Minimal Session Types for the π-calculus

Alen Arslanagić
University of Groningen
The Netherlands

Anda-Amelia Palamariuc
University of Groningen
The Netherlands

Jorge A. Pérez
University of Groningen and CWI
The Netherlands

ABSTRACT

Session types enable the static verification of message-passing programs. A session type specifies a channel's protocol as *sequences* of messages. Prior work established a *minimality result*: every process typable with standard session types can be compiled down to a process typable using *minimal session types*: session types without the sequencing construct. This result justifies session types in terms of themselves; it holds for a higher-order session π-calculus, where values are abstractions (functions from names to processes).

This paper establishes a new minimality result but now for the session π-calculus, the language in which values are names and for which session types have been more widely studied. Remarkably, this new minimality result can be obtained by composing known results. We develop optimizations of our new minimality result, and establish its static and dynamic correctness.

CCS CONCEPTS

• **Theory of computation** → **Process calculi**; **Type structures**.

KEYWORDS

Concurrency, Session Types, Process Calculi, Expressiveness

ACM Reference Format:
Alen Arslanagić, Anda-Amelia Palamariuc, and Jorge A. Pérez. 2021. Minimal Session Types for the π-calculus. In *23rd International Symposium on Principles and Practice of Declarative Programming (PPDP 2021), September 6–8, 2021, Tallinn, Estonia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3479394.3479407

1 INTRODUCTION

Session types are a type-based approach to statically ensure correct message-passing programs [6, 7]. A session type stipulates the sequence and payload of the messages exchanged along a channel. In this paper, we investigate a *minimal formulation* of session types for the π-calculus, the paradigmatic calculus of concurrency. This elementary formulation is called *minimal session types* (MSTs).

The gap between standard and minimal session types concerns *sequentiality* in types. Sequentiality, denoted by the action prefix ' ; ', is arguably the most distinguishing construct of session types. For instance, in the session type $S = ?(\mathsf{Int}); ?(\mathsf{Int}); !\langle \mathsf{Bool} \rangle; \mathsf{end}$, this construct conveniently specifies a channel protocol that *first* receives (?) two integers, *then* sends (!) a Boolean, and *finally* ends.

PPDP 2021, September 6–8, 2021, Tallinn, Estonia

ACM ISBN 978-1-4503-8689-0/21/09…$15.00
https://doi.org/10.1145/3479394.3479407

Because sequentiality is so useful for protocol specification and verification, a natural question is whether it could be recovered by other means. To this end, Arslanagić et al. [2] defined MSTs as the sub-class of session types without the ' ; ' construct. They established a *minimality result*: every well-typed session process can be *decomposed* into a process typable with MSTs. Their result is inspired by Parrow's work on *trios processes* for the π-calculus [12]. The minimality result justifies session types in terms of themselves, and shows that sequentiality in types is useful but not indispensable, because it can be precisely mimicked by the process decomposition.

The minimality result in [2] holds for a *higher-order* process calculus called HO, in which values are only abstractions (functions from names to processes). HO does not include name passing nor process recursion, but it can encode them precisely [9, 11]. An important question left open in [2] is whether the minimality result holds for the session π-calculus (dubbed π), the language in which values are names and for which session types have been more widely studied from foundational and practical perspectives.

In this paper, we positively answer this question. Our approach is simple, perhaps even deceptively so. In order to establish the minimality result for π, we compose the decomposition in [2] with the mutual encodings between HO and π given in [9, 11].

Let $\mu\pi$ and μHO denote the process languages π and HO with MSTs (rather than with standard session types). Also, let $\mathcal{D}(\cdot)$ denote the decomposition function from HO to μHO defined in [2]. Kouzapas et al. [9, 11] gave typed encodings from π to HO (denoted $[\![\cdot]\!]^1_g$) and from HO to π (denoted $[\![\cdot]\!]^2$). Therefore, given $\mathcal{D}(\cdot)$, $[\![\cdot]\!]^1_g$, and $[\![\cdot]\!]^2$, to define a decomposition function $\mathcal{F}(\cdot) : \pi \to \mu\pi$, it suffices to follow Figure 1. This is sound for our purposes, because $[\![\cdot]\!]^1_g$ and $[\![\cdot]\!]^2$ preserve sequentiality in processes and their types.

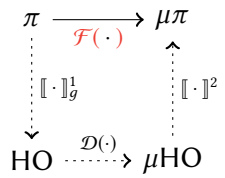

Figure 1: Decomposition by composition.

The *first contribution* of this paper is the decomposition function $\mathcal{F}(\cdot)$, whose correctness follows from that of its constituent functions. $\mathcal{F}(\cdot)$ is significant, as it provides an elegant, positive answer to the question of whether the minimality result in [2] holds for π. Indeed, it proves that the values exchanged do not influence sequentiality in session types: the minimality result of [2] is not specific to the abstraction-passing language HO.

