally agreed-upon definition of what precisely is entailed by the term 'reflection' but the concept harkens back at least to the work of Smith [35] on procedural reflection in LISP, and the aforementioned features seem at least to be in accordance with this.

we say that x is the *subject* and z is the *object*, and likewise in an *input* construct $x(y).Q$ with y here as the object. Computation is then modelled in the π-calculus through communication of names on named channels, such as

$$\overline{x}\langle z\rangle.P \mid x(y).Q \xrightarrow{\tau} P \mid Q\{z/y\}$$

where the data term z is sent along the channel x and replaces the variable y within the continuation Q.

This marks the π-calculus as belonging purely within the first-order paradigm, since it only allows transmission of channel names, and nothing else. Consequently, the π-calculus itself does not immediately appear to be a suitable model of reflection. However, this style of computational model, dubbed the *object paradigm* by Milner [24], has since seen the development of many other π-like process calculi, and amongst these is one of particular interest for our endeavour: The *Reflective Higher-Order* calculus (RHO or ρ) by Meredith and Radestock [20] (cf. appendix A for a detailed review).

1.2 Reflection in process calculi

The ρ-calculus departs from the tradition of other π-like calculi in that it does *not* assume a set of atomic names $\mathcal{N}$. Instead, it lets names be built from the *same* syntax as processes, but *quoted*, like a fragment of program code that is put in quotes and then treated like any other text string. Thus, if P is a ρ-process, then $\ulcorner P \urcorner$ is a *quoted* process, and therefore a *name*. This yields a very small language, even by process-calculi standards, consisting of just five syntactic constructs:

$$P, Q, R ::= \mathbf{0} \;\Big|\; P \mid Q \;\Big|\; \ulcorner P \urcorner \langle\!| Q |\!\rangle \;\Big|\; \ulcorner P \urcorner (\ulcorner R \urcorner).Q \;\Big|\; \urcorner\!\ulcorner R \urcorner\!\urcorner$$

The $\mathbf{0}$ and parallel constructs are similar to their π-calculus counterparts, and so is the input construct $\ulcorner P \urcorner (\ulcorner R \urcorner).Q$, save only that both the subject $\ulcorner P \urcorner$ and the object $\ulcorner R \urcorner$ are structured terms (i.e. quoted processes) rather than atomic entities.

The *lift* operation, $\ulcorner P \urcorner \langle\!| Q |\!\rangle$, corresponds to a π-like *output* operation, except that it takes a *process* Q as its object, rather than a *name* $\ulcorner Q \urcorner$, because its purpose precisely is to quote the object Q, before it is sent along $\ulcorner P \urcorner$. Thence it can be received by an input as the *name* $\ulcorner Q \urcorner$, e.g. as in

$$\ulcorner P \urcorner \langle\!| P |\!\rangle \mid \ulcorner P \urcorner (\ulcorner R \urcorner).Q \to Q\{\ulcorner P \urcorner / \ulcorner R \urcorner\}$$

and thus the calculus can generate new names at runtime, thereby making a π-like ν-operator superfluous. And lastly, the *drop* operation, $\urcorner\!\ulcorner R \urcorner\!\urcorner$, is a request to run the process within a name, by removing the quotes around it. It is not performed by a reduction, but rather by a form of substitution

$$\urcorner\!\ulcorner R \urcorner\!\urcorner\{\ulcorner P \urcorner / \ulcorner R \urcorner\} = P$$

where the entire *process* $\urcorner R \urcorner$ is replaced with the process P that was found within the name. We call this an *eager drop* mechanism, because the drop is performed immediately, everywhere in the continuation of an input.[2] Thus, $\urcorner R \urcorner$ will only ever be reached in a reduction, if $\ulcorner R \urcorner$ is not bound by an input (i.e. it is a free name), and since it has no reduction, it actually represents a deadlock.

Together, the lift and drop operations precisely enable the ρ-calculus to turn 'code' (i.e. processes) into data, and the converse. They are the source of its reflective capability. Indeed, there is little else *in* the language apart from parallelism, communication and reflection, and taken together, these features are enough to yield a succinct model of both parallel and distributed computation.

1.3 Reflection and higher-order characteristics

The ρ-calculus' reflective capability also makes it an *inherently higher-order* calculus. Indeed, higher-order characteristics appear as just a limited form of reflection, at least according to the (rather broad) view of reflection adopted here, since it allows code to be passed around as data, and then reactivated at the reception point.

Code mobility may be convenient, but it adds nothing in terms of expressivity to a language that is already computationally complete. Several other process calculi have been extended with a construct for code mobility, to allow processes to be passed around directly, rather than by reference: notable examples include Thomsen's CHOCS calculus [36], and the Higher-Order π-calculus, HOπ, by Sangiorgi [32], yet these variants are precisely *extensions* of preexisting computational models; here of CCS [23] and the π-calculus, respectively. But not the ρ-calculus: *Its* higher-order characteristics derive from its reflective capability, and there seems to be no obvious way to reduce it to a first-order calculus without also limiting its expressivity.

This is curious, in light of a remark by Davide Sangiorgi in his Ph.D thesis [31, p. 8], where he notes that

> *[...] the* HOπ *is representable within the* π*-calculus. This proves that the first-order paradigm, being by far simpler, should be taken as basic. Such a conclusion takes away the interest in the opposite direction, namely the representability of the* π*-calculus within a language using purely communications of agents ...*

Yet the ρ-calculus does not immediately *appear* any more complicated than the π-calculus, despite having higher-order characteristics: Indeed, it can seem even *more* basic, when measured in terms of syntactic constructs or semantic rules.

[2] This is in contrast to a *delayed drop* that instead would use a reduction rule $\ulcorner\urcorner R \ulcorner\urcorner \to R$ to perform the drop. In [2] we conjecture that the mechanism of eager drop cannot be captured by a delayed drop semantics.

1.4 Reflection and encodability

The above considerations led us in [2] to investigate the relationship, in terms of encodability and separation, between the ρ- and π-calculi: The ρ-calculus can, perhaps not unexpectedly, encode the π-calculus in a reasonably straightforward way: only the ν-operator and replication require some care to ensure that all generated names will in fact be unique.

More surprisingly, however, the π-calculus does not seem to be able to encode the ρ-calculus, as we conjectured in [2]. To see why, recall that Carbone and Maffeis [7] formally proved the separation between the π-calculus and their ${}^e\pi$-extension. This theorem was later rederived and reformulated by Gorla [14] as follows: Let the *match degree* $\text{MD}(\mathcal{L})$ of a language $\mathcal{L}$ denote the least upper bound on the number of names that must be matched to yield a reduction in $\mathcal{L}$. Thus, for example $\text{MD}(\pi) = 1$ because π-calculus communications are of the form

$$\overline{x}\!<\!z\!>\!.P \mid x(y).Q \rightarrow P \mid Q\{z/y\}$$

i.e. the number of names in subject position is exactly 1. By allowing up to n-ary *vectors* $\widetilde{x}$ of names in subject position $\widetilde{x}(y).P$ and $\widetilde{x}\!<\!z\!>\!.Q$, we obtain the language π_n, for any $n \geq 0$, and the language ${}^e\pi$ is then

$${}^e\pi \triangleq \bigcup_{n=0}^{\infty} \pi_n$$

i.e. the union of these languages for all n. Thus, for any such n we have that $\text{MD}(\pi_n) = n$, and the language ${}^e\pi$ itself will have $\text{MD}({}^e\pi) = \infty$.

The theorem then states that for two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, satisfying some very general assumptions, if $\text{MD}(\mathcal{L}_1) > \text{MD}(\mathcal{L}_2)$ then there does not exist a *valid* encoding of $\mathcal{L}_1$ into $\mathcal{L}_2$. Both the assumptions and the criteria for encoding validity are described in [14], but it suffices to note that they are satisfied by all calculi we shall consider here. Thus the 'tower of expressiveness' of Carbone and Maffeis [7] also follows immediately, since

$$\text{MD}(\pi_1) < \ldots < \text{MD}(\pi_n) < \text{MD}({}^e\pi)$$

where $\pi_1 = \pi$, and hence that the π-calculus cannot encode the ${}^e\pi$-extension.

But the ρ-calculus *can* encode it. By using a suitable convention on name generation, like the one we define in [2], we can encode ${}^e\pi$-like name composition

$$x_1 \cdot x_2 \cdot \ldots \cdot x_n$$

in the ρ-calculus by composing the processes within the names. Assume that X_i is the process within the x_i'th name. Then we could define

$$\ulcorner X_1 \urcorner \cdot \ulcorner X_2 \urcorner \cdot \ldots \cdot \ulcorner X_n \urcorner \triangleq \ulcorner \prod_{i=1}^{n} X_i \urcorner = \ulcorner X_1 \mid X_2 \mid \ldots \mid X_n \urcorner$$

which can then be extended to vectors of arbitrary length. This composition can be performed at runtime through the ρ-calculus' lift operator, and we can thus encode ${}^{e}\pi$ input and output operations with n-ary subjects as follows:

$$[\![x_1 \cdot \ldots \cdot x_n(y).P]\!]_a \triangleq a\langle\!|\ulcorner x_1 \urcorner \mid \ldots \mid \ulcorner x_n \urcorner|\!\rangle \mid a(v).v(y).[\![P]\!]_{a+}$$
$$[\![\overline{x_1 \cdot \ldots \cdot x_n}\!<\!z\!>]\!]_a \triangleq a\langle\!|\ulcorner x_1 \urcorner \mid \ldots \mid \ulcorner x_n \urcorner|\!\rangle \mid a(v).v\langle\!|\ulcorner z \urcorner|\!\rangle$$

where we assume all the names x_i are implemented as quoted processes $\ulcorner X_i \urcorner$; and where the parameter a is an internal name that is chosen fresh for each translation, and $a+$ is derived from a by a suitable convention of name incrementation, that is ensured to never cause a name clash (e.g. the method we describe in [2]).

The above argument is merely a sketch of a formal proof of separation, but its implications should be immediately clear: there cannot be an encoding of the ρ-calculus into the π-calculus, because if such an encoding existed, then it could be composed with the above encoding to yield an encoding of ${}^{e}\pi$ into the π-calculus, in contradiction of the aforementioned theorem by Gorla [14].

1.5 Typing reflection in higher-order Ψ-calculi

As previously noted, a plethora of type systems have been developed for the π-calculus, to capture many different kinds of safety-features. If we could simply adapt them to the ρ-calculus, then our task would be easy: indeed, we used this approach in [2], where we, as a means to adapt a π-calculus type system to the ρ-calculus, also attempted to define a first-order form of ρ-calculus with inspiration from the Fusion calculus [29]. However, it turned out to be little else than the π-calculus in disguise, and, not surprisingly in light of the previous discussion, it failed to encode the ρ-calculus. Following the argument above, we cannot hope to be able to type the ρ-calculus with a π-calculus type system by simply encoding the ρ-calculus into the π-calculus and then typing the result, because the ρ-calculus is *not* just the π-calculus in disguise. We shall need a different approach.

Bengtson et al. [3, 4] define Ψ-*calculi* as a generalisation of various extensions of the π-calculus [26]: by choosing appropriate settings for a small number of parameters, a range of (first order) π-like calculi can be expressed as *instantiations* of the Ψ-calculus framework.[3] Of particular interest is here the ability to declare arbitrary assertions in the syntax, and to transmit *structured* data terms, rather than names alone: As these authors show, it enables them to represent, amongst other, the full ${}^{e}\pi$-calculus of Carbone and Maffeis [7], which we already know is beyond the representability of the π-calculus.

[3]Note that the authors of [3; 4] usually speak of Ψ-*calculi* in plural, when referring to the general framework. However, we find this usage to be unnecessarily unnatural and cumbersome, so in the following, when we speak of *the* Ψ-calculus (in definite singular), we refer not to any particular instantiation (which is *a* Ψ-calculus, in indefinite singular), but rather to the framework itself, i.e. the abstract calculus, with its abstract semantics without any parameter settings.

Furthermore, Hüttel [17] has created a *generic* type system for the first-order Ψ-calculus framework, that likewise generalises several of the type systems for the π-calculus and its many extensions. The generic type system can similarly be instantiated through parameter setting to yield both well-known and new type systems for the calculi that are representable as first-order Ψ-calculi, including Dπ by Hennessy and Riely [15]; the calculus of explicit fusions [13]; and not least the aforementioned ${}^{e}\pi$-calculus.

Parrow et al. [28] later extend the Ψ-calculus framework to also include *higher order* communication, thus creating the *Higher-Order* Ψ-calculi (HOΨ), which thereby further widens the range of calculi that can be expressed as a Ψ-calculus. For example, they show that both the CHOCS- and HOπ-calculi are representable as HOΨ-instantiations, as well as every calculus that the 'first-order' Ψ-calculus framework can represent. The authors of HOΨ never create an instantiation of the ρ-calculus in their examples, but we nevertheless suspect that it *is* representable therein, since the HOΨ-framework can encompass both the name-generating capabilities of ${}^{e}\pi$, *and* can extend these with higher-order characteristics.

Thus, to be able to type reflection through the HOΨ-framework, we shall therefore firstly have to create such an instantiation of the ρ-calculus, and secondly to extend the generic type system of Hüttel [17] to the HOΨ-calculus. Like its predecessor, our generic type system for the HOΨ-calculus should satisfy a subject reduction property, and furthermore, every *instance* of the generic type system should satisfy a given appropriate definition of safety, expressing (at least) a notion of absence of channel type errors.

However, unlike its predecessor, the generic type system must additionally be able to type higher-order behaviour, such that mobile processes may be restricted with respect to the names they are allowed to use, and whether or not they should be allowed to be re-executed at the reception point. Taken together, these two properties should allow us to capture both the name-generating and higher-order capabilities of the ρ-calculus.

1.6 Summary of our approach

Given the considerations above, the rest of the report is structured as follows: Chapter 2 reviews the syntax and labelled semantics of the higher-order Ψ-calculus. In chapter 3 we then give a reduction semantics for HOΨ to facilitate proofs for the generic type system, which we develop in chapter 5. Inbetween, chapter 4 concerns the development of an instantiation of ρ-calculus, which we arrive at through an analysis of several other instantiations of first- and higher-order calculi, and in chapter 6 we demonstrate how the generic type system may be instantiated to yield type systems for some of the aforementioned instantiations of the HOΨ-calculus. Lastly, in chapter 7 we conclude on our endeavours and discuss some alternative approaches to the problem of typing reflection.

2 The Higher-Order Ψ-calculus

Our purpose in the present chapter is to give a structured and concise presentation of the syntax and operational semantics of the higher-order Ψ-calculus. Thus in the following, we shall forgo the longer, motivating examples, some of which have been deferred to chapter 4, as well as the separate definition of the first-order Ψ-calculus. Instead we shall proceed directly to the definition of the higher-order Ψ-calculus, since it contains all of the former's syntax and semantics, plus the extension by Parrow et al. [28]. Many of the definitions in the following will therefore apply equally to both the first-order and higher-order variants, and we shall not distinguish explicitly between Ψ- and HOΨ-calculi, except when a definition applies solely to the higher-order variant.

2.1 Nominal datatypes

Ψ-calculi rely on a notion of *nominal datatypes*, which are *nominal sets* in the style of Gabbay and Pitts [12], equipped with a set of *equivariant* functions, which are functions that obey the restriction that they are invariant under *name swapping*. We shall briefly review these concepts in the following:

Definition 2.1 (Transposition function)**.** *We assume a countably infinite set of atomic names $\mathcal{N}$, ranged over by a, b, x, y etc. For any set $\mathcal{X}$ of structures in which these names may occur, with $X \in \mathcal{X}$, a* transposition function *is a function*

$$(\cdot, \cdot)\cdot\cdot : \mathcal{N} \times \mathcal{N} \times \mathcal{X} \to \mathcal{X}$$

written $(a, b)\cdot X$, that replaces each occurrence in X of a with b and vice versa.

A transposition function is thus a *reversible name swapping* function, i.e. such that

$$(a, b)\cdot\big((a, b)\cdot X\big) = X$$

As a simple example of such an $\mathcal{X}$, consider the set of n-tuples of $\mathcal{N}$ for any $n \in \mathbb{Z}$. An example of X could then be the tuple $\langle a, x, b, a\rangle$, which by application of the transposition function yields

$$(a, b)\cdot\langle a, x, b, a\rangle = \langle b, x, a, b\rangle$$

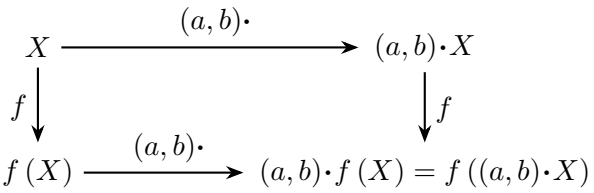

Figure 2.1: Commutativity diagram illustrating that f is an equivariant function.

Definition 2.2 (Equivariant function)**.** *A function $f : \mathcal{X} \to \mathcal{X}$ is* equivariant *if it holds for all $a, b \in \mathcal{N}$, and for all structures $X \in \mathcal{X}$ in which a, b may occur, that*

$$(a,b)\cdot f(X) = f((a,b)\cdot X)$$

In other words, the effect of f on some structure X is unaffected by name swapping, so we can draw the usual commutativity diagram for f w.r.t. transposition, cf. figure 2.1. The definition of equivariance is then extended to functions and relations of any arity.

Definition 2.3 (Nominal datatype)**.** *A* nominal set *is a set equipped with at least one transposition function. A* nominal datatype *is then a nominal set with a number of equivariant endofunctions defined on the set.*

Definition 2.4 (Support and freshness)**.** *A name a* occurs *in X if it is affected by transposition. The* support *of X, written $\mathrm{n}(X)$, is the set of names that occur in X. Formally*

$$\mathrm{n}(X) \triangleq \{\, a \in \mathcal{N} \mid \{\, b \in \mathcal{N} \mid (a,b)\cdot X \neq X \,\} \text{ is not finite} \,\}$$

i.e. $a \notin \mathrm{n}(X)$ iff $(a,b)\cdot X = X$ is true for all but finitely many $b \in \mathcal{N}$. A name a is fresh for *X, written $a\#X$, if $a \notin \mathrm{n}(X)$. This is then extended to* sets *of names A such that $A\#X$ if it is the case that $\forall a \in A.a \notin \mathrm{n}(X)$.*

See Gabbay and Pitts [12, p. 345] for further details on this definition. It is worth noting that for a structure such as the aforementioned tuple

$$T = \langle a, x, b, a \rangle$$

the support of T is (obviously) just the set of *all* names that occur syntactically in T, so $\mathrm{n}(T) = \{\, a, b, x \,\}$. However, in other structures where notions of binders and α-equivalence exist, such as e.g. terms of the λ-calculus, the support of such a structure will only correspond to the *free* names, because any bound name could always be α-converted to some other name. Thus, since

$$\lambda x.e \equiv_\alpha \lambda y.e\,\{y/x\} \qquad \text{and} \qquad x \notin \mathrm{n}(\lambda y.e\,\{y/x\})$$

then it should also be the case that $x \notin \mathrm{n}(\lambda x.e)$.

Lastly, for the purpose of defining Ψ-calculi it is required that there exists an equivariant *substitution function* defined on the nominal datatype $\mathcal{X}$, and parametrised with a substitution σ mapping names from $\mathcal{N}$ to structures of a set $\mathcal{Y}$:

Definition 2.5 (Substitution)**.** *Let $\mathcal{Y}$ be a set of structures, with $\mathcal{N} \subseteq \mathcal{Y}$. A* substitution *$\sigma$ is a (non-injective) function*

$$\sigma : \mathcal{N}^n \to \mathcal{Y}^n$$

mapping vectors of names $\widetilde{x} \in \mathcal{N}^n$ to vectors of structures $\widetilde{Y} \in \mathcal{Y}^n$ of equal arity. We shall often only write the non-trivial part of a substitution as $[\widetilde{x} := \widetilde{Y}]$, which is the substitution that maps each element $x_i \in \widetilde{x}$ to the corresponding element $Y_i \in \widetilde{Y}$ and all other names to themselves.

This definition is slightly more complicated than what is commonly seen in process calculi, because the Ψ-calculus framework does not require its nominal datatypes to *be* sets of names themselves. Hence, to ensure that σ is well-defined for all $x \in \mathcal{N}$ we let $\mathcal{Y}$ contain the set of names.

Definition 2.6 (Substitution function)**.** *A* substitution function

$$(\cdot)\,[\cdot := \cdot] : \mathcal{X} \times (\mathcal{N}^n \to \mathcal{Y}^n) \to \mathcal{X}$$

on a nominal datatype $\mathcal{X}$, written $X[\widetilde{a} := \widetilde{Y}]$, is an equivariant endofunction on $\mathcal{X}$, parametrised with a substitution σ.

In the above definition, each Y_i *can* be of the same type as X, but it is not required. Now, rather than defining the substitution function explicitly, it is enough to define a few requirements that any substitution function must satisfy:

Definition 2.7 (Substitution laws)**.** *The substitution function must then be defined such that it satisfies the following* substitution laws:

1. *If $\widetilde{a} \subseteq \mathrm{n}(X)$ and $b \in \mathrm{n}\big(\widetilde{Y}\big)$ then $b \in \mathrm{n}\Big(X[\widetilde{a} := \widetilde{Y}]\Big)$*
2. *If $\widetilde{u}\#X, \widetilde{v}$ then $X[\widetilde{v} := \widetilde{Y}] = \big((\widetilde{u}, \widetilde{v})\cdot X\big)[\widetilde{u} := \widetilde{Y}]$*

The requirements are quite general and should be satisfied by any ordinary definition of substitution: The first law states that names cannot be lost in substitution, i.e. the names present in $\widetilde{Y}$ must also be present when the substitution has been performed; whilst the second law states that substitution cannot be affected by transposition.

2.2 Parameters

As previously mentioned, the Ψ-calculus is a general framework for expressing many different kinds of π-like calculi. This is done by setting a few *parameters*, consisting of three nominal datatypes and four equivariant operators defined on the datatypes.

Definition 2.8 (Ψ-calculus datatypes)**.** *Any Ψ-calculus requires a specification of three nominal datatypes,* terms, conditions *and* assertions *denoted by the following sets and metavariables:*

$$\begin{aligned} M, N &\in \mathbb{T} && \textit{data terms} \\ \varphi &\in \mathbb{C} && \textit{conditions} \\ \Psi &\in \mathbb{A} && \textit{assertions} \end{aligned}$$

The three sets are not necessarily disjoint. Note further that the definition allows data terms (channels) to be structured objects of arbitrary complexity. Thus, they could be just single names, as in the monadic π-calculus, or vectors of names as in the polyadic π-calculus, or even whole terms of another language such as the λ-calculus.

Definition 2.9 (Ψ-calculus parameters)**.** *Any Ψ-calculus requires a definition of two equivariant operators,* channel equivalence $\dot{\leftrightarrow}$ *and* assertion composition $\otimes$*, a* unit element $\mathbf{1}$ *of assertions, and an* entailment relation $\Vdash$*, defined on the respective nominal datatypes and with the following signatures:*

$$\begin{aligned} \dot{\leftrightarrow} &: \mathbb{T} \times \mathbb{T} \to \mathbb{C} && \textit{channel equivalence} \\ \otimes &: \mathbb{A} \times \mathbb{A} \to \mathbb{A} && \textit{assertion composition} \\ \mathbf{1} &\in \mathbb{A} && \textit{assertion unit} \\ \Vdash &\subseteq \mathbb{A} \times \mathbb{C} && \textit{entailment relation} \end{aligned}$$

where we write the entailment relation as $\Psi \Vdash \varphi$ to denote the conditions φ entailed by the assertions Ψ. Lastly, a substitution function $(\cdot)\,[\widetilde{a} := \widetilde{M}]$*, substituting n-ary term tuples $\widetilde{M}$ for n-ary name tuples $\widetilde{a}$, must be defined on each of the three sets $\mathbb{T}, \mathbb{C}, \mathbb{A}$:*

$$\begin{aligned} (\cdot)\,[\cdot := \cdot] &: \mathbb{T} \times (\mathcal{N}^n \to \mathbb{T}^n) \to \mathbb{T} \\ (\cdot)\,[\cdot := \cdot] &: \mathbb{C} \times (\mathcal{N}^n \to \mathbb{T}^n) \to \mathbb{C} \\ (\cdot)\,[\cdot := \cdot] &: \mathbb{A} \times (\mathcal{N}^n \to \mathbb{T}^n) \to \mathbb{A} \end{aligned}$$

At this point we should also note a difference between our presentation and the one given by Parrow et al. [28] for the HOΨ-calculi. In this extension, the authors introduce a fourth set called *clauses*, which are of the form $M \Leftarrow P$, pronounced 'M is a *handle* for P.' Clauses are entailed by assertions, just like conditions, but in [28] the authors decide to maintain a distinction between the sets of clauses and conditions, although it makes no practical difference. However, this decision then necessitates that they overload the entailment relation, such that it is defined for both conditions and clauses. We find such excessive use of overloading unnecessarily confusing, and we shall therefore in the following treat clauses as a just a subset of conditions. Thus in the sequel we shall just assume that the set $\mathbb{C}$ of conditions be defined in such a way that

$$\{\, M \Leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{P}_\Psi \,\} \subseteq \mathbb{C}$$

where $\mathcal{P}_\Psi$ is the set of Ψ-calculus processes, *if* the framework is to be used for instantiating higher-order calculi.

For any particular choice of the aforementioned parameters to form a *valid* Ψ-calculus, it is also required that $\dot{\leftrightarrow}$ be a *partial equivalence relation* (PER), i.e. symmetric and transitive but not necessarily reflexive; that $\otimes$ be compositional; and that $(\mathbb{A}_{/\simeq}, \otimes, \mathbf{1})$ be an abelian monoid. We express these requirements as follows:

Definition 2.10 (Assertion equivalence)**.** *Two assertions are* assertion equivalent*, written $\Psi_1 \simeq \Psi_2$ if it is the case that*

$$\forall \varphi. \Psi_1 \Vdash \varphi \iff \Psi_2 \Vdash \varphi$$

That is, two assertions are equivalent if they entail the same conditions. With this notion of assertion equivalence we can now specify the requirements for valid parameter settings:

Definition 2.11 (Valid Ψ-calculus parameters)**.** *An instantiation is a* valid Ψ*-calculus if the following conditions all hold:*

1. *Channel equivalence is symmetric and transitive:*

$$[\textsc{ceq-sym}]\frac{\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} N}{\Psi \Vdash N \mathrel{\dot{\leftrightarrow}} M} \qquad [\textsc{ceq-trans}]\frac{\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} N \quad \Psi \Vdash N \mathrel{\dot{\leftrightarrow}} L}{\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} L}$$

2. *Assertion composition is compositional:*

$$[\textsc{ass-comp}]\frac{\Psi_1 \simeq \Psi_2}{\Psi_1 \otimes \Psi' \simeq \Psi_2 \otimes \Psi'}$$

3. $(\mathbb{A}_{/\simeq}, \otimes, \mathbf{1})$ *is an abelian monoid:*

$$\begin{array}{ll} [\textsc{ass-ident}] & \Psi \otimes \mathbf{1} \simeq \Psi \\ [\textsc{ass-assoc}] & (\Psi_1 \otimes \Psi_2) \otimes \Psi_3 \simeq \Psi_1 \otimes (\Psi_2 \otimes \Psi_3) \\ [\textsc{ass-comm}] & \Psi_1 \otimes \Psi_2 \simeq \Psi_2 \otimes \Psi_1 \end{array}$$

2.3 Syntax

With the definition of the three nominal sets, *terms, conditions*, and *assertions*, in place, we can now move on to defining the syntax of HOΨ-calculus processes (or agents):

Definition 2.12 (HOΨ-calculus processes)**.** *The set of* HOΨ*-calculus* processes *$\mathcal{P}_\Psi$, ranged over by P, Q etc., are generated by the following formation rules:*

$$
\begin{array}{rll}
P, Q \in \mathcal{P}_\Psi ::= & \mathbf{0} & \textit{Nil} \\
\mid & P \mid Q & \textit{Parallel} \\
\mid & \overline{M}N.P & \textit{Output} \\
\mid & \underline{M}(\lambda \widetilde{x})N.P & \textit{Input} \\
\mid & \mathbf{run}\ M & \textit{Invocation} \\
\mid & \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} & \textit{Selection} \\
\mid & (\boldsymbol{\nu} x)\, P & \textit{Restriction} \\
\mid & !P & \textit{Replication} \\
\mid & (\!|\Psi|\!) & \textit{Assertion}
\end{array}
$$

where $\widetilde{\varphi} : \widetilde{P}$ *is a shorthand:*

$$\widetilde{\varphi} : \widetilde{P} \triangleq \varphi_1 : P_1 \ [] \ \ldots \ [] \ \varphi_n : P_n$$

2.3.1 Comments on the syntax

Some of the syntactic constructs are similar to those found in the π-calculus [26] and other π-like calculi: In particular, $\mathbf{0}$ is the inactive process; $P \mid Q$ is the parallel composition of P and Q; $(\boldsymbol{\nu} x)\, P$ restricts the visibility of the name x to P, and $!P$ is replication of P. Lastly, $(\!|\Psi|\!)$ is included to allow assertions to appear in the syntax, such that they can become enabled during the course of program execution. The remaining constructs are mostly generalisations, and we shall comment on each in greater detail below:

The *output* construct $\overline{M}N.P$ sends the object data term N along the channel named M and continues as P, just as in the π-calculus. The main difference is that both the subject M, and the object N, can be *structured data terms*, and not merely atomic names. Thus, both subject and object could be e.g. vectors of names $\langle x, y, z \rangle$, or even complex terms of another language entirely.

The *input* construct $\underline{M}(\lambda \widetilde{x})N$ receives along M a structured term, e.g. K, and this K is then matched against the pattern N, where $(\lambda \widetilde{x})$ is the argument list. For example, the ordinary π-calculus input construct $x(y).P$ can be expressed in this format as $\underline{x}(\lambda y)y$, since all terms consist of just a single name each. The polyadic π-calculus input $x(\widetilde{y}).P$ can similarly be expressed as $\underline{x}(\lambda \widetilde{y})\widetilde{y}.P$ by merely letting terms be tuples of any arity.

The *selection* construct $\mathbf{case}\ \varphi_1 : P_1 \ [] \ \ldots \ [] \ \varphi_n : P_n$ selects *one* P_i process for which the corresponding condition φ_i is entailed by the assertions in effect, and discards the rest. If more than one φ_i is entailed by the assertions, then the P_i is chosen non-deterministically. This too is a generalisation; it combines the π-calculus' match and mismatch operators and the various other choice constructs, which can all be expressed in this general format. Notably:

- If there is only one condition φ and one process P, then it corresponds to an if-then construct:

$$\mathbf{if}\ \varphi\ \mathbf{then}\ P \triangleq \mathbf{case}\ \varphi : P$$

- If conditions such as $a = b$ and $a \neq b$ are included in $\mathbb{C}$, then, given the aforementioned encoding of if-then, then match and mismatch can be expressed as

$$[a = b]P \triangleq \textbf{if } a = b \textbf{ then } P \qquad \text{and} \qquad [a \neq b]P \triangleq \textbf{if } a \neq b \textbf{ then } P$$

- If there exists a condition $\top$ such that $\forall \Psi . \Psi \Vdash \top$, i.e. $\top$ is true for all assertions, then the π-calculus' non-deterministic choice $P + Q$ can be expressed as

$$P + Q = \textbf{case } \top : P \; [] \; \top : Q$$

Lastly, the *invocation* construct **run** M will run the process P if M is a *handle* for P, written $M \Leftarrow P$, and this condition is entailed by the assertions in effect. This construct is the sole extension to the syntax, introduced by Parrow et al. [28], to create the *higher-order* Ψ-calculus from the (first-order) Ψ-calculus.

2.3.2 Criteria for well-formedness

Not all processes generated by the syntax in definition 2.12 will be meaningful. For example, a replicated assertion $!(\!|\Psi|\!)$ is allowed, even though it cannot do anything. We shall therefore introduce a few restrictions on the forms of processes that we shall allow:

Definition 2.13 (Prefix, subject, object)**.** *A* prefix π *is either an input or an output construct:*

$$\pi ::= \underline{M}(\lambda \widetilde{x})N \;\; \Big| \;\; \overline{M}N$$

and for either kind of prefix, we say that M *is the* subject *and* N *is the* object *of the prefix action.*

Definition 2.14 (Assertion guarded processes)**.** *An assertion* $(\!|\Psi|\!)$ *is* guarded *if it occurs under a prefix* π*: If* $(\!|\Psi|\!)$ *is a subterm of* P *then* $(\!|\Psi|\!)$ *is guarded in* $\pi.P$*. A process is* assertion guarded *if all its assertion subterms are guarded. Assertion guarded processes* G *are hence built by the formation rules:*

$$G ::= \mathbf{0} \;\; \Big| \;\; G_1 \mid G_2 \;\; \Big| \;\; \textbf{case } \widetilde{\varphi} : \widetilde{G} \;\; \Big| \;\; \pi.P \;\; \Big| \;\; (\boldsymbol{\nu} x)\, G \;\; \Big| \;\; !G \;\; \Big| \;\; \textbf{run } M$$

Definition 2.15 (Well-formed processes)**.** *A process is* well-formed *if it satisfies all of the following criteria:*

- $\widetilde{x} \subseteq \mathrm{n}(N)$ *in* $\underline{M}(\lambda \widetilde{x})N$ *is a sequence without duplicates.*
- *Every replication is assertion guarded:* $!G$
- *Every choice is assertion guarded:* **case** $\widetilde{\varphi} : \widetilde{G}$
- *Every handle is assertion guarded:* $M \Leftarrow G$
- *Every handle for* P *must contain* P*'s names:* $M \Leftarrow P \implies \mathrm{n}(P) \subseteq \mathrm{n}(M)$

The last criterion in particular may require some further explanation: It states that when a term M is a handle for the process P, then M must contain *at least* all the names occurring in P, and possibly more. This ensures that any restriction binding names in P will also bind the same names in M. Thus, if M is passed around, these scopes must firstly be extruded to encompass the reception sites, and *this*, in turn, ensures that if M is used in a **run** M term, leading to P being executed, then all restricted names in P will still be restricted at the execution site. In other words, this criterion merely ensures that any restricted names in P cannot suddenly become unrestricted when P is run at some other site that (initially) may have been outside the scope of the restriction.

2.4 Labelled semantics

Both the original presentation of (first order) Ψ-calculi by Bengtson et al. [3] and Bengtson et al. [4], and the later (higher order) extension by Parrow et al. [28], on which we also base our presentation, all give the semantics in terms of a labelled transition system. Here, we shall essentially repeat their presentation in the following, with only some minor clarifications. We shall firstly need the notion of a *frame*, which is an environment consisting of the restrictions and assertions in effect, in which the process executes:

Definition 2.16 (Frame of a process)**.** *The* frame *of a process consist of the set of all top-level restrictions and the composition of all unguarded assertions of the process. Formally, we define the frame of a process P by the two recursive functions $\mathcal{F}_\nu(P)$ and $\mathcal{F}_\Psi(P)$ that collect the top level restrictions resp. assertions in the expected way. The relevant clauses are:*

$$\begin{aligned} \mathcal{F}_\Psi(P \mid Q) &\triangleq \mathcal{F}_\Psi(P) \otimes \mathcal{F}_\Psi(Q) & \qquad \mathcal{F}_\nu(P \mid Q) &\triangleq \mathcal{F}_\nu(P) \cup \mathcal{F}_\nu(Q) \\ \mathcal{F}_\Psi((\boldsymbol{\nu} x)\, P) &\triangleq \mathcal{F}_\Psi(P) & \qquad \mathcal{F}_\nu((\boldsymbol{\nu} x)\, P) &\triangleq \{\, x \,\} \cup \mathcal{F}_\nu(P) \\ \mathcal{F}_\Psi(\llparenthesis \Psi \rrparenthesis) &\triangleq \Psi \end{aligned}$$

and with all remaining clauses of the forms:

$$\mathcal{F}_\Psi(P) \triangleq \mathbf{1} \qquad \textit{and} \qquad \mathcal{F}_\nu(P) \triangleq \emptyset$$

respectively. Frames are identified up to α-equivalence, and in the case of parallel composition $\mathcal{F}(P \mid Q)$ it is furthermore required that

$$\mathcal{F}_\nu(Q) \,\#\, \mathcal{F}_\nu(P)\,, \mathcal{F}_\Psi(P) \qquad \textit{and} \qquad \mathcal{F}_\nu(P) \,\#\, \mathcal{F}_\nu(Q)\,, \mathcal{F}_\Psi(Q)$$

i.e. that the restricted names in P must be fresh for the whole frame of Q, and conversely for the restricted names in Q.

Note that the format of this definition differs markedly from the presentation by Bengtson et al. [3, 4]; Parrow et al. [28], although the results are equivalent. The aforementioned authors would write $\mathcal{F}(P) = (\boldsymbol{\nu} \widetilde{x}_P)\, \Psi_P$ for the frame of P,

where we instead write $\mathcal{F}_\nu(P), \mathcal{F}_\Psi(P)$, but the difference is purely syntactic. The benefit of their notation is that it suggests that the restrictions $(\boldsymbol{\nu}\widetilde{x}_P)$ bind into the assertions Ψ_P (which they do). Thus, frames (and particularly assertions) are identified by α-equivalence, and α-conversion may be required to ensure that name clashes cannot occur in frames.

