



A generic type system for higher-order Ψ -calculi

Hans Hüttel^{a,*}, Stian Lybech^{b,*}, Alex R. Bendixen^c, Bjarke B. Bojesen^c

^a Department of Computer Science, University of Copenhagen, Universitetsparken 1, Copenhagen, 2100, Denmark

^b Department of Computer Science, Reykjavík University, Menntavegur 1, Reykjavík, 102, Iceland

^c Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, Aalborg Ø, 9220, Denmark

ARTICLE INFO

Article history:

Received 31 March 2023

Received in revised form 1 July 2024

Accepted 4 July 2024

Available online 10 July 2024

ABSTRACT

The Higher-Order Ψ -calculus framework ($\text{HO}\Psi$) by Parrow et al. is a generalisation of many first- and higher-order extensions of the π -calculus. In this paper we present a generic type system for $\text{HO}\Psi$ -calculi. It satisfies a subject reduction property and can be instantiated to yield both existing and new type systems for calculi, that can be expressed as $\text{HO}\Psi$ -calculi. In this paper, we consider the type system for termination in $\text{HO}\pi$ by Demangeon et al. Moreover, we derive a new type system for the ρ -calculus of Meredith and Radestock and present a type system for non-interference for mobile code.

© 2024 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Process calculi are formalisms for modelling and reasoning about concurrent and distributed computations; a prominent example being the π -calculus of Milner et al. [1,2], which models computation as communication between processes, by passing messages on named channels.

Since its inception, a multitude of variants of the π -calculus has appeared; e.g. $\text{D}\pi$ [3], the calculus of explicit fusions [4], the spi-calculus with correspondence assertions [5] and the $^e\pi$ -calculus [6]. These calculi are all *first-order*, in the sense that only atomic channel names can be passed around, not processes themselves. Moreover, the calculi have semantics that share characteristics such as the treatment of scope extension and name passing but also differ in specific aspects that pertain to the calculus under consideration.

Bengtson et al. [7,8] created Ψ -calculi as a generalisation of the first-order variants and extensions of the π -calculus, allowing them to be expressed as *instances* of the Ψ -calculus framework through appropriate settings of a small number of parameters. One of these parameters is the set of messages or *terms*, which can be passed around and used as channels, and the choice of this set is subject only to some very mild constraints. Thus, even in the ‘first-order’ Ψ -calculus, the set of terms may be chosen such that it contains the set of processes, thereby allowing processes to be passed around as messages.

However, the Ψ -calculus lacks the one feature that is common to process calculi with higher-order communication, namely, the ability to execute a received message as a process. This feature was added as an extension to the framework by Parrow et al. [9], who thus created the *Higher-Order* Ψ -calculus, $\text{HO}\Psi$. This framework can be parametrised to yield instances corresponding to higher-order calculi such as $\text{HO}\pi$ and CHOCS, as well as every calculus that can be expressed in the first-order Ψ -calculus, including $\text{D}\pi$ [3], the calculus of explicit fusions [4], the spi-calculus with correspondence

* Corresponding author.

E-mail addresses: hans.huttel@di.ku.dk (H. Hüttel), stian21@ru.is (S. Lybech).

¹ Supported by the Icelandic Research Fund Grant No. 218202-05(1-3).

assertions [5] and the $e\pi$ -calculus [6]. One advantage of this generalised framework is the ability to reason about these many extensions and variants of the π -calculus collectively. For example, in [8,9], the authors define a notion of strong bisimilarity for the π -calculus, which then is automatically inherited by every instance of the calculus.

An important approach for reasoning about processes is that of *type systems*. For process calculi, this approach originates with Milner [1] whose first type system deals with the notion of correct usage of channels in the π -calculus: In a well-typed process only names of the correct type can be communicated. Pierce and Sangiorgi [10] later described a type system that uses subtyping and capability tags to control the use of names as input or output channels. Many of the aforementioned first-order extensions of the π -calculus have also been given type systems that capture properties such as secrecy, authenticity and safe migration.

In [11], Hüttel noted that these type systems, despite arising in different settings, share certain characteristics: The type judgments for processes P are all of the form $\Gamma \vdash P$ where Γ is a type environment recording the types of the free names in P , so processes are only classified as being either well-typed or not. On the other hand, terms M are given a type T , so type judgements for terms are of the form $\Gamma \vdash M : T$. Based on these shared characteristics, Hüttel then created a generic type system for the first-order Ψ -calculus framework, that generalises several of the type systems for the π -calculus and its variants. This generic type system can similarly be instantiated through parameter settings to yield both well-known and new type systems for the calculi that are representable as first-order Ψ -calculi. An important advantage of this approach is that a general result of type system soundness can be formulated, which is then inherited by all instances of the type system. The soundness result is one of subject reduction along with a generalized version of the channel usage property from Milner's original system.

There has been other work on generic type systems, notably work by König [12], Caires [13] and Igarashi and Kobayashi [14]. As these contributions demonstrate, the notion of genericity can be understood in different ways. The type system by Caires [13] is formulated only for the first-order monadic π -calculus, but is parametrised with a subtyping relation, which allows different typing disciplines to be expressed. These include a simple, sorting-like system for correct channel, input/output-capabilities and certain behavioural properties. Seen in this light, the type system by Caires generalises more typing disciplines, but it is only formulated for a specific calculus, whereas other generic type systems can only express a more narrow collection of properties directly, but can be instantiated for a broad range of calculi.

However, all generic type systems mentioned in the above are formulated for variants of the first-order π -calculus. This means that they cannot be instantiated to yield type system for *higher-order* calculi, such as $\text{HO}\pi$ or CHOCs . Both of these higher-order calculi can be encoded into the first-order π -calculus, as shown by Sangiorgi in [15], and may therefore also be represented in just the first-order Ψ -calculus. Not surprisingly, there is therefore little work on type systems for higher-order calculi, since these encodings allow us to disregard the higher-order behaviour and instead just type the first-order translations.

One exception is the type system for termination in variants of $\text{HO}\pi$, due to Demangeon et al. [16]. As the authors argue, it may not always be desirable (or even possible) to type a higher-order language through a first-order representation, if the language contains features that are difficult (or impossible) to encode. Another example of this is the Reflective Higher-Order (RHO or ρ) calculus of Meredith and Radestock [17], which is a name-passing process calculus in which names are generated by “quoting” process expressions and higher-order communication is obtained by passing around such names which can then be “unquoted” (or “dropped”). The ρ -calculus cannot be uniformly encoded in the π -calculus, as shown in [18] so we here have an example of a language that cannot easily be represented in the first-order paradigm. This makes it difficult (even impossible) to adapt any of the existing first-order type systems to this language.

In the present paper, we create a new, generic type system for higher-order Ψ -calculi, which extends the generic type system of Hüttel [11] for the first-order Ψ -calculus, and we show both how existing type systems for higher-order calculi can be expressed as instances of this generic type system, and how new type systems can be obtained. Like its predecessor, our generic type system satisfies a general subject reduction property that is inherited by all instances.

Our generic type system is able to handle subtyping at the level of terms, as the type rules for terms are specific for each instance of it. It is therefore simple to incorporate a subsumption rule. Other forms of polymorphism can be handled by means of the channel compatibility predicate. On the other hand, the type rule for parallel composition reveals that we cannot handle linear type systems since the premise says that the type environment to be used for typing each parallel component is the same; this will also be clear by our requirement that every instance must satisfy the notions of weakening and strengthening.

We use our generic type system to formulate simple type systems for $\text{HO}\pi$, and show that the type system for termination by Demangeon et al. can also be captured as an instance of our type system. Moreover, we present new type systems. First, we demonstrate that our generic type system can be instantiated to yield a type system for the ρ -calculus, and this establishes that our type system is richer than the first-order type systems. Finally we use the generic type system to present a new type system for non-interference for mobile code.

The paper is structured as follows: In Section 2 we introduce the syntax and semantics of the higher-order Ψ -calculus. Then, in Section 3, we present the generic type system, and in Section 4 we establish important properties of the type system. In particular, we prove a subject reduction property (Theorem 1) for the type system and introduce a general notion of safety. Finally, in Section 5 we give some examples of instantiations to show how a number of existing type systems for higher-order calculi can be seen as instances of the generic type system; and also how the generic type system can be instantiated to yield new type systems.

The present paper is an extension of [19]. Compared to the previous version, the review of the $\text{HO}\Psi$ -calculus and the presentation of the generic type system and the instance assumptions have been substantially rewritten and expanded to provide a more thorough and detailed presentation. We also provide full proofs for the subject reduction property (Theorem 1) and most of the associated lemmas (Lemmas 1–8), and for the property of operational correspondence for the ρ -calculus instance (Theorem 3). Lastly, the type system instance for non-interference presented in Section 5.4 is also new.

2. The higher-order Ψ -calculus

The first-order Ψ -calculus [7,8] generalises the common characteristics of variants of the π -calculus that allow for transmission of structured message terms M , including, in principle, also processes. The Higher-Order Ψ -calculus then extends the original Ψ -calculus with an invocation construct, **run** M , which allows a received message to be executed as a running process. In this section we first review the syntax of $\text{HO}\Psi$ as given in [9], and then proceed to give a reduction semantics for the calculus.

2.1. Parameters

The Higher-Order Ψ -calculus is a general framework, which is intended to allow many different calculi to be obtained as instances, by setting a small number of parameters in the form of definitions of three (not necessarily disjoint) sets of *terms*, *conditions* and *assertions*, and four operations on these sets. Elements of these sets appear in the syntax of the Ψ -calculus, and thus different calculi can be obtained as instances by making different choices for the definitions of these sets.

To allow the framework to be as general and flexible as possible, the authors of [7,9] identify only a few restrictions that must be imposed on the sets of terms, conditions and assertions: they must be *nominal datatypes*. Informally, a *nominal set*, in the sense of Gabbay and Pitts [20], is a set whose members can be affected by names being bound or swapped. Firstly, assume a countably infinite set of names $\mathcal{N}$, ranged over by $a, \dots, z$. If a, b are names and X is an element of a nominal set, then the *transposition* of a and b on X , written $(a, b) \cdot X$, swaps all occurrences of a for b in X and vice versa. A function f on a nominal set is *equivariant*, if it is unaffected by name swapping; i.e. if $(a, b) \cdot f(X) = f((a, b) \cdot X)$, and a *nominal datatype* is a nominal set together with a set of equivariant functions on it. This requirement is very mild and allows e.g. non-well-founded sets to be used in an instantiation; for example, the set of processes can itself be included in the sets of terms, conditions or assertions.

Two other, important notions are those of support and freshness: if X is an element of a nominal set, a name a is said to *occur* in X , if it can be affected by transposition, and the *support* of X , written $\text{n}(X)$, is the set of names that occur in X . Conversely, a name a is *fresh* for X , written $a \# X$, if $a \notin \text{n}(X)$. This notion extends to sets of names A , such that $A \# X$ if it is the case that $\forall a \in A. a \notin \text{n}(X)$, and this is pointwise extended to lists of elements $X_1, \dots, X_n$, so we write $A \# X_1, \dots, X_n$ for $A \# X_1 \wedge \dots \wedge A \# X_n$.

As mentioned above, any Ψ -calculus instance requires a specification of three nominal datatypes: the terms, conditions and assertions. Unlike in the original presentations of Ψ -calculi [7–9], we shall be using a *typed* syntax, with types appearing in the name binders, so we therefore add a fourth parameter to be specified in an instantiation, namely the nominal datatype of *types*. This is further described in section 3.1. The sets to be specified are thus:

$$\begin{aligned} M, N &\in \mathbb{T} \text{ Terms} \\ \varphi &\in \mathbb{C} \text{ Conditions} \\ \Psi &\in \mathbb{A} \text{ Assertions} \\ T &\in \mathbf{Types} \text{ Types} \end{aligned}$$

Terms M, N are used as subjects and objects in communication; they could be e.g. single names, as in the monadic π -calculus, vectors of names as in $^e\pi$ and the polyadic π -calculus; or elements of a composite datatype. Conditions φ are used in conditional expressions; e.g. matching and mismatch checks. Lastly, assertions Ψ can appear in the syntax and become enabled during the execution of a process, where they can affect the evaluation of conditions.

Each of the nominal datatypes must include the definition of an equivariant *substitution function* σ , written $(\cdot)[\tilde{a} := \tilde{M}]$ for the non-trivial part, which is the substitution of tuples of terms $\tilde{M}$ for tuples of names $\tilde{a}$ of equal arity. It must be defined such that it satisfies the following *substitution laws*:

- If $\tilde{a} \subseteq \text{n}(X)$ and $b \in \text{n}(\tilde{Y})$ then $b \in \text{n}(X[\tilde{a} := \tilde{Y}])$
- If $\tilde{u} \# X, \tilde{v}$ then $X[\tilde{v} := \tilde{Y}] = ((\tilde{u}, \tilde{v}) \cdot X)[\tilde{u} := \tilde{Y}]$

These 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 $\tilde{Y}$ must also be present when the substitution has been performed. The second law states that substitution cannot be affected by transposition.

For the purpose of defining the type system, we need to impose two further restrictions on the definition of the nominal sets, which are not present in the original formulation of Ψ -calculi: We require that the nominal sets of terms, conditions,

assertions and types can be described as term algebras; i.e. that they are generated by a signature of constructors. This follows the presentation of abstract syntax given in [21].

Since we allow term constructors with binders, we define capture-avoiding substitutions as follows, also following [21]. For any n -ary term constructor f with arity $(\tilde{s}_1.s_1, \dots, \tilde{s}_n.s_n)$ and substitution $\sigma = [\tilde{x} \mapsto \tilde{M}]$ we require that σ satisfies the following conditions.

1. $x\sigma = M$ if $\sigma(x) = M$ and $y\sigma = y$ if $y \notin \text{dom}(\sigma)$
2. $(f(\tilde{x}_1.M_1, \dots, \tilde{x}_n.M_n))\sigma = f(\tilde{x}_1.M'_1, \dots, \tilde{x}_n.M'_n)$ where for each $1 \leq i \leq n$ we require that $\tilde{x}_i \# M_i$ and we have $M'_i = M'_i\sigma$ if $\tilde{x}_i \# \tilde{x}_i$ and $M'_i = M_i$ otherwise.

In other words, term substitution distributes over function symbols. These requirements are also imposed in the type system proposed in [11], and they will ensure that a standard substitution lemma for type judgments will hold in our generic type system introduced later.

The Ψ -calculus allows arbitrary terms to be used as channels. Any Ψ -calculus instance therefore requires a definition of two equivariant operators, *channel equivalence* $\leftrightarrow$ 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:

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

We write the entailment relation as $\Psi \Vdash \varphi$ to denote that the condition φ holds, given the assertions Ψ . Our notation differs from e.g. that of [7], since we use the $\vdash$ symbol in the type judgements introduced later.

Note that comparison by channel equivalence $M_1 \leftrightarrow M_2$ is itself a condition, which may or may not be entailed by some assertions Ψ , according to the definition of the entailment relation.

In $\text{HO}\Psi$, higher-order communication is handled by declaring *handles* for processes, written $M \Leftarrow P$, which denotes that the term M is a handle for the process P . We assume the existence of conditions of the form $M \Leftarrow P$ in the definition of the set of conditions $\mathbb{C}$, such that the entailment $\Psi \Vdash M \Leftarrow P$ is always well-defined. Note that the set of processes may itself be included in the set of terms, thus allowing conditions of the form $P \Leftarrow P$ whereby a process becomes a handle for itself.

The authors of [9] impose two restrictions for the parameter settings to be *valid*: Channel equivalence must be symmetric and transitive, and $\otimes$ must satisfy the commutative monoidal laws, with $\mathbf{1}$ as the unit element.

Finally, for the purpose of defining the type system, we need to impose a constraint on the entailment of some conditions, specifically channel equivalence and handles.

Both must have a weakening property:

- If $\Psi \Vdash M_1 \leftrightarrow M_2$ then $\Psi \otimes \Psi' \Vdash M_1 \leftrightarrow M_2$.
- If $\Psi \Vdash M \Leftarrow P$ then $\Psi \otimes \Psi' \Vdash M \Leftarrow P$.

These requirements ensure that an extension of the assertions Ψ cannot invalidate channel equivalence, nor delete a handle for a process P . This could of course be generalised by requiring the weakening property to hold for all conditions, but since this requirement is not present in the original presentation of Ψ -calculi [7–9], we prefer this less constraining formulation.