However, $\mathcal{F}(\cdot)$ is not entirely satisfactory. A side effect of composing $\mathcal{D}(\cdot)$, $[\![\cdot]\!]^1_g$, and $[\![\cdot]\!]^2$ is that the resulting decomposition of π into $\mu\pi$ is inefficient, as it includes redundant synchronizations. These shortcomings are particularly noticeable in the treatment of recursive processes. The *second contribution* of this paper is an optimized variant of $\mathcal{F}(\cdot)$, dubbed $\mathcal{F}^*(\cdot)$, in which we remove redundant synchronizations and target recursive processes and

$$
\begin{array}{rcl}
n & ::= & a, b \mid s, \overline{s} \\
u, w & ::= & n \mid x, y, z \\
V, W & ::= & u \mid \boxed{\lambda x.\, P} \mid \boxed{x, y, z} \\
P, Q & ::= & u!\langle V\rangle.P \mid u?(x).P \\
& & \mid \boxed{V\, u} \mid P \mid Q \mid (\nu\, n)P \mid \mathbf{0} \mid X \mid \mu X.P
\end{array}
$$

Figure 2: Syntax of HOπ. The sub-language HO lacks shaded constructs, while π lacks boxed constructs.

variables directly, exploiting the fact that π supports recursion natively.

Contributions. The main contributions of this paper are:

(1) A minimality result for π, based on the function $\mathcal{F}(\cdot)$.
(2) $\mathcal{F}^*(\cdot)$, an optimized variant of $\mathcal{F}(\cdot)$, without redundant communications and with native support for recursion.
(3) Examples for $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$.

Due to space limits, omitted material (definitions, correctness proofs for $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$, additional examples) can be found in [1]. Throughout the paper, we use red and blue colors to distinguish elements of the first and second decompositions, respectively.

2 PRELIMINARIES

HO and π are actually sub-languages of HOπ [10, 11], for which we recall its syntax, semantics, and session type system. We also recall the mutual encodings between HO and π [9, 11]. Finally, we briefly discuss MSTs for HO, and the minimality result in [2].

2.1 HOπ (and its Sub-languages HO and π)

Fig. 2 gives the **syntax** of processes $P, Q, \ldots$, values $V, W, \ldots$, and conventions for names. Identifiers $a, b, c, \ldots$ denote *shared names*, while $s, \overline{s}, \ldots$ are used for *session names*. Duality is defined only on session names, thus $\overline{\overline{s}} = s$, but $\overline{a} = a$. *Names* (shared or sessions) are denoted by $m, n \ldots$, and $x, y, z, \ldots$ range over variables. We write $\widetilde{x}$ to denote a tuple $(x_1, \ldots, x_n)$, and use ϵ to denote the empty tuple.

An abstraction $\lambda x.\, P$ binds x to its body P. In processes, sequencing is specified via prefixes. The *output* prefix, $u!\langle V\rangle.P$, sends value V on name u, then continues as P. Its dual is the *input* prefix, $u?(x).P$, in which variable x is bound. Parallel composition, $P \mid Q$, reflects the combined behaviour of two processes running simultaneously. Restriction $(\nu\, n)P$ binds the endpoints $n, \overline{n}$ in process P. Process $\mathbf{0}$ denotes inaction. Recursive variables and recursive processes are denoted X and $\mu X.P$, respectively. Replication is denoted by the shorthand notation $*P$, which stands for $\mu X.(P \mid X)$.

The sets of free variables, sessions, and names of a process are denoted $\mathtt{fv}(P)$, $\mathtt{fs}(P)$, and $\mathtt{fn}(P)$. A process P is *closed* if $\mathtt{fv}(P) = \emptyset$. We write $u!\langle\rangle.P$ and $u?().P$ when the communication objects are not relevant. Also, we omit trailing occurrences of $\mathbf{0}$.