Compared to the above, our notation is less suggestive, although the same considerations w.r.t. α-equivalence and α-convertibility still apply. However, our purpose in choosing this alternative notation is twofold: Firstly, it means that we, unlike the aforementioned authors, can avoid overloading the $\otimes$ operator for frames (cf. e.g. [28, p. 7]); and secondly, for any P it will allow us to refer to the set of restricted names $\widetilde{x}$ and the assertions Ψ separately, by the functions $\mathcal{F}_\nu(P)$ and $\mathcal{F}_\Psi(P)$ respectively. The utility of this shall become apparent below, in our presentation of the semantics for the Ψ-calculus:

Definition 2.17 (Ψ-calculus actions)**.** *The set of Ψ-calculus* actions *$\mathcal{A}$ (or labels), ranged over by α, is defined by the following syntax:*

$$\alpha \in \mathcal{A} ::= \overline{M}\,(\boldsymbol{\nu}\widetilde{x})\,N \;\mid\; \underline{M}N \;\mid\; \tau$$

where $\overline{M}\,(\boldsymbol{\nu}\widetilde{x})\,N$ denotes sending, $\underline{M}N$ denotes reception, and τ is an internal action. For both input and output it must be the case that $\widetilde{x} \subseteq \mathrm{n}(N)$. For the output label, $(\boldsymbol{\nu}\widetilde{x})$ represents name binding, and the bound names $\mathrm{bn}\,(\alpha)$ *of a label α is defined in the obvious way:*

$$\mathrm{bn}\,(\alpha) = \begin{cases} \{\,\widetilde{x}\,\} & \text{if } \alpha = \overline{M}\,(\boldsymbol{\nu}\widetilde{x})\,N \\ \emptyset & \text{otherwise} \end{cases}$$

Definition 2.18 (HOΨ-calculus labelled semantics)**.** *The semantics of higher order Ψ-calculi is given in terms of a labelled transition system, where transitions are of the form*

$$\Psi \rhd P \xrightarrow{\alpha} P'$$

and the transition relation $\cdot \rhd \cdot \xrightarrow{\alpha} \cdot$ *is given by the following rules:*

$$[\textsc{Case}]\frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \quad \Psi \Vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \varphi_1 : P_1\ []\ \ldots\ []\ \varphi_n : P_n \xrightarrow{\alpha} P'}$$

$$[\textsc{Open}]\frac{\Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P'}{\Psi \rhd (\boldsymbol{\nu}z)\,P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x}z)N} P'} \left(\begin{array}{c} z\#\widetilde{x}, \Psi, M \\ z \in \mathrm{n}(N) \end{array} \right)$$

$$[\textsc{Com}]\frac{\begin{array}{c}\mathcal{F}_\Psi(P) \otimes \mathcal{F}_\Psi(Q) \otimes \Psi \Vdash M \leftrightarrow K \\ \mathcal{F}_\Psi(Q) \otimes \Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P' \\ \mathcal{F}_\Psi(P) \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{x})\,(P' \mid Q')} \left(\begin{array}{c} \widetilde{x}\#Q \\ \mathcal{F}_\nu(P)\,\#\mathcal{F}_\nu(Q), Q, M, \Psi \\ \mathcal{F}_\nu(Q)\,\#\mathcal{F}_\nu(P), P, M, \Psi \end{array} \right)$$

$$[\textsc{In}]\frac{\Psi \Vdash M \dot{\leftrightarrow} K}{\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P \xrightarrow{\underline{K}N[\widetilde{x}:=\widetilde{L}]} P[\widetilde{x} := \widetilde{L}]} \qquad [\textsc{Out}]\frac{\Psi \Vdash M \dot{\leftrightarrow} K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}(\boldsymbol{\nu}\epsilon)N} P}$$

$$[\textsc{Res}]\frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\boldsymbol{\nu}z)\,P \xrightarrow{\alpha} (\boldsymbol{\nu}z)\,P'}\ (z\#\alpha, \Psi) \qquad [\textsc{Rep}]\frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

$$[\textsc{Par}]\frac{\mathcal{F}_\Psi(Q) \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \begin{pmatrix}\mathrm{bn}\,(\alpha)\,\#Q \\ \mathcal{F}_\nu(Q)\,\#\alpha, P, \Psi\end{pmatrix} \qquad [\textsc{Run}]\frac{\Psi \Vdash M \Leftarrow P \quad \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd \mathbf{run}\ M \xrightarrow{\alpha} P'}$$

We omit the symmetric forms of the [Par] and [Com] rules. Note also that both frames, processes and transitions are (implicitly) identified up to α-equivalence. Hence, we can use α-conversion in the premises to choose new bound names in order to satisfy the freshness criteria listed in the side conditions.

We have made a few cosmetic changes of the rules, compared to the presentation found in [28]. Most notably, in the rule [Out] we use the label $\overline{K}\,(\boldsymbol{\nu}\epsilon)\,N$ where $(\boldsymbol{\nu}\epsilon)$ represents the empty list of restricted names.[1] We also consistently use the frame functions $\mathcal{F}_\nu(P)$ and $\mathcal{F}_\Psi(P)$ in side conditions to refer to the two parts of the frame of a process P.

2.4.1 Comments on the rules

The transition rules in definition 2.18 are clearly of a highly abstract nature, and hence not necessarily clearly understandable. There are a number of subtle points, especially regarding the freshness conditions on some of the rules, and the interplay between the parameter settings from definition 2.9, so we shall comment on each rule in some detail below. Notice firstly that all transitions are of the form

$$\Psi \rhd P \xrightarrow{\alpha} P'$$

Here $\Psi \rhd$ represents an environment of active assertions in which the transition itself takes place. New assertions may also appear in the syntax, as $(\!|\Psi|\!)$, and if they are free (i.e. do not stand under a prefix), then they will be collected in the [Par] and [Com] rules, through application of the frame function $\mathcal{F}_\Psi(\cdot)$ from definition 2.16, and composed with the active assertions in the premises of these rules. For example in the [Par] rule (where we omit the side conditions for clarity):

$$[\textsc{Par}]\frac{\mathcal{F}_\Psi(Q) \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

the idea is that free assertions $(\!|\Psi_1|\!) \mid \ldots \mid (\!|\Psi_n|\!)$ may appear in Q, and these will also form part of the environment for P; they may even entail conditions that

[1] In contrast, Parrow et al. [28] write $\overline{K}N$, but this label format is actually not included in the set of actions given in definition 2.17.

enable P's transition. Therefore, they are collected by the frame function $\mathcal{F}_\Psi(Q) = \Psi_1 \otimes \ldots \otimes \Psi_n$ and composed with the other active assertions in the premise

$$\mathcal{F}_\Psi(Q) \otimes \Psi \rhd P \xrightarrow{\alpha} P'$$

The rule may have to be applied several times (or the [Com] rule), each time collecting new free assertions. Thus the active assertions will increase towards the leaves in the derivation tree for a given transition. Notice also that assertions $(\!|\Psi|\!)$ are never *removed* in transitions, and since previously guarded assertions may become free after a transition, the number of free assertions will therefore also increase during the course of program execution.

A second characteristic feature of this semantics, which sets it apart from many other 'conventional' labelled semantics for process calculi, is the explicit use of freshness criteria in the side conditions of several rules. This is necessary because the set $\mathcal{N}$ of atomic names can be used in the definition of any or all of the three nominal datatypes $\mathbb{T}, \mathbb{C}$ and $\mathbb{A}$ from definition 2.8. Hence, the usual assumption, that a name x in e.g. a restriction $(\boldsymbol{\nu} x)$ P is always fresh, may not necessarily hold for the whole program, and instead the freshness conditions make it clear where x at least has to be chosen such that it is *locally* fresh to ensure that a name clash does not occur. We shall see some examples of this in the following, when we comment on the individual rules. Consider firstly the [Case] rule:

$$[\text{Case}]\frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \quad \Psi \Vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \varphi_1 : P_1\ [\!]\ \ldots\ [\!]\ \varphi_n : P_n \xrightarrow{\alpha} P'}$$

Here we select (possibly non-deterministically) one P_i from the list, for which the condition φ_i is entailed by the active assertions Ψ, *and* this P_i must furthermore be able to do a transition $\xrightarrow{\alpha}$. Both criteria must be satisfied, so if a process has no available transitions, it cannot be selected, even if its condition is entailed. Thus, even if we assume $\top$ is a condition that is entailed by all assertions, then a process

$$\mathbf{case}\ \top : (\boldsymbol{\nu} a)\, \overline{a} x.\mathbf{0}$$

will not have any transitions. To see this, consider next the [Res] rule:

$$[\text{Res}]\frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\boldsymbol{\nu} z)\, P \xrightarrow{\alpha} (\boldsymbol{\nu} z)\, P'}\ (z \# \alpha, \Psi)$$

Here, the freshness condition states that z must be fresh for both α and Ψ. Hence, if P was an output with restriction $(\boldsymbol{\nu} z)\, \overline{z} x$, as in the example above, then α were to have been an output label $\overline{z}\, (\boldsymbol{\nu} \epsilon)\, x$, which would then violate the freshness condition, since it does not hold that $z \# \overline{z}\, (\boldsymbol{\nu} \epsilon)\, x$. This is intuitively what we should expect, since z is a restricted name, but the freshness condition makes this intuition explicit.

Secondly, this also means that the action α concluded using this rule must either be an input or output involving some *other* name(s), not under restriction, or be an

internal communication, in which case the label is τ. If instead the action *is* an output of the restricted name, it must be concluded by application of the [Open] rule:

$$[\textsc{Open}]\frac{\Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P' \quad \left(\begin{array}{l} z\#\widetilde{x}, \Psi, M \\ z \in \mathrm{n}(N) \end{array}\right)}{\Psi \rhd (\boldsymbol{\nu} z)\, P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x}z)N} P'}$$

The purpose of this rule is to open the scope of $(\boldsymbol{\nu} z)$ by adding it to the list of restricted names in the label in the conclusion. Thus, if P can do a transition

$$\Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P'$$

in the premise, then z is added to the label $\overline{M}\,(\boldsymbol{\nu}\widetilde{x}z)\,N$ in the conclusion. Thus, if N contained a vector of n restricted names $(\boldsymbol{\nu}\widetilde{x})$, then this rule would be applied repeatedly, up to n times, to build the list of restricted names in the label $\overline{M}\,(\boldsymbol{\nu}\widetilde{x})\,N$.

We note in passing, that this is another point where our notation differs insignificantly from that of Parrow et al. [28], who instead write $\overline{M}\,(\boldsymbol{\nu}\widetilde{x} \cup \{\, z \,\})\,N$ to signify that z can be inserted *anywhere* in $\widetilde{x}$. We regard $\widetilde{x}$ as a list, rather than a set, so we find this implicit overloading of $\cup$ to be an unwarranted and unnecessary complication. Since the name z indeed can occur anywhere, we simply choose to always concatenate it to the end of the list.

Conversely, if there is only a single restricted name $(\boldsymbol{\nu} z)$, then the list of restricted names will be empty in the premise, so the label will be precisely $\overline{M}\,(\boldsymbol{\nu}\epsilon)\,N$, as used also in the [Out] rule:

$$[\textsc{Out}]\frac{\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}(\boldsymbol{\nu}\epsilon)N} P}$$

Here, the only other notable subtlety is that channel equivalence $\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} K$ is checked in the premise, and it is K that appears in the label's subject position, rather than M. Thus we must always choose an *equivalent* term for the label's subject, rather than M itself, *unless* $\mathrel{\dot{\leftrightarrow}}$ is also defined to be reflexive, which, as noted in definition 2.11, is allowed but not *required*. The check occurs in the premise of both the [In] and [Com] rules, which thus e.g. may allow the original M to be recovered, even if $\mathrel{\dot{\leftrightarrow}}$ is not reflexive. Bengtson et al. [4] give a longer example of why this may be necessary with some parameter settings, such as an instantiation of the Fusion Calculus [29], where assertions may declare arbitrary pairs of names to be equivalent during the course of program execution.

Apart from the aforementioned detail with $\mathrel{\dot{\leftrightarrow}}$, the [In] rule is fairly straightforward:

$$[\textsc{In}]\frac{\Psi \Vdash M \mathrel{\dot{\leftrightarrow}} K}{\Psi \rhd \underline{M}(\lambda\widetilde{x})N.P \xrightarrow{\underline{K}N[\widetilde{x}:=\widetilde{L}]} P[\widetilde{x} := \widetilde{L}]}$$

However, note the specification of pattern matching on the received object: The label $\underline{K}N[\widetilde{x} := \widetilde{L}]$ states that there must exist some vector $\widetilde{L}$ of terms, matching the arity of $\widetilde{x}$, *and* the received object must exactly match N with $\widetilde{x}$ substituted for $\widetilde{L}$. This is a somewhat convoluted way of expressing that the subterms $\widetilde{L}$, matching the pattern of $\widetilde{x}$ within N, are extracted from the received object, and then substituted into the continuation. For example, suppose a term could have an email address-like format $N_1@N_2\,.\,N_3$. If this term was received on M, then an input pattern of the form $\underline{M}(\lambda x_1, x_2, x_3)x_1@x_2\,.\,x_3.P$ would extract the three subterms N_1, N_2, N_3 and bind them to x_1, x_2, x_3 within the continuation P. Thus, the input construct can also be used to extract subterms from a term, as in e.g. macro programming with pattern matching. This also implies that *if* e.g. the composition

$$\underline{M_1}(\lambda\widetilde{x})N_1.P_1 \mid \overline{M_2}N_2.P_2$$

is able to communicate, *then* it must also be the case firstly that $M_1 \overset{\cdot}{\leftrightarrow} M_2$, and secondly that N_2 must match the pattern found in N_1. Both criteria must be satisfied if the communication is to be enabled.

The [Com] rule ties together all of the aforementioned [In], [Out] and [Open] rules:

$$[\textsc{Com}]\frac{\begin{array}{c}\mathcal{F}_\Psi(P)\otimes\mathcal{F}_\Psi(Q)\otimes\Psi \Vdash M \overset{\cdot}{\leftrightarrow} K \\ \mathcal{F}_\Psi(Q)\otimes\Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P' \\ \mathcal{F}_\Psi(P)\otimes\Psi \rhd Q \xrightarrow{\underline{K}N} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{x})\,(P' \mid Q')} \left(\begin{array}{c}\widetilde{x}\#Q \\ \mathcal{F}_\nu(P)\,\#\mathcal{F}_\nu(Q)\,, Q, M, \Psi \\ \mathcal{F}_\nu(Q)\,\#\mathcal{F}_\nu(P)\,, P, M, \Psi\end{array}\right)$$

Firstly, this rule closes any scopes $(\boldsymbol{\nu}\widetilde{x})$ that were opened by application of the [Open] rule in the derivation of the premise

$$\mathcal{F}_\Psi(Q)\otimes\Psi \rhd P \xrightarrow{\overline{M}(\boldsymbol{\nu}\widetilde{x})N} P'$$

which explains the appearance of the restriction $(\boldsymbol{\nu}\widetilde{x})\,(P' \mid Q')$ after the transition in the conclusion of this rule. Note that if there are no restricted names involved in the derivation, i.e. the label contains $(\boldsymbol{\nu}\epsilon)$, then obviously

$$(\boldsymbol{\nu}\epsilon)\left(P' \mid Q'\right) = P' \mid Q'$$

in the conclusion. Otherwise, since the names $\widetilde{x}$ came from P, the first freshness requirement in the side condition ensures that these names are also fresh for Q, thereby allowing the scope of $(\boldsymbol{\nu}x)$ to be extruded from P to encompass Q. This is similar to the usual capture-avoiding requirement that $x \notin \mathrm{fn}\,(Q)$, when a structural congruence axiom $(\boldsymbol{\nu}x)\,P \mid Q \equiv (\boldsymbol{\nu})\,(P \mid Q)$ is used to handle scope extrusion.

Then, in the premises, the unguarded assertions in Q are adjoined to the environment for the derivation of P's transition, and vice versa for Q, similar to the

[Par] rule, and lastly, *all* unguarded assertions from both P and Q are added to the environment to check the entailment of the channel equivalence.

The other two freshness criteria mirror each other: They state that neither P nor Q may bind any names in each other (which would cause a name clash), nor in the subject M (which would disable communication), nor in the environment Ψ. A subtlety here is that it is required for $\mathcal{F}_\nu(P)$ to be fresh for *both* Q *and* the restricted names $\mathcal{F}_\nu(Q)$ in Q (and vice versa for $\mathcal{F}_\nu(Q)$ w.r.t. P). This may seem redundant, but recall from definition 2.4 that freshness w.r.t. e.g. Q is defined in terms of the *support* $\mathrm{n}(Q)$, that precisely disregards the *bound* names. However, the frame function extracts the restricted names into two sets of names, which may then be compared. Thus, the requirement $\mathcal{F}_\nu(P) \# \mathcal{F}_\nu(Q)$ is merely a complicated way of stating that the intersection of the two sets of restricted names must be empty.

Lastly, the [Run] rule[2] enables a form of higher-order process mobility:

$$[\textsc{Run}]\frac{\Psi \Vdash M \Leftarrow P \quad \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd \mathbf{run}\ M \xrightarrow{\alpha} P'}$$

If $M \Leftarrow P$ and this is entailed by the active assertions, *and* P can do a transition to P', then $\mathbf{run}\ M \xrightarrow{\alpha} P'$. Thus, this rule allows processes to be bound to terms, that may be passed around, and then later used to invoke the process. This obviously places the restriction on P that it must be able to do at least *one* transition: $\mathbf{run}\ M$ cannot be used to invoke a deadlocked process.

Secondly, $M \Leftarrow P$ is a condition φ, so as mentioned in definition 2.11 it is assumed that the set $\mathbb{C}$ of conditions is defined such that

$$\mathbb{C} \triangleq \{\ldots\} \cup \{\, M \Leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{P}_\Psi \wedge \mathrm{n}(P) \subseteq \mathrm{n}(M) \,\}$$

where $\{\ldots\}$ represents the definition of other forms of conditions. Which processes are bound to which terms can then be controlled in the definition of the set $\mathbb{A}$ of assertions, as well as in the entailment relation $\Vdash$. Suppose for example that the definition of $\Vdash$ contains the rule

$$(M, P) \in \Psi \implies \Psi \Vdash M \Leftarrow P$$

and with $\mathbb{A}$ defined as the set of tuples $\mathbb{A} \triangleq \mathbb{T} \times \mathcal{P}_\Psi$ and $\otimes \triangleq \cup$. Then processes can be bound to terms during the course of program execution, by declaring the bindings as assertions ($\!|\{\ (M, P)\ \}|\!$). However, this is by no means the only possible parameter setting: Parrow et al. [28] give even more involved examples, for instance using parametrised clauses, but we defer them to chapter 4 where we shall give a more detailed account of the parameter settings necessary to instantiate some well-known first- and higher-order calculi.

[2] Note: Parrow et al. [28] named it [Invocation], but we preferred a shorter name.

* * *

In this chapter we have attempted to create a structured and succinct presentation of the fundamental definitions pertaining to nominal datatypes, and the the syntax and labelled semantics of (higher-order) Ψ-calculi. Notably, the semantics here presented were given in terms of a labelled transition system, which may be well-suited for some tasks, but less so for others. Thus, with these definitions in place we can now proceed to discuss developments of a reduction semantics for the HOΨ-calculus framework.

3 Reduction semantics

The presentations of the (first and higher order) Ψ-calculus framework by Bengtson et al. [3, 4] and Parrow et al. [28] all give their semantics in terms of labelled transition systems. As argued by e.g. Sangiorgi [31, p. 27], labelled semantics may be advantageous when we wish to reason about behavioural equivalences such as bisimulation, because it describes the behaviour of each process individually, irrespective of context.

On the other hand, a reduction system is usually simpler, by having fewer rules, because both the symmetric forms of [Com] and [Par] rules can be subsumed under structural congruence, and complementary interaction rules like [Input] and [Output] are are not used. This may simplify proofs by induction in the rules; for example invariance[1] proofs for type systems, which may be useful for our later endeavours.

Åman Pohjola [39] defines a reduction semantics for the first-order Ψ-calculus framework, but to our knowledge, there does not exist a reduction semantics for the higher-order Ψ-calculus. Our purpose in the present chapter is therefore to discuss the development of such a reduction semantics, which (as shall become apparent) is not an entirely trivial task.

3.1 Reduction with structural rules

The idea in a reduction semantics, as described by Milner [25], is to define the reduction relation $\rightarrow$ such that it coincides (preferably exactly) with the τ-labelled transitions $\xrightarrow{\tau}$ of the labelled semantics. Furthermore, the redex of a process calculus term, consisting usually of parallel compositions, will be distributed over the entire term, and hence the task of defining $\rightarrow$ may be simplified if it is defined not between pairs of individual processes, but between equivalences classes, induced by a structural congruence relation on the set of processes.

3.1.1 A first attempt

The task of defining $\rightarrow$ such that it matches $\xrightarrow{\tau}$ exactly may however not always be entirely trivial. Consider for example the following attempt: Assume we let struc-

[1] I.e. subject reduction/type preservation and the accompanying lemmas.

tural congruence $\equiv$ be defined in the usual way, as the least congruence containing α-equivalence; where $(\mathcal{P}_{\Psi/\equiv},\ \mid\ , \mathbf{0})$ is an abelian monoid; and such that it contains the following axioms:

$$\begin{aligned} !P &\equiv P \mid\ !P \\ (\boldsymbol{\nu} x)\, P \mid\ Q &\equiv (\boldsymbol{\nu} x)\,(P \mid\ Q) \text{ if } x\#Q \\ (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\, P &\equiv (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\, P \\ (\boldsymbol{\nu} x)\, \mathbf{0} &\equiv \mathbf{0} \end{aligned}$$

corresponding to most of the equational properties of bisimilarity for HOΨ, as proved by Parrow et al. [28, theorem 4.5]. Then suppose we let the reduction relation be defined by the following rules:

$$[\textsc{Com}]\frac{\Psi \Vdash M \overset{\cdot}{\leftrightarrow} K}{\Psi \rhd \overline{M} N[\widetilde{x} := \widetilde{L}].P \mid\ \underline{K}(\lambda \widetilde{x}) N.Q \to P \mid\ Q[\widetilde{x} := \widetilde{L}]}$$

$$[\textsc{Res}]\frac{\Psi \rhd P \to P'}{\Psi \rhd (\boldsymbol{\nu} x)\, P \to (\boldsymbol{\nu} x)\, P'}\ (x\#\Psi) \qquad [\textsc{Run}]\frac{\Psi \rhd P \to P' \quad \Psi \Vdash M \Leftarrow P}{\Psi \rhd \mathbf{run}\ M \to P'}$$

$$[\textsc{Struct}]\frac{P \equiv Q \quad \Psi \rhd Q \to Q' \quad Q' \equiv P'}{\Psi \rhd P \to P'}$$

$$[\textsc{Par}]\frac{\mathcal{F}_\Psi(Q) \otimes \Psi \rhd P \to P'}{\Psi \rhd P \mid\ Q \to P' \mid\ Q}\ (\mathcal{F}_\nu(Q)\, \#P, \Psi)$$

$$[\textsc{Case}]\frac{\Psi \rhd P_i \to P' \quad \Psi \Vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \varphi_1 : P_1\ []\ \ldots\ []\ \varphi_n : P_n \to P'}$$

These rules are certainly simpler than the labelled semantics of definition 2.18, but unfortunately also too simple: $\to$ is too small compared to $\xrightarrow{\tau}$. Specifically, the problems pertain to the [Case] and [Run] rules: In the premise of the proposed [Case] rule we require of the selected process P_i that

$$\Psi \rhd P_i \to P'$$

i.e. that it must be able to perform an entirely *internal* communication. The labelled semantics places no such restriction on P_i; there, it is enough that P can do *either* an input (inclusive) *or* an output. The problem with the proposed [Run] rule is similar.

One possible solution for [Case] could be to require that $\mathbb{C}$ and $\Vdash$ always be defined in such a way that there exists a condition $\top \in \mathbb{C}$ such that

$$\forall \Psi \in \mathbb{A}.\Psi \Vdash \top$$

i.e. $\top$ is entailed by all assertions. This imposes only a slight extra restriction on the generality of the framework. Then we could extend the definition of $\equiv$ with the

axiom

$$\textbf{case}\ \top : P \equiv P$$

which, when read from the right, allows us to rewrite any process term to a **case** that always holds. Clearly, if $P \xrightarrow{\tau} P'$ by the labelled semantics, then we also have that

$$P \to P' \iff \textbf{case}\ \top : P \to P'$$

by the present rules. We can then amend the [Com] rule as follows:

$$[\textsc{Com'}]\frac{\Psi \Vdash \varphi_1 \quad \Psi \Vdash \varphi_2 \quad P_1 \equiv \overline{M}N[\widetilde{x} := \widetilde{L}].P \mid P' \quad P_2 \equiv \underline{K}(\lambda\widetilde{x})N.Q \mid Q' \quad \Psi \Vdash M \leftrightarrow K}{\Psi \rhd (\textbf{case}\ \ldots\ []\ \varphi_1 : P_1\ []\ \ldots) \mid (\textbf{case}\ \ldots\ []\ \varphi_2 : P_2\ []\ \ldots) \to P \mid P' \mid Q[\widetilde{x} := \widetilde{L}] \mid Q'}$$

This rule relies on a kind of (implicitly defined) pattern matching on **case** terms, where all $\ldots$ represent (possibly empty) sequences of case expressions. Alternatively we could also extend $\equiv$ with commutative monoidal rules for **case**, by treating each case expression $\varphi : P$ as a pair (φ, P), with $(\top, \mathbf{0})$ as the identity element and $[]$ as the binary operator.

3.1.2 The unfolding problem

Unfortunately, even the revised [Com] rule is not enough to fully capture the $\xrightarrow{\tau}$ relation of the labelled semantics of HOΨ. There are still some cases which it cannot handle; notably the **run** M construct. Consider again the [Run] rule:

$$[\textsc{Run}]\frac{\Psi \rhd P \to P' \quad \Psi \Vdash M \Leftarrow P}{\Psi \rhd \textbf{run}\ M \to P'}$$

By this rule, P must perform an *internal* communication, if **run** M is allowed to reduce. This excludes the possibility that P might be able to reduce by communicating with another process in the *context* of **run** M instead. Assume, for the sake of simplicity, a π-calculus-like instantiation:

$$\emptyset \rhd \overline{x}z.P \mid \underline{x}(\lambda y)y.Q \xrightarrow{\tau} P \mid Q[y := z]$$

Intuitively, we would expect to be able to replace one (or both) of these terms with **run** M processes, but this is not the case. For example, choose any M such that $\mathrm{n}(\overline{x}z.P) \subseteq \mathrm{n}(M)$ to satisfy the requirement for names in handles. Then

$$\{\, M \Leftarrow \overline{x}z.P \,\} \rhd \textbf{run}\ M \mid \underline{x}(\lambda y)y.Q \not\to$$

because $\overline{x}z.P \not\to$, as the [Run] rule requires. Thus, the proposed [Run] rule cannot capture this behaviour of the corresponding labelled semantics. Furthermore, this problem propagates to the aforementioned revised [Com'] rule, because a **case** cannot be unfolded, if the inner process is a **run** M. Again, by the present [Com'] rule we have that

$$\emptyset \rhd (\textbf{case}\ \top : \overline{x}z.P) \mid (\textbf{case}\ \top : \underline{x}(\lambda y)y.Q) \xrightarrow{\tau} P \mid Q[y := z]$$

but no corresponding reduction is possible, if we rewrite one of the processes to a **run** M:

$$\{\, M \Leftarrow \overline{x}z.P \,\} \rhd (\textbf{case } \top : \textbf{run } M) \;\mid\; (\textbf{case } \top : \underline{x}(\lambda y)y.Q) \not\rightarrow$$

again because **run** M cannot on its own reduce, nor can it be converted to a prefixed form, as required in the [Com'] rule.

Intuitively, we might expect that this issue could be resolved by adding another axiom to structural congruence, along the lines of

$$\Psi \rhd \textbf{run } M \equiv P \qquad \text{if } \Psi \Vdash M \Leftarrow P$$

to allow **run** M to be unfolded, but, unfortunately, this will not work as expected, because it would allow *any* **run** M to unfold at any time, regardless of whether P can perform any reduction at all; for example, if $\tau.Q \xrightarrow{\tau} Q$ by any rule, then

$$\{\, M \Leftarrow \overline{x}z.P \,\} \rhd \textbf{run } M \;\mid\; \tau.Q \equiv \overline{x}z.P \;\mid\; \tau.Q \to \overline{x}z.P \;\mid\; Q$$

but no similar transition can be derived by the rules of the labelled semantics. This axiom is thus unsound.

3.1.3 A second attempt

Rather than trying to find separate solutions for the problems of **case** and **run** M separately, we might try to resolve both issues simultaneously: Firstly, we shall assume that the set $\mathbb{C}$ of conditions is closed under conjunction; i.e. that there exists a binary operator

$$\wedge : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$$

similar to logical $\wedge$, that may allow any two conditions φ_1 and φ_2 to be composed, such that

$$\Psi \Vdash \varphi_1 \wedge \varphi_2 \iff \Psi \Vdash \varphi_1 \wedge \Psi \Vdash \varphi_2$$

Then we add another axiom

$$\textbf{case } \ldots \;[]\; \varphi_i : \Big(\textbf{case } \widetilde{\varphi} : \widetilde{P}\Big) \;[]\; \ldots \equiv \textbf{case } \ldots \;[]\; \varphi_i \wedge \widetilde{\varphi} : \widetilde{P} \;[]\; \ldots$$

to structural congruence, to allow a hierarchy of nested **case** expressions to be flattened by composing the outer condition φ_i with each of the inner conditions in $\widetilde{\varphi}$; i.e. $\varphi_i \wedge \widetilde{\varphi} : \widetilde{P}$ is an abbreviation for

$$\varphi_i \wedge \varphi^1 : P^1 \;[]\; \ldots \;[]\; \varphi_i \wedge \varphi^n : P^n$$

where the superscripted φ^i and P^i denote the components of the inner $\widetilde{\varphi} : \widetilde{P}$ list.

Recall now from definition 2.9 that we specifically included the process binding clauses $M \Leftarrow P$ as a subset of the set of conditions. Using this fact, we, at last, add the rule

$$\textbf{run } M \equiv \textbf{case } M \Leftarrow P : P$$

to the definition of structural congruence. This allows **run** M terms to be handled as just a special instance of a **case** expression, and the reduction can then be concluded by either [Case] or the amended [Com'] rule. Assume for example that $P \xrightarrow{\tau} P'$ and $\mathrm{n}(P) \subseteq \mathrm{n}(M)$ and $M \Leftarrow P \in \Psi$. Then we could conclude e.g.

$$[\textsc{Struct}]\frac{\textbf{run}\ M \equiv \textbf{case}\ M \Leftarrow P : P \quad \Psi \rhd \textbf{case}\ M \Leftarrow P : P \to P' \quad P' \equiv P'}{\Psi \rhd \textbf{run}\ M \to P'}$$

by the [Struct] rule, and then conclude the premise by the [Case] rule:

$$[\textsc{Case}]\frac{\Psi \rhd P \to P' \quad \Psi \Vdash M \Leftarrow P}{\Psi \rhd \textbf{case}\ M \Leftarrow P : P \to P'}$$

As can be seen, the previously proposed [Run] rule now appears as just a specialised instance of the [Case] rule. Thus, [Run] can now be subsumed under [Case], and removed as an explicit rule, thereby simplifying the definition of $\to$.

3.1.4 The unfolding problem again: Mixed case

Unfortunately, even with this extended form of structural congruence, the current definition of $\to$ still fails to capture one class of $\xrightarrow{\tau}$ transitions. Consider a pair of processes P, Q where $\Psi \rhd P \not\xrightarrow{\tau}$ and $\Psi \rhd Q \not\xrightarrow{\tau}$ but $\Psi \rhd P \mid Q \xrightarrow{\tau}$ for any Ψ, and suppose that we then place P within a **case** expression:

$$(\textbf{case}\ \top : P) \mid Q$$

Obviously, this process should still be able to reduce, since the condition is $\top$, but suppose then further that P itself consists of an inner **case** expression *and* some other subterms in parallel:

$$P \triangleq (\textbf{case}\ \varphi_1 : P_1) \mid P_2$$

and such that $P_1 \mid Q \xrightarrow{\tau}$ but $P_2 \mid Q \not\xrightarrow{\tau}$, i.e. such that the communication with Q happens with the process within the *inner* **case**. Then

$$\Psi \rhd (\textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1) \mid P_2) \mid Q \not\to$$

because the current form of structural congruence only allows a hierarchy of nested cases to be flattened, if the *entire* inner term is a **case**, which is not the case here.

Even though this preliminary attempt a definition of a $\to$ relation fails to fully capture $\xrightarrow{\tau}$, it nevertheless highlights some relevant issues: Firstly, that the main difficulties with defining $\to$ seem to pertain to **case** and **run** M; and secondly that the semantics of these two constructs, with just a few, very reasonable assumptions about the way $\mathbb{C}$ is defined, can be handled together, as instances of the same rule.

3.2 Reduction with contexts

The problems pertaining to **case** unfolding are not particular to the higher-order Ψ-calculi; they also appear in the first-order variant, since the **case** construct exists in both. Åman Pohjola [39] defines a reduction semantics for first-order Ψ-calculi by using *reduction contexts*, rather than structural congruence, to isolate the redex in **case** expressions. Given that HOΨ is obtained from the first-order Ψ-calculus just by the addition of the **run** M construct, and this, in turn, may be seen as just a special instance of **case**, we may thus hope to be able to adapt the first-order reduction semantics to the higher-order variant. We shall therefore review the first-order semantics in the following, with a few insignificant changes compared to [39]:

Definition 3.1 (Reduction contexts)**.** *The set of* reduction contexts $C \in \mathcal{C}$ *is built by the formation rules:*

$$
\begin{array}{rll}
C \in \mathcal{C} ::= & [\,] & \textit{hole} \\
\mid & G & \textit{assertion guarded process} \\
\mid & C_1 \mid C_2 & \textit{parallel composition} \\
\mid & \mathbf{case}\ \widetilde{\varphi}_1 : \widetilde{G}_1\ [\!]\ \varphi : C\ [\!]\ \widetilde{\varphi}_2 : \widetilde{G}_2 & \textit{case composition}
\end{array}
$$

and the notation $C[\,\widetilde{G}\,]$ denotes the process obtained by substituting each element of $\widetilde{G}$ into the corresponding hole in C. The number of holes must match the arity of $\widetilde{G}$; otherwise, the substitution is undefined.

Note here that reduction contexts are defined, not in terms of processes P in general, but in terms of *assertion guarded* processes G, as described in definition 2.14 (cf. page 13). The idea is to use reduction contexts to isolate the communicating (sub)terms, and then collect the remainder of the contextual processes after the communication has occurred. To this end, the semantics relies on two auxiliary functions to collect the parallel compositions and conditions:

Definition 3.2 (Conditions and parallel)**.** *The* conditions $\textsc{conds}\,(C)$ *and* parallel *compositions* $\textsc{ppr}\,(C)$ *of a context C are defined by the recursive equations:*

$$
\begin{aligned}
\textsc{conds}\,([\,]) &\triangleq \emptyset \\
\textsc{conds}\,(G) &\triangleq \emptyset \\
\textsc{conds}\,(C_1 \mid C_2) &\triangleq \textsc{conds}\,(C_1) \cup \textsc{conds}\,(C_2) \\
\textsc{conds}\left(\mathbf{case}\ \widetilde{\varphi}_1 : \widetilde{G}_1\ [\!]\ \varphi : C\ [\!]\ \widetilde{\varphi}_2 : \widetilde{G}_2\right) &\triangleq \{\,\varphi\,\} \cup \textsc{conds}\,(C)
\end{aligned}
$$

$$
\begin{aligned}
\textsc{ppr}\,([\,]) &\triangleq \mathbf{0} \\
\textsc{ppr}\,(G) &\triangleq G \\
\textsc{ppr}\,(C_1 \mid C_2) &\triangleq \textsc{ppr}\,(C_1) \mid \textsc{ppr}\,(C_2) \\
\textsc{ppr}\left(\mathbf{case}\ \widetilde{\varphi}_1 : \widetilde{G}_1\ [\!]\ \varphi : C\ [\!]\ \widetilde{\varphi}_2 : \widetilde{G}_2\right) &\triangleq \textsc{ppr}\,(C)
\end{aligned}
$$

Definition 3.3 (Structural congruence)**.** *Let* structural congruence*, written* $\equiv$*, be the least equivalence on process terms containing the congruence rules:*

$$[\text{S-CON-PAR}]\frac{P_1 \equiv P_2}{P_1 \mid R \equiv P_2 \mid R} \qquad [\text{S-CON-RES}]\frac{P_1 \equiv P_2}{(\boldsymbol{\nu} x)\, P_1 \equiv (\boldsymbol{\nu} x)\, P_2}$$

$$[\text{S-CON-REP}]\frac{G_1 \equiv G_2}{!G_1 \equiv !G_2} \qquad [\text{S-CON-OUT}]\frac{P_1 \equiv P_2}{\overline{M}N.P_1 \equiv \overline{M}N.P_2}$$

$$[\text{S-CON-CASE}]\frac{\forall i. G_{1i} \equiv G_{2i}}{\mathbf{case}\ \widetilde{\varphi} : \widetilde{G_1} \equiv \mathbf{case}\ \widetilde{\varphi} : \widetilde{G_2}}$$

and the commutative monoidal rules for $(\mathcal{P}_{\Psi/\equiv},\ \mid\ , \mathbf{0})$*, and the algebraic axioms:*

$$\begin{aligned}
(\boldsymbol{\nu} x)\, \mathbf{0} &\equiv \mathbf{0} \\
(\boldsymbol{\nu} x)\, P \mid Q &\equiv (\boldsymbol{\nu} x)\, (P \mid Q) \ \ \textit{if}\ x \# Q \\
(\boldsymbol{\nu} x)\, (\boldsymbol{\nu} y)\, P &\equiv (\boldsymbol{\nu} y)\, (\boldsymbol{\nu} x)\, P \\
\mathbf{case}\ \widetilde{\varphi} : (\boldsymbol{\nu} x)\, \widetilde{P} &\equiv (\boldsymbol{\nu} x)\, \left(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\right) \ \ \textit{if}\ x \# \widetilde{\varphi} \\
\overline{M}N.\, (\boldsymbol{\nu} x)\, P &\equiv (\boldsymbol{\nu} x)\, \overline{M}N.P \ \textit{if}\ x \# M, N \\
\underline{M}(\lambda \widetilde{z})N.\, (\boldsymbol{\nu} x)\, P &\equiv (\boldsymbol{\nu} x)\, \underline{M}(\lambda \widetilde{z})N.P \ \textit{if}\ x \# M, N, \widetilde{z} \\
!P &\equiv P \mid !P
\end{aligned}$$

and lastly the axioms for unit and composition of assertions:

$$\begin{aligned}
(\!|\Psi_1|\!) \mid (\!|\Psi_2|\!) &\equiv (\!|\Psi_1 \otimes \Psi_2|\!) \\
(\!|\mathbf{1}|\!) \mid P &\equiv P
\end{aligned}$$

The definition adds all the congruence properties and algebraic properties of HOΨ bisimilarity (cf. Åman Pohjola [39, p. 11-12]). Notably, bisimilarity is not a congruence for input, and it is only a congruence for **case** and replication, if the processes are assertion guarded. This is as expected, since **case** and replication are only *well-formed*, if the process subterms are assertion guarded (cf. definition 2.15).