In summary, we say that a Ψ -calculus instance is *valid*, if its parameter settings satisfy the following requirements:

Definition 1 (*Valid instance*). A Ψ -calculus instance is *valid*, if the parameter settings satisfy the following:

1. Substitution for each nominal datatype satisfies the substitution laws:

$$\begin{aligned} \tilde{a} \subseteq n(X) \wedge b \in n(\tilde{Y}) &\implies b \in n(X[\tilde{a} := \tilde{Y}]) \\ \tilde{u} \# X, \tilde{v} &\implies X[\tilde{v} := \tilde{Y}] = ((\tilde{u}, \tilde{v}) \cdot X)[\tilde{u} := \tilde{Y}] \end{aligned}$$

2. $\mathbb{T}$, $\mathbb{C}$, $\mathbb{A}$ and **Types** are generated by a signature of constructors, and term substitutions σ distribute over function symbols f :

$$f(M_1, \dots, M_n)\sigma = f(M_1\sigma, \dots, M_n\sigma)$$

3. Handles are included in the set of conditions: $M \Leftarrow P \in \mathbb{C}$.
4. Channel equivalence $\leftrightarrow$ is symmetric and transitive.
5. $(\mathbb{A}, \otimes, \mathbf{1})$ is a commutative monoid w.r.t. an equivalence on assertions, written $\Psi_1 \simeq \Psi_2$. This equivalence is defined such that $\Psi_1 \simeq \Psi_2$ if for all φ it holds that $\Psi_1 \Vdash \varphi \iff \Psi_2 \Vdash \varphi$.

6. Entailment satisfies a weakening property for channel equivalence and handles:

$$\begin{aligned}\Psi \Vdash M_1 \dot{\leftrightarrow} M_2 &\implies \Psi \otimes \Psi' \Vdash M_1 \dot{\leftrightarrow} M_2 \\ \Psi \Vdash M \Leftarrow P &\implies \Psi \otimes \Psi' \Vdash M \Leftarrow P\end{aligned}$$

7. Entailment is compositional for assertion equivalence:

$$\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''.$$

We shall only consider valid instances in the following.

2.2. Syntax

The set of HO Ψ -calculus processes $\mathcal{P}_\Psi$ is built by the formation rules

$P \in \mathcal{P}_\Psi ::=$	0	Nil
	$P_1 \mid P_2$	Parallel
	$\overline{M}N.P$	Output
	$\underline{M}(\lambda\tilde{x}:\tilde{T})N.P$	Input
	run M	Invocation
	case $\tilde{\varphi} : \tilde{P}$	Selection
	$(\nu x : T) P$	Restriction
	$!P$	Replication
	(Ψ)	Assertion

where the shorthand $\tilde{\varphi} : \tilde{P}$ denotes a list of cases $\varphi_1 : P_1 \sqcap \dots \sqcap \varphi_n : P_n$.

The nil, parallel, restriction and replication constructs are those of the π -calculus.

The input and output prefixes generalise those of the π -calculus, since here both subject and object are *terms* rather than just names. Thus $\overline{M}N.P$ outputs the term N on M and continues as P , whilst $\underline{M}(\lambda\tilde{x}:\tilde{T})N.P$ receives a term (e.g. K) on M that must match the pattern N . Here, $\tilde{x}$ is a list of pattern variables, binding into N and P : they are used to extract subterms from K that will then be substituted for the occurrences of $\tilde{x}$ within the continuation P . Any term K received on M must match this pattern for the communication to succeed; this means that it must be possible to obtain K from N by instantiating the variables $\tilde{x} = x_1, \dots, x_k$ in N with terms $N_1, \dots, N_k$, such that $N[x_1, \dots, x_k := N_1, \dots, N_k] = K$.

Unlike the presentation in [9], we here use an explicitly typed version of the language: the types of the pattern variables are found in the list $\tilde{T}$ where $|\tilde{x}| = |\tilde{T}|$. Likewise, in the restriction $(\nu x : T) P$, we annotate the name x bound in P with its type T .

The selection construct **case** $\tilde{\varphi} : \tilde{P}$ is a shorthand for a list of cases and is to be understood as saying: If condition φ_i is entailed by the assertions Ψ , we continue as P_i . If more than one condition is entailed, the process is chosen non-deterministically among those where the condition holds. This construct thus generalises the choice and matching operators of the π -calculus.

Higher-order communication is achieved by declaring terms as handles for processes; this may typically be done in an assertion. If the condition $M \Leftarrow P$ is entailed, we can then send P by sending its handle M , and P may thence be executed by using the invocation construct **run** M .

Finally, the authors in [9] impose the restriction that processes must be *well-formed*, which is defined as follows:

Definition 2 (*Assertion-guarded processes*). An assertion (Ψ) is said to be *guarded*, if it occurs as a sub-process within an input or output, and *unguarded* otherwise. A process is *assertion guarded* if all its assertions are guarded.

Definition 3 (*Well-formed processes*). Let G be an assertion-guarded process. A process P is *well-formed* if it satisfies the following:

- In every sub-process of P , every process in replication, case expressions and handles are assertion-guarded: $!G$, **case** $\tilde{\varphi} : \tilde{G}$ and $M \Leftarrow G$.
- In input prefixes $\underline{M}(\lambda\tilde{x})N$, $\tilde{x} \subseteq \text{n}(N)$ is a sequence without duplicates.
- If $M \Leftarrow P$ then $\text{n}(P) \subseteq \text{n}(M)$.

The third criterion states that if M is a handle for P , then M must contain *at least* all the names of P . This is imposed in [9] to ensure that a bound name cannot be sent out of its scope and become exposed by means of the invocation construct.

$$\begin{array}{c}
\text{[E-REP]} \frac{}{\Psi \triangleright ! P \ggg P \mid ! P} \quad \text{[E-STRUCT]} \frac{P \equiv P'}{\Psi \triangleright P \ggg P'} \quad \text{[E-RES]} \frac{\Psi \triangleright P \ggg P'}{\Psi \triangleright (\nu x : T) P \ggg (\nu x : T) P'} \quad (x \# \Psi) \\
\text{[E-CASE]} \frac{\Psi \Vdash \varphi_i}{\Psi \triangleright \text{case } \tilde{\varphi} : \tilde{P} \ggg P_i} \quad \text{[E-RUN]} \frac{\Psi \Vdash M \Leftarrow P}{\Psi \triangleright \text{run } M \ggg P} \quad \text{[E-PAR]} \frac{\Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \ggg P'_1}{\Psi \triangleright P_1 \mid P_2 \ggg P'_1 \mid P_2} \quad (\mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1)
\end{array}$$

Fig. 1. The rules defining the evaluation relation.

2.3. Reduction semantics

Previous presentations of Ψ -calculi [7–9] give the semantics in terms of a labelled transition system, and the type system by Hüttel [11] also used the labelled semantics. In this presentation, we shall instead give a reduction semantics for $\text{HO}\Psi$, to simplify some of the proofs for the type system.

There already exists a reduction semantics for the first-order Ψ -calculus, given by Åman Pohjola in [22], but this has not been extended to the higher-order Ψ -calculus, and also uses reduction contexts, rather than inference rules and structural congruence, to handle the unfolding of case expressions. In the Higher-Order Ψ -calculus, we also need to handle the unfolding of **run** M terms, but this seems difficult to achieve with reduction contexts, since the process P , for which M is a handle, does not have been explicitly mentioned. For example, it could be defined as part of the entailment relation that $M \Leftarrow P$. Thus, we shall instead give a more conventional reduction semantics, where processes may be rewritten to build redexes.

Firstly, we define a notion of structural congruence, $\equiv$, containing α -equivalence $\equiv_\alpha$, nil-absorption, the commutative monoidal rules for parallel composition, and the rule for scope extrusion:

Definition 4 (Structural congruence). Structural congruence is the least congruence on process terms containing the following rules:

$$\begin{array}{l}
\text{[S-ALPHA]} \quad P_1 \equiv_\alpha P_2 \implies P_1 \equiv P_2 \\
\text{[S-SCOPE]} \quad (\nu x : T) P_1 \mid P_2 \equiv (\nu x : T) (P_1 \mid P_2) \quad \text{if } x \# P_2 \\
\text{[S-RES}_1\text{]} \quad (\nu x : T) \mathbf{0} \equiv \mathbf{0} \\
\text{[S-RES}_2\text{]} \quad (\nu x_1 : T_1) (\nu x_2 : T_2) P \equiv (\nu x_2 : T_2) (\nu x_1 : T_1) P \\
\text{[S-IDENT]} \quad P \mid \mathbf{0} \equiv P \\
\text{[S-ASSOC]} \quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \\
\text{[S-COMM]} \quad P_1 \mid P_2 \equiv P_2 \mid P_1
\end{array}$$

Next we shall need the notion of a *frame* of a process. In Ψ -calculi, new assertions (Ψ) may appear in the syntax and therefore become enabled during the evolution of the program. These are collected by the functions, $\mathcal{F}_\Psi(P)$ and $\mathcal{F}_\nu(P)$, which extract respectively the free assertions and locally declared names from P (the latter of which may bind into the assertions). We call the pair $(\mathcal{F}_\nu(P), \mathcal{F}_\Psi(P))$ the *frame* of P .

Definition 5 (Frame of a process). The relevant clauses for $\mathcal{F}_\Psi(P)$ and $\mathcal{F}_\nu(P)$ are:

$$\begin{array}{l}
\mathcal{F}_\Psi(P_1 \mid P_2) \triangleq \mathcal{F}_\Psi(P_1) \otimes \mathcal{F}_\Psi(P_2) \quad \mathcal{F}_\nu(P_1 \mid P_2) \triangleq \mathcal{F}_\nu(P_1) \cup \mathcal{F}_\nu(P_2) \\
\mathcal{F}_\Psi((\nu x : T) P) \triangleq \mathcal{F}_\Psi(P) \quad \mathcal{F}_\nu((\nu x : T) P) \triangleq \{x\} \cup \mathcal{F}_\nu(P) \\
\mathcal{F}_\Psi(\langle \Psi \rangle) \triangleq \Psi
\end{array}$$

and with all remaining clauses of the forms $\mathcal{F}_\Psi(P) \triangleq \mathbf{1}$ and $\mathcal{F}_\nu(P) \triangleq \emptyset$ respectively.

Structural congruence will be used for rewriting terms, but as this relation is symmetric, we cannot use it to unfold case expressions and **run** M terms, since that would also allow a reverse reading of the rules. Instead, we introduce a parametrised, asymmetric *evaluation relation* $\cdot \triangleright \cdot \ggg \cdot$ to properly handle unfolding, which may depend on the assertions currently in effect. It is defined as the least preorder closed under the rules given in Fig. 1. It replaces the usual structural congruence rule in the reduction semantics to ensure that neither of these operations may be reversed by a reverse reading of the rules, whilst including $\equiv$ for the other kinds of process rewrites where symmetry is unproblematic.

Lastly, the *reduction relation* $\cdot \triangleright \cdot \rightarrow \cdot$ is given by the rules in Fig. 2. Reductions are thus on the form $\Psi \triangleright P \rightarrow P'$, i.e. relative to a global Ψ containing the assertions currently in effect.

The frame of a process is used in the **[R-PAR]** and **[E-PAR]** rules, where the free assertions (Ψ) occurring in the second component P_2 of a parallel composition are composed with the global Ψ , since they may affect the evaluation (resp. reduction) of the first component, P_1 . However, P_2 may also contain local, new names $(\nu x : T)$, which may bind into (Ψ),

$$\begin{array}{c}
\text{[R-COM]} \frac{\Psi \Vdash M_1 \leftrightarrow M_2}{\Psi \triangleright \overline{M_1} N[\tilde{x} := \tilde{K}].P_1 \mid \underline{M_2}(\lambda \tilde{x} : \tilde{T}).P_2 \rightarrow P_1 \mid P_2[\tilde{x} := \tilde{K}]} \\
\text{[R-EVAL]} \frac{\Psi \triangleright P_1 \ggg P'_1 \quad \Psi \triangleright P'_1 \rightarrow P_2}{\Psi \triangleright P_1 \rightarrow P_2} \\
\text{[R-RES]} \frac{\Psi \triangleright P \rightarrow P'}{\Psi \triangleright (\nu x : T) P \rightarrow (\nu x : T) P'} (x \# \Psi) \\
\text{[R-PAR]} \frac{\Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \rightarrow P'_1}{\Psi \triangleright P_1 \mid P_2 \rightarrow P'_1 \mid P_2} (\mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1)
\end{array}$$

Fig. 2. The rules defining the reduction relation.

and it must therefore be ensured that these are fresh w.r.t. both the free and locally declared names in P_1 , as well as any names found in the global Ψ . This is stated in the side conditions of these rules, where the local names are collected by $\mathcal{F}_\Psi(P_2)$.

The reduction relation defined above does not coincide exactly with the τ -labelled transition relation $\xrightarrow{\tau}$ from the original presentation in [9]. It is slightly larger, since, in the ‘unfolding rules’ [E-CASE], [E-REP] and [E-RUN], we allow the respective constructs to unfold to a process P , regardless of whether P itself can perform a reduction step or not. In contrast, the labelled semantics requires that a transition $P \xrightarrow{\alpha} P'$ can be concluded for the unfolded process. However, the following result can be shown:

Proposition 1. $\Psi \triangleright P \xrightarrow{\tau} P' \implies \Psi \triangleright P \rightarrow P'$ where $\xrightarrow{\tau}$ is the τ -labelled transition relation given in [9].

The proof in itself is not difficult: it proceeds by induction in the derivation of $\Psi \triangleright P \xrightarrow{\tau} P'$, where we show that for each possible transition of this form, we can derive a corresponding conclusion of the form $\Psi \triangleright P \rightarrow P'$ by using the rules of the reduction semantics. The main importance of this is that even though $\rightarrow$ only approximates the relation $\xrightarrow{\tau}$, it will suffice for the purpose of showing subject reduction.

3. The generic type system

As in other type systems, we need to describe when processes are well-typed, but since we in the HO Ψ -calculus can have arbitrary terms, conditions and assertions, we shall also need a way to decide when *they* are well-typed. As they are parameters to the HO Ψ -calculus, we cannot specify a set of type rules for them, as we can with processes. Instead, such rules must likewise be provided as parameters to create an instance of the generic type system. The parameters to be specified consist of *rules* for concluding type judgments of the form

$$\Gamma, \Psi \vdash M : T \quad \Gamma, \Psi \vdash \varphi \quad \Gamma, \Psi \vdash \Psi'$$

and two *predicates* $T_1 \dot{\leftrightarrow} T_2$ and $T \dot{\cap} \Gamma$. These rules must then satisfy a number of requirements, here denoted *instance assumptions*, which we shall need in the proof for subject reduction. These are described in the following sections.

3.1. Types and environments

In type systems for process calculi, types can contain names, as exemplified by the instances considered in the work on first-order typed psi-calculi[11]. We therefore assume that the set of types **Types** is a nominal datatype ranged over by T and that substitutions can affect types. Furthermore, we need the concept of a type environment Γ to record the types of free names; thus Γ is a partial function with finite support $\Gamma : \mathcal{N} \rightarrow \mathbf{Types}$. Sometimes it can be convenient to think of Γ as a set of tuples $\Gamma \subseteq \mathcal{N} \times \mathbf{Types}$ where $(x, T) \in \Gamma$ if $\Gamma(x) = T$. We write $\Gamma, x : T$ to denote the type environment Γ extended by the name x with type T .

As usual, our type judgments will be relative to a type environment Γ . However, due to the presence of assertions which may affect the well-typedness of a process, term, condition, or indeed an assertion, our type judgments must also be relative to a global assertion Ψ . Thus, type judgements for processes will be of the form $\Gamma, \Psi \vdash P$. As the global assertion may be composed with assertions appearing in a process, we shall therefore also need the notion of an extension ordering on assertions, and we also define a corresponding notion for type environments. To this end, we identify assertions up to commutativity and associativity of composition. The resulting notion of equivalence is denoted $=_\Psi$.

Definition 6 (Extension ordering). We say that $\Psi_1 \leq \Psi_2$ if $n(\Psi_1) \subseteq n(\Psi_2)$ and there exists a Ψ such that $\Psi_1 \otimes \Psi =_\Psi \Psi_2$. Likewise, we say that $\Gamma_1 \leq \Gamma_2$ if $\text{dom}(\Gamma_1) \subseteq \text{dom}(\Gamma_2)$ and there exists a Γ such that $\Gamma_1, \Gamma = \Gamma_2$.

The extension ordering is superficially similar to the notion of static implication, but differs from it as no notion of assertion entailment is involved. We deliberately use the notation $\Psi_1 \leq \Psi_2$ in order to be reminiscent of the extension ordering for type environments.

Lastly, the environments Γ, Ψ appearing in type judgments must be *well-formed*. As **Types** is a nominal datatype, names can also appear inside a type T , and we must therefore require that if Γ contains a type annotation $x : T$, with names with type annotations appearing inside T , then these must coincide with those found in Γ .

Example 1. Consider a size-dependent list type $\text{List}(x : \text{Int})$ where the size x has been annotated with a type. Then for a type environment $\Gamma = x : T_x, y : \text{List}(x : \text{Int})$ we require that $T_x : \text{Int}$.