As Fig. 2 details, the **sub-languages** π and HO of HOπ differ as follows: application Vu is only present in HO; constructs for recursion $\mu X.P$ are present in π but not in HO.

$$
\begin{array}{rl}
(\lambda x.\, P)\, u \longrightarrow P\{u/x\} & \text{[App]} \\
n!\langle V\rangle.P \mid \overline{n}?(x).Q \longrightarrow P \mid Q\{V/x\} & \text{[Pass]} \\
P \longrightarrow P' \Rightarrow (\nu\, n)P \longrightarrow (\nu\, n)P' & \text{[Res]} \\
P \longrightarrow P' \Rightarrow P \mid Q \longrightarrow P' \mid Q & \text{[Par]} \\
P \equiv Q \longrightarrow Q' \equiv P' \Rightarrow P \longrightarrow P' & \text{[Cong]}
\end{array}
$$

$$
P_1 \mid P_2 \equiv P_2 \mid P_1 \quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3
$$
$$
P \mid \mathbf{0} \equiv P \quad P \mid (\nu\, n)Q \equiv (\nu\, n)(P \mid Q)\ (n \notin \mathtt{fn}(P))
$$
$$
(\nu\, n)\mathbf{0} \equiv \mathbf{0} \qquad \mu X.P \equiv P\{\mu X.P/X\} \qquad P \equiv Q \text{ if } P \equiv_\alpha Q
$$

Figure 3: Operational Semantics of HOπ.

$$
\begin{array}{rcl@{\qquad}rcl}
U & ::= & C \mid L & C & ::= & S \mid \langle S\rangle \mid \langle L\rangle \\
L & ::= & U \to \diamond \mid U \multimap \diamond & S & ::= & !\langle U\rangle; S \mid ?(U); S \\
 & & & & & \mid \mu \mathrm{t}.S \mid \mathrm{t} \mid \mathsf{end}
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
U & ::= & \widetilde{C} \to \diamond \mid \widetilde{C} \multimap \diamond & C & ::= & M \mid \langle U\rangle \\
\gamma & ::= & \mathsf{end} \mid \mathrm{t} & M & ::= & \gamma \mid !\langle\widetilde{U}\rangle; \gamma \mid ?(\widetilde{U}); \gamma \mid \mu \mathrm{t}.M
\end{array}
$$

Figure 4: STs for HOπ (top) and MSTs for HO (bottom).

The **operational semantics** of HOπ, enclosed in Figure 3, is expressed through a *reduction relation*, denoted $\longrightarrow$. Reduction is closed under *structural congruence*, $\equiv$, which identifies equivalent processes from a structural perspective. We write $P\{V/x\}$ to denote the capture-avoiding substitution of variable x with value V in P. We write '$\{\}$' to denote the empty substitution. In Figure 3, Rule [App] denotes application, which only concerns names. Rule [Pass] defines a shared or session interaction on channel n's endpoints. The remaining rules are standard.

2.2 Session Types for HOπ

Fig. 4 (top) gives the syntax of types. Value types U include the first-order types C and the higher-order types L. Session types are denoted with S and shared types with $\langle S\rangle$ and $\langle L\rangle$. We write $\diamond$ to denote the *process type*. The functional types $U \to \diamond$ and $U \multimap \diamond$ denote *shared* and *linear* higher-order types, respectively. The *output type* $!\langle U\rangle; S$ is assigned to a name that first sends a value of type U and then follows the type described by S. Dually, $?(U); S$ denotes an *input type*. Type end is the termination type. We assume the *recursive type* $\mu \mathrm{t}.S$ is guarded, i.e., the type variable t only appears under prefixes. This way, e.g., the type $\mu \mathrm{t}.\mathrm{t}$ is not allowed. The sets of free/bound variables of a type S are defined as usual; the sole binder is $\mu \mathrm{t}.S$. Closed session types do not have free type variables.

Session types for HO exclude C from value types U; session types for π exclude L from value types U and $\langle L\rangle$ from C.

We write S_1 dual S_2 if S_1 is the *dual* of S_2. Intuitively, duality converts ! into ? (and vice-versa). This intuitive definition is enough for our purposes; the formal definition is co-inductive, see [10, 11].

Typing *environments* are defined below:

$$\Gamma ::= \emptyset \mid \Gamma, x : U \to \diamond \mid \Gamma, u : \langle S \rangle \mid \Gamma, u : \langle L \rangle \mid \Gamma, X : \Delta$$
$$\Lambda ::= \emptyset \mid \Lambda, x : U \multimap \diamond \qquad \Delta ::= \emptyset \mid \Delta, u : S$$

Γ, Λ, and Δ satisfy different structural principles. Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. While Λ maps variables to linear higher-order types, Δ maps session names to session types. Both Λ and Δ are only subject to exchange. The domains of Γ, Λ and Δ (denoted $\mathsf{dom}(\Gamma)$, $\mathsf{dom}(\Lambda)$, and $\mathsf{dom}(\Delta)$) are assumed pairwise distinct.