Note also that the two last axioms, for unit and composition of assertions, are not part of the original presentation by Åman Pohjola [39], but have been added by us to simplify the notation of assertions by allowing them to be combined by structural congruence. We then enforce this combination by only allowing reductions on process terms with a single assertion $(\!|\Psi|\!)$ at the left-most position in a parallel composition. The unit rule $(\!|\mathbf{1}|\!) \mid P \equiv P$ then ensures that an assertion always *can* be present at that position, by allowing the unit assertion $(\!|\mathbf{1}|\!)$ to be inserted anywhere. The reduction relation $\rightarrow$ can then be defined as follows:

Definition 3.4 (Reduction relation)**.** *The* reduction relation $\rightarrow \subseteq \mathcal{P}_\Psi \times \mathcal{P}_\Psi$ *is defined inductively by the following rules:*

$$[\text{R-CTX}]\frac{\Psi \Vdash M \dot{\leftrightarrow} K \quad \forall \varphi \in \text{CONDS}\,(C)\,.\Psi \Vdash \varphi}{(\!|\Psi|\!) \mid C\,[\,\overline{M}N[\widetilde{x} := \widetilde{L}].P;\ \underline{K}(\lambda\widetilde{x})N.Q\,] \rightarrow (\!|\Psi|\!) \mid P \mid Q[\widetilde{x} := \widetilde{L}] \mid \text{PPR}\,(C)}$$

$$[\text{R-STRUCT}]\frac{P \equiv Q \quad Q \rightarrow Q' \quad Q' \equiv P'}{P \rightarrow P'} \qquad [\text{R-RES}]\frac{P \rightarrow P'}{(\boldsymbol{\nu} x)\,P \rightarrow (\boldsymbol{\nu} x)\,P'}$$

Compared to 'usual' formulations of reduction semantics, such as our own previous attempt in section 3.1, the formulation in definition 3.4 differs significantly in one particular regard: It does not contain a rule for reduction of a term in parallel composition. Instead, this is handled solely by the reduction context in the [R-CTX] rule: The context is used to *pattern match* against the term, to isolate two subterms that are able to communicate. Any processes parallel to these terms are then collected by $\text{PPR}\,(C)$ and composed with the reduct, such that they are preserved for the next reduction step. Thus, the derivation of a reduction would first use the [R-STRUCT] rule (and possibly the [R-RES] rule) to arrange the process in the required format, and then isolate two subterms that shall communicate, by pattern matching with the context. According to Åman Pohjola [39], this definition of $\rightarrow$ matches exactly the $\xrightarrow{\tau}$ relation of the first-order Ψ-calculus labelled semantics.

Note that our formulation of the rule [R-CTX] in definition 3.4 differs insignificantly from the presentation in [39] in that we only allow a single $(\!|\Psi|\!)$ at the leftmost position of the redex and reduct, whilst the original formulation here has a parallel composition

$$\widetilde{(\!|\Psi|\!)} \triangleq (\!|\Psi_1|\!) \mid \ldots \mid (\!|\Psi_n|\!)$$

of arbitrarily many assertions, which furthermore is just assumed to be equal to $(\!|\mathbf{1}|\!)$ if the sequence is empty. It was precisely to avoid this assumption and extra notation that we instead introduced the aforementioned two extra axioms for assertion unit and composition in structural congruence.

Now, the reduction relation in definition 3.4 is only defined for the first-order variant of the Ψ-calculus framework, but we can easily extend it to the higher-order calculus by just including the axiom

$$\textbf{run}\ M \equiv \textbf{case}\ M \Leftarrow P : P$$

as previously described. We do not even need to add the axioms for collapsing nested **case** terms, since they can be built by the syntax for contexts. It also allows a mixing of **case** and parallel compositions, and we thus avoid the problem of mixed **case** that our previous attempt could not handle. Thus, we have the following result:

Theorem 1 (Semantic equivalence). *Let P be any higher-order Ψ-calculus process, and let $\rightarrow$ be defined as in definition 3.4 with $\equiv$ extended with the axiom for* **run** M. *Then*

$$P \xrightarrow{\tau} P \iff P \rightarrow P'$$

The theorem has been proved by Åman Pohjola [39] for the first-order calculus. Thus we only need to extend this proof to cases where the redex is, or includes, a **run** M term. The details of this proof is given in appendix B.

3.3 Reduction with an evaluation relation

As we have seen in the previous section, the difficulties with creating a reduction semantics for HOΨ pertain particularly to the **run** M and **case** constructs, because the reduction of both these terms is conditional upon a subterm, which in the case of **run** M is not even present in the syntax of the redex. In both cases, the problem is how to ensure that these subterms perform an actual reduction step, and are not merely rewritten by structural congruence.

Reduction contexts sidestep the issue by using pattern matching, rather than structural rules, to isolate reducible subterms to build a redex. This ensures precisely that rewriting by structural congruence cannot happen at arbitrary points in the tree. However, the downside of this technique is that the context rule [R-CTX] becomes complicated and unwieldy to use in proofs, because one would have to do induction in the formation rules for contexts, rather than just in the hight of the derivation tree. It may thus defy our very purpose of defining a reduction semantics; namely to make the semantics *simpler* to use in proofs for e.g. type systems.

In this section, we shall therefore consider a third alternative, which is to build a reduction relation that is slightly *larger* than the $\xrightarrow{\tau}$ transition relation. The benefit of working with a larger relation is that every property of the reduction relation then also holds for the τ-transition, since we will have that

$$P \rightarrow P' \implies P \xrightarrow{\tau} P'$$

Specifically, we will allow **case** and **run** M to unfold, regardless of whether they contain reducible subterms that can perform an internal or external communication: For example, if we assume that $\Psi \Vdash \varphi$, $\Psi \Vdash M \Leftarrow P$, $P \overset{\alpha}{\not\rightarrow}$ and $Q \xrightarrow{\tau} Q'$, then we will allow reductions such as

$$\Psi \rhd (\mathbf{case}\ \varphi : P) \mid Q \rightarrow P \mid Q' \quad \text{and} \quad \Psi \rhd \mathbf{run}\ M \mid Q \rightarrow P \mid Q'$$

even though they cannot be concluded by the rules of the labelled semantics. Lifting this restriction removes most of the aforementioned difficulties pertaining to the behaviour of these two constructs.

In effect, we shall make the process **run** M *equivalent* to P, if $M \Leftarrow P$ is entailed by Ψ, and likewise for **case** $\widetilde{\varphi} : \widetilde{P}$ which we shall regard as equivalent to P_i if $\Psi \Vdash \varphi_i$. Thus we shall again make use of a notion of process equivalence similar to

structural congruence, but this time we shall add axioms for unfolding **run** M and **case** constructs, rather than using structural rules.

However, this unfolding is contingent upon the free assertions $(\!|\Psi|\!)$ present in the process to be rewritten, and we shall therefore need to *parametrise* our equivalence relation with the active assertions. We *could* require processes to be on a certain a normal form, such as

$$(\boldsymbol{\nu}\widetilde{x})\left(\widetilde{(\!|\Psi|\!)} \mid P\right)$$

where we require all scopes to have been fully extruded, and all assertions to appear at the left-most position, but this would precisely *require* a notion of structural congruence to allow an arbitrary process to be rewritten to this form, thereby making the definition circular.

Instead we shall proceed as Bengtson et al. [3, 4] and Parrow et al. [28] and use the frame function $\mathcal{F}_\Psi(\cdot)$ from definition 2.16 to collect the free assertions in the redex. Before we can proceed to define our parametrised equivalence relation, we shall then firstly define a version of structural congruence that is simpler than the one given previously, in definition 3.3:

Definition 3.5 (Simplified structural congruence)**.** *We define the* simplified structural congruence *written,* $\equiv_S$*, as the least congruence on process terms containing* α*-equivalence, the commutative monoidal rules for parallel composition, and the rule for scope extrusion:*

$$[\text{S-Scope}]\ \ (\boldsymbol{\nu}x)\,P \mid Q \equiv_S (\boldsymbol{\nu}x)\,(P \mid Q) \quad \text{if } x\#Q$$

With this definition in place, we could then try to proceed by extending structural congruence with a parametrisation, i.e.

$$\cdot \rhd \cdot \equiv_S \cdot \subseteq \mathbb{A} \times \mathcal{P}_\Psi \times \mathcal{P}_\Psi$$

and then add the two unfolding rules:

$$\frac{\Psi \Vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \equiv_S P_i} \qquad \frac{\Psi \Vdash M \Leftarrow P}{\Psi \rhd \mathbf{run}\ M \equiv_S P}$$

for unfolding **run** M and **case**s, if their conditions are entailed by the Ψ parameter. Unfortunately, this definition of structural congruence is unsound: The relation is symmetric, and we can therefore use the rule for **case** unfolding to create an arbitrary process P: Assume for example that $\Psi \Vdash \varphi_1$ and $\Psi \Vdash \varphi_2$; then

$$\Psi \rhd \mathbf{0} \equiv_S \mathbf{case}\ \varphi_1 : \mathbf{0}\ [\!]\ \varphi_2 : P \equiv_S P$$

because we can read the rule from right to left. A related problem arises with the **run** M construct, where the unfolding rule now allows one process to be substituted

for another, if they share a handle: Assume for example that $M \Leftarrow P$ and $M \Leftarrow Q$; then the we may perform the two rewrites

$$\Psi \rhd P \equiv_S \mathbf{run}\ M \equiv_S Q$$

by using the unfolding rule twice, first from right to left, and then left to right. In both cases, the problem derives from the symmetry of $\equiv_S$, and we shall therefore instead create an *asymmetric* relation $\underline{\ggg}$, that we term the *evaluation relation*, defined as follows:

Definition 3.6 (Evaluation relation)**.** *We define the parametrised* evaluation relation

$$\cdot \rhd \cdot \underline{\ggg} \cdot \subseteq \mathbb{A} \times \mathcal{P}_\Psi \times \mathcal{P}_\Psi$$

by the following rules:

$$[\text{E-Res}]\frac{\Psi \rhd P \underline{\ggg} P'}{\Psi \rhd (\boldsymbol{\nu} x)\, P \underline{\ggg} (\boldsymbol{\nu} x)\, P'}\ (x \# \Psi) \qquad [\text{E-Case}]\frac{\Psi \Vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \underline{\ggg} P_i}$$

$$[\text{E-Struct}]\frac{P \equiv_S P' \quad \Psi \rhd P' \underline{\ggg} Q' \quad Q' \equiv_S Q}{\Psi \rhd P \underline{\ggg} Q} \qquad [\text{E-Run}]\frac{\Psi \Vdash M \Leftarrow P}{\Psi \rhd \mathbf{run}\ M \underline{\ggg} P}$$

$$[\text{E-Par}]\frac{\Psi \otimes \mathcal{F}_\Psi(Q) \rhd P \underline{\ggg} P'}{\Psi \rhd P \mid Q \underline{\ggg} P' \mid Q}\ (\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P)\,, P)$$

$$[\text{E-Rep}]\frac{}{\Psi \rhd\, !P \underline{\ggg} P \mid\, !P}$$

This definition ensures that the two unfolding rules, [E-Run] and [E-Case], cannot be applied in the other direction, and thus we avoid the aforementioned problem of symmetry. The [E-Struct] rule then allows $\underline{\ggg}$ to rewrite processes by our simplified structural congruence, where symmetry is unproblematic; and, lastly, the rule [E-Par] is used to ensure that the free assertions in Q are taken into account when evaluating P in a parallel composition, similar to the [Par] rule in the labelled semantics. Then at last we can define the reduction relation:

Definition 3.7 (Reduction relation)**.** *We define the* reduction relation

$$\cdot \rhd \cdot \to \cdot \subseteq \mathbb{A} \times \mathcal{P}_\Psi \times \mathcal{P}_\Psi$$

by the following rules:

$$[\text{r-com}]\frac{\Psi \Vdash M \leftrightarrow K}{\Psi \rhd \overline{M} N[\widetilde{x} := \widetilde{L}].P \mid \underline{K}(\lambda \widetilde{x}) N.Q \to P \mid Q[\widetilde{x} := \widetilde{L}]}$$

$$[\text{R-EVAL}]\frac{\Psi \rhd P \ggg Q \quad \Psi \rhd Q \to P'}{\Psi \rhd P \to P'} \qquad [\text{R-RES}]\frac{\Psi \rhd P \to P'}{\Psi \rhd (\boldsymbol{\nu} x)\, P \to (\boldsymbol{\nu} x)\, P'}\ (x \# \Psi)$$

$$[\text{R-PAR}]\frac{\Psi \otimes \mathcal{F}_\Psi(Q) \rhd P}{\Psi \rhd P \mid Q \to P' \mid Q}\ (\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P)$$

As mentioned above, this definition yields a reduction relation that is strictly larger than the τ-labelled transition relation; hence we cannot have a full correspondence, but only one direction of implication, as expressed in the following theorem:

Theorem 2. $\Psi \rhd P \xrightarrow{\tau} P' \implies \Psi \rhd P \to P'$

Proof. The proof is by induction in the derivation of $\Psi \rhd P \xrightarrow{\tau} P'$, where we show that for each possible transition of this form, we can derive a corresponding conclusion of the form $\Psi \rhd P \to P'$ by using the rules of the reduction semantics. □

* * *

In this chapter we have discussed the development of a reduction semantics for the higher-order Ψ-calculus framework. We have given three examples of the definition of a reduction relation: One (the first) that was strictly smaller than $\xrightarrow{\tau}$; one (the second) that coincided with $\xrightarrow{\tau}$ but used reduction contexts and reinterpreted **run** M as a special **case** construct; and lastly one (the third) that was strictly larger than $\xrightarrow{\tau}$ and used an asymmetric evaluation relation rather than the usual structural congruence.

4 Instantiations

Our purpose in the present chapter is twofold: Firstly, our presentation of the HOΨ-calculus in chapter 2 was rather terse and devoid of any illustrative examples of instantiations; these are instead deferred to the present chapter, where we shall review and expand on some of the examples of both first- and higher-order instantiations by Bengtson et al. [3, 4] and Parrow et al. [28].

Secondly, as described in chapter 1, one of our main purposes in this thesis is to show that the ρ-calculus can be represented within the HOΨ-framework. We shall therefore use the aforementioned examples to build towards such an instantiation.

4.1 First-order instantiations

We shall firstly consider instantiations of the π-calculus [26], both its monadic and polyadic variants, as well as the ${}^{e}\pi$-calculus of Carbone and Maffeis [7]. They appear much alike, when viewed through the abstract lens of the Ψ-calculus framework, since only a small change in parameter settings is required to obtain one from the other.

The *monadic* π-calculus allows only a *single* name, both in subject and in object position, whilst the *polyadic* π-calculus allows *vectors* of names in object position. Lastly, the ${}^{e}\pi$-calculus reverses this pattern by allowing vectors of names in *subject* position, but only single names in object position. The syntactic difference between the monadic, polyadic and ${}^{e}\pi$ variants thus lies in the form of the input and output operators, and we shall therefore define the common syntax here at the outset for later reference:

Definition 4.1 (Common π-syntax)**.** *The common syntax of π-calculi with matching is given by the following formation rules:*

$$P \in \mathcal{P} ::= \mathbf{0} \;\mid\; \textstyle\prod_{i=1}^{n} P_i \;\mid\; \sum_{i=1}^{n} \alpha_i.P_i \;\mid\; [x = y]P \;\mid\; (\boldsymbol{\nu} x)\, P \;\mid\; !P$$

where $\prod$ denotes parallel composition, $\sum$ denotes choice, and α is a communication prefix. Sums and products with only a single term denote just the term itself; i.e.

$$\prod_{i=1}^{1} P_1 = \sum_{i=1}^{1} P_1 = P_1$$

To obtain the various π-calculi in the following, we need only redefine the communication prefix α accordingly, and perhaps amend the syntax slightly. The meaning of the remaining operators is the standard interpretation, as described in e.g. [27], but rather than repeating the semantics here we shall instead define a general translation into Ψ-calculus syntax, that we shall reference in the following sections:

Definition 4.2 (Common π-translation)**.** *The translation of the common π-calculus syntax is given by the following recursive equations:*

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \mathbf{0} \\
[\![\textstyle\prod_{i=1}^{n} P_i]\!] &= [\![P_1]\!] \mid \ldots \mid [\![P_n]\!] \\
[\![\textstyle\sum_{i=1}^{n} \alpha_i.P_i]\!] &= \mathbf{case}\ \top : [\![\alpha_1.P_1]\!]\ [\!]\ \ldots\ [\!]\ \top : [\![\alpha_n.P_n]\!] \\
[\![[\,x = y\,]P]\!] &= \mathbf{case}\ x \dot{\leftrightarrow} y : [\![P]\!] \\
[\![(\boldsymbol{\nu} x)\,P]\!] &= (\boldsymbol{\nu} x)\,[\![P]\!] \\
[\![!P]\!] &= ![\![P]\!]
\end{aligned}$$

Note that the translation is homomorphic for most of the operators, and introduces only a minimum of encoding for the rest. Specifically, nondeterministic choice $P_1 + P_2$ is encoded by using the **case** expression with $\top$ as the condition, as we also described in our comments on the Ψ-calculus' syntax (cf. section 2.3.1). Matching $[\,x = y\,]P$ is similarly encoded with a **case** expression, this time with a comparison by $\dot{\leftrightarrow}$ as the condition, similar to an **if** φ **then** P construct.[1]

The semantics will then follow from the specific parameter settings and translations of the communication operators that we shall define for each variant below.

4.1.1 The monadic π-calculus

The *monadic* π-calculus is the most basic variant, since it only allows a single name in both subject and in object position. We define the prefix α accordingly:

$$\alpha ::= x\,(y) \quad \mid \quad x\langle z\rangle$$

Now, to describe the monadic π-calculus as a Ψ-calculus instantiation, we shall expand on an example by Bengtson et al. [4]. We set the parameters as follows:

Definition 4.3 (Monadic π-instantiation)**.** *The monadic π-calculus is obtained by the following Ψ-calculus parameter settings:*

$$\begin{aligned}
&\mathbb{T} \triangleq \mathcal{N} \\
&\mathbb{C} \triangleq \left\{ M \dot{\leftrightarrow} N \mid M, N \in \mathbb{T} \right\} \cup \{ \top \} && \mathbb{A} \triangleq \{ \mathbf{1} \} \\
&\Vdash \triangleq \left\{ (\mathbf{1}, x \dot{\leftrightarrow} x) \mid x \in \mathcal{N} \right\} \cup \{ (\mathbf{1}, \top) \} && \otimes \triangleq \lambda x.\lambda y.\mathbf{1}
\end{aligned}$$

and with substitution defined as the usual capture-avoiding replacement function of names for names.

[1] Here we deviate from the instantiation by Bengtson et al. [4] who instead explicitly define $\dot{\leftrightarrow}$ to be the $=$ relation. However, we found this double definition to be an unnecesssary complication.

Thus, the set of terms, $\mathbb{T}$, is just the set of names $\mathcal{N}$, and the set of conditions, $\mathbb{C}$ consists of comparisons of terms (which are just atomic names) by the $\dot{\leftrightarrow}$ operator. We also include the $\top$ symbol to represent a condition that is true for all assertions. The π-calculus does not use assertions, so we define them to be empty, by letting the set of assertions $\mathbb{A}$ contain only the unit assertion $\mathbf{1}$, and with the composition operator as a function that for any two arguments will return $\mathbf{1}$.

Lastly, the entailment relation is defined as a set of $\mathbb{A}\times\mathbb{C}$ pairs of the form $(\mathbf{1}, \varphi)$, since $\mathbf{1}$ is our only valid assertion, and consisting of the pair $(\mathbf{1}, \top)$, to let $\top$ be a condition that is always entailed, and of pairings of $\mathbf{1}$ with precisely all conditions where a name x is compared to itself by the channel equivalence operator $\dot{\leftrightarrow}$. Thus we implicitly define $\dot{\leftrightarrow}$ to be exact syntactic equivalence between names.

Given these settings, the translation of the α-prefixes is straightforward:

Definition 4.4 (Monadic π-translation)**.** *The translation of the monadic π-calculus extends the translation of the common syntax from definition 4.2 with the following clauses for input and output:*

$$[\![x\,(\,y\,)\,.P]\!] = \underline{x}(\lambda y)y.[\![P]\!]$$
$$[\![\overline{x}\!<\!z\!>\!.P]\!] = \overline{x}z.[\![P]\!]$$

4.1.2 The polyadic π-calculus

The polyadic variant of the π-calculus allows transmission of *vectors* of names, $\widetilde{x}$, rather than just a single name. We shall represent such vectors as dot-separated lists:

$$\widetilde{x} \triangleq x_1 \cdot \ldots \cdot x_n$$

and we change the definition of prefixes accordingly:

$$\alpha ::= x\,(\,y_1 \cdot \ldots \cdot y_n\,) \;\mid\; \overline{x}\!<\!z_1 \cdot \ldots \cdot z_n\!>$$

It requires just a small addition to the definition of the set of terms $\mathbb{T}$ to accommodate this change, whilst the remaining parameter settings are as in the monadic variant:

Definition 4.5 (Polyadic π-instantiation)**.** *The polyadic π-calculus is obtained with the following definition for Ψ-calculus terms:*

$$\mathbb{T} \triangleq \{\, M \cdot x \mid M \in \mathbb{T} \wedge x \in \mathcal{N} \,\} \cup \mathcal{N}$$
$$\mathbb{C} \triangleq \{\, M \dot{\leftrightarrow} N \mid M, N \in \mathbb{T} \,\} \cup \{\, \top \,\}$$
$$\Vdash \triangleq \{\, (\mathbf{1}, x \dot{\leftrightarrow} x) \mid x \in \mathcal{N} \,\} \cup \{\, (\mathbf{1}, \top) \,\}$$

and with assertions $\mathbb{A} \triangleq \{\, \mathbf{1} \,\}$ and composition $\otimes \triangleq \lambda x.\lambda y.\mathbf{1}$ as before.

As can be seen, the settings for $\mathbb{C}, \mathbb{A}, \mathbf{1}, \otimes$ and the entailment relation $\Vdash$ is exactly as in definition 4.3. The only difference is in the definition of the set of terms, $\mathbb{T}$, where we adjoin a new set of *composite names* (vectors) onto the basic set of atomic names.

This setting allows *all* terms to be vectors of any length, which technically also would allow vectors to appear in *subject* position, and in the conditions with the channel equivalence relation. However, our definition of entailment is still as in the monadic instance, so only conditions with exact syntactic equivalence between *single* names $x \leftrightarrow x$ are entailed, and since channel equivalence is used in the communication rules on precisely the terms appearing in subject position, only input and output operations with a single name in subject position will give rise to a transition.

Lastly, the translation of the polyadic α-prefixes is as expected:[2]

Definition 4.6 (Polyadic π-translation)**.** *The translation of the polyadic π-calculus extends the translation of the common syntax from definition 4.2 with the following clauses for input and output:*

$$[\![x\,(y_1 \cdot \ldots \cdot y_n)\,.P]\!] = \underline{x}(\lambda y_1, \ldots, y_n) y_1 \cdot \ldots \cdot y_n . [\![P]\!]$$
$$[\![\overline{x}\!<\!z_1 \cdot \ldots \cdot z_n\!>\!.P]\!] = \overline{x} z_1 \cdot \ldots \cdot z_n . [\![P]\!]$$

4.1.3 The ${}^e\pi$-calculus

The polyadic π-calculus has atomic names in subject position, but allows composite names in object position. The ${}^e\pi$-calculus of of Carbone and Maffeis [7] reverses this pattern, by instead having composite names in *subject* position, but only atomic names as objects (cf. also section 1.4). We once again redefine the α prefixes accordingly:

$$\alpha ::= x_1 \cdot \ldots \cdot x_n\,(y) \;\;\Big|\;\; \overline{x_1 \cdot \ldots \cdot x_n}\!<\!z\!>$$

and we can furthermore remove the matching construct $[\,x = y\,]P$ from the common syntax of definition 4.1, since it can be encoded in ${}^e\pi$.

As mentioned above, the definition for the polyadic π-calculus allowed vectors in both subject and object position, but only communication operations with single names in subject position would be entailed and thus yield transitions. All we therefore need is to lift this restriction, by changing the definition of the entailment relation to let syntactic equivalence between composite names of any length be entailed:

Definition 4.7 (${}^e\pi$-instantiation)**.** *The ${}^e\pi$-calculus is obtained with the following Ψ-calculus parameters:*

[2] This is another point where we deviate from the presentation by Bengtson et al. [3], who instead use a tupling symbol $t_n(x_1, \ldots, x_n)$ for a vector of length n to represent the composite names. We saw no point in complicating the presentation with this extra symbol, and preferred instead the dot-notation which also is directly transferable to the ${}^e\pi$-syntax.

$$\mathbb{T} \triangleq \{\, M \cdot x \mid M \in \mathbb{T} \wedge x \in \mathcal{N} \,\} \cup \mathcal{N}$$
$$\mathbb{C} \triangleq \{\, M \dot{\leftrightarrow} N \mid M, N \in \mathbb{T} \,\} \cup \{\, \top \,\}$$
$$\Vdash \triangleq \{\, (\mathbf{1}, M \dot{\leftrightarrow} M) \mid M \in \mathbb{T} \,\} \cup \{\, (\mathbf{1}, \top) \,\}$$

and with assertions $\mathbb{A} \triangleq \{\, \mathbf{1} \,\}$ *and composition* $\otimes \triangleq \lambda x.\lambda y.\mathbf{1}$ *as before.*

Thus, the settings for $\mathbb{T}, \mathbb{C}, \mathbb{A}$ and $\otimes$ are the same as definition 4.5 for the polyadic π-calculus, and the only change in the definition of $\Vdash$ is that syntactic equivalence of all terms $M \dot{\leftrightarrow} M$, rather than just atomic names $x \dot{\leftrightarrow} x$, is now entailed. Note that the definition of terms $\mathbb{T}$ includes vectors of unit length (i.e. single atomic names), so the entire monadic π-calculus is a sub-language of ${}^e\pi$, as expected. The translation is therefore again straightforward:

Definition 4.8 (${}^e\pi$-translation)**.** *The translation of the* ${}^e\pi$*-calculus extends the translation of the common syntax from definition 4.2 with the following clauses for input and output:*

$$[\![x_1 \cdot \ldots \cdot x_n\,(y)\,.P]\!] = \underline{x_1 \cdot \ldots \cdot x_n}(\lambda y)y.[\![P]\!]$$
$$[\![\overline{x_1 \cdot \ldots \cdot x_n}\!<\!z\!>\!.P]\!] = \overline{x_1 \cdot \ldots \cdot x_n}z.[\![P]\!]$$

Note a curious detail: The variant of ${}^e\pi$ defined by Carbone and Maffeis [7] is monadic, since it only allows single names in object position. This is reflected in the above translation, but the Ψ-calculus instantiation does not explicitly disallow the use of such vectors in object position. Thus, this instantiation actually represents a *polyadic* form of ${}^e\pi$.

Furthermore, in the ordinary ${}^e\pi$-calculus the length of a name vector cannot grow at runtime, but in the present Ψ-calculus instantiation it *can*, because the substitution function is defined such that it substitutes whole *terms* for names, and terms are vectors of names. Thus, if we let *single* names appear in object position of *inputs*, but use vectors in *outputs*, then we can compose new name vectors of arbitrary length at runtime. Consider the following transition:

$$\underline{x}(\lambda y)y.\overline{w}z_1 \cdot y \mid \overline{x}z_2 \cdot z_3 \xrightarrow{\tau} \overline{w}z_1 \cdot z_2 \cdot z_3$$

Here the single name y is bound in object position of the input, but the composite name vector $z_2 \cdot z_3$ appears in object position of the output. The polyadic π-calculus would use a sorting system like Milner's [26] to disregard such processes, since they would not be meaningful in the ordinary π-calculus, where polyadic communication is encoded with monadic operators, but we *can* allow them in the Ψ-calculus representation, precisely because the substitution function here can be defined such that it replaces single names with whole terms. Thus, when $z_2 \cdot z_3$ replaces y within the continuation we obtain the *new* composite name vector $z_1 \cdot z_2 \cdot z_3$, which is then ready to be sent out on w. Thence it may be received by some other process, and used for further communication.

This instantiation thus allows us to generate new names at runtime, thereby making the ν-operator redundant. In other words, we have obtained something

akin to the name generating capability of the ρ-calculus, but without the higher-order characteristics

4.2 Higher-order instantiations

The **run** M construct adds higher-order process mobility to the Ψ-calculus framework, thereby allowing any representable first-order calculus to be lifted to a higher-order variant, including all of the previous examples. Parrow et al. [28] give a very general presentation of how this may be done in a generic way for a number of representable calculi, including, at least in principle, the aforementioned polyadic variant of ${}^{e}\pi$ with name-generating capabilities, although these authors never explicitly explore the properties of this particular higher-order variant. Yet, lifting the polyadic ${}^{e}\pi$-instantiation would yield a π-like calculus with both name-generating capabilities *and* higher-order process mobility; that is, precisely the characteristic features of the ρ-calculus. Thus, we might say that whilst the polyadic ${}^{e}\pi$ instantiation seems to approach the ρ-calculus 'from below,' a higher-order variant might very well approach it 'from above.'

We shall return to this possibility below, after reviewing a few other examples of instantiations of higher-order calculi to further explore the capabilities afforded by **run** M construct. As with the first-order instantiations, we shall begin with a basic language and proceed in an incremental fashion, by adding complexities to the basic definition.

4.2.1 Plain process mobility

Consider, as our first example, the following prototypical higher-order language, which is reminiscent of the Plain CHOCS language of Thomsen [37], except that we have removed the restriction and renaming operators and replaced them with the scoping ν-operator:

Definition 4.9 (Plain process mobility)**.** *Assume a set $\mathcal{N}$ of atomic names, partitioned into two disjoint sets of* channel names *$\mathcal{C}$, ranged over by a, x, y, and of* process variables *$\mathcal{V}$, ranged over by X, Y, Z; i.e.*

$$a, x, y \in \mathcal{C} \subseteq \mathcal{N} \quad \wedge \quad X, Y, Z \in \mathcal{V} \subseteq \mathcal{N} \quad \wedge \quad \mathcal{C} \cap \mathcal{V} = \emptyset$$

and then let the set $\mathcal{P}$ of processes be built by the following formation rules:

$$P \in \mathcal{P} ::= \mathbf{0} \;\Big|\; P \mid Q \;\Big|\; P + Q \;\Big|\; (\boldsymbol{\nu} a)\, P \;\Big|\; a(X).P \;\Big|\; \overline{a}\langle Q \rangle.P \;\Big|\; X$$

As can be seen, this CHOCS-like language is merely a simplified form of the general π-calculus syntax (cf. definition 4.1), with processes rather than names appearing in object position of communication prefixes, and with replication replaced by process variables X. The idea is that communication should yield transitions like the following:

$$a(X).X \mid \overline{a}\langle P \rangle.Q \xrightarrow{\tau} P \mid Q$$

where the process P itself is sent along a and immediately substituted for X in the continuation. It is not much, but it is clearly enough to represent infinite behaviour. We can, for example, use the same method as Meredith and Radestock [20] for encoding replication in the ρ-calculus, and define a copying process D, as

$$D\,(a) \triangleq a(X).\,(X \mid \overline{a}\langle X\rangle)$$

and let it copy itself, to encode infinite behaviour. Clearly, if we let

$$\begin{aligned} P &\triangleq (\boldsymbol{\nu} a)\,(D \mid \overline{a}\langle D\rangle) \\ &= (\boldsymbol{\nu} a)\,(a(X).\,(X \mid \overline{a}\langle X\rangle) \mid \overline{a}\langle a(X).\,(X \mid \overline{a}\langle X\rangle)\rangle) \end{aligned}$$

then $P \to P \to^{\omega}$, and from this we can encode replication. We shall now describe the CHOCS-like language as a HOΨ-calculus, where we expand on an example by Parrow et al. [28, p. 10].

Definition 4.10 (CHOCS-instantiation)**.** *To obtain the instantiation of CHOCS we let the* HOΨ*-parameters be defined as follows:*

$$\begin{aligned} &\mathbb{T} \triangleq \mathcal{N} \cup \mathcal{P} \\ &\mathbb{C} \triangleq \left\{\, a \dot{\leftrightarrow} b \mid a, b \in \mathcal{N} \,\right\} \cup \left\{\, P \Leftarrow Q \mid P, Q \in \mathcal{P} \,\right\} \cup \{\, \top \,\} \\ &\Vdash \triangleq \left\{\, (\mathbf{1}, a \dot{\leftrightarrow} a) \mid a \in \mathcal{C} \,\right\} \cup \{\, (\mathbf{1}, P \Leftarrow P) \mid P \in \mathcal{P} \,\} \cup \{\, (\mathbf{1}, \top) \,\} \end{aligned}$$

and with assertions $\mathbb{A} \triangleq \{\, \mathbf{1} \,\}$ *and composition* $\otimes \triangleq \lambda x.\lambda y.\mathbf{1}$ *as before.*

The definition states that terms $\mathbb{T}$ can be atomic names or whole processes, and conditions $\mathbb{C}$ are either on the form of a comparison of atomic names by the $\dot{\leftrightarrow}$ operator, or a declaration that one process is a handle for another. The latter is the important part: This is the setting that will allow the process code itself to be transmitted in a communication, and it is possible precisely because we let processes themselves be terms.

Assertions are again not needed and hence defined with an empty value as in the first-order examples, and lastly we define the entailment relation to again let exact syntactic equality of channel names be entailed. But furthermore we also let all conditions be entailed where a process is *a handle for itself*; i.e. such that $P \Leftarrow P$ is entailed for all processes P.

The syntactic translation is now straightforward, with the clauses for $\mathbf{0}$, $P \mid Q$, $P + Q$ and $(\boldsymbol{\nu} a)\,P$ exactly as in the translation of the common π-calculus syntax, and with the input and output operators changed in the obvious way to use process variables rather than channel names in object position:

Definition 4.11 (CHOCS-translation)**.** *The translation of CHOCS extends the translation of the common syntax in definition 4.1 with the following clauses for the communication operators and process variables:*

$$\begin{aligned} [\![a(X).P]\!] &= \underline{a}(\lambda X)X.[\![P]\!] \\ [\![\overline{a}\langle Q\rangle.P]\!] &= \overline{a}[\![Q]\!].[\![P]\!] \\ [\![X]\!] &= \mathbf{run}\ X \end{aligned}$$

and with substitution defined as the capture-avoiding replacement of names (process variables X) for terms (processes P).

The Ψ-calculus' term language can be arbitrarily complex, and by letting processes be handles for themselves we are automatically ensured that the well-formedness criterion for handles (cf. definition 2.15) is satisfied, because we obviously have that $\mathrm{n}(P) \subseteq \mathrm{n}(P)$. However, it also has another curious, but useful effect: Consider again our previous communication example, that through the translation now becomes

$$\mathbf{1} \rhd \underline{a}(\lambda X)X.\mathbf{run}\ X \mid \overline{a}P.Q \xrightarrow{\tau} \mathbf{run}\ P \mid Q$$

which is immediately concluded by the [Com] rule. Consider then the term **run** P that results from the communication: Here **run** $P \xrightarrow{\alpha} P'$ if $P \xrightarrow{\alpha} P'$, which is concluded by the [Run] rule:

$$[\textsc{Run}]\frac{\mathbf{1} \Vdash P \Leftarrow P \quad \mathbf{1} \rhd P \xrightarrow{\alpha} P'}{\mathbf{1} \rhd \mathbf{run}\ P \xrightarrow{\alpha} P'}$$

so in other words, **run** $P \simeq P$ for any reasonable definition of a behavioural equivalence $\simeq$. Thus, the transmitted process does not need a number of 'extra' transition steps to be able to resume its operation; it behaves similar to the 'eager drop' mechanism employed in the ρ-calculus' higher-order substitution function (see appendix A), as long as the dropped process is not a deadlock. This is relevant for our endeavour to create an instantiation of the ρ-calculus, because, as we argued in [2], the 'eager drop' method of higher-order substitution cannot be encoded with a 'delayed drop' semantics rule, or its equivalent, a 'trigger process' $!x.P$ as one would use in the π-calculus to represent the higher-order paradigm (cf. [32]), because a reduction always will require at least one extra step to perform the communication on x that will trigger the process.

4.2.2 The HOπ-calculus

The CHOCS-like language from the previous section allowed *only* process terms to appear in object position of a communication. Thus, one may send an entire input operation, but not just the name to input something on.