Definition 7 (Well-formed environments). Let Γ_T denote the type annotations extracted from a type T and let $\text{an}(T)$ denote the set of annotated names occurring in T . We say the environments Γ, Ψ are *well-formed* if $\text{fn}(\Psi) \subseteq \text{dom}(\Gamma)$, and for all names $x \in \text{dom}(\Gamma)$ the following holds:

$$\Gamma(x) = T \implies \text{fn}(T) \subseteq \text{dom}(\Gamma)$$

$$\Gamma(x) = T \implies \forall y \in \text{an}(T). \Gamma_T(y) = \Gamma(y)$$

3.2. Channel compatibility

When we type an input or output prefix term, the type of the subject M and the type of the object (the term transmitted on channel M) must be compatible w.r.t. a *compatibility predicate* $\leftrightarrow$ that describes which types of values can be carried by channels of a given type. Thus, $T_1 \leftrightarrow T_2$ denotes that channels of type T_1 can carry terms of type T_2 , and we require that the set of types be defined such that this holds. Furthermore, we distinguish between output compatibility $\leftrightarrow^+$, and input compatibility $\leftrightarrow^-$, and we write $T_1 \leftrightarrow T_2$ if both $T_1 \leftrightarrow^+ T_2$ and $T_1 \leftrightarrow^- T_2$.

Example 2. Consider the channel types in the sorting system by Milner [1]. Here, a name has type $\text{ch}(T)$, if it is a channel that can be used to transmit names of type T , so in that case we would therefore require that $\leftrightarrow$ be defined such that $\text{ch}(T) \leftrightarrow T$.

In our definition of compatibility, we assume given a subtype ordering $\leq$ on types. If $T_1 \leq T_2$, a term of type T_1 can be used wherever a term of type T_2 is needed. Thus we require the usual subsumption rule for types, namely that a term of a given type T_1 can also be typed with a supertype T_2 :

$$[\text{T-SUBS}] \frac{\Gamma, \Psi \vdash M : T_1}{\Gamma, \Psi \vdash M : T_2} (T_1 \leq T_2)$$

The compatibility predicate for a type T must further satisfy the following requirements w.r.t. the subtyping relation:

1. If a channel type can carry two distinct types, then the types have to be related by the subtype ordering. That is, if $* \in \{+, -\}$, $T \leftrightarrow^* T_1$ and $T \leftrightarrow^* T_2$ with $T_1 \neq T_2$, then $T_1 \leq T_2$ or $T_2 \leq T_1$.
2. Output compatibility is *contravariant*. That is, if $T \leftrightarrow^+ T_2$ and $T_1 \leq T_2$, then also $T \leftrightarrow^+ T_1$. This requirement mirrors that of [10]. If $T_1 \leq T_2$, then a term of type T_1 can be used wherever a term of type T_2 is needed, and a channel that outputs terms of the more general type T_2 can therefore be used, wherever a channel of the specialized type T_1 is required.
3. Input compatibility is *covariant*. That is, if $T \leftrightarrow^- T_1$ and $T_1 \leq T_2$, then also $T \leftrightarrow^- T_2$. This requirement, too, mirrors that of [10]. Here, if $T_1 \leq T_2$, a channel that accepts terms of type T_1 can also be used to accept terms of type T_2 .

Like the other assumptions, the compatibility assumptions are needed to ensure soundness. Consider, for instance, the requirement that channels cannot carry values that are not related by subtyping. Here is an example that illustrates the need for this particular requirement:

Example 3. Assume that the constant $+$ has its type given by $+: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$, that $\text{Int} \not\leq \text{Bool}$ and $\text{Bool} \not\leq \text{Int}$. Finally, assume that $c : T$ where $T \leftrightarrow \text{Int}$ and $T \leftrightarrow \text{Bool}$. The process

$$\bar{c}17 \mid \bar{c}\text{true} \mid \underline{c}(\lambda x).\bar{c}x + x$$

can reduce to an ill-typed process, namely

$$\bar{c}17 \mid \bar{c}\text{true} + \text{true}$$

If we require that all types carried by a channel must be related by the subtype ordering, the problem disappears, as we can then type the subject x of the input prefix in the receiving subprocess with the least supertype of the message terms that can be bound to x .

3.3. Instance assumptions

The type rules for terms, assertions and conditions will depend on how these parameters are defined for a specific instance of the $\text{HO}\Psi$ -calculus, and they must therefore be provided as part of the instantiation of the generic type system.

However, like type judgments for processes, they must also be relative to a type environment Γ and a global Ψ , so we require that they be of the form $\Gamma, \Psi \vdash \mathcal{J}$, where $\mathcal{J}$ is defined by the formation rules:

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

We introduce a collection of assumptions that must hold for an instance of the generic type system to be valid. They pertain to the rules for type judgments $\Gamma, \Psi \vdash \mathcal{J}$ for terms, conditions and assertions, on which we must impose certain restrictions to allow us to prove subject reduction for the generic type system. Whether or not the following requirements are minimal is an open problem at the time of writing.

Definition 8 (*Qualified judgement*). We say a judgement $\Gamma, \Psi \vdash \mathcal{J}$ is *qualified*, if Γ, Ψ are well-formed, and if $\text{fn}(\mathcal{J}) \subseteq \text{dom}(\Gamma)$.

Type rules have zero or more *premises* and a *conclusion* that are qualified judgements of the form $\Gamma, \Psi \vdash \mathcal{J}$. They may also include a *side condition*, which is a predicate that is not a qualified type judgement, but which can depend on the judgements in the rule. Thus, they are of the following form:

$$[\text{NAME}] \frac{\Gamma, \Psi \vdash \mathcal{J}_1 \quad \dots \quad \Gamma, \Psi \vdash \mathcal{J}_n}{\Gamma, \Psi \vdash \mathcal{J}} (\text{PRED})$$

where PRED is a (possibly empty) predicate, and $\Gamma, \Psi \vdash \mathcal{J}_i$ is a (possibly empty) list of premises. We assume the type rules for terms, conditions and assertions are of this form.

Furthermore, we assume that the type rules for terms, conditions and assertions are defined such that the usual properties of weakening and strengthening of the type environment hold for all type judgements $\mathcal{J}$. These properties are summarised in the following definition:

Definition 9 (*Weakening and strengthening*). Judgements of the form $\Gamma, \Psi \vdash \mathcal{J}$ satisfy

1. The *environment weakening property*, if whenever $\Gamma, \Psi \vdash \mathcal{J}$, then for any $x \notin \text{dom } \Gamma$ and type T we have $\Gamma, x : T, \Psi \vdash \mathcal{J}$
2. The *environment strengthening property*, if whenever $\Gamma, x : T, \Psi \vdash \mathcal{J}$ and $x \# \mathcal{J}$ we have $\Gamma, \Psi \vdash \mathcal{J}$.
3. The *assertion weakening property*, if whenever $\Gamma, \Psi \vdash \mathcal{J}$ and $\Psi \leq \Psi'$ and $\text{n}(\Psi') \subseteq \text{dom}(\Gamma)$, then also $\Gamma, \Psi' \vdash \mathcal{J}$.

The weakening assumptions in Definition 9 tell us that judgements are *monotonic* w.r.t. the extension ordering: If $\Gamma, \Psi \vdash \mathcal{J}$ and $\Gamma \leq \Gamma_1$ then also $\Gamma_1, \Psi \vdash \mathcal{J}$ and if $\Psi \leq \Psi_1$ then also $\Gamma, \Psi_1 \vdash \mathcal{J}$. Likewise, the strengthening assumption tells us that the extension ordering $\leq$ for type environments is *well-founded*: If $\Gamma, \Psi \vdash \mathcal{J}$ then there is a least $\Gamma_0 \leq \Gamma$ such that $\Gamma_0, \Psi \vdash \mathcal{J}$. We do not need to require a similar well-foundedness property to hold for the assertion environment, but it would also be quite natural to assume that there exists a least $\Psi_0 \leq \Psi$ such that $\Gamma, \Psi_0 \vdash \mathcal{J}$.

Weakening and strengthening properties tell us that we can introduce additional hypotheses and remove superfluous hypotheses and preserve typability. In type systems for process calculi that have a notion of communication between parallel components P and Q as well as the notion of restriction known from the π -calculus, properties of strengthening and weakening are necessary for soundness because of scope extrusion. In this setting, the need for these properties mirrors the need for open/close rules in a labelled semantics for name-passing calculi [23].

Example 4. Consider a process $((\nu x)P) \mid Q$ where the bound name x is sent by P and received by Q . Weakening is necessary here if we are to establish that P and Q can be typed within the same context Γ , which includes x , and strengthening is needed in order to deal with the fact that the resulting process $(\nu x)(P \mid Q)$ must be typable in the same Γ , which assumes that the x that was local to one component is still not a free name.

Next, we must require type judgments for terms to respect channel equivalence w.r.t. the type annotations in Γ :

$$[\text{T-EQUAL}] \frac{\Gamma, \Psi \vdash M : T}{\Gamma, \Psi \vdash N : T} \left(\Psi \Vdash M \dot{\leftrightarrow} N \right)$$

Thus, if two terms are channel equivalent, then they must also have the same type.

Another group of assumptions concern *compositionality*: for each of the three nominal datatypes of terms, conditions and assertions, we require that there is a type rule, that allows us to type a composite element by typing each of its constituents. These assumptions are similar to those found in [11]. For assertions, the rule must be of the following form:

$$[\text{T-COMP-ASS}] \frac{\Gamma, \Psi \vdash \Psi_1 \quad \Gamma, \Psi \vdash \Psi_2}{\Gamma, \Psi \vdash \Psi_1 \otimes \Psi_2}$$

For conditions and terms, we make use of the assumption that these nominal datatypes must be describable as term algebras with constructor symbols f . If a type rule types a composite condition $f(\varphi_1, \dots, \varphi_n)$, it must be of the following form:

$$[\text{T-COMP-COND}] \frac{\Gamma, \Psi \vdash \varphi_i \quad 1 \leq i \leq n}{\Gamma, \Psi \vdash f(\varphi_1, \dots, \varphi_n)}$$

Likewise for terms, the rule must be of the form

$$[\text{T-COMP-TERM}] \frac{\Gamma, \Psi \vdash M_i : T_i \quad 1 \leq i \leq n}{\Gamma, \Psi \vdash f(M_1, \dots, M_n) : F(T_1, \dots, T_n)}$$

where $F(T_1, \dots, T_n)$ is the type of the composite term and F is a function over types. This assumption will guarantee that the type of a composite term $f(M_1, \dots, M_n)$ can be determined from the types T_i (for $1 \leq i \leq n$) using F .

3.4. Higher-order assumptions

In the higher-order Ψ -calculus, terms M serve a dual purpose: They can be used as simple terms or as handles, that is, terms for which the **run** M primitive can be used to obtain a process P . As an example, in the ρ -calculus a name can be used both as the name of a channel and as the handle of a process.

Processes do not have types in our type system; only terms do. However, we can view the ‘type’ of a process P as a type environment Γ that make P well-typed. Thus we require that the type T of a term M , and the ‘type’ (environment) of a process P must be compatible w.r.t. a *higher-order compatibility predicate* $\curvearrowright$, if M is a handle for P . Thus $T \curvearrowright \Gamma$ denotes that terms of type T can be used to obtain processes that are well-typed w.r.t. Γ , if the term is executed with the invocation construct. $\curvearrowright$ must therefore be defined such that the following holds:

$$[\text{T-HANDLE}] \quad \Gamma, \Psi \vdash M : T \wedge \Psi \Vdash M \Leftarrow P \implies \exists \Gamma'. T \curvearrowright \Gamma' \wedge \Gamma', \Psi \vdash P$$

This requirement is necessitated by the formulation of the $\text{HO}\Psi$ -calculus itself. The $\text{HO}\Psi$ -calculus does not require that a process P , for which M is a handle, must be defined within the syntax of a program where M may be invoked. It is entirely possible to define it directly as part of the entailment relation.

Example 5. Assume for some $\text{HO}\Psi$ -instance that entailment contains the following definition:

$$\Vdash \triangleq \dots \cup \{ (\mathbf{1}, M \Leftarrow \mathbf{0}) \mid M \in \mathbb{T} \}$$

This would make *every* term M be a handle for the $\mathbf{0}$ process, which would be entailed by the unit assertion $\mathbf{1}$. Because of the weakening requirement of Definition 1 (Valid parameters, requirement 6), handles are not allowed to be altered by composing new assertions onto this minimal assertion environment; hence *no* other handles can be declared by this setting.

In the above example, $\mathbf{0}$ was chosen for simplicity, but indeed *any* assertion-guarded process P could have been used. As our type system is syntactic (i.e. it analyses the syntax of programs), we have no means to check that such processes are well-typed, and we therefore have no other option than to simply require them to be well-typed. If handles are defined in this way, well-typedness of the handled process must therefore be shown manually.

Usually, however, handles would be defined in assertions appearing in the syntax of a program, e.g. as $\langle \{ M \Leftarrow P \} \rangle$, and with the definition of entailment containing e.g. the following rule:

$$[\text{H-ENTAIL}] \frac{\{ M \Leftarrow P \} \in \Psi}{\Psi \Vdash M \Leftarrow P}$$

If that is the case, then $[\text{T-HANDLE}]$ can be satisfied by including a type rule for typing handles in the type rules for assertions:

$$[\text{T-ASS-HANDLE}] \frac{\Gamma, \Psi \vdash M : T \quad \Gamma', \Psi \vdash P}{\Gamma, \Psi \vdash \{ M \Leftarrow P \}} \left(\begin{array}{l} T \curvearrowright \Gamma' \\ \Gamma' \leq \Gamma \end{array} \right)$$

Example 6. A simple way to define types of handles, such that $\curvearrowright$ holds, is to let **Types** be defined as

$$T \in \mathbf{Types} ::= \dots \mid (T, \Gamma)$$

and then let $\curvearrowright$ be defined such that $(T, \Gamma) \curvearrowright \Gamma$. Indeed, we shall do that in our type system instances in Section 5.

$$\begin{array}{c}
\text{[T-CASE]} \frac{\Gamma, \Psi \vdash \varphi_i \quad \Gamma, \Psi \vdash P_i}{\Gamma, \Psi \vdash \text{case } \tilde{\varphi} : \tilde{P}} \quad \text{[T-REP]} \frac{\Gamma, \Psi \vdash P}{\Gamma, \Psi \vdash !P} \quad \text{[T-IN]} \frac{\Gamma, \Psi \vdash M : T \quad \Gamma, \tilde{x} : \tilde{T}, \Psi \vdash N : T' \quad \Gamma, \tilde{x} : \tilde{T}, \Psi \vdash P}{\Gamma, \Psi \vdash \underline{M}(\lambda \tilde{x} : \tilde{T})N.P} (T \leftarrow P^- T') \\
\text{[T-RES]} \frac{\Gamma, x : T, \Psi \vdash P}{\Gamma, \Psi \vdash (\nu x : T)P} (x \# \Psi) \quad \text{[T-ASS]} \frac{\Gamma, \Psi \vdash \Psi'}{\Gamma, \Psi \vdash \langle \Psi' \rangle} \quad \text{[T-OUT]} \frac{\Gamma, \Psi \vdash M : T \quad \Gamma, \Psi \vdash N : T' \quad \Gamma, \Psi \vdash P}{\Gamma, \Psi \vdash \overline{M}N.P} (T \leftarrow P^+ T') \\
\text{[T-RUN]} \frac{\Gamma, \Psi \vdash M : T}{\Gamma, \Psi \vdash \text{run } M} \left(\frac{T \curvearrowright \Gamma'}{\Gamma' \leq \Gamma} \right) \quad \text{[T-NIL]} \frac{}{\Gamma, \Psi \vdash \mathbf{0}} \quad \text{[T-PAR]} \frac{\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1 \quad \Gamma, \mathcal{F}_\Gamma(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2}{\Gamma, \Psi \vdash P_1 \mid P_2} \left(\frac{\mathcal{F}_\Psi(P_1) \# \Psi, \mathcal{F}_\Psi(P_2), P_2}{\mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1} \right)
\end{array}$$

Fig. 3. Type judgements for processes.

We shall also need to impose a requirement on the definition of $\curvearrowright$, when handles are subjected to a term substitution σ :

$$\begin{array}{c}
\Gamma, \Psi \vdash M : T_1 \\
\wedge \Gamma, \Psi \vdash M\sigma : T_2 \\
\wedge T_1 \curvearrowright \Gamma_1 \\
\implies \exists \Gamma_2. T_2 \curvearrowright \Gamma_2 \wedge \Gamma_2 \leq \Gamma_1
\end{array}$$

This requirement essentially states that if M has a higher-order type T_1 , i.e. such that $T_1 \curvearrowright \Gamma_1$, and M is subjected to a term substitution σ , such that the resulting term $M\sigma$ has type T_2 , then T_2 too must be a higher-order type. Moreover, the Γ_2 we obtain from the type of $M\sigma$ by means of $T_2 \curvearrowright \Gamma_2$ must be such that $\Gamma_2 \leq \Gamma_1$; i.e. it can *at most* contain the same names as Γ_1 . In other words, a substitution is not allowed to alter the fact that a term has a higher-order type; and the new type must not allow the substituted term to be used as handle for a process containing more free names than the type of the original term would allow, prior to substitution. This is easily satisfied by any ordinary definition of substitution.

Example 7. Consider a $\text{HO}\Psi$ -instance where terms are just atomic names: $\mathcal{T} \triangleq \mathcal{N}$, and with simple channel types as in Example 2. Assume that $\Gamma, \Psi \vdash x : \text{ch}(T)$ and $\Gamma, \Psi \vdash z : T$, where $T = (T', \Gamma')$ for some T', Γ' , and with $\curvearrowright$ defined such that $(T', \Gamma') \curvearrowright \Gamma'$. Thus z has a higher-order type, and it may therefore be used as a handle, so we want to be able to pass it around.