Given Γ, we write $\Gamma \backslash x$ to denote the environment obtained from Γ by removing the assignment $x : U \to \diamond$, for some U. This notation applies similarly to Δ and Λ; we write $\Delta \backslash \Delta'$ (and $\Lambda \backslash \Lambda'$) with the expected meaning. Notation $\Delta_1 \cdot \Delta_2$ means the disjoint union of Δ_1 and Δ_2. We define *typing judgements* for values V and processes P:

$$\Gamma; \Lambda; \Delta \vdash V \triangleright U \qquad\qquad \Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$$

The judgement on the left says that under environments Γ, Λ, and Δ value V has type U; the judgement on the right says that under environments Γ, Λ, and Δ process P has the process type $\diamond$. The typing rules are presented in [1].

Type soundness for HOπ relies on two auxiliary notions:

Definition 2.1 (Session Environments: Balanced/Reduction). Let Δ be a session environment.

- Δ is *balanced* if whenever $s : S_1, \overline{s} : S_2 \in \Delta$ then S_1 dual S_2.
- We define reduction $\longrightarrow$ on session environments as:

$$\Delta, s : !\langle U \rangle; S_1, \overline{s} : ?(U); S_2 \longrightarrow \Delta, s : S_1, \overline{s} : S_2$$

Theorem 2.2 (Type Soundness [10]). *Suppose $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$ with Δ balanced. Then $P \longrightarrow P'$ implies $\Gamma; \emptyset; \Delta' \vdash P' \triangleright \diamond$ and $\Delta = \Delta'$ or $\Delta \longrightarrow \Delta'$ with Δ' balanced.*

2.3 Mutual Encodings between π and HO

The encodings $[\![\cdot]\!]^1_g : \pi \to \mathsf{HO}$ and $[\![\cdot]\!]^2 : \mathsf{HO} \to \pi$ are *typed*: each consists of a translation on processes and a translation on types. This way, $(\!| \cdot |\!)^1$ translates types for first-order processes into types for higher-order processes, while $(\!| \cdot |\!)^2$ operates in the opposite direction—see Figures 5 and 6, respectively. Remarkably, these translations on processes and types do not alter their sequentiality.

From π to HO. To mimic the sending of name w, the encoding $[\![\cdot]\!]^1_g$ encloses w within the body of an input-guarded abstraction. The corresponding input process receives this higher-order value, applies it on a restricted session, and sends the encoded continuation through the session's dual.

Several auxiliary notions are used to encode recursive processes; we describe them intuitively (see [11] for full details). The key idea is to encode recursive processes in π using a "duplicator" process in HO, circumventing linearity by replacing free names with variables. The parameter g is a map from process variables to sequences of name variables. Also, $\|\cdot\|$ maps sequences of session names into sequences of variables, and $\llbracket \cdot \rrbracket_\emptyset$ maps processes with free names to processes without free names (but with free variables instead).

The encoding $(\!| \cdot |\!)^1$ depends on the auxiliary function $\lfloor \cdot \rfloor^1$, defined on value types. Following the encoding on processes, this mapping on values takes a first-order value type and encodes it into a linear higher-order value type, which encloses an input type that expects to receive another higher-order type. Notice how the innermost higher-order value type is either shared or linear, following the nature of the given type.

Terms:

$$[\![u!\langle w \rangle.P]\!]^1_g \stackrel{\text{def}}{=} u!\langle \lambda z.\ z?(x).(x\ w) \rangle.[\![P]\!]^1_g$$
$$[\![u?(x:C).Q]\!]^1_g \stackrel{\text{def}}{=} u?(y).(\nu\, s)(y\, s \mid \overline{s}!\langle \lambda x.\ [\![Q]\!]^1_g \rangle.\mathbf{0})$$
$$[\![P \mid Q]\!]^1_g \stackrel{\text{def}}{=} [\![P]\!]^1_g \mid [\![Q]\!]^1_g$$
$$[\![(\nu\, n)P]\!]^1_g \stackrel{\text{def}}{=} (\nu\, n)[\![P]\!]^1_g$$
$$[\![\mathbf{0}]\!]^1_g \stackrel{\text{def}}{=} \mathbf{0}$$
$$[\![\mu X.P]\!]^1_g \stackrel{\text{def}}{=} (\nu\, s)(\overline{s}!\langle V \rangle.\mathbf{0} \mid s?(z_X).[\![P]\!]^1_{g,\{X \to \tilde{n}\}}) \quad \text{where } (\tilde{n} = \mathsf{fn}(P))$$
$$V = \lambda(\|\tilde{n}\|, y).\ y?(z_X).\llbracket [\![P]\!]^1_{g,\{X \to \tilde{n}\}} \rrbracket_\emptyset$$
$$[\![X]\!]^1_g \stackrel{\text{def}}{=} (\nu\, s)(z_X\, (\tilde{n}, s) \mid \overline{s}!\langle z_X \rangle.\mathbf{0}) \quad (\tilde{n} = g(X))$$