To make the calculus a little more flexible, we can allow both by amending the syntax for the communication prefixes as follows:

Definition 4.12 (Simplified HOπ-syntax)**.** *Let the sets $a, x, y \in \mathcal{C} \subseteq \mathcal{N}$ and $X, Y \in \mathcal{V} \subseteq \mathcal{N}$ with $\mathcal{C} \cap \mathcal{V} = \emptyset$ be as in definition 4.9. Then let the input and output prefixes be redefined as follows:*

$$\begin{aligned} \alpha &::= \overline{a}\!<\!K\!>\!.P \mid a(U).P \\ K &::= x \mid P \\ U &::= y \mid Y \end{aligned}$$

Here K, appearing in object position of outputs, is either a name or a process; and U, appearing as object of inputs, is either an ordinary variable or a process variable. This yields a simplified form of monadic HOπ, similar to the one described by Parrow [27] in his overview of variants of the π-calculus.

This 'simplified HOπ' extension requires no change to the parameter settings from definition 4.10, and the translation is the same as in definition 4.11, so one might almost say that it was implied in the settings from the outset, with the only change residing in the syntax of the input language. It is, however, quite far from the original form of the HOπ-calculus, as defined by Sangiorgi in [31; 32], which has both polyadicity; process constants D instead of replication; and an *application* construct $D\langle\widetilde{K}\rangle$ and $X\langle\widetilde{K}\rangle$ allowing processes to be parametrised with vectors of data terms, like a function call:

Definition 4.13 (HOπ-calculus syntax)**.** *Let the set of names $\mathcal{N}$ be partitioned into* three *disjoint sets of channel names $\mathcal{C}$ and process variables $\mathcal{V}$, as previously defined, and then thirdly a set $\mathcal{D}$ of* process constants*, ranged over by D. The syntax of* HOπ *is then similar to the common π-calculus syntax of definition 4.1, but without replication, and extended with the following constructs:*

$$\begin{aligned} P &::= \ldots \mid D\langle\widetilde{K}\rangle \mid X\langle\widetilde{K}\rangle \\ \alpha &::= \overline{x}\texttt{<}\widetilde{K}\texttt{>} \mid x\texttt{(}\widetilde{U}\texttt{)} \\ K &::= x \mid D \mid P \\ U &::= y \mid Y \end{aligned}$$

where each D has a defining equation on the form $D \triangleq (\lambda\widetilde{U})P$, with the notation $(\lambda\widetilde{U})P$ denoting process abstraction, with $\widetilde{U}$ binding a number of names or process variables within P. We write D and X for $D\langle\rangle$ and $X\langle\rangle$, i.e. when the length of $\widetilde{K}$ is zero.

The purpose of process abstraction $(\lambda\cdot)\cdot$ in the definition of constants is to act as a list of formal parameters that will be bound by the application operator $\cdot\langle\cdot\rangle$ on process constants and process variables. The semantic rule for process constants with application is therefore

$$[\textsc{Const}]\frac{P\{\widetilde{K}/\widetilde{U}\} \to P'}{D\langle\widetilde{K}\rangle \to P'} \quad \left(D \triangleq (\lambda\widetilde{U})P\right)$$

with substitution being used to bind the formal parameters $\widetilde{U}$ to the actual $\widetilde{K}$.

The application operator can likewise occur on process variables, to enable a form of dynamic parametrisation of received processes. The variable $X\langle\widetilde{K}\rangle$ itself has no reduction rule but will instead be evaluated *eagerly* by the higher-order substitution; i.e. the X will have been replaced by some other term before it is reached in the reduction. Suppose now that a process constant D with a zero-length argument list is received and substituted for X, but the *process* abstraction bound to D

has a formal parameter list $\widetilde{U}$ of non-zero length; i.e. that $D \triangleq (\lambda\widetilde{U})P$ and that we have a reduction sequence like

$$a(X).X\langle\widetilde{K}\rangle \mid \overline{a}\langle D\rangle \rightarrow D\langle\widetilde{K}\rangle \rightarrow P'$$

where $P\{\widetilde{K}/\widetilde{U}\} \rightarrow P'$. Thus we may even parametrise processes with other process abstractions, that again expects parameters with process abstractions, and so on.[3]

Polyadicity can, as we have seen above, be handled straightforwardly in the Ψ-calculus framework, but to represent process constants and parametrisability we shall have to fundamentally alter our parameter settings. Firstly, we shall define a form of parametrised process handles, similar to those in [28], and its associated form of entailment:

Definition 4.14 (Parametrised handle)**.** *Let the* application operator $\cdot\langle\cdot\rangle$ *be any binary operator*

$$\cdot\langle\cdot\rangle : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$$

defined on the set of Ψ-calculus terms. The corresponding form of abstraction is then a parametrised handle *of the form $M(\lambda\widetilde{x})N \Leftarrow P$, with $\widetilde{x}$ binding into N and P; and the associated* entailment rule *for application of parametrised handles is then*

$$[\textsc{Para}]\frac{M(\lambda\widetilde{x})N \Leftarrow P \in \Psi}{\Psi \Vdash M\langle N[\widetilde{x} := \widetilde{L}]\rangle \Leftarrow P[\widetilde{x} := \widetilde{L}]}$$

The [Para] rule is quite general and not limited to the HOπ-calculus, but can be used in a variety of instantiations where this form of parametrisation is desirable, simply by including $M\langle N\rangle$ in the term language, and sets of parametrised handles $M(\lambda\widetilde{x})N \Leftarrow P$ in the definition of assertions, and then lastly adding the [Para] rule to the corresponding entailment relation. Indeed, Parrow et al. [28] use it, in a slightly different formulation, in their generic procedure for lifting first-order Ψ-calculi to higher order. With this definition in place we can now define the Ψ-calculus parameters for HOπ:

Definition 4.15 (HOπ-calculus instantiation)**.** *Let the sets of Ψ-calculus terms, conditions and assertions be defined as follows:*

$$\begin{aligned}
&\mathbb{T} \triangleq \mathcal{N} \cup \mathcal{P} \cup \{\, M \cdot N \mid M, N \in \mathbb{T} \,\} \cup \{\, M\langle N\rangle \mid M, N \in \mathbb{T} \,\} \\
&\mathbb{C} \triangleq \{\, x \leftrightarrow y \mid x, y \in \mathcal{N} \,\} \cup \{\, M \Leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{P} \,\} \cup \{\, \top \,\} \\
&\mathbb{A} \triangleq \mathscr{P}(\{\, M(\lambda\widetilde{x})N \Leftarrow P \mid M, N \in \mathbb{T} \wedge P \in \mathcal{P} \,\})
\end{aligned}$$

with $\mathbf{1} \triangleq \emptyset$ and $\otimes \triangleq \cup$, and with the entailment relation defined by the [Para] *rule from definition 4.14 and the following rules:*

[3] It is on this basis that Sangiorgi [32] states that whilst the π-calculus is of 1st order, and CHOCS is 2nd order, any HOπ-process can be of arbitrarily high order and the HOπ-calculus is therefore of ω order.

$$[\textsc{Chaneq}]\frac{}{\Psi \Vdash x \leftrightarrow\!\!\!\!\cdot\; x} \qquad\qquad [\textsc{True}]\frac{}{\Psi \Vdash \top}$$

Terms $\mathbb{T}$ are here defined to be vectors of names or processes (or even mixes, although we shall not use that possibility here), and and they may optionally be parametrised, since we also include the application operator $\cdot\langle\cdot\rangle$ in the term language. Conditions $\mathbb{C}$ are again defined as channel equivalence tests between vectors of names, the true condition $\top$, and the set of process handles; note that we here only let terms be handles for *single* processes P, and not vectors of processes. Assertions $\mathbb{A}$ are defined as *sets* of parametrised handles, as described in definition 4.14, with assertion composition $\otimes$ as set union and the unit assertion $\mathbf{1}$ as the empty set. And lastly, the entailment relation $\Vdash$ is defined with the [Para] rule as prescribed, as well as the two usual rules for syntactic equivalence of channel names and the true condition.

Now, before we can proceed to define the translation, we must handle one remaining difficulty: Recall that process constants D are declared 'outside' of the syntax of processes, even though the declaration obviously uses the syntax of processes on the right-hand side. To enable the translation to be simple and homomorphic even for process constant declarations we shall therefore assume that each such declaration $D \triangleq (\lambda\widetilde{U})P$ actually *does* appear in the syntax of processes, as unguarded parallel compositions at top level. That is, we assume a HOπ-program is of the form

$$\left(\prod_{i=1}^{n} D_i \triangleq (\lambda\widetilde{U})P_i\right) \mid P$$

with no declarations appearing within any of the P_i processes or within P. We can now define the translation, which is mostly similar to the translation of the common π-calculus syntax, except for the input and output operations, process constants and process variables.

Definition 4.16 (HOπ-calculus translation)**.** *The* HOπ*-translation extends the common translation of the π-calculus syntax in definition 4.2 with the following clauses:*

$$[\![P]\!] = \begin{cases} [\![D \triangleq (\lambda\widetilde{U})P]\!] = (\!|\{\, D(\lambda\widetilde{U})\widetilde{U} \Leftarrow [\![P]\!] \,\}|\!) \\ [\![D\langle\widetilde{K}\rangle]\!] = \mathbf{run}\ D\langle[\![\widetilde{K}]\!]\rangle \\ [\![X\langle\widetilde{K}\rangle]\!] = \mathbf{run}\ X\langle[\![\widetilde{K}]\!]\rangle \\ [\![x(\widetilde{U}).P]\!] = \underline{x}(\lambda\widetilde{U})\widetilde{U}.[\![P]\!] \\ [\![\overline{x}\langle\widetilde{K}\rangle.P]\!] = \overline{x}[\![\widetilde{K}]\!].[\![P]\!] \end{cases}$$

$$[\![\widetilde{K}]\!] = \begin{cases} [\![\widetilde{x}]\!] = \widetilde{x} \\ [\![P_1 \cdot \ldots \cdot P_n]\!] = [\![P_1]\!] \cdot \ldots \cdot [\![P_n]\!] \\ [\![D]\!] = D \end{cases}$$

The translation firstly converts all process constant declarations to top-level assertions, where the process constant now appears as a parametrised handle for the process. This is possible, because we defined the set of process constants $\mathcal{D}$ to be a subset of the set of names $\mathcal{N}$, and thus included in the set of terms.

Technically, we are required by the well-formedness criterion in definition 2.15 to ensure that $\mathrm{n}(M) \subseteq \mathrm{n}(P)$ if $M \Leftarrow P$, if we intend to use the labelled semantics. This is, however, difficult to do within the recursive translation function itself, but one solution could be to use the possibility of having composite terms, consisting of mixed vectors of names and processes, and let the handles be defined as composite terms $D \cdot P$, consisting of the constant itself and the process it represents.

4.3 Towards reflection

We are come at last to the problem of creating an instantiation of the ρ-calculus, our model of reflection. Throughout the previous sections we have seen how the HOΨ-calculus framework can represent calculi with features such as names with a structure ($^e\pi$), binding processes to names (HOπ) and code mobility (CHOCS); features that are all characteristic of the ρ-calculus, and which we now shall attempt to combine to obtain an instantiation.

We briefly described the syntax and semantics of the ρ-calculus in the introduction, section 1.2, and more at length in appendix A. Recall that the ρ-calculus has no atomic names *at all*, but builds its names from the syntax of processes by quoting, including even the bound names $\ulcorner R \urcorner$ appearing in the input construct $\ulcorner P \urcorner(\ulcorner R \urcorner).Q$. But this name is precisely *bound* and can therefore be α-converted to any other name $\ulcorner R_1 \urcorner$ as long as it does not cause a name clash.

The *free* names, like $\ulcorner P \urcorner$ in the previous example, can all be dropped (at least in principle), and thus be converted back into the process found within the name; hence, the structure of a free name has a form of 'meaning' in the sense that it denotes the behaviour of the process within. But a *bound* name can never be dropped, and thus its internal structure only serves to distinguish it from any other bound name, but the actual *format* of this difference is irrelevant, as long as it can be used to build a notion of α-equivalence. In this sense, the bound names might equally well be regarded as atomic entities. Thus we shall begin by slightly amending the syntax of the ρ-calculus, by letting bound names *be* atomic, drawn from the set $\mathcal{N}$, rather than built from quoted processes:

Definition 4.17 (Amended ρ-syntax)**.** *Assume $\mathcal{N}$ is a countably infinite set of atomic names, ranged over by x. The sets of amended ρ-calculus processes and names are then built by the formation rules:*

$$\begin{array}{l} P ::= \mathbf{0} \;\mid\; P_1 \mid P_2 \;\mid\; n(x).P \;\mid\; n\langle\!|P|\!\rangle \;\mid\; \urcorner n \ulcorner \\ n ::= x \;\mid\; \ulcorner P \urcorner \end{array}$$

where we assume all free *names are quoted processes.*

Note that we can easily convert a 'pure' ρ-calculus process into this amended form, as long as we are able to choose such a bound name x afresh whenever we encounter an input operation, by using the following translation:

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \mathbf{0} \\
[\![P_1 \mid P_2]\!] &= [\![P_1]\!] \mid [\![P_2]\!] \\
[\![n(\ulcorner Q \urcorner).P]\!] &= [\![n]\!](x).[\![P\{x/\ulcorner Q \urcorner\}]\!] \qquad \text{where } x\#\,\mathrm{n}(P) \\
[\![n\langle\!| P |\!\rangle]\!] &= [\![n]\!]\langle\!| [\![P]\!] |\!\rangle \\
[\![\ulcorner n \urcorner]\!] &= \ulcorner [\![n]\!] \urcorner \\
[\![\ulcorner P \urcorner]\!] &= \ulcorner [\![P]\!] \urcorner \\
[\![x]\!] &= x
\end{aligned}$$

However, we shall not explicitly use this translation function. Rather, in the sequel, we shall simply assume that any ρ-calculus process is written by using the amended syntax of definition 4.17, with all free names being quoted processes and all bound names being atomic and chosen distinct from all others (i.e. the Barendregt convention).

Given these preliminary considerations we shall now proceed to build two different instantiations of the ρ-calculus: one using code mobility directly, like the CHOCS instantiation; and another using names as process constants, like the second instantiation of the HOπ-calculus.

4.3.1 The ρ-calculus with code mobility

Before we build our first instantiation of the ρ-calculus, we should at least attempt to clarify why doing so may not be entirely trivial. The main difficulty lies not in the higher-order process mobility itself, which, as we have seen above, can easily be handled within the HOΨ-framework. Instead, it derives from the ρ-calculus' ability to generate names with *structure* at runtime, and the fact that this structure determines the equivalence of names in a way that is different from purely syntactic equivalence. Indeed, *name equivalence*, $\equiv_{\mathcal{N}}$, in the ρ-calculus is generated by the two rules

$$[\textsc{Nameeq}_1]\frac{P_1 \equiv P_2}{\ulcorner P_1 \urcorner \equiv_{\mathcal{N}} \ulcorner P_2 \urcorner} \qquad [\textsc{Nameeq}_2]\frac{}{\ulcorner\ulcorner P \urcorner\urcorner \equiv_{\mathcal{N}} \ulcorner P \urcorner}$$

that are also found in definition A.3 (cf. appendix A). The first rule, [NAMEEQ$_1$], declares that two names are equivalent if the processes *within* them are structurally congruent; and structural congruence, in turn, is a *congruence* which thus again may require us to compare the *free names* of processes, to decide whether they are structurally congruent or not.

Suppose for example that we must decide whether

$$\ulcorner\ulcorner P_1 \urcorner \langle\!| \mathbf{0} |\!\rangle \urcorner \stackrel{?}{\equiv_{\mathcal{N}}} \ulcorner\ulcorner P_2 \urcorner \langle\!| \mathbf{0} |\!\rangle \urcorner$$

This would, by the [NAMEEQ$_1$] rule, require us to decide whether the two processes are structurally congruent:

$$\ulcorner P_1 \urcorner \langle\!| \mathbf{0} |\!\rangle \stackrel{?}{\equiv} \ulcorner P_2 \urcorner \langle\!| \mathbf{0} |\!\rangle$$

which again comes down to deciding whether

$$\ulcorner P_1 \urcorner \stackrel{?}{\equiv_{\mathcal{N}}} \ulcorner P_2 \urcorner$$

The two relations depend on each other in a mutual recursion, and deciding name equivalence between any two names thus becomes an actual *computation*, albeit one that will always terminate in a finite number of steps, since the sets of names and processes are both well-founded. Hence, the level of quoting within a name cannot be infinite. Consider for example the following process and reduction:

$$\begin{aligned} \ulcorner R \urcorner \langle\!| P |\!\rangle \ \mid \ulcorner R \urcorner (x).x \langle\!| \mathbf{0} |\!\rangle \ \mid \ulcorner Q \urcorner (y).\checkmark \\ \rightarrow \ulcorner P \urcorner \langle\!| \mathbf{0} |\!\rangle \ \mid \ulcorner Q \urcorner (y).\checkmark \end{aligned}$$

Whether or not this process will reduce to $\checkmark$ in the next step depends on whether $\ulcorner P \urcorner \equiv_{\mathcal{N}} \ulcorner Q \urcorner$, which again depends on whether $P \equiv Q$, as described above. But P may have been composed at runtime, and thus be contingent upon the reduction history of the process. *This* is the main difficulty in creating an encoding of the ρ-calculus into another process calculus: The target language must also be able to perform the computational steps involved in deciding name equivalence between objects that are built at runtime.

Yet the HOΨ-framework *can* accommodate this, precisely because it is parametric in the *channel equivalence* operator $\dot{\leftrightarrow}$ and the entailment relation. Thus, our strategy is to build the equivalent of name equivalence into the definition of entailment. We begin by defining the other parameters:

Definition 4.18 (ρ-calculus instantiation)**.** *To obtain the ρ-calculus instantiation we let the set of terms $\mathbb{T}$ and conditions $\mathbb{C}$ be defined thus*

$$\begin{aligned} \mathbb{T} &\triangleq \mathcal{N} \cup \{ \ulcorner P \urcorner \mid P \in \mathcal{P} \} \cup \{ \langle\!| \ulcorner P \urcorner |\!\rangle \mid P \in \mathcal{P} \} \\ \mathbb{C} &\triangleq \{ M \dot{\leftrightarrow} N \mid M, N \in \mathbb{T} \} \\ &\quad \cup \{ P_1 \equiv P_2 \mid P_1, P_2 \in \mathcal{P} \} \\ &\quad \cup \{ M \Leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{P} \} \end{aligned}$$

with assertions $\mathbb{A} \triangleq \{ \mathbf{1} \}$ and composition $\otimes \triangleq \lambda x.\lambda y.\mathbf{1}$, and with the substitution function $[x := M]$ on terms defined as

$$\begin{aligned} \langle\!| \ulcorner P \urcorner |\!\rangle [x := M] &= \langle\!| \ulcorner P[x := M] \urcorner |\!\rangle \\ \ulcorner P \urcorner [x := M] &= \ulcorner P \urcorner \\ z[x := M] &= \begin{cases} M & \text{if } z = x \\ z & \text{if } z \neq x \end{cases} \end{aligned}$$

Note that we have not yet defined the entailment relation: We postpone its definition until after we have created the translation, since they are rather closely entwined. However, for the set of conditions $\mathbb{C}$ we have, apart from the usual comparisons of terms by channel equivalence $M \overset{\cdot}{\leftrightarrow} N$, also included another kind of condition, $P_1 \equiv P_2$, foreshadowing some of the definitions to come.

As for the set of terms, $\mathbb{T}$, we include two kinds of structured terms, both built from the syntax of processes like names in the ρ-calculus: $\ulcorner P \urcorner$ and $\langle\!|\ulcorner P \urcorner|\!\rangle$, where the latter is intended to remind us of the *lift* operator $\ulcorner R \urcorner \langle\!| P |\!\rangle$. Here, the operand (or object) is a *process*, which means that names within it *can* be replaced by substitution, whilst the subject $\ulcorner R \urcorner$ is a *quoted* process, i.e. a name, which is unaffected by substitution.

The purpose of the two different kinds of structured terms is precisely to maintain this distinction, although mainly for the sake of clarity. The substitution function will of course only ever replace *atomic* names, and in our amended ρ-calculus syntax we ensured that only *bound* names would be replaced by atomic names. Thus, if an atomic name x appears in a term, then that name *must* be bound by some input prefix, and if that prefix occurs outside the term, then that term can only have been a lifted process in the source language. That is, for *some* $x \in \mathcal{N}$ we may *possibly* have that $x \in \mathrm{n}(\langle\!|\ulcorner P \urcorner|\!\rangle)$, but for *all* $x \in \mathcal{N}$ we have that $x \notin \mathrm{n}(\ulcorner P \urcorner)$. Indeed, if this was not the case, then the substitution function would fail to satisfy the first of the first substitution law in definition 2.7.

Given these definitions and considerations we can now create the translation:

Definition 4.19 (ρ-calculus translation)**.** *The translation to* HOΨ *of ρ-calculus processes and names, using the amended syntax of definition 4.17, is given by the equations:*

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \mathbf{0} \\
[\![P_1 \mid P_2]\!] &= [\![P_1]\!] \mid [\![P_2]\!] \\
[\![n\,(x)\,.P]\!] &= \overline{[\![n]\!]}(\lambda x)\,\langle\!|x|\!\rangle\,.[\![P]\!] \\
[\![n\,\langle\!|P|\!\rangle]\!] &= \overline{[\![n]\!]}\,\langle\!|\ulcorner [\![P]\!] \urcorner|\!\rangle\,.\mathbf{0} \\
[\![\ulcorner n \urcorner]\!] &= \mathbf{run}\ [\![n]\!] \\
[\![\ulcorner P \urcorner]\!] &= \ulcorner [\![P]\!] \urcorner \\
[\![x]\!] &= x
\end{aligned}$$

The translation is quote straightforward, with only a minimum of encoding. The only noteworthy detail occurs in the translation of input, where the *pattern* $\langle\!|x|\!\rangle$ ensures that the brackets are removed from any received term $\langle\!|\ulcorner P \urcorner|\!\rangle$, resulting in only $\ulcorner P \urcorner$ being substituted for x within the continuation. The translation also gives us the syntax of any HOΨ-process that may *represent* a ρ-calculus process in this instantiation, and with this we can now at last define the entailment relation:

Definition 4.20 (ρ-calculus entailment)**.** *Let the entailment relation for conditions of* channel equivalence $\overset{\cdot}{\leftrightarrow}$ *be generated by the rules:*

$$[\textsc{ChanEq}_1]\frac{}{\Psi \Vdash \ulcorner \mathbf{run}\ M \urcorner \dot{\leftrightarrow} M} \qquad [\textsc{ChanEq}_2]\frac{\Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \ulcorner P_1 \urcorner \dot{\leftrightarrow} \ulcorner P_2 \urcorner}$$

and including the symmetric and transitive closure of $\dot{\leftrightarrow}$ *as described in definition 2.11. Then let the entailment relation for conditions of* structural congruence $\equiv$ *be defined such that* $\equiv$ *contains* α*-equivalence; that* $(\mathcal{P}_{/\equiv},\ |\ ,\mathbf{0})$ *is an abelian monoid; and containing the following congruence rules:*

$$[\textsc{Par}]\frac{\Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash P_1 \mid R \equiv P_2 \mid R} \qquad [\textsc{In}]\frac{\Psi \Vdash M_1 \dot{\leftrightarrow} M_2 \quad \Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \underline{M_1}(\lambda x_1)\, \langle\!|x_1|\!\rangle\, .P_1 \equiv \underline{M_2}(\lambda x_2)\, \langle\!|x_2|\!\rangle\, .P_2}$$

$$[\textsc{Run}]\frac{\Psi \Vdash M_1 \dot{\leftrightarrow} M_2}{\Psi \Vdash \mathbf{run}\ M_1 \equiv \mathbf{run}\ M_2} \qquad [\textsc{Out}]\frac{\Psi \Vdash M_1 \dot{\leftrightarrow} M_2 \quad \Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \overline{M_1}\, \langle\!|\ulcorner P_1 \urcorner|\!\rangle \equiv \overline{M_2}\, \langle\!|\ulcorner P_2 \urcorner|\!\rangle}$$

and lastly, let the entailment of handles *be defined by the rule*

$$[\textsc{Handle}]\frac{}{\Psi \Vdash \ulcorner P \urcorner \Leftarrow P}$$

The rules $[\textsc{ChanEq}_1]$ and $[\textsc{ChanEq}_2]$ correspond directly to the $[\textsc{NameEq}_1]$ and $[\textsc{NameEq}_2]$ rules, and the structural congruence rules then mirror the rules of structural congruence as defined for the original ρ-calculus, with recursive calls to channel equivalence on subjects of input and output.

Returning to our previous example of $\ulcorner\ulcorner P_1 \urcorner\, \langle\!|\mathbf{0}|\!\rangle \urcorner \equiv_{\mathcal{N}} \ulcorner\ulcorner P_2 \urcorner\, \langle\!|\mathbf{0}|\!\rangle \urcorner$, its translated counterpart would then be concluded in the expected way:

$$[\textsc{ChanEq}_2]\ \frac{[\textsc{Out}]\ \dfrac{[\textsc{ChanEq}_2]\ \dfrac{\dfrac{\cdots}{\mathbf{1} \Vdash P_1 \equiv P_2}}{\mathbf{1} \Vdash \ulcorner P_1 \urcorner \dot{\leftrightarrow} \ulcorner P_2 \urcorner} \qquad \dfrac{}{\mathbf{1} \Vdash \mathbf{0} \equiv \mathbf{0}}}{\mathbf{1} \Vdash \overline{\ulcorner P_1 \urcorner}\, \langle\!|\ulcorner \mathbf{0} \urcorner|\!\rangle \equiv \overline{\ulcorner P_2 \urcorner}\, \langle\!|\ulcorner \mathbf{0} \urcorner|\!\rangle}}{\mathbf{1} \Vdash \ulcorner \overline{\ulcorner P_1 \urcorner}\, \langle\!|\ulcorner \mathbf{0} \urcorner|\!\rangle \urcorner \dot{\leftrightarrow} \ulcorner \overline{\ulcorner P_2 \urcorner}\, \langle\!|\ulcorner \mathbf{0} \urcorner|\!\rangle \urcorner}$$

4.3.2 The ρ-calculus with process handles

In [2] we attempted to extend the π-calculus with a Fusion construct inspired by the Fusion calculus [13]. The idea was to let processes be 'fused' with atomic names, written $[z = P]$ and stating that 'z *represents* the process P,' as a means to use atomic names rather than quoted processes directly in the syntax. The ρ-calculus' *lift* operator would then be encoded as

$$[\![n\, \langle\!|P|\!\rangle]\!] = (\boldsymbol{\nu} z)\, \overline{n}\!<\!z\!>\!.[z = P]$$

This calculus, called the ρ-fusion, was intended to act as an intermediary step between the ρ-calculus and the π-calculus, to facilitate encoding of the first into the

latter – something that we now know to be impossible, as we argued in chapter 1. It failed for precisely the same reasons that we described in the previous section: because the purely syntactic equivalence of atomic names cannot capture the name equivalence relation of the ρ-calculus. It equates too few names.

But as we also saw above, the name equivalence relation can be built into the definition of entailment, and we can therefore now create another instantiation of the ρ-calculus, where we reuse this idea of fusing processes to atomic names by letting $[z = P]$ be an *assertion* $(\!|\{\, z \Leftarrow P \,\}|\!)$, stating that '$z$ is a *handle* for the process P.' In fact, it is similar to using process constants as handles, like we did in the HOπ-instantiation.

However, this approach faces one technical difficulty, that we also mentioned for HOπ: By the well-formedness criteria of definition 2.15 we are required to ensure that $\mathrm{n}(P) \subseteq \mathrm{n}(M)$ if $M \Leftarrow P$, which rather limits the kinds of processes that can be fused, if we only use a single name z as the handle. An obvious solution would be to let processes be part of the term language and include the whole of P into the term M, for example as $z \cdot P$, but then there seems little point in adding the extra name z in the first place, since we instead could just let each process be a handle for itself, $P \Leftarrow P$, like we did in the CHOCS instantiation.

The well-formedness criterion thus seems quite restrictive and limiting to the usefulness of having terms as handles, and we might wonder whether we can avoid it. According to Parrow et al. [28], this particular well-formedness criterion was introduced to ensure that restricted names cannot suddenly be moved outside of their scope by the **run** M construct. Suppose for example we had the following generic HOΨ-process:

$$(\boldsymbol{\nu} x)\left(\overline{a}z.(\!|\{\, z \Leftarrow \overline{x}z.\mathbf{0} \,\}|\!)\right) \mid \underline{a}(\lambda y)y.\mathbf{run}\ y \mid \underline{x}(\lambda y)y.\checkmark$$

Here x is a restricted name inside the scope, and clearly $x \in \mathrm{n}(\overline{x}z.\mathbf{0})$ and also $x \notin \mathrm{n}(z)$. Thus, if we simply ignored the well-formedness criterion and let the process reduce, we would not open the scope of x in the communication, and yet the process containing the restricted name x would be run, thereby causing a name clash with the free name x outside the scope:

$$\begin{aligned}(\boldsymbol{\nu} x)\left(\overline{a}z.(\!|\{\, z \Leftarrow \overline{x}z.\mathbf{0} \,\}|\!)\right) \mid \underline{a}(\lambda y)y.\mathbf{run}\ y \mid \underline{x}(\lambda y)y.\checkmark \\ \rightarrow (\boldsymbol{\nu} x)\left((\!|\{\, z \Leftarrow \overline{x}z.\mathbf{0} \,\}|\!)\right) \mid \mathbf{run}\ z \mid \underline{x}(\lambda y)y.\checkmark \\ \stackrel{?}{\rightarrow} (\boldsymbol{\nu} x)\left((\!|\{\, z \Leftarrow \overline{x}z.\mathbf{0} \,\}|\!)\right) \mid \checkmark\end{aligned}$$

The purpose of the well-formedness criterion is precisely to ensure that this situation cannot arise, by requiring that any restricted names used within P must also occur in the handle of P, thereby ascertaining that all scopes will be extruded if the handle is sent out. To circumvent the problem, we shall therefore instead have to assume that all scopes are always maximally extruded: that is, we shall assume that all processes are always on a *normal form*

$$(\boldsymbol{\nu}\widetilde{x})\left((\!|\widetilde{\Psi}|\!) \mid P\right)$$

where all unguarded assertions are gathered in the leftmost position, and all scopes are extruded to encompass both assertions and processes, and all have been appropriately α-converted. This is easy if we use the reduction semantics, because we can use the scope extrusion axiom [S-Scope] of structural congruence to ensure that any process is on this form prior to any reduction.

With this assumption we can safely use single names as handles in our instantiation, because no bound name in a process can then ever be moved outside of its scope. In fact, we might as well let *all* names occurring anywhere within a ρ-process (including within other quoted processes) be represented as atomic names, and not just the bound names. The key distinction between free and bound names will then be that all free names must have a *defining equation* $x \triangleq P$, whilst bound names must not.

This is precisely similar to using HOπ process constants D as free names (which, as the reader may recall, is how we represented them in HOΨ anyway), and we shall therefore have to make a similar assumption about these defining equations: that is, we remove $\ulcorner P \urcorner$ from the syntax of names in definition 4.17, such that only atomic names remain, and instead we add the construct $x \triangleq P$ to the syntax of processes. For each $\ulcorner P \urcorner$ occurring in the 'amended syntax' of definition 4.17 we shall then instead require that a declaration $x \triangleq P$ is added in parallel at top level for some fresh name x which is then used in the syntax instead of $\ulcorner P \urcorner$; and furthermore that declarations occur do not occur anywhere else in the syntax, apart from at top level. That is, we assume now that a ρ-calculus program is on the form

$$\left(\prod_{i=0}^{n} x_i \triangleq P_i \right) \mid P$$

where each name x_i is chosen distinct from the others. The translation of this 'preprocessed' form of the ρ-calculus into HOΨ is then as expected:

Definition 4.21 (ρ-calculus translation)**.**

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \mathbf{0} \\
[\![P_1 \mid P_2]\!] &= [\![P_1]\!] \mid [\![P_2]\!] \\
[\![x\,(y)\,.P]\!] &= \underline{x}(\lambda y) y.[\![P]\!] \\
[\![x\,\langle\!|P|\!\rangle]\!] &= (\boldsymbol{\nu} z)\,\overline{x} z.(\!|\{\, z \Leftarrow [\![P]\!] \,\}|\!) \\
[\![\ulcorner x \urcorner]\!] &= \mathbf{run}\ x \\
[\![z \triangleq P]\!] &= (\!|\{\, z \Leftarrow [\![P]\!] \,\}|\!)
\end{aligned}$$

Note that lift, perhaps not surprisingly, becomes an output, combined with the equivalent of precisely a declaration $z \triangleq P$. The only difference between the new names thus created at runtime, and the 'statically quoted' names appearing as defining equations at top level in the syntax, is that the latter are defined with *free names*, whilst the former use the ν-operator to choose a fresh name z at runtime.

However, this distinction cannot have any practical meaning in the translated program, because all names in the ρ-calculus are globally visible; that is, it is always possible to construct any name anywhere in a program, so there can be no secret or unobservable names. There is simply no way to shield a name from being observed anywhere in the ρ-calculus.

We use $(\boldsymbol{\nu} z)$ in the present translation to choose a fresh name z in the translated program, but this does not mean that communication on z, or running the process P bound to z, will be limited by the scope of z; indeed, we must set our HOΨ-parameters such as to *disregard* the scoping mechanism of ν to mimic this global visibility property of the ρ-calculus.

However, before we proceed to the parameter settings, we should briefly discuss an alternative to the ν-operator, that we also mentioned earlier: namely the name-generating capability of the polyadic ${}^{e}\pi$-calculus. Rather than using ν at runtime to choose a new atomic name for each lifted process, we might instead consider using only a *finite* set of atomic names, and then composing new *name vectors* of increasing length by the method we described in section 4.1.3.

This is, indeed, a possibility, but it would mean that names are generated *independently* of their being bound to a process. Thus we would no longer be ascertained that *all* names 'point' to a process, and hence that all names $\widetilde{x}$ can be used in a **run** $\widetilde{x}$ construct. On the contrary, it would always be possible to build a new name vector with *no* associated process and then attempt to run it, which would be the equivalent of a null-pointer dereference. It is for this very reason that we here instead require *every* free or generated name to be bound to a process; and we shall consequently use the structure of their associated processes, rather than the names themselves, to decide channel equivalence between any pair of names.

Definition 4.22 (ρ-calculus instantiation)**.** *We let the sets of terms, conditions, assertions and associated operators be defined as follows:*

$$\begin{aligned}
\mathbb{T} &\triangleq \mathcal{N} \\
\mathbb{H} &\triangleq \{\, M \Leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{P} \,\} \\
\mathbb{C} &\triangleq \{\, M \mathrel{\dot{\leftrightarrow}} N \mid M, N \in \mathbb{T} \,\} \cup \{\, P_1 \equiv P_2 \mid P_1, P_2 \in \mathcal{P} \,\} \cup \mathbb{H} \\
\mathbb{A} &\triangleq \mathscr{P}(\mathbb{H}) \\
\mathbf{1} &\triangleq \emptyset \\
\otimes &\triangleq \cup
\end{aligned}$$

where $\mathbb{H}$ *is a generic set of handles. Furthermore, we let the substitution function on* terms *be the standard capture-avoiding replacement of names for terms, and for* assertions *be defined as:*

$$\{\, z \Leftarrow P \,\}\,[x := M] = \{\, z[x := M] \Leftarrow P[x := M] \,\}$$

There are no conditions appearing directly in the syntax, so we need not define the substitution function for them. For assertions appearing in the syntax as $(\!|\Psi|\!)$, each such Ψ will be of the form $\{\, z \Leftarrow P \,\}$, and we therefore do not need to define substitution for anything else than singleton sets of handles.

Lastly, we shall implement $\equiv_{\mathcal{N}}$ through the $\dot{\leftrightarrow}$ operator in the entailment relation as before. Many of the rules for $\equiv$ are structurally similar to the first version, differing only in the specific form of the processes, but we shall give them all regardless:

Definition 4.23 (ρ-calculus entailment)**.** *Let the entailment relation for conditions of* channel equivalence $\dot{\leftrightarrow}$ *be generated by the rules:*

$$[\textsc{Chaneq}_1]\frac{}{\{\, x_1 \Leftarrow \mathbf{run}\ x_2 \,\} \otimes \Psi \Vdash x_1 \mathrel{\dot{\leftrightarrow}} x_2}$$

$$[\textsc{Chaneq}_2]\frac{\Psi \Vdash P_1 \equiv P_2}{\{\, x_1 \Leftarrow P_1, x_2 \Leftarrow P_2 \,\} \otimes \Psi \Vdash x_1 \mathrel{\dot{\leftrightarrow}} x_2}$$

and including the symmetric and transitive closure of $\dot{\leftrightarrow}$ *as described in definition 2.11. Next, let the entailment relation for conditions of* structural congruence $\equiv$ *be defined such that* $\equiv$ *contains* α*-equivalence; that* $(\mathcal{P}_{/\equiv},\ \mid\ , \mathbf{0})$ *is an abelian monoid; and containing the following congruence rules:*

$$[\textsc{Run}]\frac{\Psi \Vdash x_1 \mathrel{\dot{\leftrightarrow}} x_2}{\Psi \Vdash \mathbf{run}\ x_1 \equiv \mathbf{run}\ x_2}$$

$$[\textsc{Par}]\frac{\Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash P_1 \mid R \equiv P_2 \mid R} \qquad [\textsc{In}]\frac{\Psi \Vdash x_1 \mathrel{\dot{\leftrightarrow}} x_2 \quad \Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \underline{x_1}(\lambda y_1)y_1.P_1 \equiv \underline{x_2}(\lambda y_2)y_2.P_2}$$

$$[\textsc{Out}]\frac{\Psi \Vdash x_1 \mathrel{\dot{\leftrightarrow}} x_2 \quad \Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash (\boldsymbol{\nu} z_1)\,\overline{x_1}z_1.(\!|\{\, z_1 \Leftarrow P_1 \,\}|\!) \equiv (\boldsymbol{\nu} z_2)\,\overline{x_2}z_2.(\!|\{\, z_2 \Leftarrow P_2 \,\}|\!)}$$

And lastly, let the entailment of handles *be defined by the rule*

$$[\textsc{Handle}]\frac{}{\{\, x \Leftarrow P \,\} \otimes \Psi \Vdash x \Leftarrow P}$$

Again, the rules [$\textsc{Chaneq}_1$] and [$\textsc{Chaneq}_2$] correspond to the [$\textsc{Nameeq}_1$] and [$\textsc{Nameeq}_2$] rules, but compared to the previous instantiation we now use the assertions Ψ to hold the information on which process is bound to each of the names. Channel equivalence is then concluded on the basis of structural congruence between these processes, as before. Thus, the name itself does not matter; only the structure of the process to which it points.