Consider now the reduction:

$$x(\lambda y : T)y.\text{run } y \mid \bar{x}z.\mathbf{0} \rightarrow \text{run } y[y := z]$$

[T-SUBS-HANDLE] amounts to saying that $\Gamma, \Psi \vdash y[y := z] : T_1$ and $T_1 \curvearrowright \Gamma''$ such that $\Gamma'' \leq \Gamma'$, which indeed holds, since $T = T_1$ and therefore $\Gamma' = \Gamma''$.

3.5. Type rules for processes

Unlike the type rules for terms, conditions and assertions, the type rules for processes are common to every instance. These are of the form $\Gamma, \Psi \vdash P$. As before, we only consider qualified judgements; i.e. where Γ, Ψ are well-formed and $\text{fn}(P) \subseteq \text{dom}(\Gamma)$, so every name mentioned in the process in the judgement is bound in the type environment.

We shall need a way to extract the type annotations of scoped names from a process P . This will be needed in the rule for typing a parallel composition $P_1 \mid P_2$, since the free assertions in P_2 must be composed onto the global assertion environment for the typing of P_1 , and vice versa. Such a free assertion $\langle \Psi \rangle$ may, however, be declared inside the scope of a local name $(\nu x : T)$, with x binding into $\langle \Psi \rangle$, so the type information must therefore be added to the type environment Γ . Thus we shall write $\mathcal{F}_\Gamma(\cdot)$ to mean $\mathcal{F}_\Gamma((\nu x : T)P) \triangleq x : T, \mathcal{F}_\Gamma(P)$, and for all other clauses this function is defined similar to $\mathcal{F}_\Psi(\cdot)$.

The type rules for processes are given in Fig. 3; they are mostly similar to those of [11], except for the rule [T-RUN] used to type the **run** M construct, which is the only construct that is new to the higher-order setting. Here we require that M must have a higher-order type, containing a Γ matching the Γ it is typed relative to, which ensures that no new names are introduced by running the process associated with the handle M .

In the rule [T-PAR] we require that for a parallel composition $P_1 \mid P_2$ to be typable, P_1 and P_2 must both be typable within type environments and assertions that add information extracted from the other component. This is a natural requirement, since P_1 can, among other things, mention handles for processes established in P_2 , and vice versa. The side condition then asserts that all new names declared in P_1 , using the $(\nu x : T)$ construct, must be fresh for Ψ and both the free and new names occurring in P_2 , and vice versa for P_2 , similar to the side conditions for the [E-PAR] and [R-PAR] rules in the semantics.

4. Properties of the generic type system

Type systems normally ensure two properties of well-typed programs: a *subject reduction* property guarantees that a well-typed program remains well-typed under reduction; and a *safety* property ensures that if a program is well-typed then

a certain safety predicate holds. The latter may depend on the particular instance of the type system and must therefore be shown individually, for each instance, but subject reduction can be shown for the generic type system.

We establish these properties through a series of lemmas, building up to the main result. Here we shall see the various instance assumptions and parameter requirements (distributivity and weakening) from section 3.3, come into play. We begin with the usual results of weakening and strengthening of the type environment Γ :

Lemma 1 (Weakening). *If $\Gamma, \Psi \vdash P$ and $x \notin \text{dom}(\Gamma)$ then $\Gamma, x : T, \Psi \vdash P$.*

Proof. By induction in the rules of the type judgment $\Gamma, \Psi \vdash P$.

The case for **0** is immediate, and the cases for $P_1 \mid P_2$, $(\nu x : T) P$ and $!P$ follow from straightforward application of the induction hypothesis. The remaining cases contain either terms M , conditions φ , or assertions (Ψ) , so here the environment weakening assumption (Definition 9, item 1) is required.

Consider the case for output: By **[T-OUT]**, we know that $\Gamma, \Psi \vdash \overline{MN}.P$, and from the premises that $\Gamma, \Psi \vdash M : T_1$, $\Gamma, \Psi \vdash N : T_2$ and $\Gamma, \Psi \vdash P$. Then $\Gamma, x : T, \Psi \vdash P$ holds by induction hypothesis, and $\Gamma, x : T, \Psi \vdash M : T_1$ and $\Gamma, x : T, \Psi \vdash N : T_2$ both hold by the environment weakening assumption (Definition 9, item 1). Thus we can conclude $\Gamma, x : T, \Psi \vdash \overline{MN}.P$ by **[T-OUT]**.

The cases for **[T-IN]** and **[T-RUN]** are similar. In **[T-CASE]**, the premise (for each φ_i in $\tilde{\varphi}$) that $\Gamma, x : T, \Psi \vdash \varphi_i$ is also satisfied because of the environment weakening assumption (Definition 9, item 1), and likewise for the premise $\Gamma, x : T, \Psi \vdash \Psi'$ in **[T-ASS]**. $\square$

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

Proof. By induction in the rules of the type judgment $\Gamma, \Psi \vdash P$.

This proof proceeds in the same way as the proof for Lemma 1 (Weakening), but this time using the environment strengthening assumption (Definition 9, item 2) for the cases where terms, conditions or assertions appear in the premise. $\square$

We shall also need a weakening result for the global assertion environment Ψ . This result is necessitated by the syntax of the $\text{HO}\Psi$ -calculus itself, which allows guarded assertions in continuations to become unguarded after a reduction, and hence they may become composed onto the global Ψ . The result we wish to have is thus that any well-typed process remains well-typed after a composition of any assertion in the assertion environment, as long as all names in the new assertion environment are in the support of the type environment:

Lemma 3 (Assertion environment weakening). *If $\Gamma, \Psi \vdash P$ and $\text{n}(\Psi') \subseteq \text{dom}(\Gamma)$ and $\Psi \leq \Psi'$ then $\Gamma, \Psi' \vdash P$.*

Proof. By induction in the rules of the type judgment $\Gamma, \Psi \vdash P$.

- For **[T-NIL]**, **[T-IN]**, **[T-OUT]**, **[T-CASE]**, **[T-REP]**, **[T-RUN]** and **[T-ASS]** the proof proceeds in the same way as the proof for Lemma 1 (Weakening), but this time using the assertion weakening assumption (Definition 9, item 3) for the cases where terms, conditions or assertions appear in the premise.
- In the cases of **[T-RES]** and **[T-PAR]** we must ensure that the freshness requirements in the side conditions are satisfied. Thus, consider the case for **[T-PAR]**, since the case for **[T-RES]** is simpler: We know that $\Gamma, \Psi \vdash P_1 \mid P_2$, and from the premise that $\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1$, and from the side condition that $\mathcal{F}_\nu(P_2) \# \Psi, \mathcal{F}_\nu(P_1), P_1$ (and conversely for P_2). We then use α -conversion as necessary to ensure that the condition $\mathcal{F}_\nu(P_2) \# \Psi'$ holds, such that $\mathcal{F}_\Gamma(P_2)$ will not capture any free names in Ψ' that are not in Ψ . We then do the same for the other premise, and then they both hold by induction hypothesis. Then we conclude $\Gamma, \Psi' \vdash P_1 \mid P_2$ by **[T-PAR]**. $\square$

Next, we need a result of *substitution*, which must be shown for both the ‘parameter’ type judgements $\Gamma, \Psi \vdash \mathcal{J}$ and type judgments for processes $\Gamma, \Psi \vdash P$:

Lemma 4 (Substitution). *Let $\mathcal{G} ::= \mathcal{J} \mid P$, and let σ be a term substitution. If $\Gamma, \Psi \vdash \mathcal{G}$, $\text{dom}(\Gamma) = \text{dom}(\sigma)$, and for all $x \in \text{dom}(\sigma)$ it holds that $\Gamma, \Psi \vdash \sigma(x) : \Gamma(x)$, then $\Gamma, \Psi \vdash \mathcal{G}\sigma$.*

Proof. There are four different kinds of type judgments to consider, since $\mathcal{G}$ can be a term, a condition, an assertion or a process:

If the judgment is of the form $\Gamma, \Psi \vdash M : T$, we proceed by induction in the structure of M . If M is a name, the result is immediate. Else, M must be a composite term with n immediate constituents; i.e. $M = f(M_1, \dots, M_n)$. Thus, this judgment must have been concluded by the (assumed) compositionality rule for terms, **[T-COMP-TERM]**, so we must have that

$$\frac{\Gamma, \Psi \vdash M_i : T_i \quad 1 \leq i \leq n}{\Gamma, \Psi \vdash f(M_1, \dots, M_n) : F(T_1, \dots, T_n)}$$

By induction hypothesis, we have that $\Gamma, \Psi \vdash M_i \sigma : T_i \sigma$ for all the constituents, and by [T-COMP-TERM] we can conclude that

$$\Gamma, \Psi \vdash f(M_1 \sigma, \dots, M_n \sigma) : F(T_1 \sigma, \dots, T_n \sigma)$$

By the distributivity assumption, we have that

$$f(M_1 \sigma, \dots, M_n \sigma) : F(T_1 \sigma, \dots, T_n \sigma) = f(M_1, \dots, M_n) \sigma : F(T_1, \dots, T_n) \sigma$$

which thus lets us conclude $\Gamma, \Psi \vdash f(M_1, \dots, M_n) \sigma : F(T_1, \dots, T_n) \sigma$ as desired.

If the judgment is of the form $\Gamma, \Psi \vdash \varphi$ or $\Gamma, \Psi \vdash \Psi'$, the same reasoning applies, using the respective compositionality assumption [T-COMP-COND] or [T-COMP-ASS].

If the judgment is of the form $\Gamma, \Psi \vdash P$, we proceed by induction in the structure of P :

- For **0**, the result is immediate. For assertions (Ψ) , it follows from the result above, and likewise for case constructs **case** $\tilde{\varphi} : \tilde{P}$, with $\Gamma, \Psi \vdash P_i \sigma$ from the induction hypothesis. For replication $!P$, it follows directly from the induction hypothesis, and also for $(\nu x : T) P$, since we know by requirement that $x \notin \text{dom}(\Gamma)$ and $\text{dom}(\Gamma) = \text{dom}(\sigma)$; hence σ cannot affect the name x .
- For input and output, $\underline{M}(\lambda \tilde{x} : \tilde{T}) N.P$ and $\overline{M} N.P$, the result follows from the induction hypothesis for the continuation P , and from the above result regarding substitution on terms for the subject and object M, N , where again we know that the substitution cannot affect any of the bound names $\tilde{x}$.
- For parallel composition we have that $\Gamma, \Psi \vdash P_1 \mid P_2$, and we wish to conclude that $\Gamma, \Psi \vdash (P_1 \mid P_2) \sigma$. From the premise we know that $\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1$, and conversely for P_2 . Applying the substitution gives us, that we must be able to conclude

$$\Gamma, \mathcal{F}_\Gamma(P_2 \sigma), \Psi \otimes \mathcal{F}_\Psi(P_2 \sigma) \vdash P_1 \sigma$$

The substitution cannot introduce new, bound names, which would be collected by $\mathcal{F}_\Gamma(\cdot)$, hence $\mathcal{F}_\Gamma(P_2) = \mathcal{F}_\Gamma(P_2 \sigma)$. Now since we know that $\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1$, then by Lemma 3 we can conclude that $\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2 \sigma) \vdash P_1$. We then apply the induction hypothesis and obtain

$$\Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2 \sigma) \vdash P_1 \sigma$$

A similar reasoning then applies for P_2 , which allows us to conclude.

- Finally for **run** M , we know that $\Gamma, \Psi \vdash \text{run } M$, which must have been concluded by [T-RUN], and we wish to conclude $\Gamma, \Psi \vdash \text{run } M \sigma$. From the premise and side condition we have that $\Gamma, \Psi \vdash M : T$ and $T \frown \Gamma'$ and $\Gamma' \leq \Gamma$. By the case for terms above, we therefore have that $\Gamma, \Psi \vdash M \sigma : T \sigma$. Then by requirement [T-SUBS-HANDLE] we get that $T \sigma \frown \Gamma''$ and $\Gamma'' \leq \Gamma'$, and by transitivity of $\leq$ therefore also $\Gamma'' \leq \Gamma$. We can then conclude $\Gamma, \Psi \vdash \text{run } M \sigma$ by [T-RUN]. $\square$

As we here use reduction semantics with an asymmetric evaluation relation to handle unfolding of **case** and **run** expressions, we shall also need two results that describe how frames can evolve during evaluation. The first result establishes that the property of being assertion-guarded is preserved by the evaluation relation $\ggg$.

Lemma 5 (Frame post evaluation). *If $\Psi \triangleright P \ggg P'$ and $\mathcal{F}_\Psi(P) \# \Psi$ and $\mathcal{F}_\Psi(P') \# \Psi$, then $\Psi \otimes \mathcal{F}_\Psi(P) =_\Psi \Psi \otimes \mathcal{F}_\Psi(P')$.*

Proof. The proof proceeds by induction in the rules for concluding the evaluation $\Psi \triangleright P \ggg P'$ (Fig. 1).

- The cases for [E-RES] and [E-PAR] are straightforward by application of the induction hypothesis.
- For [E-STRUCT] we need an extra induction in the rules of structural congruence (Definition 4), but it is immediately clear from the definition that a rewrite by structural congruence cannot expose any new assertions which were not free before.
- For the cases of [E-REP], [E-CASE] and [E-RUN], we know from the well-formedness criterion for processes that neither of these processes may contain unguarded assertions. Thus, unfolding $!P$ or **case** $\tilde{\varphi} : \tilde{P}$ or **run** M cannot expose any new assertions. $\square$

The second result states that the free assertions in a process at most can increase after a reduction:

Lemma 6 (Frame post reduction). *If $\Psi \triangleright P \rightarrow P'$ then $\mathcal{F}_\Psi(P) \leq \mathcal{F}_\Psi(P')$.*

Proof. By induction in the rules for concluding the reduction $\Psi \triangleright P \rightarrow P'$ (Fig. 2).

- Case [R-PAR] and [R-RES]: by the induction hypothesis.

- Case [R-COM]: Here the initial assertions are $\mathbf{1}$ (the unit assertion), since neither of the processes contain free assertions. Then after the reduction, we have $\mathbf{1} \leq \mathcal{F}_\Psi(P_1 \mid P_2[\tilde{x} := \tilde{K}])$, which is immediately seen to hold.
- Case [R-EVAL]: We have $\Psi \triangleright P_1 \rightarrow P_2$, and from the premise that $\Psi \triangleright P_1 \ggg P'_1$ and $\Psi \triangleright P'_1 \rightarrow P_2$. By induction hypothesis $\mathcal{F}_\Psi(P'_1) \leq \mathcal{F}_\Psi(P_2)$, and by Lemma 5 we have that $\mathcal{F}_\Psi(P_1) =_\Psi \mathcal{F}_\Psi(P'_1)$. Thus we conclude that $\mathcal{F}_\Psi(P_1) \leq \mathcal{F}_\Psi(P_2)$. $\square$

The above lemmas can now be used to prove the subject reduction result, which states that well-typed processes remain well-typed after reduction. We shall break it into three parts and first show that well-typedness is preserved by structural congruence, the evaluation relation, and lastly the reduction relation:

Lemma 7 (Structural congruence). *If $\Gamma, \Psi \vdash P_1$ and $P_1 \equiv P_2$ then $\Gamma, \Psi \vdash P_2$.*

Proof. By induction in the rules for concluding $P_1 \equiv P_2$ (Definition 4).

- The cases for rules for congruence are straightforward by induction hypothesis.
- The case for [S-ALPHA] is also straightforward, since we only change bound names, and none of these are found in Γ .
- The cases for the monoidal rules [S-IDENT], [S-ASSOC] and [S-COMM] simply require applying the rule [T-PAR] in different order, or applying it an extra time.
- Case [S-RES₁]: For the forward direction, we know $\Gamma, \Psi \vdash (\nu x : T) \mathbf{0}$ by [T-RES], and $\Gamma, x : T, \Psi \vdash \mathbf{0}$ holds by the premise. By Lemma 1 (Weakening) therefore also $\Gamma, \Psi \vdash \mathbf{0}$. For the other direction, we weaken Γ first and then conclude by rule [T-RES].
- Case [S-RES₂]: By twice application of [T-RES].
- Case [S-SCOPE]: For the forward direction, we know that

$$\Gamma, \Psi \vdash (\nu x : T) P_1 \mid P_2$$

and $x \# P_2$. This judgement must have been concluded by [T-PAR] with premises

$$\Gamma, \mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash (\nu x : T) P_1$$

$$\Gamma, \mathcal{F}_T(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2$$

The premise $\Gamma, \mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash (\nu x : T) P_1$ was concluded using [T-RES] with

$$\Gamma, x : T, \Psi \otimes \mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1$$

as premise. Since $x \# P_2$, we can then also conclude

$$\Gamma, x : T, \Psi \vdash P_2$$

by Lemma 1 (Weakening). By [T-PAR] we can then conclude

$$\Gamma, x : T, \Psi \vdash P_1 \mid P_2$$

and then by [T-RES] we conclude

$$\Gamma, \Psi \vdash (\nu x : T) (P_1 \mid P_2)$$

For the other direction, apply Lemma 2 to conclude $\Gamma, \Psi \vdash P_2$ from $\Gamma, x : T, \Psi \vdash P_2$, and then [T-RES] for P_1 and conclude by [T-PAR]. $\square$

Lemma 8 (Subject evaluation). *If $\Gamma, \Psi \vdash P \wedge \Psi \triangleright P \ggg P'$ then $\Gamma, \Psi \vdash P'$.*

Proof. By induction in the rules for concluding $\Psi \triangleright P \ggg P'$ (Fig. 1).