Types:

$$\lfloor S \rfloor^1 \stackrel{\text{def}}{=} (?((\!| S |\!)^1 \multimap \diamond); \mathsf{end}) \multimap \diamond$$
$$\lfloor \langle S \rangle \rfloor^1 \stackrel{\text{def}}{=} (?(\langle (\!| S |\!)^1 \rangle \to \diamond); \mathsf{end}) \multimap \diamond$$
$$(\!| !\langle U \rangle; S |\!)^1 \stackrel{\text{def}}{=} !\langle \lfloor U \rfloor^1 \rangle; (\!| S |\!)^1$$
$$(\!| ?(U); S |\!)^1 \stackrel{\text{def}}{=} ?(\lfloor U \rfloor^1); (\!| S |\!)^1$$
$$(\!| \langle S \rangle |\!)^1 \stackrel{\text{def}}{=} \langle (\!| S |\!)^1 \rangle \qquad (\!| \mu \mathsf{t}.S |\!)^1 \stackrel{\text{def}}{=} \mu \mathsf{t}.(\!| S |\!)^1$$
$$(\!| \mathsf{end} |\!)^1 \stackrel{\text{def}}{=} \mathsf{end} \qquad (\!| \mathsf{t} |\!)^1 \stackrel{\text{def}}{=} \mathsf{t}$$

Figure 5: Typed encoding of π into HO, selection from [11]. Above, $\mathsf{fn}(P)$ is a lexicographically ordered sequence of free names in P. Maps $\|\cdot\|$ and $\llbracket \cdot \rrbracket_\sigma$ can be found in [1].

From HO *to π.* The encoding $[\![\cdot]\!]^2$ simulates higher-order communication using first-order constructs, following Sangiorgi [13]. The idea is to use *trigger names*, which point towards copies of input-guarded server processes that should be activated. The encoding of abstraction sending distinguishes two cases: if the abstraction body does not contain any free session names (which are linear), then the server can be replicated. Otherwise, if the value contains session names then its corresponding server name must be used exactly once. The encoding of abstraction receiving proceeds inductively, noticing that the variable is now a placeholder for a first-order name. The encoding of application is also in two cases; both of them depend on the creation of a fresh session, which is used to pass around the applied name.

2.4 Minimal Session Types for HO

The syntax of MSTs for HO is in Fig. 4 (bottom). We write μHO to denote HO with MSTs. The decomposition $\mathcal{D}(\cdot)$ in [2] relies crucially on the ability of communicating tuples of values. Hence, value types are of the form $\widetilde{C} \to \diamond$ and $\widetilde{C} \multimap \diamond$. Similarly, minimal session

Terms:

$$[\![u!\langle\lambda x.\, Q\rangle.P]\!]^2 \stackrel{\text{def}}{=} \begin{cases}(\nu\, a)(u!\langle a\rangle.([\![P]\!]^2 \mid *\ a?(y).y?(x).[\![Q]\!]^2)) & \text{if } \mathsf{fs}(Q)=\emptyset\\ (\nu\, a)(u!\langle a\rangle.([\![P]\!]^2 \mid a?(y).y?(x).[\![Q]\!]^2)) & \text{otherwise}\end{cases}$$

$$[\![u?(x).P]\!]^2 \stackrel{\text{def}}{=} u?(x).[\![P]\!]^2$$

$$[\![x\, u]\!]^2 \stackrel{\text{def}}{=} (\nu\, s)(x!\langle s\rangle.\overline{s}!\langle u\rangle.\mathbf{0})$$

$$[\![(\lambda x.\, P)\, u]\!]^2 \stackrel{\text{def}}{=} (\nu\, s)(s?(x).[\![P]\!]^2 \mid \overline{s}!\langle u\rangle.\mathbf{0})$$

Types:

$$(\!|!\langle S\multimap\diamond\rangle;S_1|\!)^2 \stackrel{\text{def}}{=} !\big\langle\langle?((\!|S|\!)^2);\mathsf{end}\rangle\big\rangle;(\!|S_1|\!)^2$$

$$(\!|?(S\multimap\diamond);S_1|\!)^2 \stackrel{\text{def}}{=} ?\big(\langle?((\!|S|\!)^2);\mathsf{end}\rangle\big);(\!|S_1|\!)^2$$

Figure 6: Typed encoding of HO into π [11].

types for output and input are of the form $!\langle\widetilde{U}\rangle;\mathsf{end}$ and $?(\widetilde{U});\mathsf{end}$: they communicate tuples of values but lack a continuation.