* * *

In this chapter we have reviewed and expanded on several of the examples of instantiations of both first- and higher-order process calculi presented by Bengtson et al. [4] and Parrow et al. [28]. Most notably, we discussed the relationship between the monadic and polyadic π-calculus and the ${}^{e}\pi$ calculus, as they appear when viewed through the lens of a Ψ-calculus parameter settings, which led us to realise that a (purposefully ill-sorted) polyadic ${}^{e}\pi$-calculus may also be capable of generating names with a structure, like the ρ-calculus. Another notable contribution in this chapter was the creation of a 'full' HOπ-instantiation with both application and process constants, that follows Sangiorgi's original formulation much more closely than the 'generic' instantiation described in [28].

And lastly, we created two different instantiations of the ρ-calculus: the first quite closely mirrored the ρ-calculus' concept of letting names (here terms) be *quoted;* whilst the latter took a fundamentally different approach and let all names be the equivalent of a HOπ process constant. In both cases, the crucial detail enabling these instantiations to actually represent the ρ-calculus, was the possibility of having the equivalent of the ρ-calculus' name equivalence relation directly represented in the definition of entailment. It shows that the peculiarity of the ρ-calculus does not reside in its use of quoted processes as names per se, but in its method of deciding equivalence between names, depending on structures that are built within the calculus itself. Whether or not these structures are the names themselves, or whether names are just atomic pointers to these structures, does not matter.

5 A generic type system for the Higher-Order Ψ-calculus

In this chapter we will introduce a generic type system for the HOΨ-calculus. Unlike an ordinary type system for a specific language, the *generic* type system is itself intended to be instantiated through parameter setting, similar to the way the Ψ-calculus is instantiated to yield a representation of a specific language. Thus, it represents a general framework for type systems, and consequently, when we have an instantiation of the HOΨ-calculus, we can create a corresponding instance of the generic type system.

A process is well-behaved when the process is well-typed; that is, we are ensured that certain predefined runtime errors can not occur when the process is well-typed. As with the HOΨ-calculus, in order to make an instance of the generic type system, some parameters must be defined and some requirement/assumption must hold for the parameters. While we cannot show safety for every instance of the generic type system, since the notion of well-behaved processes is dependent on each instance, we *can* show subject reduction for every instance of the generic type system; that is, if a process is well-typed so is every reduct of the process. Since it follows from safety that a well-typed process is well-behaved, we derive that if a process is well-typed, then every following process is well-behaved.

In this chapter we firstly introduce the generic type system and the definition of the type judgement in section 5.2. Then, in section 5.3 we define the instance parameters and instance assumption, which is used to instantiate a type system for an instance of the HOΨ-calculus, and lastly, in section 5.4 we prove subject reduction and its associated lemmas. We also give a short description of how safety can be defined. Example of instances of the generic type system are deferred to chapter 6.

5.1 Preliminary considerations

There already exists a generic type system for the first-order Ψ-calculus made by Hüttel [17].[1] However, this type system, is based on the labelled semantics for the

[1]The definition of our type system will be heavily inspired by the type system of Hüttel [17], and we therefore suspect that our type system and the type system by Hüttel [17] will have same

Ψ-calculus by Bengtson et al. [4], and we wish instead to define a type system for the HOΨ-calculus and base it on the reduction semantics that we developed in chapter 3. We have two principal reasons for this choice:

- Subject reduction is always a desired property for any type system. Theorems for subject reduction with a labelled transition relation will commonly be on the form:
$$\Gamma \vdash P \wedge P \xrightarrow{\tau} P' \implies \Gamma \vdash P'$$
and in order to show that his property holds, one will often also need to show preservation of well-typedness for any labels. Theorems for subject reduction with a reduction relation will instead be on the form:
$$\Gamma \vdash P \wedge P \to P' \implies \Gamma \vdash P'$$
i.e. we do not have to consider labels, and hence and the proof is (sometimes) simpler than the proof for subject reduction with a labelled transition relation.

- A reduction semantics often comes with a definition of structural congruence, which can be used to rewrite processes. A corresponding type system will then also have a lemma stating that any well-typed process remains well-typed after a rewrite; that is, well-typedness is preserved by structural congruence. Given such a lemma, we can therefore rewrite processes such that the typing becomes more manageable. An example of this could be to extrude all scopes to the outermost position, such that we may begin typing a process by adding the types of all new names to the type environment.

In chapter 3 we formulated three different reduction semantics; one that was strictly smaller than the τ-labelled transition relation; one that matched it exactly but used reduction contexts in place of structural rules for (particularly) parallel composition; and one that was strictly larger but instead used an evaluation relation $\ggg$ to circumvent some of the problems created by the symmetry of structural congruence. In the following, we shall base our type system on the largest of the three reduction relations, cf. section 3.3, because we found it easier to work with structural rules rather than reduction contexts. Although it does not match the τ-labelled transition relation exactly, it *will* relate every pair of processes related by the τ-labelled transition relation since it is strictly larger, and our type system will therefore also be able to correctly type processes in the labelled transition system. Thus, if we have a process P, type environment Γ and an assertion environment Ψ, we can derive the following from theorem 2 and subject reduction (see theorem 3):

$$\frac{\begin{array}{c}\Psi \rhd P \xrightarrow{\tau} P' \implies \Psi \rhd P \to P' \\ \Gamma, \Psi \vdash P \wedge \Psi \rhd P \to P' \implies \Gamma, \Psi \vdash P'\end{array}}{\Gamma, \Psi \vdash P \wedge \Psi \rhd P \xrightarrow{\tau} P' \implies \Gamma, \Psi \vdash P'}$$

expressive power, when excluding higher-order behaviour. We will however not attempt to prove this, since it will not be relevant for later results.

We will generally follow the same approach as in the type system by Hüttel [17], but we shall deviate from it in the assumptions and lemmas we make. Furthermore, we will also extend it to support the higher-order behaviour found in the HOΨ-calculus.

5.2 The generic type system

The first problem that arises when defining the generic type system for the HOΨ-calculus is how to handle unguarded assertions collected from the context of a process. This is important because the unguarded assertions determine which channel names (or rather, terms) M, N are equal and therefore need the same type, and which terms are handles for which processes when used in a **run** M operator. The type system by Hüttel [17] solves this by including assertions in type environments, to be able to handle bound names in assertions correctly. The definition of type environments by Hüttel [17] is

$$\Gamma ::= \Gamma, x : T \;\mid\; \Gamma, \Psi \;\mid\; \emptyset$$

and Γ is said to be *well-formed* if Γ is on the form Γ_1, Ψ, Γ_2 and $\mathrm{n}(\Psi) \subseteq \mathrm{dom}(\Gamma_1)$.[2]

We specifically wish to exclude the assertions from the type environment and instead keep them separate for the sake of clarity, and we therefore instead define type environments in the following way:

Definition 5.1 (Type environments)**.** *Let a type environment Γ be a partial function with finite support $\Gamma : \mathcal{N} \rightharpoonup$ **Types**. We shall write this as a set of tuples $\mathcal{N} \times$ **Types**, and we say that if $(x, T) \in \Gamma$ then $\Gamma(x) = T$. We write $\Gamma, x : T$ to denote the type environment Γ extended by the name x with type T.*

However, we still need to keep track of the assertions in the outside context, and for this purpose we shall therefore introduce an *assertion environment*, containing assertions from an outside context. This creates an stronger correspondence between type judgements for processes and the reduction relation, and we can therefore write a judgement for processes as

$$\Gamma, \Psi \vdash P$$

and the subject reduction theorem will then be on the form:

$$\Gamma, \Psi \vdash P \wedge \Psi \rhd P \to P' \implies \Gamma, \Psi \vdash P'$$

Next, we shall define a notion of *well-formed* type judgements to ensure that a type environment in a type judgement supports all free names in both the process and the assertion environment:

[2] Note that type environments are denoted E in the generic type system by Hüttel [17].

Definition 5.2 (Well-formed type judgement)**.** *A type judgement* $\Gamma, \Psi \vdash P$ *is said to be* well-formed *if it is the case that*

$$\mathrm{n}(\Psi) \cup \mathrm{n}(P) \subseteq \mathrm{dom}(\Gamma)$$

This corresponds to the aforementioned definition of well-formed type environments by Hüttel [17]. Furthermore, the assertion environment in the type system also represents an over-approximation of the assertion environment in the reduction relation; that is, it is an over-approximation of itself. We over-approximate the assertion environment though weakening (cf. section 5.4 for details).

Returning briefly to the form of the type environment found in Hüttel [17], we might use two functions, $\mathbf{E}(\Gamma)$ and $\mathbf{\Psi}(\Gamma)$ respectively, to extract the name-type pairs and the assertions. The recipe we shall follow in the following for defining our rules for type judgements might then abstractly be expressed as

$$\Gamma \vdash P \implies \mathbf{E}(\Gamma), \mathbf{\Psi}(\Gamma) \vdash P$$

that is, our type judgements for all constructs in the first-order Ψ-calculus will (not surprisingly) have the same form as in the type system by Hüttel [17], save only that we separate the assertions from the type envionment, and we then extend this with a rule for typing the **run** M construct from the HOΨ-calculus. Thus, we give the type judgements for processes as follows:

Definition 5.3 (Type judgements for processes)**.** *Let type judgements be on the form* $\Gamma, \Psi \vdash P$ *and defined by the typing rules described in figure 5.1.*

The rules [T-IN] and [T-OUT] are similar to the type judgements rules by Hüttel [17]. In order to support higher order behaviour we also need a rule for the **run** M operator, which is [T-RUN]: here a term M is a handle for a process P, and we use a relation $\curvearrowleft$ to obtain the type environment for P from the type of M. This relation is one of the parameters that must be specified when instantiating the type system (cf. section 5.3 for details).

As mentioned above, the purpose of the assertion environment is to collect and represent assertions from an outer context. We can do this with one component of the frame function, $\mathcal{F}_\Psi(\cdot)$ from definition 2.16 in the [T-PAR] rules. Additionally, since the premises must be well-formed, we also need to collect the types of the bound names, for which we use the other component of the frame function, $\mathcal{F}_\nu(\cdot)$.

5.3 Parameters and assumptions

The purpose of making a generic type system is to be able to *instantiate* it with a specific parameter setting, such that we obtain a type system that handles a set of runtime errors. We shall now define these parameters for the generic type system, and we denote them *instance parameters for the generic* HOΨ*-calculus*. They consist of typing rules for terms, conditions (including clauses) and assertions, mirroring the fact that these nominal data types are also parameters in the HOΨ-calculus.

$$[\text{T-IN}]\ \frac{\begin{array}{r} T \hookleftarrow T' \\ \Gamma, \Psi \vdash M : T \\ \Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash N : T' \\ \Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \end{array}}{\Gamma, \Psi \vdash \underline{M}(\lambda \widetilde{x})N.P} \qquad [\text{T-RUN}]\ \frac{\begin{array}{r} T \curvearrowleft \Gamma' \\ \Gamma, \Psi \vdash M : T \\ \Psi \Vdash M \Leftarrow P \end{array}}{\Gamma', \Psi \vdash P} \qquad [\text{T-OUT}]\ \frac{\begin{array}{r} T \hookleftarrow T' \\ \Gamma, \Psi \vdash M : T \\ \Gamma, \Psi \vdash N : T' \end{array}}{\Gamma, \Psi \vdash P}$$

$$[\text{T-PAR}]\ \frac{\begin{array}{c} \Gamma, \mathcal{F}_\nu(Q)\,, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \\ \Gamma, \mathcal{F}_\nu(P)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \end{array}}{\Gamma, \Psi \vdash P \mid Q} \begin{pmatrix} \mathcal{F}_\nu(P) \,\#\Psi, \mathcal{F}_\nu(Q)\,, Q \\ \mathcal{F}_\nu(Q) \,\#\Psi, \mathcal{F}_\nu(P)\,, P \end{pmatrix}$$

$$[\text{T-NEW}]\ \frac{\Gamma, x : T, \Psi \vdash P}{\Gamma, \Psi \vdash (\boldsymbol{\nu} x : T)\, P}\ (x\#\Psi) \qquad [\text{T-NIL}]\ \frac{}{\Gamma, \Psi \vdash \mathbf{0}} \qquad [\text{T-REPL}]\ \frac{\Gamma, \Psi \vdash P}{\Gamma, \Psi \vdash\ !\, P}$$

$$[\text{T-CASE}]\ \frac{\begin{array}{c} \Gamma, \Psi \vdash \varphi_i \\ \Gamma, \Psi \vdash P_i \end{array}}{\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}} \qquad [\text{T-ASSERT}]\ \frac{\Gamma, \Psi \vdash \Psi'}{\Gamma, \Psi \vdash (\!|\Psi'|\!)}$$

Figure 5.1: Type judgements for processes

Secondly, they consist of the two relations $T \hookleftarrow T'$ and $T \curvearrowleft \Gamma$, where the latter specifies how how we can obtain a type environment from a type. These type judgements and relations are defined as follows:

Definition 5.4 (Instance parameters for the generic HOΨ-calculus type system)**.** *Let type judgements for terms, conditions and assertions, be of the form* $\Gamma, \Psi \vdash \mathcal{J}$*, where* $\mathcal{J}$ *is defined by the following syntax:*

$$\mathcal{J} ::= M : T \ \mid\ \Psi \ \mid\ \varphi$$

In order to instantiate the type system we require a set $\mathcal{T}$ *containing types* $T \in \mathcal{T}$ *and inference rules which define the following type judgements and relations:*

$$\begin{array}{r} [\text{T-TERM}]\ \Gamma, \Psi \vdash M : T \\ [\text{T-ASS}]\ \Gamma, \Psi \vdash \Psi' \\ [\text{T-COND}]\ \Gamma, \Psi \vdash \varphi \\ [\text{T-CHA}]\ T_1 \hookleftarrow T_2 \\ [\text{T-ENV}]\ T \curvearrowleft \Gamma \end{array}$$

The relation $\hookleftarrow$ is used in [T-IN] and [T-OUT] to obtain the type of the object in an input or output prefix. As an example, consider the channel types in the sorting system by Milner [26], where we can define the relation to be $\mathrm{ch}(T) \hookleftarrow T$.

A key motivation for the present type system is the ability to also type the higher-order behaviour afforded by the **run** M construct, which is done with the [T-RUN] type judgement rule. It is therefore perhaps here worth elaborating on some of our considerations underlying the definition of this particular rule. We *could* have let the process obtained from a **run** M operator be typed in the same type environment as the run operator, such that we would have the following rule:

$$\frac{\Psi \Vdash M \Leftarrow P \qquad \Gamma, \Psi \vdash P}{\Gamma, \Psi \vdash \mathbf{run}\ M}$$

However, in order to increase the expressive power of the type system, we also want to handle which names and types are allowed inside a process, which results in the current type judgement rule [T-RUN]. We therefore need a relation $\curvearrowleft$ to extract the type environment from a type. As a simple example, suppose that every type of a term would consists of a channel component and a run type component (T, Γ); then we could define the $\curvearrowleft$ relation to be $(T, \Gamma) \curvearrowleft \Gamma$.

In order to show subject reduction for our type system, we need to show some properties holds for the type system, described by Lemma 3, Lemma 5, Lemma 4, Lemma 6 and Lemma 7. In order to do that we need some requirements/assumptions for the instance parameters in definition 5.4, and we denote them the *instance assumptions for the generic* HOΨ*-calculus*. However, before we can define them, we need the notion of a preorder on frames:

Definition 5.5 (Frame preorder)**.** *We say that*

$$\left(\boldsymbol{\nu}\widetilde{x_1} : \widetilde{T_1}\right)\Psi_1 \leq \left(\boldsymbol{\nu}\widetilde{x_2} : \widetilde{T_2}\right)\Psi_2$$

if $\widetilde{x_1} \subseteq \widetilde{x_2}$ *and* $\Psi_2 = \Psi_1 \otimes \Psi$ *for any* Ψ*.*

Definition 5.6 (Instance assumptions for the generic HOΨ-calculus type system)**.** *To yield a valid instance of the generic type system, the following assumptions must holds for all instance parameters:*

$$\begin{aligned}
[\text{T-ENV-WEAK}]\ & \Gamma, \Psi \vdash \mathcal{J} \implies \Gamma, x : T, \Psi \vdash \mathcal{J} \\
[\text{T-ENV-STRENGTH}]\ & \Gamma, x : T, \Psi \vdash \mathcal{J} \wedge x \notin \mathrm{n}(\mathcal{J}) \implies \Gamma, \Psi \vdash \mathcal{J} \\
[\text{T-COMP-TERM}]\ & \Gamma, \Psi \vdash \mathcal{J}[\widetilde{x} := \widetilde{L}] : F(\widetilde{T}) \implies \Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \\
[\text{T-ASS-WEAK}]\ & \Gamma, \Psi \vdash \mathcal{J} \wedge \Psi \leq \Psi' \wedge \mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \implies \Gamma, \Psi' \vdash \mathcal{J} \\
[\text{T-SUBS}]\ & \Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \wedge \Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \mathcal{J} \implies \Gamma, \Psi \vdash \mathcal{J}[\widetilde{x} := \widetilde{L}] \\
[\text{T-EQUAL}]\ & \Gamma, \Psi \vdash M : T \wedge \Psi \Vdash M \leftrightarrows N \implies \Gamma, \Psi \vdash N : T \\
[\text{T-ENV-CLAUS}]\ & \Gamma, \Psi \vdash M : T \wedge T \curvearrowleft \Gamma' \implies \mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Gamma') \\
[\text{T-WEAK-ASS-CLAUS}]\ & \Psi \Vdash M \Leftarrow P \wedge \Gamma, \Psi \vdash M \Leftarrow P \wedge \Psi \leq \Psi' \wedge \mathrm{n}(\Psi) \subseteq \Gamma \\
& \implies \Psi' \Vdash M \Leftarrow P
\end{aligned}$$

$$[\text{T-SUBS-RUN}]\ \Gamma, \Psi \vdash M : T \wedge T \curvearrowleft \Gamma' \wedge \Psi \Vdash M[\widetilde{x} := \widetilde{L}] \Leftarrow P \implies \Gamma', \Psi \vdash P$$

We also require that for all types $T \in \mathcal{T}$ and names x, that $x\#T$; i.e. we do not allow atomic names in types.

We have labelled each of the assumptions such that we may refer to them individually in the proofs for the type system. [T-ENV-WEAK], [T-ENV-STRENGTH], [T-COMP-TERM], [T-ASS-WEAK], [T-SUBS] and [T-EQUAL] are similar to in the assumptions of Hüttel [17].

The assumption [T-ENV-CLAUS] is used to prove Lemma 5: Specifically, we use it to weaken the assertion environment in the type judgement of a process P for which M is a handle, as in the following example:

$$\Gamma, \Psi \vdash \underline{M}(\lambda x)x.(\!|\{\, M \Leftarrow P \,\}|\!) \mid \mathbf{run}\ M$$

Here we firstly type the process P, and since P might contain x, P is well-typed in $\Gamma, x : T$. When we then type the **run** M operator, we must have that $\Gamma, \vdash M : T$, $T \curvearrowleft \Gamma, x : T$ and $\Gamma, x : T, \Psi \vdash P$. Here we clearly see that the assumption [T-ENV-CLAUS] holds, since $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Gamma, x : T)$.

The assumption [T-WEAK-ASS-CLAUS] is used in the proof of Lemma 5. It states that if M is a handle for P, then M must till remain a handle for the same process P if the assertion environment is weakened. An example of a definition that might not comply with this assumption could be if an assertion was able to 'delete' or otherwise invalidate handles in another assertion:

$$(\!|\{\, x \Leftarrow P \,\}|\!) \otimes (\!|\{\, \mathrm{delete}\ x \,\}|\!) = \emptyset$$

Regardless, we would still be allowed to delete handles if they are never used in any conditions. And in any case it might be difficult in practice to define the composition operator $\otimes$ such that is associative, as required in definition 2.11, if such a *delete* operator was included in the assertion language, so this might arguably be a corner case.

Lastly, [T-SUBS-RUN] is used to prove Lemma 7: It states that if a term M becomes a handle for a new process P after a substitution, then the new process must still be well-typed in the environment we obtain from M's type T.

5.4 Soundness and safety

Now we wish to show subject reduction for our type system. For this, we shall use the approach to type system soundness advocated by Wright and Felleisen [38], by some called 'syntactic typing.' Here *soundness* is based upon the notion of *subject reduction*, also known as *preservation*, which states that a well-typed process remains well-typed after a transition or reduction step. This is a necessary condition for

soundness, but it is not sufficient, since we also need to prove that programs containing type errors are not typable. However, we cannot show this result with the current, generic setup, since the notion of a type error is dependent on the definition of the specific type system; that is, the *instance* obtained by parameter settings of the generic type system. We will therefore only show subject reduction, and then give a notion of how type errors can be defined (cf. theorem 4).

We shall build our way towards this result through a series of lemmas, beginning with the usual results of *weakening* and *strengthening* of the type environment:

Lemma 1 (Weakening). *If* $\Gamma, \Psi \vdash P$ *then* $\Gamma, x : T, \Psi \vdash P$

Proof. We use proof by induction on the rules of $\Gamma, \Psi \vdash P$. The proof can be found in appendix C.1. □

Lemma 2 (Strengthening). *If* $\Gamma, x : T, \Psi \vdash P$ *and* $x \# P, \Psi$ *then* $\Gamma, \Psi \vdash P$

Proof. We use proof by induction on the rules of $\Gamma, x : T, \Psi \vdash P$. The proof can be found in appendix C.2. □

We shall also need two results, called *frame post reduction* and *frame post evaluation*. The former, given in Lemma 3 is used in the proof of subject reduction (theorem 3) to find any new assertions that may have become composed onto the pre-existing assertion environment after a reduction. The latter, given in Lemma 4 states that the assertions in a process are unaltered by an evaluation: This is mainly ensured by the criteria for well-formed processes (cf. definition 2.15) asserting that all processes in a case operator, and all processes spawned by a **run** M operator, must be assertion guarded and hence cannot contain any unguarded assertions, and the proof establishes that the property of being assertion-guarded is preserved by the evaluation relation $\underline{\ggg}$:

Lemma 3 (Frame post reduction). *If* $\Psi \rhd P \to P'$ *then* $\mathcal{F}(P) \leq \mathcal{F}(P')$

Proof. Very simple induction proof on the rules of $\Psi \rhd P \to P'$. For [R-COM], $\mathcal{F}(P)$ is empty and we extend it with the frames in the continuations. For [R-EVAL] we use Lemma 4 to show that the frames are equal, and then we use the induction hypothesis. For [R-RES] we simply apply the induction hypothesis. For [R-PAR] we use the induction hypothesis and show that the extended frame simply extend the frame found in the parallel process. □

Lemma 4 (Frame post evaluation). *If* $\Psi \rhd P \underline{\ggg} P'$ *then* $\mathcal{F}_\Psi(P) = \mathcal{F}_\Psi(P')$.

Proof. We use a simple proof by induction on the rules of $\Psi \rhd P \underline{\ggg} P'$. For [E-RES] and [E-PAR] we use the induction hypothesis. For [E-CASE] and [E-RUN] we use definition 2.15 which ensures that no unguarded assertion can exist in processes of a case operator, nor in any processes spawned by the **run** M operator. □

Next we show *assertion environment weakening*, given in Lemma 5. This lemma states that any process that is well-typed remains well-typed after a composition of any assertion in the assertion environment, so long as all names in the new assertion environment are in the support of the type environment. This lemma is necessitated by the syntax of HOΨ-calculus itself, which allows guarded assertions in continuations to become unguarded after a reduction. It is used in the proof of subject reduction to extend the assertion environment with the assertion found by *frame post reduction* (Lemma 3). The instance assumptions [T-ASS-WEAK], [T-ENV-CLAUS] and [T-WEAK-ASS-CLAUS] are used to prove this lemma.[3]

Lemma 5 (Assertion environment weakening)**.**
If $\Gamma, \Psi \vdash P$, $n(\Psi') \subseteq dom(\Gamma)$ *and* $\Psi \leq \Psi'$ *then* $\Gamma, \Psi' \vdash P$.

Proof. We use proof by induction in the rules of $\Gamma, \Psi \vdash P$. The proof can be found in appendix C.3. □

Hereafter we show *subject evaluation*, given in Lemma 6. This lemma states that a well-typed process remains well-typed after an evaluation. For the proof, we use Lemma 4 to keep any assertions in a parallel process unaltered. The lemma is used in the subject reduction theorem (theorem 3) in the [R-EVAL] rule.

Lemma 6 (Subject evaluation)**.** *If* $\Gamma, \Psi \vdash P$ *and* $\Psi \rhd P \underline{\ggg} P'$ *then* $\Gamma, \Psi \vdash P'$.

Proof. We use proof by induction in the rules for $\Psi \rhd P \underline{\ggg} P'$. The proof can be found in appendix C.4. □

The last lemma is *subject substitution*, given in Lemma 7. This lemma states that a well-typed process remains well-typed after a substitution, as long as the substituted names and the terms used for the substitution have the same type. It is used in the proof of subject reduction in the [R-COM] case, when we use substitution on the continuation of an input operation. This lemma requires the instance assumptions [T-SUBS] and [T-SUBS-RUN].

Lemma 7 (Subject substitution)**.**
If $\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P$ *and* $\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T}$ *then* $\Gamma, \Psi \vdash P[x := \widetilde{L}]$.

Proof. We use proof by induction in the rules of $\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P$. The proof can be found in appendix C.5. □

Lastly we show our main result, which is the *subject reduction:*

Theorem 3 (Subject reduction)**.** *If* $\Gamma, \Psi \vdash P$ *and* $\Psi \rhd P \to P'$ *then* $\Gamma, \Psi \vdash P'$.

Proof. We use proof by induction in the rules for $\Psi \rhd P \to P'$. The proof can be found in appendix C.6. □

[3] A similar property can be found in the generic type system by Hüttel [17], where the relation $<_A$ expresses the weakening of assertions.

While we are unable to prove *safety* for the generic type system for the HOΨ-calculus, we *can* give a general notion of how a definition of safety might be expressed. The reason for this inability is that the notion of type errors are dependent on the formulation of type judgements for terms, conditions and assertions, which must alle be specified as parameters to the type system. A notion of type error can be described as

$$\Gamma, \Psi \vdash P \rightarrow \textbf{WRONG}$$

where, conversely, a process is said to be *well-behaved* if it is the case that $\Gamma, \Psi \vdash P \not\rightarrow$ **WRONG**. In order to obtain a result of safety, the following theorem would need to be shown:

Theorem 4 (Safety)**.** *Given a process P, a type environment Γ, and an assertion environment Ψ, if $\Gamma, \Psi \vdash P$ then $\Gamma, \Psi \vdash P \not\rightarrow$ **WRONG***

This result will have to be proved for each individual instance, but given such a result, in combination with the result of subject reduction which we have for any instance of the generic type system, we can then derive the following for any P':

$$\Psi \rhd P \rightarrow^* P' \wedge \Gamma, \Psi \vdash P \implies \Gamma, \Psi \vdash P' \not\rightarrow \textbf{WRONG}$$

* * *

In this chapter we have given a definition of a generic type system for the HOΨ-calculus and discussed some of our considerations underlying its definitions, including a number of instance assumptions that must be satisfied by every choice of setting of the instance parameters, both for the HOΨ-calculus and the type system itself, to yield a valid instance of the generic type system. The genetic type system extends previous work on a generic type system for the first-order Ψ-calculus by Hüttel [17], but our type system draws a stronger connexion between the assertion environment Ψ in a type judgement and in the semantics, and it also includes the ability to type the higher-order behaviour introduced in the HOΨ-calculus. Lastly, we showed safety for our generic type system, which holds for any instance of the generic type system. A discussion of an alternative for the generic type system can be found in chapter 7.

6 Type system instantiations

In this chapter we will utilize the generic type system from chapter 5 to define instances of it for different calculi. In the first section of this chapter (section 6.1), we will apply the generic type system to the simplified HOπ-calculus from section 4.2 in order to catch faults in the translation from the HOπ-calculus to the HOΨ-calculus.

In the second section of this chapter, we will apply the generic type system to the ρ-calculus. We will mainly focus on the first instantiation defined in section 4.3.1.

6.1 Type systems for the simplified HOπ-calculus

In this section we are going to instantiate the generic type system, such that we are able to type the simplified HOπ-calculus. The syntax for the simplified HOπ-calculus can be found in definition 4.12, the parameter setting can be found in definition 4.10 and the translation can be found in definition 4.11.

The syntax for the HOπ-calculus, allows for two different variables, namely first-order and higher-order variables. However this information is lost in the translation function, which can introduce communication problems. We can see this with the following example:

$$\overline{a}\!<\!P\!>\!.\mathbf{0} \mid a(x).\overline{x}\!<\!b\!>$$

Here we are attempting to send a process P over channel a, and substitute it for the variable x. The variable is then used as a channel to send the channel b. This process does not reduce since we can not substitute x for P. However when we translate this process we get:

$$\overline{a}P.\mathbf{0} \mid \underline{a}(\lambda x)x.\overline{x}b$$

which is able to reduce but will then enter a deadlock when we try to use process P as a channel to send name b. This is due to the definition of conditions in definition 4.10, which does not allow processes in the relation $\dot{\leftrightarrow}$. We could extend $\dot{\leftrightarrow}$ to allow processes, such that they are able to be used as channels, however this would introduce behaviour that is not part of the HOπ-calculus. Another problem that arises with this translation is in the process:

$$a\!<\!b\!>\!.\mathbf{0} \mid a(X).X$$

which substitutes a process variable for a channel, which is not allowed by the HOπ-calculus.

To solve this issue, we will make an instance of the generic type system, that captures the behaviour of names wrt. whether they should behave as channels or processes.

We will first define the types of terms to be:

$$T \in \mathcal{T} ::= \mathrm{ch}(T) \mid \mathrm{drop}(\Gamma)$$

The behaviour of channels and first-order variables is captured in the same manner as the simple sorting system for the π-calculus, by Milner [26]. Process terms and higher-order variables, will have the type $\mathrm{drop}(\Gamma)$, where the processes are well-typed in Γ.

Before defining an instance of the generic type system, we wish to define the type errors that we want to prevent. The type errors have to follow the structure found in section 5.4. The definition of the type errors can be found in figure 6.1.

We now define the instance parameters from definition 5.4:

$$[\textsc{t-con}]\frac{}{\Gamma, \Psi \vdash \top} \qquad [\textsc{t-ass}]\frac{}{\Gamma, \Psi \vdash (\!|\mathbf{1}|\!)}$$

$$[\textsc{t-cha}]\frac{}{\mathrm{ch}(T) \leftrightarrows T} \qquad [\textsc{t-end}]\frac{}{\mathrm{drop}(\Gamma) \curvearrowleft \Gamma}$$

$$[\textsc{t-term-1}]\frac{\Gamma(x) = \mathrm{ch}(T)}{\Gamma, \Psi \vdash x : \mathrm{ch}(T)} \qquad [\textsc{t-term-2}]\frac{\Gamma', \Psi \vdash P}{\Gamma, \Psi \vdash P : \mathrm{drop}(\Gamma')}$$

It should be fairly easy to see from definition 5.6, that all instance assumptions hold.

This type system instance is almost completely defined, but as of now it contains a flaw, namely the rule [T-TERM-2]. Here we can choose any Γ', such that the process P is well-typed, which trivialises the drop type. A better solution is to have the type environment in a drop type, be the same type environment that is exposed to the processes, when it is defined as an object in an output prefix, i.e. if we have $\Gamma, \Psi \vdash \overline{a}P$, we want $\Gamma, \Psi \vdash P : \mathrm{drop}(\Gamma)$. This way, when we run the process, we can recall the bound variables and their types at the time when the process was sent. In order to implement this feature, we need some mechanism for storing types of processes. We can not use the type environment, since processes can not be a member of the domain of a type environment. We could rewrite the translation function and parameter setting, such that we use atomic names as handles for processes, however a simpler solution exists. We can expand the definition for assertions, such that they behave as type environments for processes. We do

$$\frac{\Gamma, \Psi \vdash M : \mathrm{drop}(\Gamma') \quad \vee \quad \Gamma, \Psi \vdash Q \to \mathbf{WRONG}}{\Gamma, \Psi \vdash \overline{M}N.Q \to \mathbf{WRONG}}$$

$$\frac{\Gamma \vdash P : \mathrm{drop}(\Gamma') \quad \Gamma', \Psi \vdash P \to \mathbf{WRONG}}{\Gamma, \Psi \vdash \overline{M}P.Q \to \mathbf{WRONG}}$$

$$\frac{\Gamma, \Psi \vdash M : \mathrm{drop}(\Gamma')}{\Gamma, \Psi \vdash \underline{M}(\lambda x)x.Q \to \mathbf{WRONG}} \qquad \frac{\Gamma, \Psi \vdash M : \mathrm{ch}(T) \quad \Gamma, x : T, \Psi \vdash Q \to \mathbf{WRONG}}{\Gamma, \Psi \vdash \underline{M}(\lambda x)x.Q \to \mathbf{WRONG}}$$

$$\frac{\Gamma, \Psi \vdash M : \mathrm{ch}(T)}{\Gamma, \Psi \vdash \mathbf{run}\ M \to \mathbf{WRONG}} \qquad \frac{\Gamma, \Psi \vdash P : \mathrm{drop}(\Gamma') \quad \Gamma', \Psi \vdash P \to \mathbf{WRONG}}{\Gamma, \Psi \vdash \mathbf{run}\ P \to \mathbf{WRONG}}$$

$$\frac{\Gamma, \mathcal{F}_\nu(Q)\,, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \to \mathbf{WRONG}}{\Gamma, \Psi \vdash P \mid Q \to \mathbf{WRONG}}\ (\mathcal{F}_\nu(Q)\,\#\Psi, \mathcal{F}_\nu(P)\,, P)$$

$$\frac{\Gamma, \mathcal{F}_\nu(P)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \to \mathbf{WRONG}}{\Gamma, \Psi \vdash P \mid Q \to \mathbf{WRONG}}\ (\mathcal{F}_\nu(P)\,\#\Psi, \mathcal{F}_\nu(Q)\,, Q)$$

$$\frac{\Gamma, \Psi \vdash P \to \mathbf{WRONG}}{\Gamma, \Psi \vdash\ !\,P \to \mathbf{WRONG}} \qquad \frac{\Gamma, x : T, \Psi \vdash P \to \mathbf{WRONG}}{\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\,P \to \mathbf{WRONG}}\ (x\#\Psi)$$

$$\frac{\Gamma, \Psi \vdash P_i \to \mathbf{WRONG}}{\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\top} : \widetilde{P} \to \mathbf{WRONG}}$$

Figure 6.1: Type errors for the instance of the HOπ-calculus.

this with the following definition:

$$\begin{aligned}
&\mathbb{A} \triangleq \mathscr{P}\big(\{\, P : T \;\mid\; P \in \mathcal{P} \wedge T \in \mathcal{T} \,\}\big)\\
&\otimes \triangleq \cup\\
&\mathbf{1} \triangleq \emptyset\\
&\Vdash \triangleq \Psi \times \Big(\Big\{\, a \leftrightarrows a \;\mid\; a \in \mathcal{N} \,\Big\} \cup \{\, P \Leftarrow P \;\mid\; P \in \mathcal{P} \,\} \cup \{\, \top \,\}\Big)
\end{aligned}$$

And we extend the encoding for output such that we have:

$$[\![a\!<\!P\!>\!.Q]\!] \triangleq \overline{a}[\![P]\!].[\![Q]\!] \mid (\!|\{\, [\![P]\!] : T \,\}|\!)$$

For the instance parameters, we change the rule for process names and assertions:

$$[\text{T-TERM-2}]\frac{P : \text{drop}(\Gamma') \in \Psi \quad \Gamma', \Psi \vdash P}{\Gamma, \Psi \vdash P : \text{drop}(\Gamma')} \qquad [\text{T-ASS}]\frac{P : \text{drop}(\Gamma) \in \Psi'}{\Gamma, \Psi \vdash (\!|\Psi'|\!)}$$

We can now show safety for the type system.

Theorem 5 (Safety for the HOπ-calculus type system)**.** *If* $\Gamma, \Psi \vdash P$ *then* $P \not\to$ ***WRONG****.*

Proof. We use proof by induction on the rules of $\Gamma, \Psi \vdash P$. The proof can be found in appendix D.1. □

6.2 Type systems for the ρ-calculus

In this section we are going to instantiate the generic type system, such that we obtain a type system for the ρ-calculus. However we will see that a sound type system for the ρ-calculus is not obtainable, and argue that the generic type system for the Ψ-calculus, is not suitable for typing the ρ-calculus.

In the previous section (section 6.1), we saw a type system for the HOπ-calculus. The ρ-calculus and the modified HOπ-calculus behave mostly in the same manner, except for one difference: In the modified HOπ-calculus, names can behave either as channels or as processes, while in the ρ-calculus names can behave as channels *and* as processes.