- Case [E-RES]: By [T-RES] and the induction hypothesis.
- Case [E-STRUCT]: By Lemma 7.
- Case [E-REP]: We know that $\Gamma, \Psi \vdash !P$, which was concluded by [T-REP]. From the premise we get $\Gamma, \Psi \vdash P$, and by well-formedness requirement (Definition 3) we know that P cannot contain free assertions. Thus we can conclude $\Gamma, \Psi \vdash P \mid !P$ by [T-PAR].
- Case [E-CASE]: Straightforward from the premise of [T-CASE].

- Case **[E-PAR]**: We know that $\Psi \triangleright P_1 \mid P_2 \ggg P'_1 \mid P_2$. From the premise and side condition, we have that

$$\begin{aligned} \Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \ggg P'_1 \\ \mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1 \end{aligned}$$

We also know that $\Gamma, \Psi \vdash P_1 \mid P_2$, which was concluded by **[T-PAR]**. From the premise and side condition, we have that

$$\begin{aligned} \Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1 \\ \Gamma, \mathcal{F}_\Gamma(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2 \\ \mathcal{F}_\Psi(P_1) \# \Psi, \mathcal{F}_\Psi(P_2), P_2 \\ \mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1 \end{aligned}$$

By the induction hypothesis, the statement holds for P_1 , i.e.:

$$\Gamma, \Psi \vdash P_1 \wedge \Psi \triangleright P_1 \ggg P'_1 \implies \Gamma, \Psi \vdash P'_1$$

Combining the above with Lemma 1 (weakening) and Lemma 3 (assertion environment weakening), we can then conclude that

$$\begin{aligned} \Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1 \\ \wedge \Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \ggg P'_1 \\ \implies \Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P'_1 \end{aligned}$$

This satisfies the first premise of **[T-PAR]**. We must then show that $\Gamma, \mathcal{F}_\Gamma(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ also holds.

By Lemma 5 we know that $\mathcal{F}_\Psi(P_1) =_\Psi \mathcal{F}_\Psi(P'_1)$, since the evaluation cannot expose any new assertions. We therefore immediately have that $\Gamma, \mathcal{F}_\Gamma(P_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$. We now have three cases to consider:

1. If $\mathcal{F}_\Gamma(P_1) =_\Psi \mathcal{F}_\Gamma(P'_1)$ then $\Gamma, \mathcal{F}_\Gamma(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ follows from the definition of $=_\Psi$ and associativity and commutativity of $\otimes$.
2. If $\mathcal{F}_\Gamma(P_1) \leq \mathcal{F}_\Gamma(P'_1)$, then we use Lemma 1 (weakening) to conclude that $\Gamma, \mathcal{F}_\Gamma(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ holds.
3. Otherwise, if $\mathcal{F}_\Gamma(P_1) \not\leq \mathcal{F}_\Gamma(P'_1)$, we instead use Lemma 2 (strengthening).

If new bound names are exposed in P'_1 , such that they would be collected by $\mathcal{F}_\Gamma(P'_1)$ and $\mathcal{F}_\Psi(P'_1)$, we use α -conversion such that

$$\begin{aligned} \mathcal{F}_\Psi(P'_1) \# \Psi, \mathcal{F}_\Psi(P_2), P_2 \\ \mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P'_1), P_1 \end{aligned}$$

both hold. As both premises and side conditions now are satisfied, we can conclude $\Gamma, \Psi \vdash P'_1 \mid P_2$ by **[T-PAR]**.

- Case **[E-RUN]**: We know that $\Psi \triangleright \text{run } M \ggg P$ and $\Psi \Vdash M \Leftarrow P$, and we wish to conclude $\Gamma, \Psi \vdash P$. $\Gamma, \Psi \vdash \text{run } M$ was concluded by **[T-RUN]** with the premise $\Gamma, \Psi \vdash M : T$ and side condition $T \frown \Gamma'$ and $\Gamma' \leq \Gamma$. By requirement **[T-HANDLE]** we get that $\Gamma', \Psi \vdash P$. We can then conclude $\Gamma, \Psi \vdash P$ by Lemma 1 (Weakening). $\square$

Theorem 1 (Subject reduction). *If $\Gamma, \Psi \vdash P$ and $\Psi \triangleright P \rightarrow P'$ then $\Gamma, \Psi \vdash P'$.*

Proof. By induction in the rules for concluding $\Psi \triangleright P \rightarrow P'$ (Fig. 2).

- Case **[R-EVAL]**: By Lemma 8 (Subject evaluation) and the induction hypothesis.
- Case **[R-RES]**: By **[T-RES]** and the induction hypothesis.
- Case **[R-PAR]**: We know that $\Gamma, \Psi \vdash P_1 \mid P_2$, which was concluded by **[T-PAR]**, and $\Psi \triangleright P_1 \mid P_2 \rightarrow P'_1 \mid P_2$ which was concluded by **[R-PAR]**. Our goal is to show that $\Gamma, \Psi \vdash P'_1 \mid P_2$. From the premises and side condition of **[T-PAR]** we have

$$\begin{aligned} \Gamma, \mathcal{F}_\Gamma(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1 \\ \Gamma, \mathcal{F}_\Gamma(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2 \\ \mathcal{F}_\Psi(P_1) \# \Psi, \mathcal{F}_\Psi(P_2), P_2 \\ \mathcal{F}_\Psi(P_2) \# \Psi, \mathcal{F}_\Psi(P_1), P_1 \end{aligned}$$

and from the premises and side conditions of **[R-PAR]** we have:

$$\begin{aligned} & \mathcal{F}_v(P_2) \# \Psi, \mathcal{F}_v(P_1), P_1 \\ & \Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \rightarrow P'_1 \end{aligned}$$

By the induction hypothesis, the statement holds for P_1 , i.e.:

$$\Gamma, \Psi \vdash P_1 \wedge \Psi \triangleright P_1 \rightarrow P'_1 \implies \Gamma, \Psi \vdash P'_1$$

By the above, and Lemma 1 (weakening) and Lemma 3 (assertion environment weakening) we can then conclude that

$$\begin{aligned} & \Gamma, \mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P_1 \\ & \wedge \Psi \otimes \mathcal{F}_\Psi(P_2) \triangleright P_1 \rightarrow P'_1 \\ & \implies \Gamma, \mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2) \vdash P'_1 \end{aligned}$$

This satisfies the first premise of **[T-PAR]**. We must then show that $\Gamma, \mathcal{F}_T(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ also holds.

By Lemma 6, we have that $\mathcal{F}_\Psi(P_1) \leq \mathcal{F}_\Psi(P'_1)$. From $\Gamma, \mathcal{F}_T(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2$ and Lemma 3 we can then conclude that $\Gamma, \mathcal{F}_T(P_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$. We now have three cases to consider:

1. If $\mathcal{F}_T(P_1) =_\Psi \mathcal{F}_T(P'_1)$ then $\Gamma, \mathcal{F}_T(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ follows from the definition of $=_\Psi$ and associativity and commutativity of $\otimes$.
2. If $\mathcal{F}_T(P_1) \leq \mathcal{F}_T(P'_1)$, then we use Lemma 1 (weakening) to conclude that $\Gamma, \mathcal{F}_T(P'_1), \Psi \otimes \mathcal{F}_\Psi(P'_1) \vdash P_2$ holds.
3. Otherwise, if $\mathcal{F}_T(P_1) \not\leq \mathcal{F}_T(P'_1)$, we instead use Lemma 2 (strengthening).

If new bound names are exposed in P'_1 , such that they would be collected by $\mathcal{F}_T(P'_1)$ and $\mathcal{F}_v(P'_1)$, we use α -conversion such that

$$\begin{aligned} & \mathcal{F}_v(P'_1) \# \Psi, \mathcal{F}_v(P_2), P_2 \\ & \mathcal{F}_v(P_2) \# \Psi, \mathcal{F}_v(P'_1), P_1 \end{aligned}$$

both hold. As both premises and side conditions now are satisfied, we can conclude $\Gamma, \Psi \vdash P'_1 \mid P_2$ by **[T-PAR]**.

- Case **[R-COM]**: We know that

$$\Psi \triangleright \overline{M_1} N[\tilde{x} := \tilde{K}].P_1 \mid \underline{M_1}(\lambda \tilde{x} : \tilde{T}) N.P_2 \rightarrow P_1 \mid P_2[\tilde{x} := \tilde{K}]$$

which was concluded by **[R-COM]**, and $\Psi \Vdash M_1 \leftrightarrow M_2$ from the premise. We also know that

$$\Gamma, \Psi \vdash \overline{M_1} N[\tilde{x} := \tilde{K}].P_1 \mid \underline{M_1}(\lambda \tilde{x} : \tilde{T}) N.P_2$$

which must have been concluded by **[T-PAR]**. As there are no free assertions or $(\nu x : T)$ in either of the processes, the side condition and extensions of Γ and Ψ in **[T-PAR]** simplify to just

$$\frac{\Gamma, \Psi \vdash \overline{M_1} N[\tilde{x} := \tilde{K}].P_1 \quad \Gamma, \Psi \vdash \underline{M_2}(\lambda \tilde{y} : \tilde{T}) N.P_2}{\Gamma, \Psi \vdash \overline{M_1} N[\tilde{x} := \tilde{K}].P_1 \mid \underline{M_2}(\lambda \tilde{x} : \tilde{T}) N.P_2}$$

with **[T-OUT]** and **[T-IN]** used for the premises. From the premises of these rules we get the following:

$$\begin{aligned} & \Gamma, \Psi \vdash M_1 : T_1 \\ & \Gamma, \Psi \vdash N[\tilde{x} := \tilde{K}] : T_3 \\ & \Gamma, \Psi \vdash P_1 \\ & \Gamma, \Psi \vdash M_2 : T_2 \\ & \Gamma, \tilde{x} : \tilde{T}, \Psi \vdash N : T_4 \\ & \Gamma, \tilde{x} : \tilde{T}, \Psi \vdash P_2 \end{aligned}$$

where $M_1 \leftrightarrow T_3$ and $M_2 \leftrightarrow T_4$ from the side conditions.

Now by the requirement **[T-EQUAL]**, since $\Psi \Vdash M_1 \leftrightarrow M_2$ then $T_1 = T_2$. By the assumptions on compatibility, either $T_3 \leq T_4$ or $T_4 \leq T_3$. Assume that the common supertype of T_3 and T_4 is T_3 . By Lemma 4 (Substitution) we have that $T_3 = F(\tilde{x})[\tilde{x} := \tilde{K}]$ for some function F over types. By the compositionality requirement we therefore have that $\Gamma, \Psi \vdash \tilde{K} : \tilde{T}$. Thus $\Gamma, \Psi \vdash P_2[\tilde{x} := \tilde{K}]$ by Lemma 4 and Lemma 2 (Strengthening).

$$\begin{array}{c}
\text{[CHNS-NIL]} \frac{}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\mathbf{0})} \qquad \text{[CHNS-REP]} \frac{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(!P)} \\
\text{[CHNS-CASE]} \frac{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P_i)}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\text{case } \tilde{\varphi} : \tilde{P})} \qquad \text{[CHNS-ASS]} \frac{}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\langle \Psi' \rangle)} \\
\text{[CHNS-IN]} \frac{\Gamma, \Psi \vdash M : T \quad \Gamma, \tilde{x} : \tilde{T}, \Psi \vdash N : T' \quad (T \leftrightarrow T')}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\underline{M}(\lambda \tilde{x} : \tilde{T})N.P)} \quad \text{[CHNS-RUN]} \frac{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\text{run } M)} \quad (\Psi \Vdash M \Leftarrow P) \\
\text{[CHNS-OUT]} \frac{\Gamma, \Psi \vdash M : T \quad \Gamma, \Psi \vdash N : T' \quad (T \leftrightarrow T')}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\overline{M}N.P)} \quad \text{[CHNS-RES]} \frac{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}((\nu x : T)P)} \quad (x \# \Psi) \\
\text{[CHNS-PAR]} \frac{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\mathcal{F}_T(P_2), \Psi \otimes \mathcal{F}_\Psi(P_2)(P_1) \quad \text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\mathcal{F}_T(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1)(P_2)}{\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P_1 \mid P_2)} \quad \left(\mathcal{F}_V(P_1) \# \Psi, \mathcal{F}_V(P_2), P_2 \right. \\
\left. \mathcal{F}_V(P_2) \# \Psi, \mathcal{F}_V(P_1), P_1 \right)
\end{array}$$

Fig. 4. The $\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)$ predicate.

Since we now know that $\Gamma, \Psi \vdash P_1$ and $\Gamma, \Psi \vdash P_2[\tilde{x} := \tilde{K}]$, we can conclude

$$\begin{array}{c}
\Gamma, \mathcal{F}_T(P_2[\tilde{x} := \tilde{K}]), \Psi \vdash P_1 \\
\Gamma, \mathcal{F}_T(P_1), \Psi \vdash P_2[\tilde{x} := \tilde{K}]
\end{array}$$

by Lemma 1 (Weakening), and then

$$\begin{array}{c}
\Gamma, \mathcal{F}_T(P_2[\tilde{x} := \tilde{K}]), \Psi \otimes \mathcal{F}_\Psi(P_2[\tilde{x} := \tilde{K}]) \vdash P_1 \\
\Gamma, \mathcal{F}_T(P_1), \Psi \otimes \mathcal{F}_\Psi(P_1) \vdash P_2[\tilde{x} := \tilde{K}]
\end{array}$$

by Lemma 3 (Assertion environment weakening). Lastly, we use α -conversion as described in the case for $P_1 \mid P_2$ above to ensure that the side condition holds, and thus we can conclude $\Gamma, \Psi \vdash P_1 \mid P_2[\tilde{x} := \tilde{K}]$ as desired. $\square$

We now have that the subject reduction result also holds for the τ -labelled transitions of the labelled semantics for HO Ψ as given in [9]. This follows as a simple corollary:

Corollary 1. *If $\Gamma, \Psi \vdash P$ and $\Psi \triangleright P \xrightarrow{\tau} P'$ then $\Gamma, \Psi \vdash P'$ where $\xrightarrow{\tau}$ is the τ -labelled transition relation given in [9].*

Proof. By Proposition 1 we have that $\Psi, \triangleright P \xrightarrow{\tau} P' \implies \Psi \triangleright P \rightarrow P'$. Then apply Theorem 1 to conclude $\Gamma, \Psi \vdash P'$. $\square$

4.1. Safety in the generic type system

A type system normally guarantees two properties: A subject reduction property and a *safety* property, which must be implied by well-typedness.

The notion of safety will depend on the particular instantiation of the type system, so, unlike with the result of subject reduction, we cannot prove a general result of safety for the generic type system. This will instead have to be shown individually for each instance.

Such a result requires a definition of a *now-safe* predicate $\text{NSafe}_{\Gamma, \Psi}(P)$, which must ensure that P is safe to perform *at least one* reduction step (if it can reduce at all). Showing safety then amounts to showing that well-typedness implies now-safety:

$$\Gamma, \Psi \vdash P \implies \text{NSafe}_{\Gamma, \Psi}(P)$$

If this holds, then by subject reduction, $\text{NSafe}_{\Gamma, \Psi}(P')$ also holds for every reduct $P \rightarrow^* P'$ after any number of reductions, so now-safety is invariant under reduction of well-typed processes. Hence, well-typed processes are invariantly now-safe, so we say they are *safe*.

Although this notion of safety will depend on the instance, the generic type system does guarantee a basic notion of *channel safety*. This property ensures that channels are always used to transmit messages whose type is compatible with that of the channel. It follows as a consequence of the side conditions in the type rules [T-IN] and [T-OUT]. To see this, we use the definition of a simple now-safety predicate for channel-safety, given by the rules in Fig. 4.

As can be seen, the rules closely mirror the type rules of Fig. 3, except that we only examine the *prefixes* of input and output, without recursing into the continuations. The side conditions in [CHNS-IN] and [CHNS-OUT] ensure compatibility between the types of subject and object, matching those of the corresponding type rules.

For **run** M , we must also examine the process for which M is a handle, since it can be released immediately (i.e. *before* the next reduction step). Note that this imposes the restriction that a **run** M cannot be unguarded, if M is not a handle for a process. This is only a slight limitation, which could also be lifted by adding another rule to conclude $\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(\text{run } M)$ whenever M is *not* a handle for any process; i.e. $\Psi \not\vdash M \Leftarrow P$ for any P . However, we have omitted this for simplicity.

Given this predicate, we can then show the following result:

Lemma 9 (*Channel safety*). $\Gamma, \Psi \vdash P \implies \text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)$.