Following Parrow [12], $\mathcal{D}(\cdot)$ is defined in terms of a *breakdown function* $\mathcal{B}^k_{\widetilde{x}}(\cdot)$, which translates a process into a composition of *trios processes* (or simply *trios*). A trio is a process with exactly three nested prefixes. If P is a sequential process with k nested actions, then $\mathcal{D}(P)$ will contain k trios running in parallel: each trio in $\mathcal{D}(P)$ will enact exactly one prefix from P; the breakdown function must be carefully designed to ensure that trios trigger each other in such a way that $\mathcal{D}(P)$ preserves the prefix sequencing in P. While trios decompositions elegantly induce processes typable with MSTs, they are not goal in themselves; rather, they offer one possible path to better understand sequentiality in session types.

We use some useful terminology for trios [12]. The *context* of a trio is a tuple of variables $\widetilde{x}$, possibly empty, which makes variable bindings explicit. We use a reserved set of *propagator names* (or simply *propagators*), denoted by $c_k, c_{k+1}, \ldots$, to carry contexts and trigger the subsequent trio. Propagators with recursive types are denoted by $c^r_k, c^r_{k+1}, \ldots$. We say that a *leading trio* is the one that receives a context, performs an action, and triggers the next trio; a *control trio* only activates other trios.

The breakdown function works on both processes and values. The breakdown of process P is denoted by $\mathcal{B}^k_{\widetilde{x}}(P)$, where k is the index for the propagators c_k, and $\widetilde{x}$ is the context to be received by the previous trio. Similarly, the breakdown of a value V is denoted by $\mathcal{V}^k_{\widetilde{x}}(V)$. Table 1 gives the breakdown function defined in [2] for the sub-language of HO without recursion—this is the so-called *core fragment*. In the figure, we include side conditions that use the one-line conditional $x = (c)\,?\,s_1 : s_2$ to express that $x = s_1$ if condition c is true, and $x = s_2$ otherwise. Notice that for session types we have either $C = S$ or $C = \langle S\rangle$.

To formally define $\mathcal{D}(\cdot)$ in terms of $\mathcal{B}^k_{\widetilde{x}}(\cdot)$, we need some notation. Let $\widetilde{u} = (a, b, s, s', \ldots)$ be a finite tuple of names. We shall write $\mathsf{init}(\widetilde{u})$ to denote the tuple $(a_1, b_1, s_1, s'_1, \ldots)$. We say that a process has been *initialized* if all of its names have some index.

Definition 2.3 (Decomposing Processes [2]). Let P be a closed HO process such that $\widetilde{u} = \mathsf{fn}(P)$. The *decomposition* of P, denoted $\mathcal{D}(P)$, is defined as:

$$\mathcal{D}(P) = (\nu\,\widetilde{c})(\overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{B}^k_{\epsilon}(P\sigma))$$

where: $k > 0$; $\widetilde{c} = (c_k, \ldots, c_{k+|P|-1})$; $\sigma = \{\mathsf{init}(\tilde{u})/\widetilde{u}\}$; and the breakdown function $\mathcal{B}^k_{\widetilde{x}}(\cdot)$, is as defined in Table 1.

$$\mathcal{G}(!\langle U\rangle;S) = \begin{cases}!\langle\mathcal{G}(U)\rangle;\mathsf{end} & \text{if } S=\mathsf{end}\\ !\langle\mathcal{G}(U)\rangle;\mathsf{end}\,,\mathcal{G}(S) & \text{otherwise}\end{cases}$$

$$\mathcal{G}(?(U);S) = \begin{cases}?(\mathcal{G}(U));\mathsf{end} & \text{if } S=\mathsf{end}\\ ?(\mathcal{G}(U));\mathsf{end}\,,\mathcal{G}(S) & \text{otherwise}\end{cases}$$

$$\mathcal{G}(C\multimap\diamond) = \mathcal{G}(C)\multimap\diamond \quad \mathcal{G}(\mathsf{end}) = \mathsf{end}$$

$$\mathcal{G}(C\to\diamond) = \mathcal{G}(C)\to\diamond \quad \mathcal{G}(\langle U\rangle) = \langle\mathcal{G}(U)\rangle$$

$$\mathcal{G}(S_1,\ldots,S_n) = \mathcal{G}(S_1),\ldots,\mathcal{G}(S_n)$$

Figure 7: Decomposition of types (cf. Def. 2.4)

Definition 2.4 (Decomposing Session Types). The *decomposition function* on the types of Fig. 4, denoted $\mathcal{G}(\cdot)$, is defined in Fig. 7.

As already mentioned, the *minimality result* in [2] is that if P is a well-typed HO process then $\mathcal{D}(P)$ is a well-typed μHO process. It attests that the sequentiality in the session types for P is appropriately accommodated by the decomposition $\mathcal{D}(P)$. Its proof relies on an auxiliary result establishing the typability of $\mathcal{B}^k_{\widetilde{x}}(P)$.