This difference is reflected in the definition of types, where types reflect the behaviour of names in a disjunctive manner:

$$T \in \mathcal{T} ::= \text{ch}(T) \mid \text{drop}(\Gamma)$$

For the ρ-calculus, the types will reflect the behavior of names in a conjunctive manner:

$$T \in \mathcal{T} ::= \langle T, \Gamma \rangle$$

In section 4.3, we defined two instantiations of the ρ-calculus in the Ψ-calculus. The first instantiation uses code mobility, and the second instantiation uses process handles. We are going to mainly focus on the first instantiation, since the second instantiation assumes that all scopes are maximally extruded, which impedes typing the instance.

Before we define the instance parameters, we are going to modify the Ψ-calculus parameters. From section 6.1 we saw that we could use assertions as type environments for processes, and we will use the same mechanism for the ρ-calculus. We therefore redefine assertions, assertion composition and the assertion unit.

$$\begin{aligned}\mathbb{A} \triangleq\ & \mathscr{P}\big(\{\, \ulcorner P \urcorner : T \mid P \in \mathcal{P} \wedge T \in \mathcal{T} \,\}\big) \\ & \cup\ \mathscr{P}\big(\{\, (\!|\ulcorner P \urcorner|\!) : T \mid P \in \mathcal{P} \wedge T \in \mathcal{T} \,\}\big)\end{aligned}$$

In the same manner as the definition of terms, $\ulcorner P \urcorner : T$ indicates that we are not able to substitute inside P, and $\langle\!|\ulcorner P \urcorner|\!\rangle : T$ indicates that we are able to substitute inside P. Note this implies that $\forall x \in \mathcal{N}.x\#\ulcorner P \urcorner$.

Hereafter we can modify the translation such that we have:

$$\begin{aligned}
[\![\ulcorner R \urcorner \langle\!| P |\!\rangle]\!] &\triangleq \overline{\ulcorner [\![R]\!] \urcorner} \langle\!| \ulcorner [\![P]\!] \urcorner |\!\rangle .\mathbf{0} \mid \langle\!|\{\ \ulcorner [\![R]\!] \urcorner : T, \langle\!| \ulcorner [\![P]\!] \urcorner |\!\rangle : T' \ \}|\!\rangle \\
[\![\ulcorner R \urcorner (x) .P]\!] &\triangleq \underline{\ulcorner [\![R]\!] \urcorner}(\lambda x) \langle\!| x |\!\rangle .[\![P]\!] \mid \langle\!|\{\ \ulcorner [\![R]\!] \urcorner : T \ \}|\!\rangle
\end{aligned}$$

Now we can instantiate the generic type system, by defining the instance parameters:

$$[\textsc{t-ass}]\ \frac{\begin{array}{c} P : T \in \Psi' \implies T \curvearrowleft \Gamma \\ P \equiv Q \wedge \Gamma, \Psi \vdash P : T \wedge \Gamma, \Psi' \vdash Q : T' \implies T = T' \end{array}}{\Gamma, \Psi \vdash \langle\!|\Psi'|\!\rangle}$$

$$[\textsc{t-term-1}]\ \frac{\Gamma, \Psi \vdash P : T}{\Gamma, \Psi \vdash \ulcorner P \urcorner : \langle T, \Gamma' \rangle} \qquad [\textsc{t-term-2}]\ \frac{\Gamma, \Psi \vdash P : T}{\Gamma, \Psi \vdash \langle\!| \ulcorner P \urcorner |\!\rangle : T}$$

$$[\textsc{t-term-3}]\ \frac{\Gamma(x) = T}{\Gamma, \Psi \vdash x : T} \qquad [\textsc{t-cha}]\ \langle T, \Gamma \rangle \hookleftarrow T \qquad [\textsc{t-end}]\ \langle T, \Gamma \rangle \curvearrowleft \Gamma$$

and $\Gamma, \Psi \vdash P : T$ is defined as:

$$[\textsc{t-proc-type}]\ \frac{\begin{array}{ccc} T \curvearrowleft \Gamma' & \wedge & \Gamma', \Psi \vdash P \\ \ulcorner P \urcorner : T \in \Psi & \vee & \langle\!| \ulcorner P \urcorner |\!\rangle : T \in \Psi \end{array}}{\Gamma, \Psi \vdash P : T}$$

The rule [T-ASS] first ensures that the second component of every type in the assertion, is the same type environment as the type environment for the judgement. Next it checks the assertion agrees with the assertion environment i.e. that every process has the same type in both assertions.

The [T-PROC-TYPE] rule checks that the process is well-typed in regard to its type environment, found in the second components of its type. Hereafter it checks that the process and type pair are represented in the assertion.

While this seems to be the most approachable definition to a type system for the ρ-calculus, this type system is not sound. Admittedly, we have neglected one of the instance assumptions for the generic type system. However before showing which instance assumption, we wish to illustrate what problems arise with this type system, such that we have a better understanding of why typing the ρ-calculus is hard.

Consider the following example

$$\begin{aligned}
&\overline{\ulcorner R_1 \urcorner}\,\langle\!|\ulcorner P \urcorner|\!\rangle \\
\mid\ &\underline{\ulcorner R_1 \urcorner}(\lambda y)\,\langle\!|y|\!\rangle\,.(\overline{\ulcorner R_2 \urcorner}\,\langle\!|\ulcorner Q \urcorner|\!\rangle \ \mid\ (\!|\{\,\langle\!|\ulcorner Q \urcorner|\!\rangle : T_1\,\}|\!)) \\
\mid\ &\underline{\ulcorner R_2 \urcorner}(\lambda x)\,\langle\!|x|\!\rangle\,.\overline{x}\,\langle\!|\ulcorner R_3 \urcorner|\!\rangle \\
\mid\ &\underline{\ulcorner Q[y := P] \urcorner}(\lambda z)\,\langle\!|z|\!\rangle\,.\checkmark \ \mid\ (\!|\{\,\ulcorner Q[y := P] \urcorner : T_2\,\}|\!)
\end{aligned}$$

We have excluded assertions for R_1, R_2, R_3 and P, since they are not needed to illustrate the underlying problem. The process will reduce to success and is well-typed, however it is not necessarily sound. If $T_1 = T_2$ the process is sound, since after one reduction we have

$$\begin{aligned}
&\overline{\ulcorner R_2 \urcorner}\,\langle\!|\ulcorner Q[y := \ulcorner P \urcorner] \urcorner|\!\rangle \ \mid\ (\!|\{\,\langle\!|\ulcorner Q[y := \ulcorner P \urcorner] \urcorner|\!\rangle : T_1\,\}|\!) \\
\mid\ &\underline{\ulcorner R_2 \urcorner}(\lambda x)\,\langle\!|x|\!\rangle\,.\overline{x}\,\langle\!|\ulcorner R_3 \urcorner|\!\rangle \\
\mid\ &\underline{\ulcorner Q[y := P] \urcorner}(\lambda z)\,\langle\!|z|\!\rangle\,.\checkmark \ \mid\ (\!|\{\,\ulcorner Q[y := P] \urcorner : T_2\,\}|\!)
\end{aligned}$$

and we can see the assertions agree on the type of $Q[y := \ulcorner P \urcorner]$. However if $T_1 \neq T_2$ then the assertions will not agree.

The instance assumption that does not hold, is the [T-WEAK-ASS-CLAUS] assumption. It states:

$$\Psi \Vdash M \Leftarrow P \wedge \Gamma, \Psi \vdash M \Leftarrow P \wedge \Psi \leq \Psi' \wedge \mathrm{n}(\Psi) \subseteq \Gamma \implies \Psi' \Vdash M \Leftarrow P$$

which does not hold from the second premise in the [T-ASS] type rule.

Since we have only typed the first instantiation (from section 4.3.1) of the ρ-calculus in the HOΨ-calculus, it is worth considering the second instantiation of the ρ-calculus (from section 4.3.2). Here instead of having processes as terms, we have atomic names that handle the processes. Unfortunately, the problems in the example above would also apply to this instantiation. We can see this if we rewrite the example:

$$\begin{aligned}
&(\boldsymbol{\nu} z)\,\overline{r_1} z_1.(\!|\{\, z \Leftarrow P\,\}|\!) \\
\mid\ &\underline{r_1}(\lambda y)y.\,(\boldsymbol{\nu} z_2)\,\overline{r_2} z_2.(\!|\{\, z_2 \Leftarrow Q\,\}|\!) \\
\mid\ &\underline{r_2}(\lambda x)x.\,(\boldsymbol{\nu} r_3)\,\overline{x} r_3.(\!|\{\, r_3 \Leftarrow R_3\,\}|\!) \\
\mid\ &\underline{q}(\lambda z)\,\langle\!|z|\!\rangle\,.\checkmark \ \mid\ (\!|\{\, q \Leftarrow Q[y := P]\,\}|\!)
\end{aligned}$$

Here z_2 and q are well-typed even if they have different types, and after one reduction we have $\{\, z_2 \Leftarrow Q[y := P], q \Leftarrow Q[y := P]\,\} \rhd z_2 \mathrel{\dot{\leftrightarrow}} q$, which makes the process not well-typed.

However, we are able to make the type system sound. For the first type system instance of the ρ-calculus, we can modify the instantiation of the ρ-calculus, such that channel equivalence is also determined from the type of the terms. We do this by including the type environment in the definition of entailment. Hereafter we redefine the definition of entailment:

$$[\textsc{Chaneq}_2]\frac{\begin{array}{c}\Gamma, \Psi \Vdash P_1 \equiv P_2 \\ \Gamma, \Psi \vdash P_1 : T \iff \Gamma, \Psi \vdash P_2 : T\end{array}}{\Gamma, \Psi \Vdash \ulcorner P_1 \urcorner \dot{\leftrightarrow} \ulcorner P_2 \urcorner}$$

and remove the second premise of [T-ASS] such that it becomes

$$[\textsc{t-ass}]\ \frac{P : T \in \Psi' \implies T \curvearrowleft \Gamma}{\Gamma, \Psi \vdash (\!|\Psi'|\!)}$$

We also include the type environment in the definition of structural congruence, since channel equivalence appears inside structural congruence (see definition 4.20). While this type system is sound, it may not be desirable since:

- The definition of the reduction relation needs to include type environments, which the processes are well-typed in.
- Quoted processes that eventually becomes structurally congruent, need to have same type. We therefore need to predict which names become equal, if we want these names to behave in the same manner as the ρ-calculus. This suggest a more semantic approach to typing (see section 7.2).

* * *

In this chapter we have described two type system instances. One for the HOπ-calculus, and one for the ρ-calculus. We have shown that the type system for the HOπ-calculus, can capture defects in the translation, where incorrect use of channels results in processes that are not well-typed. We were able to formulate a type system for the ρ-calculus, however this type system was not sound, since it did not fulfill all instance assumptions. We have shown that the generic type system for the HOΨ-calculus is not well suited to type the ρ-calculus. Considerations for alternatives can be found in chapter 7.

7 Reflections on reflection

We are come at last to the end of our endeavours, and in this final chapter we shall briefly review and relate our main findings and conclude upon them. Then lastly, reflecting in hindsight upon our work, we shall also discuss two alternative approaches to the problem of typing reflection.

7.1 Discussion and conclusion

We set out in chapter 1 with the aim of investigating whether the higher-order Ψ-calculus might be able to represent reflection, and if so, whether it might be typable. Our investigations have shown that the ρ-calculus is indeed representable therein, and, equally important, that the main enabling factor lay not in possibility of having an arbitrarily complex term language, nor entirely in the higher-order capability of HOΨ, but first and foremost in the ability to define the *entailment relation* such that channel equivalence may mimic the name equivalence relation of the ρ-calculus.

This is, in hindsight, perhaps not surprising, given our findings in a previous study [2], where our failure to encode the full ρ-calculus in a variant of the π-calculus seemed to stem from its inability to represent name equivalence of structured terms through exact syntactic equivalence. We saw a glimpse of this already in chapter 1, where we sketched a proof of separation between the π-calculus and the ρ-calculus, supporting our conjecture from the previous study [2].

We have also seen, in chapter 4, how a purposefully ill-sorted, polyadic instantiation of the ${}^{e}\pi$-calculus, which is a first-order calculus, might be used to generate names with *structure* of increasing complexity at runtime, thereby obviating the need for a ν-operator. No higher-order capabilities are required for this, but if the calculus *were* to be lifted to a higher-order calculus without further modifications, it would immediately be subject to the equivalent of a null-pointer dereferencing runtime error, because there would be no guarantee that a thusly generated name vector $\widetilde{x}$ would be a handle for any process P that then might be executed with a **run** $\widetilde{x}$.

The ρ-calculus avoids this problem precisely because it combines name generation and higher-order process mobility into *one* operation, the lift $x\,\langle\!|P|\!\rangle$, which thus ensures that every name is associated with a process; namely the process found within the name itself. However, this only becomes obvious when viewed through

the lens of the HOΨ-framework, because it precisely allows us to separate these two capabilities. Also, in light of our developments in later chapters, one might wonder whether it would be possible to create an instance of the generic type system to catch this kind of runtime error, but although we have not pursued this question any further, we strongly suspect that it will not be possible, since it seems to require us to be able to predict which names will be generated at runtime.

Another contribution of our present work consists in the development of reduction semantics for the higher-order Ψ-calculus. This was done primarily to ease the task of proving subject reduction for the generic type system, but it turned out to be quite challenging to match the τ-labelled transition relation exactly, and we ended up proposing three different definitions of a reduction relation. The problems pertained particularly to unfolding **case** and **run** M expressions, which, due to the symmetry of structural congruence, might allow these expressions to be rewritten into processes without performing a reduction. Åman Pohjola [39] solved this problem for the first-order Ψ-calculus by using reduction contexts instead of a structural rule for parallel composition, and because of the similarity between the **case** and **run** M constucts, we were able to adapt his solution to the higher-order Ψ-calculus by introducing a rule allowing a **run** M to be rewritten by structural congruence into a **case** expression on a certain form that mimics the [Run] rule.[1] Regardless, the solution seems less than elegant, and we therefore instead decided to base our type system on another formulation of a reduction semantics, which is slightly larger than the τ-labelled transition relation. We do not presently know whether or how it may possible to create a reduction semantics that exactly matches the τ-labelled transition relation, without resorting to this type of rewriting trick.

Last, but not least, our main contribution in the present report consists in the development of a generic type system for the HOΨ-calculus, based upon the generic type system for first-order Ψ-calculi by Hüttel [17]; and showing an equally generic result of subject reduction. This property is then automatically inherited by any instance of the type system that satisfies a number of assumption about the types and type judgements for the sets of terms, conditions and assertions. To our knowledge, the generic type system is the first of is kind for *higher-order* calculi, yet it is still only a first step in this direction. Hüttel [18] extends his 'first-order' generic type system with capabilities, and then later, in [19] he also develops session types for the first-order Ψ-calculus. Both of these extensions might be obvious next steps to consider as future work on our generic type system as well.

We have also shown that it is indeed possible to instantiate the generic type system to yield a type system for a higher-order calculus, specifically our instantiation of the HOπ-calculus. We do not know whether this instance corresponds to any known type system for HOπ. However, we were yet again unable to type the ρ-calculus without introducing further constraints into the language. Specif-

[1] Parrow et al. [28] also note this likeness, when they discuss the conditions under which the **case** expression (and the replication operator) may be abolished, because they can be encoded with the **run** M construct. One might then say that we (ab)use this similarity in the opposite direction.

ically, we found that it would be necessary to include type information into the definition of name equivalence, creating a *typed channel equivalence* to ensure that name-equivalent names also would have to have the same *type* to be allowed to communicate. This is necessitated by the fact that the *type* of a ρ-calculus name is not derived from the *structure* of the name, which thus makes it possible to build two names that are channel equivalent, but nevertheless have different types. Thus, this example yet again serves to illustrate that the primary difficulty in typing the ρ-calculus derives from its notion of name equivalence, and the corresponding capability for generating names with structure at runtime, and not its higher-order characteristics per se.

This negative result might also, in a more general sense, indicate that the generic type system might be unable to type reflection, regardless of its manifestation. One possible reason for why this might be the case is that the HOΨ-calculus itself is based upon the Ψ-calculus, which in turn is based on the π-calculus; and this calculus has no notion of higher-order behaviour, nor of reflection. As an alternative, we might instead attempt to develop a similarly generic framework based upon an entirely different calculus, that is also able to express higher-order behaviour, such as the *blue calculus* of Boudol [5]. We discuss this possibility further in section 7.3.

Lastly, as mentioned above, we *would* be able to type the ρ-calculus, by changing the definition of name equivalence, yet even this solution would still require us to know in advance which names will become equivalent at runtime, post substitution. This could also suggest that a syntactic approach to typing the ρ-calculus might not be sufficient, and that instead a *semantic approach* might be required. One possible way forward could be to consider bisimulation and modal logic, as Meredith and Radestock [21] do in another paper; yet as these authors themselves state, it is a *logic* and not a type system [21, p. 2]. To avoid abandoning the typing approach altogether, whilst still being able to take more of the runtime behaviour into account, we might instead consider the approach of Caires [6] and others, sometimes referred to as *semantic typing*. We shall discuss this method further immediately, in the following section.

7.2 Semantic typing

The goal of this investigation has from the outset been to find a suitable method for typing reflection and, in particular, to make a type system for the ρ-calculus. But as we have seen, this task has seemed riddled with problems when using the traditional method of syntactic typing, deriving mostly from the ability of this calculus to generate names with structure at runtime.

Dreyer [10] and Dreyer et al. [11] take a different approach to type system soundness by advocating instead a proof for *semantic type soundness*, and Dreyer shows how this approach can be used to type unsafe code, notably reflective code. Reflection is a central notion of the ρ-calculus, suggesting that this approach of semantic typing instead might be a tractable alternative. In this section we shall therefore

discuss this approach in some detail, to provide a starting point for future work in this direction.

7.2.1 Brief Introduction

The idea of semantic typing hearkens back at least to Milner's work with polymorphism in Milner [22], but it is best explained by contrasting it with syntactic typing. The usual process of defining a type system is built on the method of subject reduction as discussed in chapter 5. In this approach we define the types, the typing rules and then prove the theorems of safety and subject reduction. However, we define only a *syntax* for types, hence one might therefore refer to this approach as *syntactic typing*.

In contrast, semantic typing works by giving semantics to types. There are several advantages of this alternative approach, such as not having to define typing rules as in the case of Appel and McAllester [1]. It also affords a greater modularity in proofs compared to syntactic typing [6], and not least the ability to reason about unsafe code, as argued by Dreyer [10].

Caires [6] explains that type systems are just a specific realisation of logic, and we can use this point of view to reason about the syntax and semantics of types. Within logic, syntax and semantics give rise to a distinction between what is *provable* and what is *true*. Statements that are *true* are those that can be considered true with respect to some structure, in contrast to the *provable* statements, which are those for which we can construct a proof using a system of deduction. The relevant deductive system here consists of rules of inference, as this is the form that our typing rules are on. It is befitting that syntactic typing works through these rules of inference, as they prove statements by analysing syntax, and we write that a statement φ is provable with respect to a set of formulas Γ as $\Gamma \vdash \varphi$, where we can recognise the turnstile symbol from typing rules. Similarly, we write that a statement is true as $\Gamma \vDash \varphi$, and semantic typing makes use of this double turnstile in a similar way.

To further illustrate the difference between *provable* and *true* statements, we note that it is possible for a statement to be provable, but at the same time not be true. This occurs when the rules of inference are not 'logical,' and this vague notion of 'logical' is what we formally describe as the property of *soundness*. We say that a deduction system is *sound* if it is the case that every provable statement is also true, as illustrated by the following equation:

$$\Gamma \vdash \varphi \implies \Gamma \vDash \varphi$$

which has a complement in the form of *completeness*, that holds when it is the case that every formula that is true has a proof, illustrated as follows:

$$\Gamma \vDash \varphi \implies \Gamma \vdash \varphi$$

In syntactic typing, proving safety and subject reduction means proving that the type system is *sound*. However we note that most type systems cannot also

be *complete* at the same time. This gives rise to a formal characterisation of the *slack* of a type system as: the processes that are *not well-typed*, but nevertheless are *well-behaved*. As simple example, consider processes containing unsafe code that will never run. The type system will reject these processes, due to being able to see the unsafe code, but not being able to make any guarantees about their behaviour. Thinking in terms of provability and truth and considering the statements to be about well-typed processes, the slack of the type system consists of the statements that are true, but are not provable. This shows another advantage of semantic typing, as it may allow us to reason about safety of terms that otherwise reside in the slack.

Going back to an earlier point about the turnstile symbols in rules, given a syntactic type system with typing rules, we could replace all occurrences of $\vdash$ in those rules, with $\vDash$, corresponding instead to a semantic interpretation of the typing rules: Appel and McAllester [1] do this for their example of semantic types for the λ-calculus.

Despite looking very similar, the meaning changes significantly: First of all, as the typing rules have to do with provability, the rules state that we can prove that a term is well-typed under type T. But under semantic typing, a term is not necessarily syntactically well-typed under type T: it may not be. Instead, the semantic interpretation states that the term behaves as the type T, as defined by the semantic *meaning* of the type. Additionally, typing rules define what is syntactically well-typed and will in turn affect the soundness of the syntactic type system, but in the semantic type system, we are not proving statements. Thus the semantic interpretation of the rules is not the definition of when semantic typing holds, but lemmas that will have to be proven true.

To actually define the semantics of types, we use *logical relations*. Logical relations are used for proving that the definition of some structure satisfies a certain property [34]. Rather than proving the correlation directly, logical relations proofs are structured in two parts: First proving that the definition places all processes in some relation, and then proving that all processes in the relation satisfies the desired property, thereby using the relation as an intermediate step. In semantic typing, the logical relation will be a set indexed by types containing the processes that are well-behaved. Thus when a process belongs to a type, it means that the process *behaves* as defined by the semantics of its type. The semantic approach to type soundness involves proving, instead of subject reduction and safety, *compatibility* and *adequacy*. 'Compatibility' states that syntactically well-typed processes are also semantically well-typed, and' adequacy' is the property that the semantic typing relation corresponds to safety.

7.2.2 Example

To understand semantic typing and logical relations, we will show an example of proving semantic type soundness for the π-calculus, as done by Caires [6]. We note that the specific variant of the π-calculus in this case is monadic and with a labelled transition system.

We begin by considering our notion of safe behaviour, as the purpose of a type system is to prevent certain runtime errors from happening. In the polyadic π-calculus it is normal to consider sorting systems that match the length of vectors being sent and received, but as the π-calculus under consideration here is monadic, we instead distinguish between channel names Λ_c and basic names Λ_v. In this case, we then want to ensure that only channel names are used for communication, and that basic names never appear as the subject of communication. Caires also notes that this can in fact be seen as a special case of arity, with basic names only being used at arity zero.

Definition 7.1 (Wrong predicate)**.** *The definition of* wrong *is as follows:*

$$\text{wrong}(P) = (P \equiv (\boldsymbol{\nu} m)\,(a(n).Q \mid R) \vee P \equiv (\boldsymbol{\nu} m)\,(a\langle n\rangle.Q \mid R)) \wedge a \in \Lambda_v$$

The wrong-predicate tells us that processes are wrong when they are on the form of an unguarded input or output with a basic name as the subject. Then, given a definition of wrongness, we can define *safe processes* as those that are not wrong:

Definition 7.2 (Safe predicate)**.** *The definition of* safe *is as follows:*

$$\text{safe}(P) = \neg\,\text{wrong}(p)$$

The distinction between channel names and basic names can then be further codified in our definition of the types:

Definition 7.3 (Types)**.** *The definition of the types is as follows:*

$$\begin{array}{rll} T ::= & nil & \textit{Base type} \\ \mid & \textit{ch}(T) & \textit{Channel type} \end{array}$$

Types, as is standard, are assigned to the free names and stored in a type environment, Γ. This defines the syntax of our types. However, we are using the approach of semantic typing, which makes the next natural step, to define the semantics of types. We will do this through a type environment-indexed logical relation that we refer to as the *typing interpretation:*

Definition 7.4 (Typing interpretation)**.** *The definition of the* typing interpretation *is as follows:*

$$\begin{aligned} P \in \mathcal{R}_{n:T} &\implies \text{safe}(P) \wedge (P \xrightarrow{\alpha} Q \implies Q \in \mathcal{R}_{n:T}) \\ P \in \mathcal{R}_{n:\textit{ch}(T)} &\implies (P \xrightarrow{n!x} Q \vee P \xrightarrow{n?x} Q) \implies Q \in \mathcal{R}_{n:T} \\ P \in \mathcal{R}_{n:nil} &\implies (P \xrightarrow{n} Q \implies \mathit{false}) \end{aligned}$$

Moreover, Caires defines the typing interpretation as a conjunctive mapping such that $\mathcal{R}_{\Gamma,x:T} = \mathcal{R}_\Gamma \cap \mathcal{R}_{x:T}$ holds. Firstly, the definition states that processes in the interpretation have to be safe, along with any possible reductions. Then processes using a channel type will have their reduct be in a typing interpretation of an

environment containing the object of communication. Lastly we require that base types may not be used as subjects of communication.

This defines the semantics for types and we can now see what it means for a process to be semantically well-typed under a given type, as a process will exhibit behaviour defined by its type.

Definition 7.5 (Semantically well-typed)**.** *A process P is* semantically well-typed *in a typing environment Γ when the following holds:*

$$\Gamma \vDash P \triangleq P \in \bigcup \boldsymbol{R}_\Gamma \wedge \mathrm{fn}\,(P) \in \mathrm{dom}(\Gamma)$$

The definition of the typing interpretation in this case is conjunctive, so we want P to be in the union of all typing interpretations, and we want to ensure that every free name in P is assigned a type. With this definition we can then go on to proofs of the necessary properties of the type system.

Theorem 6 (Compatibility)**.** *The semantic typing relation exhibits* compatibility *with syntactic typing when the following holds:*

$$(\Gamma \vdash P \wedge \mathrm{fn}\,(P) \in \mathrm{dom}(\Gamma)) \implies \Gamma \vDash P$$

Compatibility states that syntactically well-typed processes are also semantically well-typed. In some definitions, the condition on the free names of P is not present, like the one found in e.g. Dreyer [10]. We would prove this by induction on the syntactic type judgements.

Theorem 7 (Type safety)**.** *The semantic typing relation exhibits* type safety *with syntactic typing when the following holds:*

1. $\Gamma \vDash P \implies \mathrm{safe}(P)$
2. $(\Gamma \vDash P \implies \mathrm{safe}(P) \wedge P \to Q) \implies \Gamma \vDash Q$
3. $\Gamma \vdash P \wedge P \to Q \implies \mathrm{safe}(Q)$

In some cases we would prove adequacy instead, which, going by Dreyer [10], is defined only as

$$\Gamma \vDash P \implies \mathrm{safe}(P)$$

although the definition of type safety in Caires [6] has several more clauses. The proof of this longer type safety theorem is no more complicated, as the first and second clauses are proven simply by the definition of the typing interpretation, and the third clause follows from a combination of the two first and compatibility. The third clause is what tells us that our syntactic type system is sound by proofs through the semantic typing. Caires notes that this is perhaps not the most straightforward way to prove soundness, especially when considering a simple type system, but it serves its purpose as an example.

7.2.3 Semantic types in the ρ-calculus

To construct a semantic type system for the ρ-calculus, we once again have to consider what kind of type of errors we want to prevent. In section 6.2 we discussed conjunctive types, but we can, like Caires, consider this as just a special case of arity. Here we shall then distinguish between names used for communication, and names used for process mobility in the types:

Definition 7.6 (Types)**.** *We define the types for the ρ-calculus as follows:*

$$T ::= ch(T) \;\mid\; d$$

We can then define the notion of a *safe* process as any process where its names are used in accordance with their respective types. For this, we shall, however, need the concept of a *process context* C containing a single hole $[\,]$, to allow us to extract a drop process $\ulcorner x \urcorner$ of a bound name x from the continuation of an input construct. We define it in the usual way:

Definition 7.7 (Process context)**.** *Let the set $\mathcal{C}$ of ρ-calculus* process contexts*, ranged over by C, be defined by the formation rules:*

$$C \in \mathcal{C} ::= [\,] \;\mid\; C \mid P \;\mid\; x(y).C \;\mid\; x\langle\!|C|\!\rangle \;\mid\; \ulcorner x \urcorner$$

Definition 7.8 (Safe predicate)**.** *The definition of* safe *for the ρ-calculus is as follows:*

$$\begin{aligned}\text{safe}_\Gamma(P) = \forall P'.P \equiv P' \\ \wedge P' = x_1(y).Q_1 \mid x_2\langle\!|Q_2|\!\rangle \mid Q_3 \\ \wedge x_1 \equiv_{\mathcal{N}} x_2 \\ \implies \left(\begin{array}{l} \Gamma(x) = Ch(T) \\ \wedge\, Q_1 \equiv C[\ulcorner y \urcorner] \implies T = d \\ \wedge\, \ulcorner Q_2 \urcorner \in \text{dom}(\Gamma) \implies \Gamma(\ulcorner Q_2 \urcorner) = T) \end{array}\right)\end{aligned}$$

Here, safety actually depends on the type of names. We consider processes safe if it is the case that *when* they are on a form that can carry out a reduction, *then* the subject has the correct type, the object has the drop type if it appears in a drop in the continuation, and if the newly created name has the correct type. Note that we must quantify over all processes P' that are structurally congruent to P to ensure that this property will hold for any permutation of the parallel compositions within P: If, for example, two different processes $x_2\langle\!|Q_2|\!\rangle$ and $x_3\langle\!|Q_3|\!\rangle$ with $x_1 \equiv_{\mathcal{N}} x_2 \equiv_{\mathcal{N}} x_3$ existed within P, then the property would have to hold for both. A new name created through the lift process could already exist in the typing environment, if it has been lifted before, so this requires that the type of the name does not change from a reduction. Note that processes like the nil or drop processes are considered *safe*, since they are not on this form, so the property holds vacuously in their case.

Definition 7.9 (Logical Typing Predicate)**.** *The definition of the typing environment indexed logical relation for the ρ-calculus is as follows:*

$$R_\Gamma = \left\{\, P \;\mid\; P \to P' \implies \left(\text{safe}_\Gamma(P) \wedge P' \in R_{\Gamma, \ulcorner Q_2 \urcorner : T}\right) \right\}$$

By using this typing environment indexed logical relation, we can require that a process after a reduction should be considered in a typing environment containing the possibly newly created name. The definition is very similar to Caires' typing interpretation; the difference being that we here have a reduction semantics, so we do not include the parts of the definition that rely on the transition labels. Similarly to Caires' approach, we can define the semantic typing relation to require types for the free names in P:

Definition 7.10 (Semantically well-typed)**.** *A process P is semantically well-typed in a typing environment* Γ *in the* ρ*-calculus when the following holds:*

$$\Gamma \vDash P \triangleq P \in R_\Gamma \wedge \mathrm{fn}\,(P) \in \mathrm{dom}(\Gamma)$$

At this point, we lack the syntactic type system for comparison, and without the syntactic type judgements we cannot prove the compatibility theorem. However, we do still have the theorem of adequacy:

Theorem 8 (Adequacy)**.** *The semantic typing relation exhibits adequacy when the following holds:*

$$\Gamma \vDash P \implies \mathrm{safe}_\Gamma(P)$$

This holds fairly trivially, as safety is part of the definition of semantic typing, which is often the case [10]. So we cannot syntactically type processes. The argument for semantic typing in Dreyer [10] talks about being able to do automatic verification through the syntactic typing system, but then manually proving semantic typing lemmas for programs that cannot be automatically verified. Both Dreyer [10] and Hinrichsen et al. [16] show examples of manually proving such a typing lemma.

This approach to semantic typing means that every case will have to be proven on an ad-hoc basis, which may not seem very different from the syntactic approach, but it does seem to provide us with a viable alternative to the problem of creating a non-trivial type system for the ρ-calculus. The argument for modularity advanced by Caires [6] is also applicable here, since with our current definition, the last part of the definitions requires us to find suitable typing rules that imply semantic typing. It remains to be seen what these typing rules would look like.

7.3 The blue calculus

The π-calculus has shown itself to be a successful calculus in modelling first-order concurrent communication, and by basing the Ψ-calculus on the π-calculus, we inherit these properties. However, the π-calculus may not be particularly well-suited for modelling higher-order behaviour, regardless of the encodability of the higher-order paradigm within the first-order paradigm, as demonstrated by Sangiorgi [31]. It was precisely in an attempt to solve this for the Ψ-calculus, that Parrow et al. [28] defined the HOΨ-calculus: However, the higher-order behaviour in in HOΨ is not

based on any *inherently* higher-order calculus, but is just a simple extension of the Ψ-calculus.

A better alternative might therefore be to find a calculus that is inherently higher-order, and then generalising it in the same manner as the π-calculus and its many extensions were generalised into the Ψ-calculus. By doing this, we might have an easier way of representing the ρ-calculus in a generic framework. One promising candidate is the so-called *blue calculus* created by Boudol [5], since it is able to both express the asynchronous π-calculus and the λ-calculus well.

We will here sketch a proposal for such a generalisation of the blue calculus, which we will call the *purple calculus*. As with the Ψ-calculus, we have terms $M \in \mathbb{T}$, conditions $\varphi \in \mathbb{C}$, and assertions $\Psi \in \mathbb{A}$. The syntax for the purple calculus is then

$$P ::= M \mid (\lambda\widetilde{x}.M)P \mid P\ M \mid \langle M \Leftarrow P\rangle \mid \langle M = P\rangle \mid P \mid Q \mid (\boldsymbol{\nu} x)\, P \mid (\!|\Psi|\!)$$

The blue calculus defines two reduction semantics, namely the (β)-reduction and the resource fetching reduction. The (β)-reduction substitutes a variable, given by the λ-expression, into the continuation of the λ-expression. The resource fetching reduction handles the higher-order behaviour, i.e.

$$x \mid \langle x \Leftarrow P\rangle \to P$$

We define structural congruence in the same manner as the blue calculus, where application on assertions is defined similar to application on declarations. Instead of having two different reduction relations, we define one single reduction relation containing both the resource fetching relation and the transitive closure of the (β)-reduction relation. The reason we include the *transitive closure* of the (β)-reduction in the resource fetching relation, and not the other way around, is that the (β)-reduction relation is strongly normalizing. We begin by defining the (β)-reduction for the purple calculus:

$$\frac{}{((\lambda\widetilde{x}.M)P)(M[\widetilde{x} := \widetilde{L}]) \to_\beta P[\widetilde{x} := \widetilde{L}]}$$

$$\frac{P \to_\beta P'}{P\ M \to_\beta P'\ M} \qquad \frac{P \to_\beta P'}{(\boldsymbol{\nu} x)\, P \to_\beta (\boldsymbol{\nu} x)\, P'} \qquad \frac{P \to_\beta P'}{P \mid Q \to_\beta P' \mid Q}$$

We can hereafter define the main reduction relation:

$$\frac{\Psi \Vdash M \dot{\leftrightarrow} N}{\Psi \rhd (M \mid \langle N \Leftarrow P\rangle) \to P}$$

$$\frac{P \equiv Q \quad \Psi \rhd Q \to P'}{\Psi \rhd P \to P'} \qquad \frac{P \to_\beta Q \quad \Psi \rhd Q \to P'}{\Psi \rhd P \to P'}$$

$$\frac{\Psi \rhd P \to P'}{\Psi \rhd P\ x \to P'\ x} \qquad \frac{\Psi \rhd P \to P'}{\Psi \rhd (\boldsymbol{\nu} x)\, P \to (\boldsymbol{\nu} x)\, P'} \quad (x \# \Psi)$$

$$\frac{\Psi \otimes \mathcal{F}_\Psi(Q) \rhd P \to P'}{\Psi \rhd P \mid Q \to P' \mid Q} \quad (\mathcal{F}_\nu(Q) \,\#\Psi, \mathcal{F}_\nu(P)\,, P)$$

The definition of $\mathcal{F}(\cdot)$ is the composition of all assertions occurring outside of a declaration.

Since there exists an encoding from the π-calculus to the blue calculus, we can construct a similar encoding from the Ψ-calculus to the purple calculus. For now, we exclude the case operator, and we shall restrict ourselves to an asynchronous version of Ψ-calculus. Furthermore, we require that a replicated process must be input-guarded. The encoding is then as follows:

$$\begin{aligned}
[\![\overline{M}N]\!] &\triangleq M\ N \\
[\![\underline{M}(\lambda x)N.P]\!] &\triangleq \langle M \Leftarrow (\lambda x.N)[\![P]\!]\rangle \\
[\![\,!\,\underline{M}(\lambda x)N.P]\!] &\triangleq \langle M = (\lambda x.N)[\![P]\!]\rangle \\
[\![P \mid Q]\!] &\triangleq [\![P]\!] \mid [\![Q]\!] \\
[\![(\boldsymbol{\nu} x)\, P]\!] &\triangleq (\boldsymbol{\nu} x)\, [\![P]\!] \\
[\![(\!|\Psi|\!)]\!] &\triangleq (\!|\Psi|\!)
\end{aligned}$$

Boudol [5] also defines a type system for the blue calculus, and since this calculus is mainly based upon the λ-calculus, it can also inherit the simple type system from the λ-calculus. This means that type judgements for processes in the blue calculus have the form:

$$\Gamma \vdash P : \tau$$

We believe it is possible to generalise this type system in the same manner as Hüttel [17] generalised the simple type system by Milner [26] for the π-calculus.

* * *

We have reviewed our work, reflecting on our aim to type reflection, and although we have seen that the ρ-calculus is indeed representable within the higher-order Ψ-calculus framework, we have also glimpsed the limitations of a syntactic typing approach, and the kind of restrictions we must seemingly impose on reflection to make it typable. We have also proposed two alternative approaches, using either the *blue calculus* or a *semantic typing* approach, and in closing, we leave them here as candidates for future work.