This is shown by induction in the type rules (Fig. 3). We omit the proof, since it should be clear from the close similarity between the type rules and the rules of the $\text{NSafe}_{\Gamma, \Psi}^{\text{ch}}(P)$ -predicate that it indeed holds. By the result of subject reduction above (Theorem 1) it then follows that channel now-safety is preserved under reduction of well-typed processes. Thus well-typed processes are *channel-safe*, and this property is then also inherited by every instance of the generic type system.

5. Instances of the generic type system

We now show how our generic type system can be applied to provide sound type systems for higher-order process calculi. We first consider type systems for a version of the $\text{HO}\pi$ -calculus [15], and then a type system for the ρ -calculus [17] introduced by Meredith and Radestock.

5.1. The higher-order π -calculus

Parrow et al. [9] give several examples of $\text{HO}\Psi$ -instances with process mobility: for example, by including the set of processes $\mathcal{R}_{\Psi}$ in $\mathbb{T}$, a process P may appear as the object of an output. If for all $P \in \mathcal{R}_{\Psi}$, $P \Leftarrow P$ is entailed by all assertions, a language similar to Thomsen's Plain CHOCS [24] is obtained, and by further allowing both names and processes to appear as objects of an output, we get a simplified version of Sangiorgi's $\text{HO}\pi$ -calculus, similar to the one described in [25]. We set the parameters for $\mathbb{T}$, $\mathbb{C}$ and entailment thus:

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

and (initially) with $\mathbb{A} \triangleq \{\emptyset\}$, $\otimes \triangleq \cup$ and $\mathbf{1} \triangleq \emptyset$. We also include the symbol $\top$ in $\mathbb{C}$ to represent a condition that is entailed by all assertions, and use that for every condition in a **case** $\tilde{\varphi} : \tilde{P}$ construct to obtain a representation of non-deterministic choice. This parameter setting obviously allows unwanted processes such as

$$\bar{a}P.0 \mid \underline{a}(\lambda x)x.\bar{x}b.0 \rightarrow \bar{P}b.0$$

where the process P is substituted for the *subject* x in the output construct $\bar{x}b.0$ after a reduction step. However, we can now use our generic type system to create an instantiation that will disallow such possibilities. We define the types of terms as:

$$T ::= (T_S, \Gamma) \quad T_S ::= \text{ch}(T) \mid \bullet$$

The intention is that higher-order types of the form $(\bullet, \Gamma)$ are the types of names that correspond to processes, whereas higher-order types of the form $(\text{ch}(T), \Gamma)$ would be the types of names that could be used both as channels and as references to processes. In $\text{HO}\pi$, we disallow the latter. The behaviour of channels and first-order variables is captured in the same manner as the simple sorting system for the π -calculus [1]. In particular, the compatibility predicate is defined by $\text{ch}(T) \dot{\leftrightarrow} T$.

Run-time errors can be expressed by means of a simple error predicate of the form $\text{Wrong}_{\Gamma, \Psi}(P)$, which is just the converse of a now-safe predicate; i.e. we can define $\text{NSafe}_{\Gamma, \Psi}(P) \triangleq \neg \text{Wrong}_{\Gamma, \Psi}(P)$. The most relevant rules in the definition of the error predicate are the following, which describe that only terms of simple channel types can be used as channels and only terms of higher order types can be used as arguments of a **run** M operator.

$$\frac{\Gamma, \Psi \vdash M : (\text{ch}(T'), \Gamma)}{\text{Wrong}_{\Gamma, \Psi}(\underline{M}(\lambda x)x.P)} \quad \frac{\Gamma, \Psi \vdash M : \text{ch}(T)}{\text{Wrong}_{\Gamma, \Psi}(\text{run } M)}$$

We now define the instance parameters as follows:

$$\begin{aligned} [\text{T-CON}] &\frac{}{\Gamma, \Psi \vdash \top} & [\text{T-ASS}] &\frac{P : (\bullet, \Gamma) \in \Psi'}{\Gamma, \Psi \vdash \langle \Psi' \rangle} & [\text{TERM}_1] &\frac{\Gamma(x) = \text{ch}(T)}{\Gamma, \Psi \vdash x : \text{ch}(T)} \end{aligned}$$

$$\begin{array}{c}
\text{[NIL]} \frac{}{\Gamma \vdash \mathbf{0} : 0} \quad \text{[IN]} \frac{\Gamma, X : (k-1) \vdash P : n \quad \Gamma(a) = \text{ch}^k(\diamond)}{\Gamma \vdash a(X).P : n} \quad \text{[OUT]} \frac{\Gamma \vdash Q : m \quad \Gamma \vdash P : n \quad \Gamma(a) = \text{ch}^k(\diamond) \quad m < k}{\Gamma \vdash \bar{a} < Q >.P : \max(k, n)} \\
\text{[PAR]} \frac{\Gamma \vdash P : m \quad \Gamma \vdash Q : n}{\Gamma \vdash P \mid Q : \max(m, n)} \quad \text{[VAR]} \frac{\Gamma(X) = n}{\Gamma \vdash X : n} \quad \text{[NEW]} \frac{\Gamma, a : \text{ch}^k(\diamond) \vdash P : n}{\Gamma \vdash (\nu a : T) P : n}
\end{array}$$

Fig. 5. The type rules of Demangeon et al. [16] for HOpi₂.

We can now show safety for the type system instance:

Theorem 2. *If $\Gamma, \Psi \vdash P$ then $\neg \text{Wrong}_{\Gamma, \Psi}(P)$.*

The proof is by induction in the rules of $\Gamma, \Psi \vdash P$. Details are given in [26].

5.2. A type system for termination

We now turn our attention to an instance of the generic type system that captures a liveness property. Demangeon et al. [16] present a type system for checking termination in variants of the HO π -calculus: for any well-typed process P we have that $P \rightarrow^* P' \not\rightarrow$. The version of the HO π -calculus, HOpi₂, is a higher-order process calculus in which only processes can be communicated and replication is absent. Its syntax is given by the formation rules

$$P ::= \mathbf{0} \mid a(X).P \mid \bar{a} < Q >.P \mid P_1 \mid P_2 \mid (\nu a : T) P \mid X$$

Type judgements in the type system for HOpi₂ are of the form $\Gamma \vdash P : n$, where n is a natural number called the *level* of P . Names a have types of the form $\text{ch}^k(\diamond)$, where $\diamond$ denotes the type of processes and k is a natural number, the level of a . The intention is that names with type $\text{ch}^k(\diamond)$ can only be used to carry processes whose level n is less than k .

The type rules, shown in Fig. 5, ensure that the level of processes that are sent on any channel a will be strictly smaller than that of a . Because of the absence of replication, this will ensure that well-typed processes will always terminate.

It is straightforward to represent the HOpi₂ calculus as an instance of the Higher-Order Ψ -calculus, using a variant of the parameter setting described in section 5.1. In order to represent the type system, we introduce assertions of the form

$$\Psi ::= n \mid n^- \mid n^+$$

We use assertions to indicate in which way a channel is to be used; an input use can only be typed in the presence of an assertion n^- and output use must be used with an assertion n^+ . We have that $n \otimes n^- = n^- \otimes n = n$; that $n \otimes n^+ = n^+ \otimes n = n$; and that $n_1 \otimes n_2 = \max(n_1, n_2)$. We distinguish explicitly between input uses ($\text{ch}^k_-(\diamond)$) and output uses ($\text{ch}^k_+(\diamond)$) of channels.

$$T \in \mathbf{Types} ::= n \mid \text{ch}^k_-(\diamond) \mid \text{ch}^k_+(\diamond)$$

and we let $\text{ch}^n(\diamond) \Leftarrow (n-1)$ and $\text{ch}^n(\diamond) \Leftarrow k$ whenever $k < n$. Type judgements are of the form $\Gamma, m \vdash M : T$ for terms and $\Gamma, m \vdash P$ for processes. We represent the judgement $\Gamma \vdash P : n$ as $\Gamma, n \vdash P$. The type rules for channels are the following.

$$\begin{array}{c}
\text{[CH-IN]} \frac{\Gamma(a) = \text{ch}^k_-(\diamond)}{\Gamma, n^- \vdash a : \text{ch}^k_-(\diamond)} \quad \text{[CH-OUT]} \frac{\Gamma(a) = \text{ch}^k_+(\diamond)}{\Gamma, n^+ \vdash a : \text{ch}^k_+(\diamond)}
\end{array}$$

5.3. The ρ -calculus

The Reflective Higher-Order calculus of Meredith and Radestock [17] is less well-known than e.g. CHOCS and HO π , so we recall it in some detail. Unlike other calculi, the ρ -calculus does not assume an infinite set of names: instead, names and processes are both built from the same syntax, so names are structured terms, rather than atomic entities. The syntax for both processes and names is given by the formation rules:

$$\begin{array}{c}
P ::= \mathbf{0} \mid P \mid P \mid x \langle P \rangle \mid x(y).P \mid \ulcorner x \urcorner \\
x, y ::= \ulcorner P \urcorner
\end{array}$$

where the syntax for names is $\ulcorner P \urcorner$, pronounced *quote P*. Names can be passed around as in the π -calculus, as well as un-quoted, written $\ulcorner x \urcorner$ (pronounced *drop x*), and thus higher-order behaviour becomes an inherent property of the calculus, rather than just an extension on top of an already computationally complete language.

The parallel, and the input construct $x(y).P$, are similar to their π -calculus counterparts. The *lift* operation, $x \langle P \rangle$ is an output construct that quotes the process P , thereby creating the *name* $\ulcorner P \urcorner$, and sends it out on x ; thus the calculus can

generate new names at runtime without the need of a ν -operator. The converse of lift is the *drop* operation, $\ulcorner x \urcorner$: it is a request to run the process within a name, by removing the quotes around it. This is not performed by a reduction, but rather by a form of substitution

$$\ulcorner x \urcorner \{ \ulcorner P \urcorner / x' \} = P \quad \text{if } x \equiv_{\mathcal{N}} x'$$

where $\equiv_{\mathcal{N}}$ is the *name equivalence* relation, defined further down. Here, the entire *process* $\ulcorner x \urcorner$ is replaced with the process P found within the substituted name, similar to how process variables are replaced by processes in e.g. $\text{HO}\pi$. Notably, this means that if x is a *free* name, then $\ulcorner x \urcorner$ will be a *deadlock*, since x can never be touched by a substitution at runtime. Otherwise, substitution is the standard, capture-avoiding substitution of names for names, and note in particular that substitution does *not* recur into processes under quotes; i.e. $\ulcorner P \urcorner \{x/y\} = \ulcorner P \urcorner$ if $y \not\equiv_{\mathcal{N}} \ulcorner P \urcorner$ regardless of whether the name y exists somewhere within $\ulcorner P \urcorner$.

The reduction semantics is given by the standard rules for parallel composition and structural congruence (as in e.g. the π -calculus) plus a rule for communication:

$$\begin{array}{c} [\rho\text{-PAR}] \frac{P_1 \rightarrow P'_1}{P_1 \mid P_2 \rightarrow P'_1 \mid P_2} \quad [\rho\text{-STRUCT}] \frac{P_1 \equiv P'_1 \quad P'_1 \rightarrow P'_2 \quad P'_2 \equiv P_2}{P_1 \rightarrow P_2} \quad [\rho\text{-COM}] \frac{x_1 \equiv_{\mathcal{N}} x_2}{x_1(y).P_1 \mid x_2\langle P_2 \rangle \rightarrow P_1\{\ulcorner P_2 \urcorner / y\}} \end{array}$$

One subtlety of this calculus concerns the notion of structural congruence, $\equiv$. It is the usual least congruence on processes, containing α -equivalence, $\equiv_{\alpha}$, and the abelian monoid rules for parallel composition with $\mathbf{0}$ as the unit element. However, with *structured* terms as names, $\equiv_{\alpha}$ in turn requires a notion of *name equivalence*, written $\equiv_{\mathcal{N}}$, that is also used for comparing subjects in the $[\rho\text{-COM}]$ rule above. It is defined as the smallest equivalence relation on *quoted* processes, closed forward under the rules:

$$\begin{array}{c} [\rho\text{-NAMEEQ}_1] \frac{P \equiv Q}{\ulcorner P \urcorner \equiv_{\mathcal{N}} \ulcorner Q \urcorner} \quad [\rho\text{-NAMEEQ}_2] \frac{}{\ulcorner \ulcorner x \urcorner \urcorner \equiv_{\mathcal{N}} x} \end{array}$$

This yields a mutual recursion between name equivalence, structural congruence and α -equivalence, albeit one that always terminates as proved in [17], because both the sets of names and processes are well-founded; their smallest elements being $\mathbf{0}$ (the inactive process) and $\ulcorner \mathbf{0} \urcorner$ respectively.

5.3.1. Instantiation as a Ψ -calculus

The ρ -calculus is interesting in the present setting, because it cannot be encoded in the π -calculus in a way that satisfies a number of generally accepted criteria of encodability, similar to those of [27].² This has been established by one of the authors in [18].

The key reason for this impossibility lies in the ability of the ρ -calculus to generate new, *free*, and hence observable, names at runtime, whilst this is not possible in the π -calculus; and, dually, its use of name equivalence, which will equate more names than strict syntactic equality. However, the ρ -calculus *can* be represented in the $\text{HO}\Psi$ -framework as follows. We define

$$\begin{aligned} \mathbb{T} &\triangleq \mathcal{N} \cup \{ \ulcorner P \urcorner \mid P \in \mathcal{R}_{\Psi} \} \cup \{ \langle \ulcorner P \urcorner \rangle \mid P \in \mathcal{R}_{\Psi} \} \\ \mathbb{C} &\triangleq \{ M \leftrightarrow N \mid M, N \in \mathbb{T} \} \cup \{ P_1 \equiv P_2 \mid P_1, P_2 \in \mathcal{R}_{\Psi} \} \\ &\quad \cup \{ M \leftarrow P \mid M \in \mathbb{T} \wedge P \in \mathcal{R}_{\Psi} \} \end{aligned}$$

and (initially) with $\mathbb{A} \triangleq \{ \emptyset \}$, $\otimes \triangleq \cup$ and $\mathbf{1} \triangleq \emptyset$ as before. Note the two different kinds of terms: we use terms of the form $\ulcorner P \urcorner$ to represent a *statically* quoted name in the ρ -calculus, which can never be dropped and never substituted into. Conversely, we use $\langle \ulcorner P \urcorner \rangle$ for the equivalent of the object of a $x\langle P \rangle$, which in the ρ -calculus is a *process* that therefore *can* be substituted into, and which later may be dropped. Furthermore, we shall assume that all *bound names* are implemented as distinct atomic names $x \in \mathcal{N}$; this is a trivial conversion, since their structure has no semantic meaning in the ρ -calculus. The encoding is then given by the translation:

$$\begin{array}{ll} \llbracket \mathbf{0} \rrbracket = \mathbf{0} & \llbracket \ulcorner x \urcorner \rrbracket = \mathbf{run} \ x \\ \llbracket P_1 \mid P_2 \rrbracket = \llbracket P_1 \rrbracket \mid \llbracket P_2 \rrbracket & \llbracket \ulcorner \ulcorner P \urcorner \urcorner \rrbracket = \mathbf{0} \\ \llbracket n \langle P \rangle \rrbracket = \llbracket n \rrbracket \langle \llbracket \ulcorner P \urcorner \urcorner \rrbracket \rangle . \mathbf{0} & \llbracket \ulcorner P \urcorner \rrbracket = \ulcorner \mathcal{N} \llbracket P \rrbracket \urcorner \\ \llbracket n(x).P \rrbracket = \llbracket n \rrbracket (\lambda x) \langle x \rangle . \llbracket P \rrbracket & \llbracket x \rrbracket = x \end{array}$$

where $\mathcal{N} \llbracket P \rrbracket$ is identical to $\llbracket P \rrbracket$ except that $\mathcal{N} \llbracket \ulcorner P \urcorner \urcorner \rrbracket = \mathbf{run} \ \ulcorner \mathcal{N} \llbracket P \rrbracket \urcorner$.

² The criteria are: Preserving the degree of parallelism, correspondence of substitutions (the ability to move a substitution into/out of the translation, up to a notion of behavioural equivalence), observational correspondence (the same names must be observable in the source and translated target terms), operational correspondence, and divergence reflection (if a translated target term diverges, then the source term must diverge).

Note the two translations of drop for processes: the process $\neg P \neg$ has no reduction in the ρ -calculus and is therefore behaviourally equivalent to $\mathbf{0}$; but its counterpart $\mathbf{run} \neg P \neg$ might have a reduction, since $\mathbf{run} M$ is not evaluated eagerly in the $\text{HO}\Psi$ -calculus. For the purpose of preserving operational correspondence, we therefore translate the drop of a *free name* $\neg P \neg$ as $\mathbf{0}$, and the drop of an atomic name x as $\mathbf{run} x$, since atomic names are bound by construction. However, we cannot do this within *names*, since name equivalence is determined by the structure, rather than the behaviour of the process within quotes. Thus we use the second level translation $\mathcal{N}[\![P]\!]$ for statically quoted names, since these can never be dropped.