Theorem 2.5 (Minimality Result [2]). *Let P be a closed* HO *process with $\widetilde{u} = \mathsf{fn}(P)$ and $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$. If $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ then*

$$\mathcal{G}(\Gamma\sigma);\emptyset;\mathcal{G}(\Delta\sigma) \vdash \mathcal{D}(P) \triangleright \diamond$$

Having summarized the results on which our developments stand, we now move on to establish the minimality result but for π.

3 DECOMPOSE BY COMPOSITION

We define a decomposition function $\mathcal{F}(\cdot) : \pi \to \mu\pi$, given in terms of a breakdown function denoted $\mathcal{A}^k_{\widetilde{x}}(\cdot)_g$ (cf. Tab. 2). Following Figure 1, this breakdown function will result from the composition of $[\![\cdot]\!]^1_g$, $\mathcal{B}^k_{\widetilde{x}}(\cdot)$, and $[\![\cdot]\!]^2$, i.e., $\mathcal{A}^k_{\widetilde{x}}(\cdot)_g = [\![\mathcal{B}^k_{\widetilde{x}}([\![\cdot]\!]^1_g)]\!]^2$. Using $\mathcal{F}(\cdot)$, we obtain a minimality result for π, given by Theorem 3.11.

3.1 Key Idea

Conceptually, $\mathcal{F}(\cdot)$ can be obtained in two steps: first, the composition $\mathcal{B}^k_{\widetilde{x}}([\![\cdot]\!]^1_g)$, which returns a process in μHO; second, a step that transforms that μHO process into a $\mu\pi$ process using $[\![\cdot]\!]^2$. We illustrate these two steps for output and input processes.

Output Let us write $\mathcal{A}'^k_{\widetilde{x}}(\cdot)_g$ to denote the (partial) composition involved in the first step. Given $P = u_i!\langle w_j\rangle.Q$, we first obtain:

$$\mathcal{A}'^k_{\widetilde{x}}(u_i!\langle w_j\rangle.Q)_g = c_k?(\tilde{x}).u_i!\big\langle W\big\rangle.\overline{c_{k+3}}!\langle\tilde{x}\rangle \mid \mathcal{A}'^{k+3}_{\widetilde{x}}(Q\sigma)_g$$

where $\sigma = (u_i : S)\,?\,\{u_{i+1}/u_i\}\colon\{\}$ and

$$W = \lambda z_1.\,(\overline{c_{k+1}}!\langle\rangle \mid c_{k+1}?().z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid c_{k+2}?(x).(x\ \widetilde{w}))$$

P	$\mathcal{B}^k_{\tilde{x}}(P)$	
$u_i!\langle V\rangle.Q$	$c_k?(\widetilde{x}).u_i!\langle\widehat{V}\rangle.\overline{c_{k+l+1}}!\langle\widetilde{z}\rangle \mid \mathcal{B}^{k+l+1}_{\tilde{z}}(Q\sigma)$	$\widehat{V} = \mathcal{V}^{k+1}_{\tilde{y}}(V\sigma)$ $\widetilde{y} = \mathsf{fv}(V)$ $\quad \widetilde{z} = \mathsf{fv}(Q)$ $l = \lvert V\rvert$ $\quad \sigma = \mathsf{next}(u_i)$
$u_i?(y).Q$	$c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\widetilde{x}'\rangle \mid \mathcal{B}^{k+1}_{\tilde{x}'}(Q\sigma)$	$\widetilde{x}' = \mathsf{fv}(Q)$ $\sigma = \mathsf{next}(u_i)$
$V\,u_i$	$c_k?(\widetilde{x}).\mathcal{V}^{k+1}_{\tilde{x}}(V)\,\widetilde{m}$	$u_i : C$ $\quad \widetilde{m} = (u_i, \ldots, u_{i+\lvert\mathcal{G}(C)\rvert-1})$ $\widetilde{x} = \mathsf{fv}(V)$
$(\nu\, s : C)P'$	$(\nu\,\widetilde{s} : \mathcal{G}(C))\,\mathcal{B}^k_{\tilde{x}}(P'\sigma)$	$\widetilde{s} = (s_1, \ldots, s_{\lvert\mathcal{G}(C)\rvert})$ $\sigma = (C = S)\,?\,\{s_1\overline{s_1}/s\overline{s}\}\colon\{s_1/s\}$ $\quad \widetilde{x} = \mathsf{fv}(P')$
$Q \mid R$	$c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{y}\rangle.\overline{c_{k+l+1}}!\langle\widetilde{z}\rangle \mid \mathcal{B}^{k+1}_{\tilde{y}}(Q) \mid \mathcal{B}^{k+l+1}_{\tilde{z}}(R)$	$\widetilde{y} = \mathsf{fv}(Q)$ $\quad \widetilde{z} = \mathsf{fv}(R)$ $\quad l = \lvert Q\rvert$
$\mathbf{0}$	$c_k?().\mathbf{0}$	
V	$\mathcal{V}^k_{\tilde{x}}(V)$	
y	y	
$\lambda u : C^{\rightsquigarrow}.\,P$	$\lambda\widetilde{y}.\,(\nu\,\widetilde{c})(\overline{c_k}!\langle\widetilde{x}\rangle \mid \mathcal{B}^k_{\tilde{x}}(P\{y_1/y\}))$	$\widetilde{x} = \mathsf{fv}(V)$ $\widetilde{y} = (y_1, \ldots, y_{\lvert\mathcal{G}(C)\rvert})$ $\widetilde{c} = (\rightsquigarrow{=}\rightarrow)\,?\,(c_k, \ldots, c_{k+\lvert P\rvert-1})\colon\epsilon$