Bibliography

[1] Appel, A. W. and McAllester, D. (2001). An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683.

[2] Bendixen, A. R., Bojesen, B. B., and Lybech, S. L. (2020). Encodability and typability of reflective higher-order languages. 9th semester computer science project report, Aalborg University.

[3] Bengtson, J., Johansson, M., Parrow, J., and Victor, B. (2009). Psi-calculi: Mobile processes, nominal data, and logic. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48. IEEE.

[4] Bengtson, J., Johansson, M., Parrow, J., and Victor, B. (2011). Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, Volume 7, Issue 1.

[5] Boudol, G. (1997). The pi-calculus in direct style. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, page 228–242, New York, NY, USA. Association for Computing Machinery.

[6] Caires, L. (2007). Logical semantics of types for concurrency. In *International Conference on Algebra and Coalgebra in Computer Science*, pages 16–35. Springer.

[7] Carbone, M. and Maffeis, S. (2003). On the expressive power of polyadic synchronisation in pi-calculus. *Nordic Journal of Computing*, 10(2):70–98.

[8] Cristescu, I. and Hirschkoff, D. (2011). Termination in a pi-calculus with subtyping. *arXiv preprint arXiv:1107.5722*.

[9] Deng, Y. and Sangiorgi, D. (2006). Ensuring termination by typability. *Information and Computation*, 204(7):1045–1082.

[10] Dreyer, D. (2018). Milner award lecture: The type soundness theorem that you really want to prove (and now you can).

[11] Dreyer, D., Timany, A., Krebbers, R., Birkedal, L., and Jung, R. (2019). What type soundness theorem do you really want to prove?

[12] Gabbay, M. and Pitts, A. (2002). A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13:341–363.

[13] Gardner, P. and Wischik, L. (2000). Explicit fusions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 373–382. Springer.

[14] Gorla, D. (2010). Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053.

[15] Hennessy, M. and Riely, J. (2002). Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120.

[16] Hinrichsen, J. K., Louwrink, D., Krebbers, R., and Bengtson, J. (2021). Machine-checked semantic session typing. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 178–198.

[17] Hüttel, H. (2011). Typed ψ-calculi. In *International Conference on Concurrency Theory*, pages 265–279. Springer.

[18] Hüttel, H. (2014). Types for resources in ψ-calculi. In Abadi, M. and Lluch Lafuente, A., editors, *Trustworthy Global Computing*, pages 83–102, Cham. Springer International Publishing.

[19] Hüttel, H. (2016). Binary session types for psi-calculi. In Igarashi, A., editor, *Programming Languages and Systems*, pages 96–115, Cham. Springer International Publishing.

[20] Meredith, L. and Radestock, M. (2005a). A reflective higher-order calculus. *Electronic Notes in Theoretical Computer Science*, 141(5):49 – 67. Proceedings of the Workshop on the Foundations of Interactive Computation (FInCo 2005).

[21] Meredith, L. G. and Radestock, M. (2005b). Namespace logic: A logic for a reflective higher-order calculus. In *International Symposium on Trustworthy Global Computing*, pages 353–369. Springer.

[22] Milner, R. (1978). A theory of type polymorphism in programming. *Journal of computer and system sciences*, 17(3):348–375.

[23] Milner, R. (1980). *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer.

[24] Milner, R. (1992a). Functions as processes. *Mathematical structures in computer science*, 2(2):119–141.

[25] Milner, R. (1992b). Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141.

[26] Milner, R. (1993). The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer Berlin Heidelberg.

[27] Parrow, J. (2001). An introduction to the π-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier.

[28] Parrow, J., Borgström, J., Raabjerg, P., and Åman Pohjola, J. (2014). Higher-order psi-calculi. *Mathematical Structures in Computer Science*, 24(2).

[29] Parrow, J. and Victor, B. (1998). The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 98CB36226)*, pages 176–185. IEEE.

[30] Pierce, B. and Sangiorgi, D. (1993). Typing and subtyping for mobile processes. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 376–385. IEEE.

[31] Sangiorgi, D. (1993a). *Expressing mobility in process algebras: first-order and higher-order paradigms*. PhD thesis, University of Edinburgh.

[32] Sangiorgi, D. (1993b). From π-calculus to higher-order π-calculus — and back. In Gaudel, M. C. and Jouannaud, J. P., editors, *TAPSOFT'93: Theory and Practice of Software Development*, pages 151–166, Berlin, Heidelberg. Springer Berlin Heidelberg.

[33] Sangiorgi, D. and Walker, D. (2003). *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press.

[34] Skorstengaard, L. (2019). An introduction to logical relations. *CoRR*, abs/1907.11133.

[35] Smith, B. C. (1982). *Procedural Reflection in Programming Languages*. PhD thesis, Massachusetts Institute of Technology.

[36] Thomsen, B. (1989). A calculus of higher order communicating systems. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL' 89*, POPL'89, pages 143–154, New York, NY, USA. ACM Press.

[37] Thomsen, B. (1993). Plain CHOCS a second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59.

[38] Wright, A. K. and Felleisen, M. (1994). A syntactic approach to type soundness. *Information and computation*, 115(1):38–94.

[39] Åman Pohjola, J. (2019). Psi-calculi revisited: Connectivity and compositionality. In Pérez, J. A. and Yoshida, N., editors, *Formal Techniques for Distributed Objects, Components, and Systems*, pages 3–20, Cham. Springer International Publishing.

A The ρ-calculus

The Reflective Higher-Order calculus, abbreviated RHO or just ρ, by Meredith and Radestock [20], is a succinct model of computation, based on the notion of *quoting*. As the name implies, it combines *reflection*, i.e. the ability of a program to generate and inspect its own code, and *higher order* process mobility. Furthermore, both its reflective and higher-order characteristisc are *inherent*, rather than merely extensions added onto an initially non-reflective, first-order language.

A.1 Syntax and reduction semantics

We shall here briefly review the syntax and reduction semantics for the ρ-calculus, originally presented by Meredith and Radestock [20]. However, much of the text is based on the presentation in [2, chapter 2].

Definition A.1 (ρ-calculus syntax)**.** *We define the set ρ of ρ-calculus processes, ranged over by P, Q, R etc., by the formation rules:*

$$P, Q, R \in \rho ::= \mathbf{0} \;\mid\; P \mid Q \;\mid\; \ulcorner R \urcorner \langle\!| P |\!\rangle \;\mid\; \ulcorner R \urcorner (\ulcorner Q \urcorner).P \;\mid\; \urcorner\ulcorner R \urcorner\ulcorner$$

where the names $\ulcorner Q \urcorner, \ulcorner R \urcorner$ are quoted *processes. Thus, the set $\ulcorner \rho \urcorner$ of ρ-calculus* names*, ranged over by x, y, z, is defined as*

$$x, y, z \in \ulcorner \rho \urcorner \triangleq \{ \ulcorner P \urcorner \mid P \in \rho \}$$

The nil, parallel, and input constructs are similar to e.g. the π-calculus [26], with input $x(y).P$ as a name binding construct. $x \langle\!| P |\!\rangle$, pronounced 'lift' quotes P, thus dynamically creating the name $\ulcorner P \urcorner$, and sends it along x. The last construct, $\urcorner x \ulcorner$, pronounced 'drop x', removes the quotes from x which thereby executes the process within the name.

The semantics in [20] is given as a reduction system, where terms can be rewritten by structural congruence, defined as follows:

Definition A.2 (Structural congruence)**.** Structural congruence*, written $\equiv \subseteq \rho \times \rho$, is the least congruence on processes, containing α-equivalence $\equiv_\alpha$ and satisfying that $\left(\rho_{/\equiv}, \mid, \mathbf{0}\right)$ is an abelian monoid.*

Likewise, since names are quoted processes, then names too are identified by an equivalence relation on the set of names, if the processes within are structurally congruent:

Definition A.3 (Name equivalence)**.** Name equivalence*, written* $\equiv_{\mathcal{N}} \subseteq \ulcorner\rho\urcorner \times \ulcorner\rho\urcorner$*, is the smallest equivalence relation on quoted processes, closed forward under the rules:*

$$[\textsc{Nameeq}_1]\frac{P \equiv Q}{\ulcorner P\urcorner \equiv_{\mathcal{N}} \ulcorner Q\urcorner} \qquad [\textsc{Nameeq}_2]\frac{}{\ulcorner\urcorner x \ulcorner\urcorner \equiv_{\mathcal{N}} x}$$

The sets $\mathrm{fn}\,(P)$ and $\mathrm{bn}\,(P)$ of free and bound names of P are defined in the standard way, with $x\,(y)\,.P$ as the only name binding construct in the syntax. Thus y is bound, and all other names are free. This leads to a (mostly) standard definition of capture-avoiding syntactic substitution:

Definition A.4 (Syntactic substitution)**.** Syntactic substitution *is a function*

$$(\cdot)\,\{\cdot/\cdot\} : \rho \times \ulcorner\rho\urcorner \times \ulcorner\rho\urcorner \to \rho$$

from processes to processes, parametrised with two names, and defined recursively by the following equations:

$$\begin{aligned}
(\mathbf{0})\,\{u/v\} &= \mathbf{0} \\
(P \mid Q)\,\{u/v\} &= (P)\,\{u/v\} \mid (Q)\,\{u/v\} \\
(x\,(y)\,.P)\,\{u/v\} &= \begin{cases} u\,(z)\,.\,((P)\,\{z/y\})\,\{u/v\} & \text{if } x \equiv_{\mathcal{N}} v \\ x\,(z)\,.\,((P)\,\{z/y\})\,\{u/v\} & \text{if } x \not\equiv_{\mathcal{N}} v \end{cases} \\
(x\,\langle\!|P|\!\rangle)\,\{u/v\} &= \begin{cases} u\,\langle\!|(P)\,\{u/v\}|\!\rangle & \text{if } x \equiv_{\mathcal{N}} v \\ x\,\langle\!|(P)\,\{u/v\}|\!\rangle & \text{if } x \not\equiv_{\mathcal{N}} v \end{cases} \\
(\ulcorner x\urcorner)\,\{u/v\} &= \begin{cases} \ulcorner u\urcorner & \text{if } x \equiv_{\mathcal{N}} v \\ \ulcorner x\urcorner & \text{if } x \not\equiv_{\mathcal{N}} v \end{cases}
\end{aligned}$$

where z *is chosen such that if* $u = \ulcorner U\urcorner$ *then*

$$z \notin \{\, u, v \,\} \cup \mathrm{fn}\,(P) \cup \mathrm{fn}\,(U) \cup \mathrm{bn}\,(U)$$

The requirement on z ensures that z is locally fresh in P, such that it cannot inadvertently clash with another free or bound name.

Syntactic substitution is used in deciding α-equivalence, which again is contained in structural congruence, which again is used in deciding name equivalence. The definition is thus mutually recursive, but always ultimately terminating because of the $[\textsc{Nameeq}_2]$ rule, as proved in [20]. However, in the semantics a slightly different form of substitution is used:

Definition A.5 (Semantic substitution)**.** Semantic substitution*, is a function*

$$(\cdot)\{\cdot/\cdot\} : \rho \times \ulcorner\rho\urcorner \times \ulcorner\rho\urcorner \to \rho$$

from processes to processes, parametrised with two names, and defined by the same equations as the syntactic subsection, except for the last clause, which instead is defined thus:

$$(\urcorner x\ulcorner)\{\ulcorner R\urcorner/v\} = \begin{cases} R & \text{if } x \equiv_{\mathcal{N}} v \\ \urcorner x\ulcorner & \text{if } x \not\equiv_{\mathcal{N}} v \end{cases}$$

We shall not distinguish explicitly between syntactic or semantic substitution when there is no risk of confusion between the two. Semantic substitution thus performs an 'eager' drop, as part of the substitution, and it is hence *this* form of substitution that is used in the semantics:

Definition A.6 (Reduction relation)**.** *The* reduction relation $\to \subseteq \rho \times \rho$ *is given by the following rules:*

$$[\text{R-CON}]\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q} \qquad [\text{R-PAR}]\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

$$[\text{R-COM}]\frac{x_1 \equiv_{\mathcal{N}} x_2}{x_1(y).P \mid x_2\langle\!|Q|\!\rangle \to P\{\ulcorner Q\urcorner/y\}}$$

A.2 Labelled semantics

The semantics of the Ψ-calculus framework is originally defined in terms of a labelled transition system. Thus, to show that we can instantiate the ρ-calculus, we shall also define a labelled semantics for it, as follows:

Definition A.7 (Action labels)**.** *We define the set of action labels $\mathcal{A}$, ranged over by α, by the following syntax:* $\alpha \in \mathcal{A} ::= \tau \mid x!\ulcorner P\urcorner \mid x?\ulcorner P\urcorner$ *where $x!\ulcorner P\urcorner$ denotes sending $\ulcorner P\urcorner$ on x, $x?\ulcorner P\urcorner$ is the reception of $\ulcorner P\urcorner$ on x, and τ is an internal communication.*

Definition A.8 (Labelled transition system)**.** *We define the transition system as the triple*

$$\left(\rho, \mathcal{A}, \xrightarrow{\alpha} \subseteq \rho \times \mathcal{A} \times \rho\right)$$

where the transition relation $\xrightarrow{\alpha}$ is given by the following rules:

$$[\text{SOS-PAR}_1]\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad [\text{SOS-PAR}_2]\frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'}$$

$$[\text{SOS-INPUT}]\frac{x_1 \equiv_{\mathcal{N}} x_2}{x_1(y).P \xrightarrow{x_2?\ulcorner Q\urcorner} P\{\ulcorner Q\urcorner/y\}} \qquad [\text{SOS-LIFT}]\frac{x_1 \equiv_{\mathcal{N}} x_2}{x_1\langle\!|Q|\!\rangle \xrightarrow{x_2!\ulcorner Q\urcorner} \mathbf{0}}$$

$$[\text{SOS-COM}_1]\frac{P \xrightarrow{x_1!\ulcorner R\urcorner} P' \quad x_1 \equiv_{\mathcal{N}} x_2 \quad Q \xrightarrow{x_2?\ulcorner R\urcorner} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$[\text{SOS-COM}_2]\frac{P \xrightarrow{x_1?\ulcorner R\urcorner} P' \quad x_1 \equiv_{\mathcal{N}} x_2 \quad Q \xrightarrow{x_2!\ulcorner R\urcorner} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

This semantics is of the *early* kind, where the label $x?\ulcorner P\urcorner$ contains the actual value received, and substitution happens in the input rule [SOS-INPUT], similar to the Ψ-calculus semantics.

Note also that name equivalence $x_1 \equiv_{\mathcal{N}} x_2$ is used in both communication rules (as expected), *and* in both [SOS-INPUT] and [SOS-LIFT]. This is is not strictly necessary in the ρ-calculus, but is done to match the usage of channel equivalence in the Ψ-calculus (see Bengtson et al. [4, p. 13]).

A.3 Correspondence between the two semantics

We now need to ascertain that the τ-labelled transitions correspond exactly to the reductions and vice versa:

Proposition 1. $\forall P, P' : P \xrightarrow{\tau} P' \implies P \to P'$

Proof. To prove this, we shall consider each SOS-rule with a conclusion of the form $P \xrightarrow{\tau} P'$ and show how a similar conclusion can be reached using the reduction rules. There are two cases to consider:

1. If either [SOS-COM$_1$] or [SOS-COM$_2$] was the last rule used to conclude $P \xrightarrow{\tau} P'$, then the transition is of the form

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

   where both processes take a transition. The two cases are mirror images of each other, so we shall consider only [SOS-COM$_1$]. From the premise we have that

   $$P \xrightarrow{x_1!\ulcorner R\urcorner} P'$$

   which again must have been concluded by the [SOS-LIFT] rule (and possibly one of the PAR-rules). Similarly, we have that

   $$Q \xrightarrow{x_2?\ulcorner R\urcorner} Q'$$

   which must have been concluded by the [SOS-INPUT] rule. Hence, the entire expression must be of the form

   $$x_1\langle\!|R|\!\rangle \mid P'' \mid x_2(y).Q_1 \mid Q_2 \xrightarrow{\tau} P'' \mid Q_1\{\ulcorner R\urcorner/y\} \mid Q_2$$

with $x_1 \equiv_{\mathcal{N}} x_2$. This can be concluded in the reduction semantics by firstly using the [R-CON] rule to reorder the terms to build a redex, e.g. leftmost:

$$x_1 \langle\!| R |\!\rangle \mid x_2(y).Q_1 \mid P'' \mid Q_2$$

and we then use the [R-PAR] rule to select only the communicating processes, and then lastly, as the axiom we use the [R-COM] rule.

2. If either [SOS-PAR$_1$] or [SOS-PAR$_2$] was the last rule used to conclude $P \xrightarrow{\tau} P'$ then we know P is a parallel composition of the form $P = P_1 \mid P_2$. Again, the two cases are symmetrically similar, so we shall only consider the first. The transition is thus of the form

$$P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2$$

with $P_1 \xrightarrow{\tau} P_1'$ in the premise. Assume therefore that $P_1 \to P_1'$ (the induction hypothesis). Then

$$P_1 \mid P_2 \to P_1' \mid P_2$$

can be concluded immediately by the [R-PAR] rule.

□

This proves that any τ-labelled transitions, that can be proved by the rules of the labelled semantics, can also be proved by the rules of the reduction semantics. Unfortunately, we cannot prove *exactly* the same relationship for the other direction, because $\mathbf{0}$-processes may be removed in the reduction semantics, through the use of the [R-CON] rule, whilst this is not possible in the SOS. Instead, we shall show correspondence up to $\simeq$, which represents any sensible notion of behavioural equivalence:

Proposition 2. $\forall P, P' : P \to P' \implies P \xrightarrow{\tau} \simeq P'$

Proof. For this proof, we shall consider each reduction rule and show that its conclusion can also be reached by using the τ-labelled SOS-rules, whilst allowing $\mathbf{0}$ processes to be removed by rewriting under $\simeq$. This time, we thus have three cases to consider:

1. If [R-COM] was the last rule used to conclude $P \to P'$ then the transition is of the form

$$x_1(y).P \mid x_2 \langle\!| Q |\!\rangle \to P\{\ulcorner Q \urcorner / y\}$$

with $x_1 \equiv_{\mathcal{N}} x_2$ as premise. This can then be concluded by the [SOS-COM$_2$] rule, where the premises are

$$x_1(y).P \xrightarrow{x_1 ? \ulcorner Q \urcorner} P\{\ulcorner Q \urcorner / y\} \qquad \text{and} \qquad x_2 \langle\!| Q |\!\rangle \xrightarrow{x_2 ! \ulcorner Q \urcorner} \mathbf{0}$$

which can be concluded by the [SOS-INPUT] and [SOS-LIFT] rules respectively. Notice that this actually yields $P\{\ulcorner Q \urcorner/y\} \mid \mathbf{0}$, and we must therefore use the behavioural equivalence relation to conclude that

$$P\{\ulcorner Q \urcorner/y\} \mid \mathbf{0} \simeq P\{\ulcorner Q \urcorner/y\}$$

2. If [R-PAR] was the last rule used to conclude $P \to P'$ then we know the form of the transition is

$$P_1 \mid P_2 \to P_1' \mid P_2$$

with $P_1 \to P_1'$ as premise. Assume therefore that $P_1 \xrightarrow{\tau} P_1'$ (induction hypothesis). Then

$$P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2$$

can be concluded immediately by the [SOS-PAR$_1$] rule.

3. Lastly, if $P \to P'$ was concluded by the [R-CON] rule, then we know only from the premises that $P \equiv Q$ and $Q \to Q'$ and $Q' \equiv P'$. By the induction hypothesis we can assume that $Q \xrightarrow{\tau} Q'$ can be concluded by one of the other cases, and we therefore just need to show that the labelled semantics can simulate the two rewrites by structural congruence.

   Of these, the latter rewrite is used to eliminate solitary $\mathbf{0}$ processes, which as already mentioned cannot be done in the labelled semantics. However, the resulting process will be behaviourally equivalent to one where the $\mathbf{0}$-processes have not been removed, so we shall ignore here.

   The first rewrite allows redexes to be built by reordering terms to match the conclusion in the [R-COM] rule. This can be achieved by applying either [SOS-COM$_1$] or [SOS-COM$_2$], and then a number of applications of the [SOS-PAR] rules to isolate the input and lift processes, respectively.

□

B Proofs for the reduction semantics

B.1 Proof of theorem 1

This theorem states that

$$P \xrightarrow{\tau} P \iff P \to P'$$

where P is a HOΨ-calculus process, and $\to$ is defined as in definition 3.4 with $\equiv$ extended with the axiom for **run** M.

Proof. The theorem was proved by Åman Pohjola [39] for the relationship between the context reduction semantics and the *first-order* Ψ-calculus of Bengtson et al. [3]. Thus we only need to prove this theorem for the **run** M extension, i.e. where the redex P is of the form $P = \mathbf{run}\ M \mid R$ with $M \Leftarrow Q$. For both directions, we then have two cases to consider:

- The reduct is $P' \mid R$ where Q becomes P' after a one-step internal communication, whilst the context process R is unaltered.
- The reduct is $P' \mid R'$ where Q becomes P' after a one-step interaction with the context process R.

For the forward direction, the two cases are as follows:

1. If $\mathbf{run}\ M \mid R \xrightarrow{\tau} P' \mid R$ then the τ-label is the result of an entirely internal communication within the process for which M is a handle. Thus we have that

$$[\textsc{Run}]\frac{\Psi \Vdash M \Leftarrow Q \quad \Psi \rhd Q \xrightarrow{\tau} P'}{\Psi \rhd \mathbf{run}\ M \xrightarrow{\tau} P'}$$

   where we know from the definition of well-formed processes (definition 2.15) that Q is an assertion guarded process. We then derive the corresponding reduction as follows: Firstly, by application of the [R-STRUCT] rule we can bring the entire process on the form

$$(\boldsymbol{\nu}\widetilde{x})\left(\langle\!|\Psi|\!\rangle \mid \mathbf{run}\ M \mid R\right)$$

where R represents the rest of the entire process which in the labelled semantics would have been disregarded in the derivation tree by application of the [Par] and [Com] rules. The restrictions are disregarded by the [r-res] rule, and with one further rewrite by structural congruence, we obtain the process

$$(\!|\Psi|\!) \mid (\textbf{case}\ M \Leftarrow Q : Q) \mid R$$

for which we can build a matching reduction context C_τ as follows:

$$C_\tau \triangleq (\textbf{case}\ M \Leftarrow Q : C) \mid G$$

Here, both holes must occur in the C part, matching the process Q, and G matches the remainder R where no holes occur. Now $M \Leftarrow Q \in \textsc{conds}\,(C_\tau)$, but we know from the premise of [Run] that $\Psi \Vdash M \Leftarrow Q$. Thus we can satisfy the premise of the [r-ctx] rule, and by application of this rule we obtain the reduction

$$(\!|\Psi|\!) \mid (\textbf{case}\ M \Leftarrow Q : Q) \mid R \to (\!|\Psi|\!) \mid P' \mid R$$

where $\textsc{ppr}\,(C_\tau) = R$ and $Q \xrightarrow{\tau} P'$ by an internal communication.

2. If $\textbf{run}\ M \mid R \xrightarrow{\tau} P' \mid R'$ then we have that $\textbf{run}\ M \xrightarrow{\alpha} P'$ by doing an α-action, and $R \xrightarrow{\overline{\alpha}} R'$ by performing the corresponding co-action. Again we assume that $M \Leftarrow Q$ and thus that $Q \xrightarrow{\alpha} P'$. Unlike in the previous case, we here do have an interaction with the context R, and the τ-transition will therefore have been derived by the [Com] rule with

$$[\textsc{Com}]\frac{\mathcal{F}_\Psi(R) \otimes \Psi \Vdash \alpha \dot{\leftrightarrow} \overline{\alpha} \quad \mathcal{F}_\Psi(R) \otimes \Psi \rhd \textbf{run}\ M \xrightarrow{\alpha} P' \quad \Psi \rhd R \xrightarrow{\overline{\alpha}} R'}{\Psi \rhd \textbf{run}\ M \mid R \xrightarrow{\tau} P' \mid R'}$$

where $\alpha \dot{\leftrightarrow} \overline{\alpha}$ represents the channel equivalence check for the corresponding channel names used within the $\alpha$and $\overline{\alpha}$ labels. To derive the corresponding reduction we proceed almost exactly as before. The only difference is in the construction of the reduction context C_α which this time instead is built in the following way:

$$C_\alpha \triangleq (\textbf{case}\ M \Leftarrow Q : C_1) \mid C_2$$

with one hole occurring within C_1, matching Q, and another occurring within C_2, matching R.

Thus, from the assumption that $P \xrightarrow{\tau} P'$ we can derive the reduction $P \to P$. The proof for the other direction proceeds in a similar fashion, where again we assume that $M \Leftarrow Q$, and here the two cases are as follows:

1. If **run** $M \mid R \to P' \mid R$, then this will have been concluded by first an application of the [R-STRUCT] rule to rewrite the **run** M term to

$$(\!|\Psi|\!) \mid (\textbf{case}\ M \Leftarrow Q : Q) \mid R$$

and possibly some further rewrites within Q to bring it on the necessary form (e.g. if Q itself contained further **run** M terms). The context is then like C_τ described above. The corresponding $\xrightarrow{\tau}$ transition can then be concluded by application of the [PAR] rule to isolate **run** M and then the [RUN] rule to conclude $Q \xrightarrow{\tau} P'$. The assertion $(\!|\Psi|\!)$ may have been formed from several assertions $(\!|\Psi_1|\!) \mid \ldots \mid (\!|\Psi_n|\!)$ found within the R process, but these will have been collected in the [PAR] rule and added to the environment.

2. If **run** $M \mid R \to P' \mid R'$ then the case is again similar to the above, except that the context here will have been like C_α instead. This time there is a communication with the R component, so we conclude by the [COM] rule instead, and then [RUN] to conclude **run** $M \xrightarrow{\alpha} P'$ in the premise.

□

C Proofs for the type system

C.1 Proof of Lemma 1

The lemma states:

If $\Gamma, \Psi \vdash P$ *then* $\Gamma, x : T, \Psi \vdash P$

Proof. We use proof by induction on the rules of $\Gamma, \Psi \vdash P$.

[T-IN] Assume

$$\Gamma, \Psi \vdash \underline{M}(\lambda \widetilde{y}) N.P$$

which we can conclude with following assumptions

$$T \leftarrow\!\!\!\rho\ T' \tag{C.1}$$
$$\Gamma, \Psi \vdash M : T \tag{C.2}$$
$$\Gamma, \widetilde{y} : \widetilde{T}, \Psi \vdash N : T' \tag{C.3}$$
$$\Gamma, \widetilde{y} : \widetilde{T}, \Psi \vdash P \tag{C.4}$$

we want to show

$$\Gamma, x : T, \Psi \vdash \underline{M}(\lambda \widetilde{x}) N.P \tag{C.5}$$

Using [T-ENV-WEAK] for (C.2) and (C.3), and using the induction hypothesis for (C.4) we can conclude

$$\Gamma, x : T, \Psi \vdash M : T \tag{C.6}$$
$$\Gamma, x : T, \widetilde{y} : \widetilde{T}, \Psi \vdash N : T' \tag{C.7}$$
$$\Gamma, x : T, \widetilde{y} : \widetilde{T}, \Psi \vdash P \tag{C.8}$$

Using [T-IN] with (C.1), (C.6), (C.7) and (C.8) we can conclude (C.5).

[T-RUN] Assume

$$\Gamma, \Psi \vdash \mathbf{run}\ M$$

which we can conclude with following assumptions

$$T \curvearrowleft \Gamma' \tag{C.9}$$
$$\Gamma, \Psi \vdash M : T \tag{C.10}$$
$$\Psi \Vdash M \Leftarrow P \tag{C.11}$$
$$\Gamma', \Psi \vdash P \tag{C.12}$$

we want to show

$$\Gamma, x : T, \Psi \vdash \mathbf{run}\ M \tag{C.13}$$

Using [T-ENV-WEAK] for (C.10) we can conclude

$$\Gamma, x : T, \Psi \vdash M : T \tag{C.14}$$

Using [T-RUN] with (C.9), (C.13), (C.11) and (C.12) we can conclude (C.13).

[T-OUT] Assume

$$\Gamma, \Psi \vdash \overline{M}N.P$$

which we can conclude with following assumptions

$$T \leftharpoonup T' \tag{C.15}$$
$$\Gamma, \Psi \vdash M : T \tag{C.16}$$
$$\Gamma, \Psi \vdash N : T' \tag{C.17}$$
$$\Gamma, \Psi \vdash P \tag{C.18}$$

we want to show

$$\Gamma, x : T, \Psi \vdash \overline{M}N.P \tag{C.19}$$

Using [T-ENV-WEAK] for (C.16) and (C.17), and using the induction hypothesis with (C.18) we can conclude

$$\Gamma, x : T, \Psi \vdash M : T \tag{C.20}$$
$$\Gamma, x : T, \Psi \vdash N : T' \tag{C.21}$$
$$\Gamma, x : T, \Psi \vdash P \tag{C.22}$$

Using [T-OUT] with (C.15), (C.20), (C.21) and (C.22) we can conclude (C.19).

[T-PAR] Assume

$$\Gamma, \Psi \vdash P \mid Q$$

which we can conclude with following assumptions

$$\Gamma, \mathcal{F}_\nu(Q)\,, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.23}$$
$$\Gamma, \mathcal{F}_\nu(P)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.24}$$
$$\mathcal{F}_\nu(P) \,\#\, \Psi, \mathcal{F}_\nu(Q)\,, Q \tag{C.25}$$
$$\mathcal{F}_\nu(Q) \,\#\, \Psi, \mathcal{F}_\nu(P)\,, P \tag{C.26}$$

we want to show

$$\Gamma, x : T, \Psi \vdash P \mid Q \tag{C.27}$$

Using the induction hypothesis for (C.23) and (C.24) we can conclude

$$\Gamma, \mathcal{F}_\nu(Q)\,, x : T, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.28}$$
$$\Gamma, \mathcal{F}_\nu(P)\,, x : T, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.29}$$

Using [T-PAR] with (C.28), (C.29), (C.25) and (C.26) we can conclude (C.27).

[T-CASE] Assume

$$\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$$

which we can conclude with following assumptions

$$\Gamma, \Psi \vdash \varphi_i \tag{C.30}$$
$$\Gamma, \Psi \vdash P_i \tag{C.31}$$

we want to show

$$\Gamma, x : T, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.32}$$

Using the [T-ENV-WEAK] for (C.30) and using the induction hypothesis with (C.31) we can conclude

$$\Gamma, x : T, \Psi \vdash \varphi_i \tag{C.33}$$
$$\Gamma, x : T, \Psi \vdash P_i \tag{C.34}$$

Using [T-CASE] with (C.33) and (C.34) we can conclude (C.32).

[T-ASSERT] Assume

$$\Gamma, \Psi \vdash (\!|\Psi'|\!)$$

which we can conclude with following assumptions

$$\Gamma, \Psi \vdash \Psi' \tag{C.35}$$

we want to show

$$\Gamma, x : T, \Psi \vdash (\!|\Psi'|\!) \tag{C.36}$$

Using the [T-ENV-WEAK] for (C.35) we can conclude

$$\Gamma, x : T, \Psi \vdash \Psi' \tag{C.37}$$

Using [T-ASSERT] with (C.37) we can conclude (C.36).

[T-NIL] We want to show

$$\Gamma, \Psi \vdash \mathbf{0} \implies \Gamma, x : T, \Psi \vdash \mathbf{0}$$

which naturally holds.

[T-NEW] Assume

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} y)\, P$$

which we can conclude with following assumptions

$$\Gamma, y : T, \Psi \vdash P \tag{C.38}$$

$$x \# \Psi \tag{C.39}$$

we want to show

$$\Gamma, x : T, \Psi \vdash (\boldsymbol{\nu} y)\, P \tag{C.40}$$

Using the induction hypothesis with (C.40) we can conclude

$$\Gamma, x : T, y : T, \Psi \vdash P \tag{C.41}$$

Using [T-NEW] with (C.41) and (C.39) we can conclude (C.40).

[T-REPL] Assume

$$\Gamma, \Psi \vdash\ !\, P$$

which we can conclude with following assumptions

$$\Gamma, \Psi \vdash P \tag{C.42}$$

we want to show

$$\Gamma, x : T, \Psi \vdash\ !\, P \tag{C.43}$$

Using the induction hypothesis with (C.44) we can conclude

$$\Gamma, x : T, \Psi \vdash P \tag{C.44}$$

Using [T-REPL] with (C.44) we can conclude (C.43).

□

C.2 Proof of Lemma 2

The lemma states:

If $\Gamma, x : T, \Psi \vdash P$ *and* $x\#P, \Psi$ *then* $\Gamma, \Psi \vdash P$

Proof. We use proof by induction on the rules of $\Gamma, x : T, \Psi \vdash P$.

[T-IN] Assume

$$\Gamma, x : T, \Psi \vdash \underline{M}(\lambda\widetilde{x})N.P \tag{C.45}$$
$$x\#\Psi \tag{C.46}$$
$$x\#M \tag{C.47}$$
$$x\#P \tag{C.48}$$

which we can conclude with

$$T \hookleftarrow T' \tag{C.49}$$
$$\Gamma, x : T, \Psi \vdash M : T \tag{C.50}$$
$$\Gamma, x : T, \widetilde{x} : \widetilde{T}, \Psi \vdash N : T' \tag{C.51}$$
$$\Gamma, x : T, \widetilde{x} : \widetilde{T}, \Psi \vdash P \tag{C.52}$$

We want to show

$$\Gamma, \Psi \vdash \underline{M}(\lambda\widetilde{x})N.P \tag{C.53}$$

Using [T-ENV-STRENGTH] with (C.50) and (C.51) and using the induction hypothesis with (C.48) we can conclude

$$\Gamma, \Psi \vdash M : T \tag{C.54}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash N : T' \tag{C.55}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \tag{C.56}$$

Using [T-IN] with (C.54), (C.55), (C.56) and (C.49) we can conclude (C.53).

[T-RUN] Assume

$$\Gamma, x : T, \Psi \vdash \mathbf{run}\ M$$
$$x\#\mathbf{run}\ M$$
$$x\#\Psi$$

which we can conclude with

$$T \curvearrowleft \Gamma' \tag{C.57}$$
$$\Gamma, x : T, \Psi \vdash M : T \tag{C.58}$$
$$\Psi \Vdash M \Leftarrow P \tag{C.59}$$
$$\Gamma', \Psi \vdash P \tag{C.60}$$
$$x\#M$$

We want to show

$$\Gamma, \Psi \vdash \textbf{run}\ M \tag{C.61}$$

Using [T-ENV-STRENGTH] with (C.58) we can conclude

$$\Gamma, \Psi \vdash M : T \tag{C.62}$$

Using [T-RUN] with (C.57), (C.62), (C.59) and (C.60) we can conclude (C.61).

[T-OUT] Assume

$$\begin{aligned} \Gamma, x : T, \Psi \vdash \overline{M} N.P \\ x \# \overline{M} N.P \\ x \# \Psi \end{aligned}$$

which we can conclude with

$$T \leftharpoonup T' \tag{C.63}$$
$$\Gamma, x : T, \Psi \vdash M : T \tag{C.64}$$
$$\Gamma, x : T, \Psi \vdash N : T' \tag{C.65}$$
$$\Gamma, x : T, \Psi \vdash P \tag{C.66}$$
$$x \# M$$
$$x \# N$$
$$x \# P$$

We want to show

$$\Gamma, \Psi \vdash \overline{M} N.P \tag{C.67}$$

Using [T-ENV-STRENGTH] with (C.64), (C.65), (C.66) we can conclude

$$\Gamma, \Psi \vdash M : T \tag{C.68}$$
$$\Gamma, \Psi \vdash N : T' \tag{C.69}$$
$$\Gamma, \Psi \vdash P \tag{C.70}$$

Using [T-OUT] with (C.63), (C.68), (C.69) and (C.70) we can conclude (C.67).

[T-PAR] Assume

$$\begin{aligned} \Gamma, x : T, \Psi \vdash P \mid Q \\ x \# \Psi, P \mid Q \end{aligned}$$

which we can conclude with

$$\Gamma, x : T, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.71}$$
$$\Gamma, x : T, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.72}$$
$$\mathcal{F}_\nu(P) \,\#\, \Psi, \mathcal{F}_\nu(Q)\,, Q \tag{C.73}$$
$$\mathcal{F}_\nu(Q) \,\#\, \Psi, \mathcal{F}_\nu(P)\,, P \tag{C.74}$$
$$x \# \Psi, P, Q$$

We want to show

$$\Gamma, \Psi \vdash P \mid Q \tag{C.75}$$

Using the induction hypothesis with (C.71) and (C.72) we can conclude

$$\Gamma, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.76}$$
$$\Gamma, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.77}$$

Using [T-PAR] with (C.76), (C.77), (C.73) and (C.74) we can conclude (C.75).

[T-CASE] Assume

$$\begin{gathered}\Gamma, x : T, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \\ x \# \Psi, \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\end{gathered} \tag{C.78}$$

which we can conclude with

$$\Gamma, x : T, \Psi \vdash \varphi_i \tag{C.79}$$
$$\Gamma, x : T, \Psi \vdash P_i \tag{C.80}$$
$$x \# \widetilde{\varphi}, \widetilde{P}$$

We want to show

$$\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.81}$$

Using [T-ENV-STRENGTH] with (C.79) and the induction hypothesis with (C.80) we can conclude

$$\Gamma, \Psi \vdash \varphi_i \tag{C.82}$$
$$\Gamma, \Psi \vdash P_i \tag{C.83}$$

Using [T-CASE] with (C.82) and (C.83) we can conclude (C.83).