Lastly, we shall define entailment such that it contains the rule $\Psi \Vdash \neg P \neg \Leftarrow P$, making every term $\neg P \neg$ a handle for the process P within, to mirror the duality of names and processes in the ρ -calculus. We furthermore include the following rules for entailment of *channel equivalence* $\Leftarrow$, mirroring the rules $[\rho\text{-NAMEEQ}_1]$ and $[\rho\text{-NAMEEQ}_2]$ for concluding name equivalence:

$$[\text{CHANEQ}_1] \frac{\Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \neg P_1 \neg \Leftarrow \neg P_2 \neg} \quad [\text{CHANEQ}_2] \frac{}{\Psi \Vdash \neg \mathbf{run} M \neg \Leftarrow M}$$

including the symmetric and transitive closure of $\Leftarrow$. We 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 four congruence rules derived from the above translation:

$$\begin{aligned} [\text{PAR}] & \frac{\Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash P_1 \mid R \equiv P_2 \mid R} & [\text{RUN}] & \frac{\Psi \Vdash M_1 \Leftarrow M_2}{\Psi \Vdash \mathbf{run} M_1 \equiv \mathbf{run} M_2} \\ [\text{OUT}] & \frac{\Psi \Vdash M_1 \Leftarrow M_2 \quad \Psi \Vdash P_1 \equiv P_2}{\Psi \Vdash \overline{M_1} \langle \neg P_1 \neg \rangle \equiv \overline{M_2} \langle \neg P_2 \neg \rangle} & [\text{IN}] & \frac{\Psi \Vdash M_1 \Leftarrow M_2 \quad \Psi \Vdash P_1[x_1 := z] \equiv P_2[x_2 := z]}{\Psi \Vdash \underline{M_1}(\lambda x_1) \langle x_1 \rangle . P_1 \equiv \underline{M_2}(\lambda x_2) \langle x_2 \rangle . P_2} \quad (z \# P_1, P_2) \end{aligned}$$

Given that this instantiation is a little more involved (i.e. requires a little more encoding) than the $\text{HO}\pi$ -instantiation, we shall now formally prove that it indeed preserves the semantics of the ρ -calculus. In other words, we shall prove (strict) operational correspondence between the two semantics, up to a reasonable notion of behavioural equivalence $\asymp$. We need not choose any particular notion, but we require that it contains at least structural congruence and the axiom $\mathbf{run} \neg P \neg \asymp P$. This is reasonable to include, since if P cannot perform any reductions, then unfolding $\mathbf{run} \neg P \neg$ cannot enable any reductions either; and conversely, if $P \rightarrow P'$ then $\mathbf{run} \neg P \neg \rightarrow P'$ after unfolding by the evaluation relation.

As a first step, we need a lemma relating name equivalence in the ρ -calculus with the entailment of channel equivalence in our $\text{HO}\Psi$ -instance:

Lemma 10. $x_1 \asymp x_2 \iff \emptyset \Vdash \llbracket x_1 \rrbracket \Leftarrow \llbracket x_2 \rrbracket$

Proof sketch. By induction in the rules of name equivalence and structural congruence, and the entailment of channel equivalence. This is straightforward, as the latter is defined as a one-to-one match of the rules of name equivalence and structural congruence. $\square$

Secondly, we shall need a lemma relating substitution in the two calculi:

Lemma 11. Let $\sigma_s \triangleq \{\neg Q \neg / x\}$ be a substitution in the ρ -calculus (source language), and let $\sigma_t \triangleq [x := \llbracket \neg Q \neg \rrbracket]$ denote the corresponding substitution in the $\text{HO}\Psi$ -calculus (target language). Then $\llbracket P\sigma_s \rrbracket \asymp \llbracket P \rrbracket \sigma_t$ and $\emptyset \Vdash \llbracket n\sigma_s \rrbracket \Leftarrow \sigma_t(\llbracket n \rrbracket)$.

Proof. By induction in the clauses of the translation function:

The cases for $\mathbf{0}$, $P_1 \mid P_2$ and $x \langle P \rangle$ are simple: For the left-hand side apply the substitution first and then perform the translation, and conversely for the right-hand side. Then apply the induction hypothesis (where necessary) to conclude.

The other cases are more interesting:

- Case input: We show that $\llbracket (n(y).P)\sigma_s \rrbracket = \llbracket n(y).P \rrbracket \sigma_t$. For the left-hand side we apply the substitution first, and then the translation:

$$\begin{aligned} & \llbracket (n(y).P)\sigma_s \rrbracket \\ &= \llbracket \sigma_s(n)(y).(P)\sigma_s \rrbracket \\ &= \llbracket \sigma_s(n) \rrbracket (\lambda y) \langle y \rangle . \llbracket P\sigma_s \rrbracket \end{aligned}$$

Assuming all bound names are distinct, we omit the extra step of α -converting y .

For the right-hand side we perform the translation first and then the substitution and obtain:

$$\begin{aligned} & \llbracket n(y).P \rrbracket \sigma_t \\ &= (\llbracket n \rrbracket (\lambda y) \langle y \rangle . \llbracket P \rrbracket) \sigma_t \\ &= \sigma_t(\llbracket n \rrbracket) (\lambda y) \langle y \rangle . (\llbracket P \rrbracket) \sigma_t \end{aligned}$$

Then apply the induction hypothesis to conclude that $\emptyset \Vdash \llbracket \sigma_s(n) \rrbracket \leftrightarrow \sigma_t(\llbracket n \rrbracket)$ and $\llbracket P\sigma_s \rrbracket \asymp \llbracket P \rrbracket \sigma_t$.

- Case bound drop: We show that $\llbracket \ulcorner x \urcorner \sigma_s \rrbracket \asymp \llbracket \ulcorner x \urcorner \rrbracket \sigma_t$. Note firstly that of $x \notin \text{dom}(\sigma_s)$, then we trivially obtain $\llbracket \ulcorner x \urcorner \rrbracket = \mathbf{run} \ x$ in both cases, so we shall focus on the other case. For the left-hand side we apply the substitution first, and then the translation:

$$\llbracket \ulcorner x \urcorner \{ \ulcorner Q \urcorner / x \} \rrbracket = \llbracket Q \rrbracket$$

For the right-hand side we perform the translation first and then the substitution and obtain:

$$\begin{aligned} & \llbracket \ulcorner x \urcorner \{ x := \llbracket \ulcorner Q \urcorner \rrbracket \} \rrbracket \\ &= \mathbf{run} \ x[x := \llbracket \ulcorner Q \urcorner \rrbracket] \\ &= \mathbf{run} \ \llbracket \ulcorner Q \urcorner \rrbracket \end{aligned}$$

and by our requirement to the behavioural equivalence relation, we precisely get that $Q \asymp \mathbf{run} \ \ulcorner Q \urcorner$.

- Case free drop: We show that $\llbracket \ulcorner P \urcorner \sigma_s \rrbracket \asymp \llbracket \ulcorner P \urcorner \rrbracket \sigma_t$. Note that $\ulcorner P \urcorner$ here is a *free* name, so by construction *no* atomic names can appear within it. Thus, in either case, the substitution will have no effect; and by applying our translation function, we then in both cases get that $\llbracket \ulcorner P \urcorner \rrbracket = \mathbf{0}$.
- Case quote: We show that $\emptyset \Vdash \llbracket \sigma_s(\ulcorner P \urcorner) \rrbracket \leftrightarrow \sigma_t(\llbracket \ulcorner P \urcorner \rrbracket)$. Like above, $\ulcorner P \urcorner$ is a free name which by construction cannot contain any atomic names, so in either case the substitution has no effect. By applying the translation, we then trivially obtain that $\emptyset \Vdash \ulcorner \mathcal{N}[\ulcorner P \urcorner] \urcorner \leftrightarrow \ulcorner \mathcal{N}[\ulcorner P \urcorner] \urcorner$.
- Case atomic: We show that $\emptyset \Vdash \llbracket \sigma_s(x) \rrbracket \leftrightarrow \sigma_t(\llbracket x \rrbracket)$. For the left-hand side we apply the substitution first, and then the translation and obtain:

$$\begin{aligned} & \llbracket x \{ \ulcorner Q \urcorner / x \} \rrbracket \\ &= \llbracket \ulcorner Q \urcorner \rrbracket \\ &= \ulcorner \llbracket Q \rrbracket \urcorner \end{aligned}$$

For the right-hand side we perform the translation first and then the substitution and obtain:

$$\begin{aligned} & \llbracket x \rrbracket [x := \llbracket \ulcorner Q \urcorner \rrbracket] \\ &= x[x := \ulcorner \llbracket Q \rrbracket \urcorner] \\ &= \ulcorner \llbracket Q \rrbracket \urcorner \end{aligned}$$

and we then trivially obtain that $\emptyset \Vdash \ulcorner \llbracket Q \rrbracket \urcorner \leftrightarrow \ulcorner \llbracket Q \rrbracket \urcorner$. $\square$

We can now state our desired result. The translation is sound and complete w.r.t. operational correspondence up to a reasonable notion of behavioural equivalence $\asymp$:

Theorem 3 (Operational correspondence). *Let $\asymp$ be a notion of behavioural equivalence for processes of the HO Ψ -instance of the ρ -calculus, that includes at least structural congruence and the axiom $\mathbf{run} \ \ulcorner \llbracket P \rrbracket \urcorner \asymp \llbracket P \rrbracket$. Then*

$$P \rightarrow P' \iff \llbracket P \rrbracket \rightarrow \asymp \llbracket P' \rrbracket$$

Proof. For the forward direction (completeness), we use induction in the reduction rules of the ρ -calculus semantics. The case for $[\rho\text{-PAR}]$ and $[\rho\text{-STRUCT}]$ are given by a straightforward application of the induction hypothesis. Thus, the only interesting case is for the communication rule, $[\rho\text{-COM}]$: We have that

$$n_1 \langle P_1 \rangle \mid n_2(y).P_2 \rightarrow P_2 \{ \ulcorner P_1 \urcorner / y \}$$

with the premise that $n_1 \equiv_{\mathcal{N}} n_2$. We perform the translation on both sides and obtain

$$\emptyset \triangleright \overline{\llbracket n_1 \rrbracket} \langle \llbracket \ulcorner P_1 \urcorner \rrbracket \rangle . \mathbf{0} \mid \llbracket n_2 \rrbracket (\lambda y) \langle y \rangle . \llbracket P_2 \rrbracket \rightarrow \mathbf{0} \mid \llbracket P_2 \rrbracket [y := \llbracket \ulcorner P_1 \urcorner \rrbracket]$$

This can be concluded by the HO Ψ -calculus reduction rule $[\mathbf{R-COM}]$. The premise requires that $\emptyset \Vdash \llbracket n_1 \rrbracket \leftrightarrow \llbracket n_2 \rrbracket$, which holds, since by Lemma 10 we have that

$$n_1 \equiv_{\mathcal{N}} n_2 \implies \emptyset \Vdash \llbracket n_1 \rrbracket \leftrightarrow \llbracket n_2 \rrbracket$$

Then by Lemma 11, we have that

$$\llbracket P_2 \rrbracket [y := \llbracket \ulcorner P_1 \urcorner \rrbracket] \asymp \llbracket P_2 \{ \ulcorner P_1 \urcorner / y \} \rrbracket$$

and by our requirements to the behavioural equivalence relation, that

$$\mathbf{0} \mid \llbracket P_2 \{ \ulcorner P_1 \urcorner / y \} \rrbracket \asymp \llbracket P_2 \{ \ulcorner P_1 \urcorner / y \} \rrbracket$$

which is precisely the translation of our reduct. Thus completeness is proved.

For the other direction (soundness), we proceed similarly by examining the reduction rules of the HO Ψ -calculus semantics. Note that as restriction (νx) does not appear in our translation, we can disregard the $[\mathbf{R-RES}]$ rule. The case for $[\mathbf{R-PAR}]$

and **[R-EVAL]** are again given by a straightforward application of the induction hypothesis. Thus again, the only interesting case is the communication rule, **[R-COM]**: We have that

$$\emptyset \triangleright \overline{\llbracket n_1 \rrbracket} \langle \llbracket \ulcorner P_1 \urcorner \rrbracket \rangle . \mathbf{0} \mid \llbracket n_2 \rrbracket (\lambda y) \langle y \rangle . \llbracket P_2 \rrbracket \rightarrow \mathbf{0} \mid \llbracket P_2 \rrbracket [y := \llbracket \ulcorner P_1 \urcorner \rrbracket]$$

where we know from the premise that $\emptyset \Vdash \llbracket n_1 \rrbracket \leftrightarrow \llbracket n_2 \rrbracket$. Let P be the redex, and T' be the reduct. Just like above, we note that

$$\mathbf{0} \mid \llbracket P_2 \rrbracket [y := \llbracket \ulcorner P_1 \urcorner \rrbracket] \asymp \llbracket P_2 \rrbracket [y := \llbracket \ulcorner P_1 \urcorner \rrbracket] \asymp \llbracket P_2 \{ \ulcorner P_1 \urcorner / y \} \rrbracket$$

where the last rewrite is concluded by Lemma 11. Let this form be our $\llbracket P' \rrbracket$; thus $P' = P_2 \{ \ulcorner P_1 \urcorner / y \}$. Then take the following ρ -calculus reduction:

$$n_1 \langle P_1 \rangle \mid n_2 (y) . P_2 \rightarrow P_2 \{ \ulcorner P_1 \urcorner / y \}$$

The redex correspond exactly to the $\text{HO}\Psi$ redex, so let it be our P . This reduction can be concluded by the **[ρ -COM]** rule. The premise requires that $n_1 \equiv_{\mathcal{N}} n_2$, but by Lemma 10 we have that

$$\emptyset \Vdash \llbracket n_1 \rrbracket \leftrightarrow \llbracket n_2 \rrbracket \implies n_1 \equiv_{\mathcal{N}} n_2$$

so the premise holds. Thus soundness is proved. $\square$

5.3.2. A type system for reflection

Other higher-order calculi such as CHOCS and $\text{HO}\pi$ can be encoded in the π -calculus and may thus be typable through translation, but as we noted above there cannot be such an encoding of the ρ -calculus into the π -calculus. Thus, we cannot hope to create a type system for the ρ -calculus by adapting an existing first-order type system. In fact, we are not aware of any general type system for the ρ -calculus, so we shall now create one by instantiating our generic type system.³ We let types for names be of the form

$$T \in \mathbf{Types} ::= \langle T, \Gamma \rangle \mid \langle B, \Gamma \rangle$$

where B is a base type, and Γ is a type environment representing the possibility of executing the process within the name. Furthermore we shall use assertions as type environments for processes as we previously did with $\text{HO}\pi$, so we update the definition accordingly.

$$\mathbb{A} \triangleq \wp \left(\begin{array}{l} \{ \ulcorner P \urcorner : T \mid P \in \mathcal{P}_{\Psi} \wedge T \in \mathbf{Types} \} \\ \cup \{ \langle \ulcorner P \urcorner \rangle : T \mid P \in \mathcal{P}_{\Psi} \wedge T \in \mathbf{Types} \} \end{array} \right)$$

with assertion unit and composition as $\mathbf{1} \triangleq \emptyset$ and $\otimes \triangleq \cup$ respectively. Note that by construction $\forall x \in \mathcal{N}. x \# \ulcorner P \urcorner$, so substitution can only occur in terms of the form $\langle \ulcorner P \urcorner \rangle : T$. We then append an assertion to the encoding of input and output:

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

Lastly, we also need to take the type information into account when concluding channel equivalence, to ensure that two terms with initially dissimilar types cannot become channel equivalent after a substitution. Thus we redefine the entailment rule **[CHANEQ₁]** as follows:

$$[\text{CHANEQ}_1] \frac{\Gamma, \Psi \Vdash P_1 \equiv P_2 \quad \Gamma, \Psi \vdash \ulcorner P_1 \urcorner : T \iff \Gamma, \Psi \vdash \ulcorner P_2 \urcorner : T}{\Gamma, \Psi \Vdash \ulcorner P_1 \urcorner \leftrightarrow \ulcorner P_2 \urcorner}$$

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

$$[\text{TERM-1}] \frac{\ulcorner P \urcorner : \langle T, \Gamma' \rangle \in \Psi \quad \Gamma', \Psi \vdash P}{\Gamma, \Psi \vdash \ulcorner P \urcorner : \langle T, \Gamma' \rangle} \quad [\text{TERM-2}] \frac{\langle \ulcorner P \urcorner \rangle : \langle T, \Gamma' \rangle \in \Psi \quad \Gamma', \Psi \vdash P}{\Gamma, \Psi \vdash \langle \ulcorner P \urcorner \rangle : \langle T, \Gamma' \rangle}$$

The rules **[TERM-1]** and **[TERM-2]** tell us that the process found within a term must be well-typed w.r.t. the type environment in the second component of its type, and that the process-type pair must be represented in the assertion. The other instance rules ensure that names, that are quoted processes according to an assertion Ψ , correspond to processes that are typable within the same environment (this is expressed by **[T-Ass]**) and can carry processes. These are as follows:

³ A recent paper, [18], describes a form of type system for checking channel safety in the ρ -calculus. However, it has certain limitations that only make it suitable for checking encoded π -calculus processes, so we would not describe it as a *general* type system for the ρ -calculus. We discuss this further at the end of this section.

$$\begin{array}{c}
\text{[T-ASS]} \frac{P : T \in \Psi' \implies T \frown \Gamma}{\Gamma, \Psi \vdash \langle \Psi' \rangle} \quad \text{[T-CHAN]} \frac{}{\langle T, \Gamma \rangle \leftarrow T} \quad \text{[T-END]} \frac{}{\langle T, \Gamma \rangle \frown \Gamma} \quad \text{[T-TERM-3]} \frac{\Gamma(x) = T}{\Gamma, \Psi \vdash x : T}
\end{array}$$

Example 8. Consider the ρ -calculus process:

$$P \triangleq x \langle Q \rangle \mid x(y) \cdot (y \langle \mathbf{0} \rangle \mid \neg y^\top)$$

for some process Q and some free name $x \triangleq \top X^\top$. A single reduction step looks as follows:

$$x \langle Q \rangle \mid x(y) \cdot (y \langle \mathbf{0} \rangle \mid \neg y^\top) \rightarrow \top Q^\top \langle \mathbf{0} \rangle \mid Q$$

so in this simple example, we have both created a new name $\top Q^\top$ and released a process, Q . The $\text{HO}\pi$ -encoding of P yields the following:

$$\llbracket P \rrbracket = \overline{\mathcal{N}[\llbracket R \rrbracket]^\top} \langle \top \llbracket Q \rrbracket^\top \rangle . \mathbf{0} \mid \mathcal{N}[\llbracket R \rrbracket]^\top (\lambda y) \langle y \rangle . (\overline{y} \langle \top \mathbf{0}^\top \rangle . \mathbf{0} \mid \text{run } y)$$

We wish to allow both usages of the newly created name $\top Q^\top$, so we append the following two assertions to the input- and output processes:

$$\langle \{ \mathcal{N}[\llbracket R \rrbracket]^\top : \langle T, \emptyset \rangle \} \rangle \quad \text{and} \quad \langle \{ \langle \top \llbracket Q \rrbracket^\top \rangle : \langle T, \Gamma \rangle \} \rangle$$

where Γ is a type environment that makes Q well-typed. The type T must describe the usage of the newly created name $\top Q^\top$, when it is used *as a name*. Here, it is used to send the process $\mathbf{0}$; hence we can choose T to be a higher-order type with both the name component and the higher-order component being empty, i.e. $T = \langle \emptyset, \emptyset \rangle$, since the process $\mathbf{0}$ contains no names and has no behaviour.

The type system works, but it does have certain limitations: Since we include [\[CHANEQ₁\]](#) in order to properly simulate the ρ -calculus, all names, that eventually become equivalent during reduction, *must* have the same type. This amounts to requiring that the programmer must know in advance *all* the names that will be generated by the program during execution. This is not entirely satisfactory, since the ρ -calculus can be used to create processes that may generate infinitely many new names, e.g. a ‘name generator’ process. Such a process is for example needed in the encoding of the π -calculus into the ρ -calculus given in [\[18\]](#). However, this encoding would not be typable using the present type system.

The aforementioned paper also offers a different approach to creating a type system for the ρ -calculus, by assuming a default type for names *not* found in the type environment Γ . This allows it to be used to type encoded π -calculus processes. However, this assumption, in turn, leads to other limitations, since it necessitates a ‘manual’ proof that the typed process will never generate new names of the non-default type at runtime (unless they are listed in Γ). Hence, that type system is not really usable in the general case of typing arbitrary ρ -calculus processes. We have yet to find a type system for the ρ -calculus that does not impose such limiting restrictions on the runtime-generated names.

5.4. A type system for non-interference

As our final example, we present a type system for security properties of mobile code in the version of the higher-order π -calculus that we studied in Section 5.1. This system makes use of the model of non-interference proposed by Goguen and Meseguer [\[28\]](#). Our type system will classify processes according to their security levels, which can be high (hi) or low (lo), and the system will guarantee that actions of high-security users do not affect observations of low-security users in that a process can be high-security or low-security.

In the following we assume that the set of messages is that of the set of names but this is not essential to the underlying ideas of the type system. Security levels are described by means of extending the set of assertions such that assertions are now defined thus:

$$\Psi \in \mathbb{A} ::= x \leftarrow P \mid \text{hi} \mid \text{lo}$$

We define an ordering of the levels by $\text{lo} \leq \text{hi}$. Assertion composition now corresponds to finding a lower bound of security levels. Thus we define

$$\text{lo} \otimes \text{lo} = \text{lo} \quad \text{lo} \otimes \text{hi} = \text{lo} = \text{hi} \otimes \text{lo} \quad \text{hi} \otimes \text{hi} = \text{hi}$$

Simple types T_S are defined as

$$T_S ::= B^\Psi \mid \text{drop}(\Gamma)^\Psi \mid \text{ch}^\Psi(T)$$

where B ranges over an unspecified set of base types and $\text{drop}(\Gamma)$ is the type of process handles. The set of types is then defined thus:

$$T ::= T_S \mid (T_S, \Gamma)$$

Next, we define the *level* of a type T , written $\text{level}(T)$, as follows:

$$\begin{aligned} \text{level}(B^\Psi) &= \Psi \\ \text{level}(\text{drop}(\Gamma)^\Psi) &= \Psi \\ \text{level}(\text{ch}^\Psi(T)) &= \Psi \\ \text{level}(T_S, \Gamma) &= \text{level}(T_S) \end{aligned}$$

In this type system, the idea is that $\Gamma, \Psi \vdash M : T$ holds if M has type T at security level Ψ . Likewise, we have that $\Gamma, \Psi \vdash P$ if all communication in P communicates data at security level Ψ .

Channels of type $\text{ch}^{\text{hi}}(T)$ can transmit content with high security level but can also safely be used to transmit content whose security level is low. On the other hand, channels of type $\text{ch}^{\text{lo}}(T)$ can only transmit content with low security level. The definition of the compatibility relation is therefore that $T \leftrightarrow \text{ch}^\Psi(T)$ if $\text{level}(T) \leq \Psi$. This allows us to send low-security data on high-security channels but prevents us from sending high-security data on low-security channels.

$$\begin{array}{c} \text{[LOW]} \frac{\Gamma, \text{hi} \vdash M : T^{\text{lo}}}{\Gamma, \text{lo} \vdash M : T^{\text{lo}}} \qquad \text{[HIGH]} \frac{\Gamma, \text{lo} \vdash M : T^{\text{lo}}}{\Gamma, \text{hi} \vdash M : T^{\text{lo}}} \end{array}$$

Moreover, we must ensure that high-security processes cannot be unleashed in low-security settings. This is handled by the rule [\[T-RUN\]](#).

We again express run-time errors by means of an error predicate for which the central rules reflecting the aforementioned requirements are given below:

$$\begin{array}{c} \frac{\Gamma, \text{lo} \vdash M : T \quad \Gamma, \text{hi} \vdash N : T}{\text{Wrong}_{\Gamma, \Psi}(M \langle N \rangle x.P)} \qquad \frac{\Gamma, \text{lo} \vdash M : (T_S, \Gamma)^{\text{hi}}}{\text{Wrong}_{\Gamma, \Psi}(\text{run } M)} \end{array}$$

We have that

Theorem 4. *If $\Gamma, \Psi \vdash P$ then $\neg \text{Wrong}_{\Gamma, \Psi}(P)$.*

The proof of this proceeds by induction in the type rules.

6. Conclusions and future work

We have presented a generic type system for higher-order Ψ -calculi, which extends a previous type system for first-order Ψ -calculi. Like its predecessor [\[11\]](#), type judgements for processes are of the form $\Gamma \vdash P$ and are given by a fixed set of rules. Terms, assertions and conditions are assumed to form nominal datatypes, and only a few requirements on type rules are imposed.

The generic type system allows us to identify what should be required of type systems for higher-order process calculi that are instances of the Ψ -calculus; these requirements take the form of instance assumptions. Thus it may also yield important insights into the general structure of type systems for higher-order calculi, and it may therefore also be taken as a starting point for developing more advanced type systems for any language that can be shown to be an instance of higher-order Ψ -calculi. Note, however, that we do not know whether the instance assumptions are *minimal* in any sense. It is possible that some of the requirements can be relaxed or further generalised to yield a smaller set of necessary assumptions. This is an open problem at the time of writing.

Our type system satisfies a general subject-reduction property and can be instantiated to yield type systems with a notion of channel safety for higher-order calculi such as CHOCS, $\text{HO}\pi$ and also the ρ -calculus. The latter in particular is interesting, as there is no valid encoding of the ρ -calculus into the π -calculus, and thus we cannot capture higher-order typability in a purely first-order setting. This establishes that our generic type system is richer than first-order type systems. However, typability in the ρ -calculus comes at the cost of necessitating that we include type information directly in the definition of channel equivalence. This amounts to saying that the programmer must know (and specify) in advance the type of all names that will be generated during the course of program evaluation. We do not know whether it is possible to create other (non-trivial) type systems for the ρ -calculus without such a restriction.

There are two important lines of future work in this direction: In [\[29\]](#), Hüttel extends the generic type system to consider more general notions of subtyping and resource awareness, and in [\[30\]](#) also considers *session types* for psi-calculi.

Both of these extensions are formulated for first-order Ψ -calculi only, and would therefore both relevant to also consider in the higher-order setting. One example is [\[31\]](#) in which Yoshida defines a type system for a higher-order π -calculus that is used in security analyses for higher-order code mobility. In this type system, the notion of linearity is central. Another is the work of Hepburn and Wright [\[32\]](#) which considers a type system for non-interference in which different parallel

components are allowed to have different security levels – unlike the one described in the present paper, for which all parallel components must agree on a security level. In both of these type systems, the context splitting rule of linear type systems is essential.

CRedit authorship contribution statement

Hans Hüttel: Writing – review & editing, Writing – original draft, Supervision, Formal analysis. **Stian Lybech:** Writing – review & editing, Writing – original draft, Investigation, Formal analysis. **Alex R. Bendixen:** Writing – original draft, Formal analysis. **Bjarke B. Bojesen:** Writing – original draft, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- [1] R. Milner, The polyadic π -calculus: a tutorial, in: *Logic and Algebra of Specification*, Springer, Berlin Heidelberg, 1993, pp. 203–246.
- [2] D. Sangiorgi, D. Walker, *The pi-Calculus: a Theory of Mobile Processes*, Cambridge University Press, 2003.
- [3] M. Hennessy, J. Riely, Resource access control in systems of mobile agents, *Inf. Comput.* 173 (1) (2002) 82–120, <https://doi.org/10.1006/inco.2001.3089>.
- [4] P. Gardner, L. Wischik, Explicit fusions, in: *International Symposium on Mathematical Foundations of Computer Science*, Springer, 2000, pp. 373–382.
- [5] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: the spi calculus, *Inf. Comput.* 148 (1) (1999) 1–70, <https://doi.org/10.1006/inco.1998.2740>.
- [6] M. Carbone, S. Maffei, On the expressive power of polyadic synchronisation in pi-calculus, *Nord. J. Comput.* 10 (2) (2003) 70–98, [https://doi.org/10.1016/S1571-0661\(05\)80361-5](https://doi.org/10.1016/S1571-0661(05)80361-5).
- [7] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: mobile processes, nominal data, and logic, in: *2009 24th Annual IEEE Symposium on Logic in Computer Science*, IEEE, 2009, pp. 39–48.
- [8] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: a framework for mobile processes with nominal data and logic, *Log. Methods Comput. Sci.* 7 (1) (Mar. 2011), [https://doi.org/10.2168/LMCS-7\(1:11\)2011](https://doi.org/10.2168/LMCS-7(1:11)2011), <https://lmcs.episciences.org/696>.
- [9] J. Parrow, J. Borgström, P. Raabjerg, J. Åman Pohjola, Higher-order psi-calculi, *Math. Struct. Comput. Sci.* 24 (2) (2014), <https://doi.org/10.1017/S0960129513000170>.
- [10] B. Pierce, D. Sangiorgi, Typing and subtyping for mobile processes, in: [1993] *Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, IEEE, 1993, pp. 376–385.
- [11] H. Hüttel, Typed ψ -calculi, in: *International Conference on Concurrency Theory*, Springer, 2011, pp. 265–279.
- [12] B. König, Analysing input/output-capabilities of mobile processes with a generic type system, *J. Log. Algebraic Program.* 63 (1) (2005) 35–58, <https://doi.org/10.1016/j.jlap.2004.01.004>, special issue on the pi-calculus.
- [13] L. Caires, Logical semantics of types for concurrency, in: T. Mossakowski, U. Montanari, M. Haveranaa (Eds.), *Algebra and Coalgebra in Computer Science*, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 16–35.
- [14] A. Igarashi, N. Kobayashi, A generic type system for the pi-calculus, *Theor. Comput. Sci.* 311 (1) (2004) 121–163, [https://doi.org/10.1016/S0304-3975\(03\)00325-6](https://doi.org/10.1016/S0304-3975(03)00325-6).
- [15] D. Sangiorgi, From π -calculus to higher-order π -calculus – and back, in: M.C. Gaudel, J.P. Jouannaud (Eds.), *TAPSOFT'93: Theory and Practice of Software Development*, Springer, Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 151–166.
- [16] R. Demangeon, D. Hirschhoff, D. Sangiorgi, Termination in higher-order concurrent calculi, *J. Log. Algebraic Program.* 79 (7) (2010) 550–577, The 20th Nordic Workshop on Programming Theory (NWPT 2008), <https://doi.org/10.1016/j.jlap.2010.07.007>.
- [17] L. Meredith, M. Radestock, A reflective higher-order calculus, *Electron. Notes Theor. Comput. Sci.* 141 (5) (2005) 49–67, *Proceedings of the Workshop on the Foundations of Interactive Computation (FInCo 2005)*, <https://doi.org/10.1016/j.entcs.2005.05.016>.
- [18] S. Lybech, The reflective higher-order calculus: encodability, typability and separation, *Inf. Comput.* 297 (2024) 105138, <https://doi.org/10.1016/j.ic.2024.105138>.
- [19] A.R. Bendixen, B.B. Bojesen, H. Hüttel, S. Lybech, A generic type system for higher-order ψ -calculi, in: V. Castiglioni, C.A. Mezzina (Eds.), *Proceedings Combined 29th International Workshop on Expressiveness in Concurrency and 19th Workshop on Structural Operational Semantics*, Warsaw, Poland, 12th, September 2022, in: *Electronic Proceedings in Theoretical Computer Science*, vol. 368, Open Publishing Association, 2022, pp. 43–59.
- [20] M. Gabbay, A. Pitts, A new approach to abstract syntax with variable binding, *Form. Asp. Comput.* 13 (2002) 341–363, <https://doi.org/10.1007/s001650200016>.
- [21] R. Harper, *Practical Foundations for Programming Languages*, 2nd edition, Cambridge University Press, USA, 2016.
- [22] J. Åman Pohjola, Psi-calculi revisited: connectivity and compositionality, in: J.A. Pérez, N. Yoshida (Eds.), *Formal Techniques for Distributed Objects, Components, and Systems*, Springer International Publishing, Cham, 2019, pp. 3–20.
- [23] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, I, *Inf. Comput.* 100 (1) (1992) 1–40.
- [24] B. Thomsen, Plain CHOCS a second generation calculus for higher order processes, *Acta Inform.* 30 (1) (1993) 1–59, <https://doi.org/10.1007/BF01200262>.
- [25] J. Parrow, An introduction to the π -calculus, in: *Handbook of Process Algebra*, Elsevier, 2001, pp. 479–543.
- [26] A.R. Bendixen, B.B. Bojesen, H. Hüttel, S. Lybech, Typing reflection in higher-order psi-calculi, *Tech. Rep.*, Department of Computer Science, Aalborg University, May 2022, <http://icetcs.ru.is/stian/2022/hopsitypes2022techreport.pdf>.
- [27] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, *Inf. Comput.* 208 (9) (2010) 1031–1053, <https://doi.org/10.1016/j.ic.2010.05.002>.
- [28] J.A. Goguen, J. Meseguer, Security policies and security models, in: *1982 IEEE Symposium on Security and Privacy*, 1982, pp. 11–20.
- [29] H. Hüttel, Types for resources in ψ -calculi, in: M. Abadi, A. Lluch Lafuente (Eds.), *Trustworthy Global Computing*, Springer International Publishing, Cham, 2014, pp. 83–102.
- [30] H. Hüttel, Binary session types for psi-calculi, in: A. Igarashi (Ed.), *Programming Languages and Systems*, Springer International Publishing, Cham, 2016, pp. 96–115.
- [31] N. Yoshida, Channel dependent types for higher-order mobile processes, *SIGPLAN Not.* 39 (1) (2004) 147–160, <https://doi.org/10.1145/982962.964014>.
- [32] M. Hepburn, D. Wright, Execution contexts for determining trust in a higher-order π -calculus, *Tech. Rep. Technical Report TR-01-2003*, School of Computing, University of Tasmania, 2003, <https://eprints.utas.edu.au/36/>.