[T-ASSERT] Assume

$$\frac{\Gamma, x : T, \Psi \vdash (\!|\Psi'|\!) \qquad x \# \Psi, (\!|\Psi'|\!)}{}$$

which we can conclude with

$$\frac{\Gamma, x : T, \Psi \vdash \Psi' \qquad x \# \Psi'}{} \tag{C.84}$$

We want to show

$$\Gamma, \Psi \vdash (\!|\Psi'|\!) \tag{C.85}$$

Using [T-ENV-STRENGTH] with (C.84) we can conclude

$$\Gamma, \Psi \vdash \Psi' \tag{C.86}$$

Using [T-ASSERT] with (C.86) we can conclude (C.85).

[T-NIL] We want to show

$$\Gamma, x : T, \Psi \vdash \mathbf{0} \implies \Gamma, \Psi \vdash \mathbf{0}$$

which naturally holds.

[T-NEW] Assume

$$\frac{\Gamma, x : T, \Psi \vdash (\boldsymbol{\nu} y)\, P \qquad x \# \Psi, (\boldsymbol{\nu} y)\, P}{}$$

which we can conclude with

$$\frac{\Gamma, x : T, y : T, \Psi \vdash P \qquad x \# P}{} \tag{C.87}$$

We want to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} y)\, P \tag{C.88}$$

Using the induction hypothesis with (C.87) we can conclude

$$\Gamma, y : T, \Psi \vdash P \tag{C.89}$$

Using [T-NEW] with (C.89) we can conclude (C.88).

[T-REPL] Assume

$$\frac{\Gamma, x : T, \Psi \vdash\ !\, P \qquad x\#\,!\, P}{}$$

which we can conclude with

$$\frac{\Gamma, x : T, \Psi \vdash P \qquad x\#P}{} \tag{C.90}$$

We want to show

$$\Gamma, \Psi \vdash\ !\, P \tag{C.91}$$

Using the induction hypothesis with (C.90) we can conclude

$$\Gamma, \Psi \vdash P \tag{C.92}$$

Using [T-REPL] with (C.92) we can conclude (C.91).

□

C.3 Proof of Lemma 5

The lemma states:

If $\Gamma, \Psi \vdash P$, $n(\Psi') \subseteq dom(\Gamma)$ *and* $\Psi \leq \Psi'$ *then* $\Gamma, \Psi' \vdash P$.

Proof. We use proof by induction on the rules of $\Gamma, \Psi \vdash P$.

[T-NIL] Assume

$$\frac{\Gamma, \Psi \vdash \mathbf{0} \qquad n(\Psi') \subseteq dom(\Gamma) \qquad \Psi \leq \Psi'}{} \tag{C.93}$$

Our desired result is

$$\Gamma, \Psi' \vdash \mathbf{0}$$

which holds.

[T-IN] Assume

$$\Gamma, \Psi \vdash \underline{M}(\lambda \widetilde{x})N.P \tag{C.94}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.95}$$
$$\Psi \leq \Psi' \tag{C.96}$$

We can conclude (C.94) with rule [T-IN] and following axioms

$$\Gamma, \Psi \vdash M : T \tag{C.97}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash N : T' \tag{C.98}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \tag{C.99}$$
$$T \hookleftarrow T' \tag{C.100}$$

We wish to show

$$\Gamma, \Psi' \vdash \underline{M}(\lambda \widetilde{x})N.P \tag{C.101}$$

Applying [T-ASS-WEAK] to (C.97) and (C.98), and using the induction hypothesis with (C.99) we have

$$\Gamma, \Psi' \vdash M : T \tag{C.102}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi' \vdash N : T' \tag{C.103}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi' \vdash P \tag{C.104}$$

With (C.101), (C.102), (C.103) and (C.99) we can conclude (C.100).

[T-OUT] Assume

$$\Gamma, \Psi \vdash \overline{M}N.P \tag{C.105}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.106}$$
$$\Psi \leq \Psi'$$

We can conclude (C.105) with [T-OUT] and axioms

$$\Gamma, \Psi \vdash M : T \tag{C.107}$$
$$\Gamma, \Psi \vdash N : T' \tag{C.108}$$
$$\Gamma, \Psi \vdash P \tag{C.109}$$
$$T \hookleftarrow T' \tag{C.110}$$

We wish to show

$$\Gamma, \Psi' \vdash \overline{M}N.P \tag{C.111}$$

Applying [T-ASS-WEAK] with (C.107) and (C.108), and using induction hypothesis with (C.109), we have

$$\Gamma, \Psi' \vdash M : T \tag{C.112}$$
$$\Gamma, \Psi' \vdash N : T' \tag{C.113}$$
$$\Gamma, \Psi' \vdash P \tag{C.114}$$

Using [T-OUT], (C.112), (C.113), (C.114) and (C.110) we can conclude (C.111).

[T-PAR] Assume

$$\Gamma, \Psi \vdash P \mid Q \tag{C.115}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.116}$$
$$\Psi \leq \Psi'$$

We can conclude (C.115) with [T-PAR] and axioms

$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.117}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.118}$$
$$\mathcal{F}_\nu(P) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{C.119}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{C.120}$$

We wish to show

$$\Gamma, \Psi' \vdash P \mid Q \tag{C.121}$$

Applying induction hypothesis with (C.117) and (C.118) we have

$$\Gamma, \mathcal{F}_\nu(Q), \Psi' \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.122}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi' \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.123}$$

We choose $\mathcal{F}_\nu(P)$ and $\mathcal{F}_\nu(Q)$ such that we have

$$\mathcal{F}_\nu(P) \# \Psi', \mathcal{F}_\nu(Q), Q \tag{C.124}$$
$$\mathcal{F}_\nu(Q) \# \Psi', \mathcal{F}_\nu(P), P \tag{C.125}$$

Using [T-PAR], (C.122), (C.123), (C.124) and (C.125) we can conclude (C.121).

[T-NEW] Assume

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P \tag{C.126}$$
$$\mathrm{n}(\Psi) \subseteq \mathrm{dom}(\Gamma) \tag{C.127}$$
$$\Psi \leq \Psi'$$

We can conclude (C.126) with [T-NEW] and axioms

$$\Gamma, x : T, \Psi \vdash P \tag{C.128}$$
$$x \# \Psi \tag{C.129}$$

We wish to show

$$\Gamma, \Psi' \vdash (\boldsymbol{\nu} x)\, P \tag{C.130}$$

Using the induction hypothesis with (C.128) we can conclude

$$\Gamma, x : T, \Psi' \vdash P \tag{C.131}$$

Using alpha renaming, we choose an x such that we have

$$x \# \Psi' \tag{C.132}$$

Using [T-NEW], (C.131) and (C.132) we can conclude (C.130).

[T-ASSERT] Assume

$$\Gamma, \Psi \vdash (\!|\Psi''|\!) \tag{C.133}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.134}$$
$$\Psi \leq \Psi'$$

We can conclude (C.133) with

$$\Gamma, \Psi \vdash \Psi'' \tag{C.135}$$

We wish to show

$$\Gamma, \Psi' \vdash (\!|\Psi''|\!) \tag{C.136}$$

Using [T-ASS-WEAK] with (C.135) we have

$$\Gamma, \Psi' \vdash (\!|\Psi''|\!) \tag{C.137}$$

Using [T-ASSERT] with (C.137) we can conclude (C.136).

[T-REPL] Assume

$$\Gamma, \Psi \vdash\ !\, P \tag{C.138}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.139}$$
$$\Psi \leq \Psi'$$

We can conclude (C.138) with [T-REPL] and axiom

$$\Gamma, \Psi \vdash P \tag{C.140}$$

We wish to show

$$\Gamma, \Psi' \vdash\ !\, P \tag{C.141}$$

Using the induction hypothesis with (C.140) we have

$$\Gamma, \Psi' \vdash P \tag{C.142}$$

Using [T-REPL] with (C.142) we can conclude (C.141).

[T-CASE] Assume

$$\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.143}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.144}$$
$$\Psi \leq \Psi'$$

We can conclude (C.143) with [T-CASE] and axioms

$$\Gamma, \Psi \vdash \varphi_i \tag{C.145}$$
$$\Gamma, \Psi \vdash P_i \tag{C.146}$$

We wish to show

$$\Gamma, \Psi' \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.147}$$

Using [T-ASS-WEAK] with (C.145), and using the induction hypothesis with (C.146) we have

$$\Gamma, \Psi' \vdash \varphi_i \tag{C.148}$$
$$\Gamma, \Psi' \vdash P_i \tag{C.149}$$

Using [T-CASE] with (C.148) and (C.149) we can conclude (C.147).

[T-RUN] Assume

$$\Gamma, \Psi \vdash \mathbf{run}\ M \tag{C.150}$$
$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma) \tag{C.151}$$
$$\Psi \leq \Psi'$$

We can conclude (C.150) with [T-RUN] and axioms

$$\Gamma, \Psi \vdash M : T \tag{C.152}$$
$$\Psi \Vdash M \Leftarrow P \tag{C.153}$$
$$\Gamma', \Psi \vdash P \tag{C.154}$$
$$T \curvearrowleft \Gamma' \tag{C.155}$$

We want to show

$$\Gamma, \Psi' \vdash \mathbf{run}\ M \tag{C.156}$$

From [T-ASS-WEAK], (C.152) we can conclude

$$\Gamma, \Psi' \vdash M : T \tag{C.157}$$

From [T-WEAK-ASS-CLAUS] and (C.153) we can conclude

$$\Psi' \Vdash M \Leftarrow P \tag{C.158}$$

From [T-ENV-CLAUS], (C.155) and (C.157), and the induction hypothesis we can conclude

$$\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Gamma') \tag{C.159}$$

together with (C.151) we have

$$\mathrm{n}(\Psi') \subseteq \mathrm{dom}(\Gamma') \tag{C.160}$$

Using the induction hypothesis with (C.160) and (C.154) we have

$$\Gamma', \Psi' \vdash P \tag{C.161}$$

Using [T-RUN], (C.157), (C.158), (C.161) and (C.155) we can conclude (C.156).

□

C.4 Proof of Lemma 6

The lemma states:

If $\Gamma, \Psi \vdash P$ *and* $\Psi \rhd P \underline{\ggg} P'$ *then* $\Gamma, \Psi \vdash P'$.

Proof. We use proof by induction on the rules for $\Psi \rhd P \underline{\ggg} P'$.

[E-RES] Assume

$$\Psi \rhd (\boldsymbol{\nu} x)\, P \underline{\ggg} (\boldsymbol{\nu} x)\, P' \tag{C.162}$$
$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P \tag{C.163}$$

where $x \# \Psi$. We can conclude (C.162) and (C.163) with [E-RES] and [T-NEW] and axioms

$$\Psi \rhd P \underline{\ggg} P' \tag{C.164}$$
$$\Gamma, x : T, \Psi \vdash P \tag{C.165}$$

We want to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P' \tag{C.166}$$

From the induction hypothesis, (C.164) and (C.165) we can conclude

$$\Gamma, \Psi \vdash P' \tag{C.167}$$

Using [T-NEW] and (C.167) we can conclude (C.166).

[E-Par] Assume

$$\Psi \rhd P \mid Q \lll P' \mid Q \tag{C.168}$$
$$\Gamma, \Psi \vdash P \mid Q \tag{C.169}$$

We can conclude (C.168) with [E-Par] and (C.169) with [T-par], and with axioms

$$\Psi \otimes \mathcal{F}_\Psi(Q) \rhd P \lll P' \tag{C.170}$$
$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.171}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.172}$$
$$\mathcal{F}_\nu(P) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{C.173}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{C.174}$$
$$\tag{C.175}$$

We want to show

$$\Gamma, \Psi \vdash P' \mid Q \tag{C.176}$$

From the induction hypothesis, (C.170) and (C.171) we can conclude

$$\Gamma, \mathcal{F}_\nu(Q)\ \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P' \tag{C.177}$$

With Lemma 4 we can conclude

$$\Gamma, \mathcal{F}_\Psi\left(P'\right), \Psi \otimes \mathcal{F}_\Psi\left(P'\right) \vdash Q \tag{C.178}$$
$$\mathcal{F}_\nu\left(P'\right) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{C.179}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu\left(P'\right), P' \tag{C.180}$$

With [T-par], (C.177), (C.178), (C.179) and (C.180) we can conclude (C.176).

[E-Case] Assume

$$\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \lll P_i \tag{C.181}$$
$$\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.182}$$

We can conclude (C.181) with [E-Case] and (C.182) with [T-case], and with axioms

$$\Psi \Vdash \varphi_i$$
$$\Gamma, \Psi \vdash \varphi_i$$
$$\Gamma, \Psi \vdash P_i \tag{C.183}$$

We want to show

$$\Gamma, \Psi \vdash P_i$$

which is given by (C.183).

[E-Run] Assume

$$\Psi \rhd \mathbf{run}\ M \ggg P \tag{C.184}$$
$$\Gamma, \Psi \vdash \mathbf{run}\ M \tag{C.185}$$

We can conclude (C.184) with [E-Run] and (C.185) with [T-run], and with premises

$$\begin{gathered}\Psi \Vdash M \Leftarrow P \\ T \curvearrowleft \Gamma' \\ \Gamma, \Psi \vdash M : T \\ \Psi \Vdash M \Leftarrow P \\ \Gamma', \Psi \vdash P\end{gathered} \tag{C.186}$$

We wish to show

$$\Gamma, \Psi \vdash P \tag{C.187}$$

From assumption about handles in the HOΨ-calculus we have

$$\mathrm{n}(P) \subseteq \mathrm{n}(M)$$

which means we have

$$\Gamma' \subseteq \Gamma$$

we can then use Lemma 1 to conclude (C.187).

[E-Struct] Assume

$$\Psi \rhd P \ggg P' \tag{C.188}$$
$$\Gamma, \Psi \vdash P \tag{C.189}$$

We can conclude (C.188) with [E-Struct] and axioms

$$P \equiv_S P' \tag{C.190}$$

We wish to show

$$\Gamma, \Psi \vdash P' \tag{C.191}$$

In order to show this case we need to show

$$P \equiv_S P' \implies (\Gamma, \Psi \vdash P \iff \Gamma, \Psi \vdash P') \tag{C.192}$$

We do this with proof by induction on $P \equiv_S P'$

[S-Scope] Assume

$$(\boldsymbol{\nu} x)\, P \mid Q \equiv_S (\boldsymbol{\nu} x)\,(P \mid Q) \tag{C.193}$$
$$x \# Q \tag{C.194}$$

First we show from left to right. Assume

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P \mid Q \tag{C.195}$$

We can conclude (C.194) with [T-par], [T-new] and premises

$$\Gamma, x : T, \mathcal{F}_\nu(Q)\,, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.196}$$
$$\Gamma, x : T, \mathcal{F}_\nu(P)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.197}$$
$$x : T, \mathcal{F}_\nu(P)\,\#\Psi, \mathcal{F}_\nu(Q)\,, Q \tag{C.198}$$
$$\mathcal{F}_\nu(Q)\,\#\Psi, x : T, \mathcal{F}_\nu(P)\,, P \tag{C.199}$$

We want to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\,(P \mid Q) \tag{C.200}$$

With [T-new], [T-par], (C.195), (C.196), (C.197) and (C.198) we can conclude (C.199). The steps from right to left is the exact same as left to right.

– The commutative, associative, identity, reflexive, symmetric and transitivity cases are trivial to show.

The result (C.191) follows from (C.192).

[E-Rep] Assume

$$\Psi \rhd !P \lll P \mid !P \tag{C.201}$$
$$\Gamma, \Psi \vdash !P \tag{C.202}$$

We can conclude (C.202) with [T-repl] and premise

$$\Gamma, \Psi \vdash P \tag{C.203}$$

Since P cannot contain any assertion we can conclude (C.203) with [T-par], [T-repl], (C.203) and (C.202).

□

C.5 Proof of Lemma 7

The lemma states:

If $\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P$ *and* $\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T}$ *then* $\Gamma, \Psi \vdash P[x := \widetilde{L}]$.

Proof. We use proof by induction on the rules of $\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P$.

[T-NIL] We want to show

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \mathbf{0} \wedge \Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \implies \Gamma, \Psi \vdash \mathbf{0}$$

which holds.

[T-IN] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \underline{M}(\lambda \widetilde{y})N.P \tag{C.204}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.205}$$

We can conclude (C.204) with rule [T-IN] and following axioms

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash M : T' \tag{C.206}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}, \Psi \vdash N : T'' \tag{C.207}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}, \Psi \vdash P \tag{C.208}$$
$$T' \leftrightarrow T'' \tag{C.209}$$

We wish to show

$$\Gamma, \Psi \vdash \underline{M[\widetilde{x} := \widetilde{L}]}(\lambda \widetilde{y})N.P[\widetilde{x} := \widetilde{L}] \tag{C.210}$$

From the induction hypothesis with (C.205) and (C.208) we have

$$\Gamma, \Psi \vdash P[\widetilde{x} := \widetilde{L}] \tag{C.211}$$

From [T-SUBS] with (C.205) and (C.206) we have

$$\Gamma, \Psi \vdash M[\widetilde{x} := \widetilde{L}] : T' \tag{C.212}$$

Using [T-IN] with (C.207), (C.209), (C.211) and (C.212) we can conclude (C.210).

[T-RUN] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \mathbf{run}\ M \tag{C.213}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.214}$$

We can conclude (C.213) with [T-RUN] and axioms

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash M : T \tag{C.215}$$
$$\Psi \Vdash M \Leftarrow P \tag{C.216}$$
$$\Gamma', \Psi \vdash P \tag{C.217}$$
$$T \curvearrowleft \Gamma' \tag{C.218}$$

We want to show

$$\Gamma, \Psi \vdash \mathbf{run}\ (M[\widetilde{x} := \widetilde{L}]) \tag{C.219}$$

From [T-SUBS] with (C.215), (C.214), (C.217) we can conclude

$$\Gamma, \Psi \vdash M[\widetilde{x} := \widetilde{L}] : T \tag{C.220}$$

Now assume that

$$\Psi \Vdash M[\widetilde{x} := \widetilde{L}] \Leftarrow Q \tag{C.221}$$

and with [T-SUBS-RUN] togther with (C.215), (C.220) and (C.218) we can conclude

$$\Gamma', \Psi \vdash Q \tag{C.222}$$

Using [T-RUN], (C.220), (C.221), (C.222) and (C.218) we can conclude (C.219).

[T-OUT] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \overline{M} N.P \tag{C.223}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.224}$$

We can conclude (C.223) with [T-OUT] and axioms

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash M : T' \tag{C.225}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash N : T'' \tag{C.226}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \tag{C.227}$$
$$T' \leftharpoonup T'' \tag{C.228}$$

We wish to show

$$\Gamma, \Psi \vdash \overline{M[\widetilde{x} := \widetilde{L}]} N[\widetilde{x} := \widetilde{L}].P[\widetilde{x} := \widetilde{L}] \tag{C.229}$$

From [T-SUBS] with (C.225) and (C.226) together with (C.224) we can conclude

$$\Gamma, \Psi \vdash M[\widetilde{x} := \widetilde{L}] : T' \tag{C.230}$$
$$\Gamma, \Psi \vdash N[\widetilde{x} := \widetilde{L}] : T'' \tag{C.231}$$

Using the induction hypothesis with (C.227) and (C.224) we can conclude

$$\Gamma, \Psi \vdash P[\widetilde{x} := \widetilde{L}] \tag{C.232}$$

Using [T-OUT] with (C.230), (C.231), (C.232) and (C.228) we can conclude (C.229).

[T-PAR] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \mid Q \tag{C.233}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.234}$$

We can conclude (C.233) with [T-PAR] and axioms

$$\Gamma, \mathcal{F}_\nu(Q), \widetilde{x} : \widetilde{T}, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.235}$$
$$\Gamma, \mathcal{F}_\nu(P), \widetilde{x} : \widetilde{T}, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.236}$$
$$\mathcal{F}_\nu(P) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{C.237}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{C.238}$$

We wish to show

$$\Gamma, \Psi \vdash P[\widetilde{x} := \widetilde{L}] \mid Q[\widetilde{x} := \widetilde{L}] \tag{C.239}$$

From the induction hypothesis with (C.235) and (C.236) we have

$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P[\widetilde{x} := \widetilde{L}] \tag{C.240}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.241}$$
$$\tag{C.242}$$

From (C.234) we can conclude

$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi\Big(Q[\widetilde{x} := \widetilde{L}]\Big) \vdash P[\widetilde{x} := \widetilde{L}] \tag{C.243}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi\Big(P[\widetilde{x} := \widetilde{L}]\Big) \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.244}$$
$$\tag{C.245}$$

From Lemma 1 we can conclude

$$\Gamma, \mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big), \Psi \otimes \mathcal{F}_\Psi\Big(Q[\widetilde{x} := \widetilde{L}]\Big) \vdash P[\widetilde{x} := \widetilde{L}] \tag{C.246}$$

$$\Gamma, \mathcal{F}_\nu\Big(P[\widetilde{x} := \widetilde{L}]\Big), \Psi \otimes \mathcal{F}_\Psi\Big(P[\widetilde{x} := \widetilde{L}]\Big) \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.247}$$

$$\tag{C.248}$$

And we choose $\mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big)$ and $\mathcal{F}_\nu\Big(P[\widetilde{x} := \widetilde{L}]\Big)$ such that

$$\mathcal{F}_\nu\Big(P[\widetilde{x} := \widetilde{L}]\Big) \# \Psi, \mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big), Q[\widetilde{x} := \widetilde{L}] \tag{C.249}$$

$$\mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big) \# \Psi, \mathcal{F}_\nu\Big(P[\widetilde{x} := \widetilde{L}]\Big), P[\widetilde{x} := \widetilde{L}] \tag{C.250}$$

Using [T-PAR], (C.246), (C.247), (C.249) and (C.250) we can conclude (C.239).

[T-NEW] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash (\boldsymbol{\nu} y)\, P \tag{C.251}$$

$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.252}$$

We can conclude (C.251) with [T-NEW] and axioms

$$\Gamma, \widetilde{x} : \widetilde{T}, y : T', \Psi \vdash P \tag{C.253}$$

$$y \# \Psi \tag{C.254}$$

We wish to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} y)\, (P[\widetilde{x} := \widetilde{L}]) \tag{C.255}$$

Using [T-NEW], the induction hypothesis, (C.252) and (C.253) we can conclude (C.255).

[T-ASSERT] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash (\!|\Psi'|\!) \tag{C.256}$$

$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.257}$$

We can conclude (C.256) with

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \Psi' \tag{C.258}$$

We wish to show

$$\Gamma, \Psi \vdash (\!|\Psi'[\widetilde{x} := \widetilde{L}]|\!) \tag{C.259}$$

Using [T-ASSERT] with [T-SUBS], (C.257) and (C.258) we can conclude (C.259).

[T-REPL] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash\ !\, P \tag{C.260}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.261}$$

We can conclude (C.260) with [T-REPL] and axiom

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P \tag{C.262}$$

We wish to show

$$\Gamma, \Psi \vdash\ !\, P[\widetilde{x} := \widetilde{L}] \tag{C.263}$$

Using [T-REPL], the induction hypothesis with (C.261) and (C.262) we can conclude (C.263).

[T-CASE] Assume

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{C.264}$$
$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.265}$$

We can conclude (C.264) with [T-CASE] and axioms

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash \varphi_i \tag{C.266}$$
$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash P_i \tag{C.267}$$

We wish to show

$$\Gamma, \Psi \vdash \mathbf{case}\ \widetilde{\varphi[\widetilde{x} := \widetilde{L}]} : \widetilde{P[\widetilde{x} := \widetilde{L}]} \tag{C.268}$$

Using [T-SUBS] with (C.265) and (C.266), and using the induction hypothesis with (C.265) and (C.267), we can conclude

$$\Gamma, \Psi \vdash \varphi_i[\widetilde{x} := \widetilde{L}] \tag{C.269}$$
$$\Gamma, \Psi \vdash P_i[\widetilde{x} := \widetilde{L}] \tag{C.270}$$

Using [T-CASE] with (C.269) and (C.270) we can conclude (C.268).

□

C.6 Proof of Theorem 3

The theorem states:

If $\Gamma, \Psi \vdash P$ *and* $\Psi \rhd P \to P'$ *then* $\Gamma, \Psi \vdash P'$.

Proof. We use proof by induction on the rules for $\Psi \rhd P \to P'$.

[R-EVAL] Assume

$$\Psi \rhd P \to P' \tag{C.271}$$
$$\Gamma, \Psi \vdash P \tag{C.272}$$

with [R-EVAL] we can conclude (C.271) with premises

$$\Psi \rhd P \lll P'' \tag{C.273}$$
$$\Psi \rhd P'' \to P' \tag{C.274}$$

we want to show

$$\Gamma, \Psi \vdash P' \tag{C.275}$$

With Lemma 6, (C.273) and (C.272) we can conclude

$$\Gamma, \Psi \vdash P'' \tag{C.276}$$

With the induction hypothesis, (C.274) and (C.276) we can conclude (C.275).

[R-PAR] Assume

$$\Psi \rhd P \mid Q \to P' \mid Q \tag{C.277}$$
$$\Gamma, \Psi \vdash P \mid Q \tag{C.278}$$

We can conclude (C.277) with [R-PAR] and (C.278) with [T-PAR], and with axioms

$$\Psi \otimes \mathcal{F}_\Psi(Q) \rhd P \to P' \tag{C.279}$$
$$\mathcal{F}_\nu(P) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{C.280}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{C.281}$$
$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{C.282}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.283}$$

We want to show

$$\Gamma, \Psi \vdash P' \mid Q \tag{C.284}$$

From the induction hypothesis with (C.279) and (C.283) we can conclude

$$\Gamma, \mathcal{F}_\nu(Q)\,, \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P' \tag{C.285}$$

From Lemma 3 and (C.279) we know

$$\mathcal{F}(P) \leq \mathcal{F}\big(P'\big) \tag{C.286}$$

using Lemma 1 with (C.283) we can conclude

$$\Gamma, \mathcal{F}_\nu\big(P'\big)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{C.287}$$

and togher with Lemma 5 we can conclude

$$\Gamma, \mathcal{F}_\nu\big(P'\big)\,, \Psi \otimes \mathcal{F}_\Psi\big(P'\big) \vdash Q \tag{C.288}$$

We choose the new names in $\mathcal{F}_\nu(P')$ and $\mathcal{F}_\nu(Q)$

$$\mathcal{F}_\nu\big(P'\big)\,\#\Psi, \mathcal{F}_\nu(Q)\,, Q \tag{C.289}$$
$$\mathcal{F}_\nu(Q)\,\#\Psi, \mathcal{F}_\nu\big(P'\big)\,, P' \tag{C.290}$$

Using [T-PAR] with (C.289), (C.290), (C.285) and (C.288) we can conclude (C.284).

[R-RES] Assume

$$\Psi \rhd (\boldsymbol{\nu} x)\, P \to (\boldsymbol{\nu} x)\, P' \tag{C.291}$$
$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P \tag{C.292}$$

We can conclude (C.291) and (C.292) with [R-RES], [T-NEW] and premises

$$\Gamma, x : T, \Psi \vdash P \tag{C.293}$$
$$\Psi \rhd P \to P' \tag{C.294}$$
$$x\#\Psi \tag{C.295}$$

We wish to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x)\, P' \tag{C.296}$$

From the induction hypothesis, (C.293) and (C.294) we can conclude

$$\Gamma, x : T, \Psi \vdash P' \tag{C.297}$$

Using rule [T-NEW] with (C.297) and (C.295) we can conclude (C.296)

[R-COM] Assume

$$\Psi \rhd \overline{M}N[\widetilde{x} := \widetilde{L}].P \mid \underline{K}(\lambda\widetilde{x})N.Q \to P \mid Q[\widetilde{x} := \widetilde{L}] \tag{C.298}$$

$$\Gamma, \Psi \vdash \overline{M}N[\widetilde{x} := \widetilde{L}].P \mid \underline{K}(\lambda\widetilde{x})N.Q \tag{C.299}$$

We can conclude (C.298) and (C.299) with [R-COM], [T-PAR], [T-OUT], [T-IN] and premises

$$\Psi \Vdash M \dot{\leftrightarrow} K \tag{C.300}$$

$$\Gamma, \Psi \vdash M : T \tag{C.301}$$

$$\Gamma, \Psi \vdash N[\widetilde{x} := \widetilde{L}] : F(\widetilde{T}) \tag{C.302}$$

$$\Gamma, \Psi \vdash P \tag{C.303}$$

$$\Gamma, \Psi \vdash K : T \tag{C.304}$$

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash N : F(\widetilde{T}) \tag{C.305}$$

$$\Gamma, \widetilde{x} : \widetilde{T}, \Psi \vdash Q \tag{C.306}$$

$$\Gamma, \Psi \vdash T \hookleftarrow F(\widetilde{T}) \tag{C.307}$$

Since we have the assumption [T-EQUAL] togher with (C.300), M and K shares the same type T in (C.301) and (C.304), and $N[\widetilde{x} := \widetilde{L}]$ and $N : F(\widetilde{T})$ share type in (C.302) and (C.305).

We wish to show

$$\Gamma, \Psi \vdash P \mid Q[\widetilde{x} := \widetilde{L}] \tag{C.308}$$

From instance assumption [T-COMP-TERM] and (C.302) we can conclude

$$\Gamma, \Psi \vdash \widetilde{L} : \widetilde{T} \tag{C.309}$$

From Lemma 7, (C.306) and (C.309) we can conclude

$$\Gamma, \Psi \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.310}$$

From Lemma 1 together with (C.303) and (C.310) we can conclude

$$\Gamma, \mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big), \Psi \vdash P \tag{C.311}$$

$$\Gamma, \mathcal{F}_\nu(P), \Psi \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.312}$$

Using Lemma 5 together with (C.311) and (C.312) we can conclude

$$\Gamma, \mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big), \Psi \otimes \mathcal{F}_\Psi\Big(Q[\widetilde{x} := \widetilde{L}]\Big) \vdash P \tag{C.313}$$

$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q[\widetilde{x} := \widetilde{L}] \tag{C.314}$$

We choose $\mathcal{F}_\nu(P)$ and $\mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big)$ such that we have

$$\mathcal{F}_\nu\Big(Q[\widetilde{x} := \widetilde{L}]\Big) \,\#\Psi, \mathcal{F}_\nu(P)\,, P \tag{C.315}$$

$$\mathcal{F}_\nu(P)\,\Psi, Q[\widetilde{x} := \widetilde{L}], Q \tag{C.316}$$

With [T-PAR], (C.313), (C.314), (C.315) and (C.316) we can conclude (C.308).

□

D Proof for instantiations of the type system

D.1 Proof of Theorem 5

The theorem states:

If $\Gamma, \Psi \vdash P$ then $P \not\rightarrow$ **WRONG**.

Proof. We use proof by induction on the rules for $\Gamma, \Psi \vdash P$. We omit the cases where the process does not match any type error rules.

[T-IN] Assume

$$\Gamma, \Psi \vdash M\,(\widetilde{x})\,N.P \tag{D.1}$$

which we can conclude with [T-IN], [T-CHA] and the following premises

$$\mathrm{ch}(T) \leftrightarrow T \tag{D.2}$$
$$\Gamma, \Psi \vdash M : \mathrm{ch}(T) \tag{D.3}$$
$$\Gamma, x : T, \Psi \vdash N : T \tag{D.4}$$
$$\Gamma, x : T, \Psi \vdash P \tag{D.5}$$

We want to show

$$\Gamma, \Psi \vdash M\,(\widetilde{x})\,N.P \not\rightarrow \textbf{WRONG} \tag{D.6}$$

In order for (D.6) to be false there are two cases

- The first rule require the following premises

  $$\Gamma, \Psi \vdash M : \mathrm{drop}(\Gamma')$$

  which contradict (D.3).
- The second rule requires

  $$\Gamma, \Psi \vdash M : \mathrm{ch}(T) \tag{D.7}$$
  $$\Gamma, x : T, \Psi \vdash Q \rightarrow \textbf{WRONG} \tag{D.8}$$

  By using the indcution hypothesis with (D.5) we contradict (D.8).

[T-RUN] Assume

$$\Gamma, \Psi \vdash \textbf{run}\ M \tag{D.9}$$

which we can conclude with [T-IN], [T-END] and premises

$$\text{drop}(\Gamma') \curvearrowleft \Gamma' \tag{D.10}$$
$$\Gamma, \Psi \vdash M : \text{drop}(\Gamma') \tag{D.11}$$
$$\Psi \Vdash M \Leftarrow P \tag{D.12}$$
$$\Gamma', \Psi \vdash P \tag{D.13}$$

We want to show

$$\Gamma, \Psi \vdash \textbf{run}\ M \not\twoheadrightarrow \textbf{WRONG} \tag{D.14}$$

In order for (D.14) to be false there are two cases

- The first rule require the following premise

  $$\Gamma, \Psi \vdash M : \text{ch}(T)$$

  which contradict (D.11).
- The second rule require the following premises

  $$\Gamma, \Psi \vdash P : \text{drop}(\Gamma') \tag{D.15}$$
  $$\Gamma', \Psi \vdash P \rightarrow \textbf{WRONG} \tag{D.16}$$

  Which contradict with the induction hypothesis together with (D.13).

[T-OUT] Assume

$$\Gamma, \Psi \vdash M\texttt{<}N\texttt{>}.P \tag{D.17}$$

which we can conclude with [T-OUT], [T-CHA] and premises

$$\text{ch}(T) \twoheadleftarrow T \tag{D.18}$$
$$\Gamma, \Psi \vdash M : \text{ch}(T) \tag{D.19}$$
$$\Gamma, \Psi \vdash N : T \tag{D.20}$$
$$\Gamma, \Psi \vdash P \tag{D.21}$$

we want to show

$$\Gamma, \Psi \vdash M\texttt{<}N\texttt{>}.P \not\twoheadrightarrow \textbf{WRONG} \tag{D.22}$$

In order for (D.22) to be false there are two rules

– The first rule require the following premise

$$\Gamma, \Psi \vdash M : \text{drop}(\Gamma') \tag{D.23}$$

or

$$\Gamma, \Psi \vdash P \to \textbf{WRONG} \tag{D.24}$$

(D.23) contradict (D.19) and (D.24) contradict the induction hypothesis together with (D.21).

– The second rule is in case N is process Q, and is concluded with the following premises

$$\Gamma \vdash Q : \text{drop}(\Gamma') \tag{D.25}$$
$$\Gamma', \Psi \vdash Q \to \textbf{WRONG} \tag{D.26}$$

When N is Q we can conclude (D.20) with rule [T-TERM-2] and premise

$$\Gamma', \Psi \vdash P$$

which together with the induction hypothesis contradict (D.26).

[T-PAR] Assume

$$\Gamma, \Psi \vdash P \mid Q \tag{D.27}$$

which we can conclude with [T-PAR] and premises

$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \tag{D.28}$$
$$\Gamma, \mathcal{F}_\nu(P), \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \tag{D.29}$$
$$\mathcal{F}_\nu(P) \# \Psi, \mathcal{F}_\nu(Q), Q \tag{D.30}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{D.31}$$

we want to show

$$\Gamma, \Psi \vdash P \mid Q \not\to \textbf{WRONG} \tag{D.32}$$

In order for (D.32) to be false there are to rules

– The first rule require premises

$$\Gamma, \mathcal{F}_\nu(Q), \Psi \otimes \mathcal{F}_\Psi(Q) \vdash P \to \textbf{WRONG} \tag{D.33}$$
$$\mathcal{F}_\nu(Q) \# \Psi, \mathcal{F}_\nu(P), P \tag{D.34}$$

Witch contradict the induction hypothesis together with (D.28).

– The second rule require premises

$$\Gamma, \mathcal{F}_\nu(P)\,, \Psi \otimes \mathcal{F}_\Psi(P) \vdash Q \to \textbf{WRONG} \tag{D.35}$$
$$\mathcal{F}_\nu(P)\,\#\,\Psi, \mathcal{F}_\nu(Q)\,, Q \tag{D.36}$$

Witch contradict the induction hypothesis together with (D.29).

[T-NEW] Assume

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x : T)\, P \tag{D.37}$$

which we can conclude with [T-NEW] and premises

$$\Gamma, x : T, \Psi \vdash P \tag{D.38}$$
$$x\#\Psi \tag{D.39}$$

we want to show

$$\Gamma, \Psi \vdash (\boldsymbol{\nu} x : T)\, P \not\to \textbf{WRONG} \tag{D.40}$$

In order for (D.40) to be false the following premises need to hold

$$\Gamma, x : T, \Psi \vdash P \to \textbf{WRONG} \tag{D.41}$$
$$x\#\Psi \tag{D.42}$$

Which contradict the induction hypothesis together with (D.38).

[T-REPL] Assume

$$\Gamma, \Psi \vdash\ !\, P \tag{D.43}$$

which we can conclude with [T-REPL] and premises

$$\Gamma, \Psi \vdash P \tag{D.44}$$

we want to show

$$\Gamma, \Psi \vdash\ !\, P \not\to \textbf{WRONG} \tag{D.45}$$

In order for (D.45) to be false following premise need to hold

$$\Gamma, \Psi \vdash P \to \textbf{WRONG}$$

which contradict the induction hypothesis together with (D.44).

[T-CASE] Assume

$$\Gamma, \Psi \vdash \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \tag{D.46}$$

which we can conclude with [T-CASE] and premises

$$\Gamma, \Psi \vdash \varphi_i \tag{D.47}$$
$$\Gamma, \Psi \vdash P_i \tag{D.48}$$

We wish to show

$$\Gamma, \Psi \vdash \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \not\to \textbf{WRONG} \tag{D.49}$$

In order for (D.49) to be false, following premise need to hold

$$\Gamma, \Psi \vdash P_i \to \textbf{WRONG}$$

which contradict with the induction hypothesis together with (D.48).

□