Table 1: The breakdown function for HO processes and values (core fragment from [2]).

We have that $\mathcal{A}'^k_{\tilde{x}}(u_i!\langle w_j\rangle.Q)_g$ is a process in μHO. The second step uses $[\![\cdot]\!]^2$ to convert it into the following $\mu\pi$ process:

$$c_k?(\widetilde{x}).(\nu\, a)(u_i!\langle a\rangle.(\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+3}_{\tilde{x}}(Q\sigma)_g \mid a?(y).y?(z_1).\overline{c_{k+1}}!\langle z_1\rangle \mid c_{k+1}?(z_1).z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid c_{k+2}?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle\widetilde{w}\rangle)))$$

The subprocess mimicking the output action on u_i is guarded by an input on c_k. Then, the output of w on u_i action is delegated to a different channel through several redirections: first a private name a is sent, then along a name for z_1 is received and so on; until, finally, the breakdown of w is sent on name s_1. These names are propagated through local trios. We can see that upon action on u_i unmodified context $\widetilde{x}$ is sent to breakdown of continuation Q.

We say names typed with tail-recursive type are *recursive names*. Another form of output is when both u_i and w_j are recursive. This case is similar to the one just discussed, and omitted from Tab. 2.

Input The breakdown of $u_i?(w).Q$ as follows:

$$c_k?(\widetilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\widetilde{x}, y\rangle \mid (\nu\, s_1)(c_{k+1}?(\widetilde{x}, y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\widetilde{x}\rangle \mid c_{k+2}?(y).(\nu\, s)(y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid c_{k+3}?(\widetilde{x}).(\nu\, a)(\overline{s_1}!\langle a\rangle.(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid a?(y').y'?(\widetilde{w}).(\overline{c_{k+4}}!\langle\widetilde{x}\rangle \mid \mathcal{A}^{k+4}_{\tilde{x}}(Q\{w_1/w\}\sigma)_g))))$$

The activation on c_k enables the input on u_i. After several redirections, the actual input of variables $\widetilde{w}$ is on a name received for y', which binds them in the decomposition of Q. Hence, context $\widetilde{x}$ does not get extended for an inductive call: it only gets extended locally (propagated by c_{k+1}). Indeed, in the core fragment, the context is always empty and propagators only enable subsequent actions. The context does play a role in breaking down recursion: variables z_X (generated to encode recursion) get propagated as context.

$$\begin{aligned} C &::= M \mid \langle M\rangle \\ \gamma &::= \mathsf{end} \mid \mathsf{t} \\ M &::= \gamma \mid !\langle\widetilde{C}\rangle;\gamma \mid ?(\widetilde{C});\gamma \mid \mu\mathsf{t}.M \end{aligned}$$

Figure 8: Minimal Session Types for π (cf. Definition 3.1)

3.2 Formal Definition

Definition 3.1 (Minimal Session Types, MSTs). *Minimal session types* for π are defined in Figure 8.

The breakdown function $\mathcal{A}^k_{\tilde{x}}(\cdot)_g$ for all constructs of π is given in Table 2, using the following definitions.

Definition 3.2 (Degree of a Process). The *degree* of a π process P, denoted $\lfloor P\rceil$, is defined as:

$$\lfloor u_i!\langle w_j\rangle.Q\rceil = \lfloor Q\rceil + 3 \quad \lfloor(\nu\, s : S)Q\rceil = \lfloor Q\rceil \quad \lfloor\mathbf{0}\rceil = 1$$
$$\lfloor u_i?(x : C).Q\rceil = \lfloor Q\rceil + 5 \quad \lfloor Q \mid R\rceil = \lfloor Q\rceil + \lfloor R\rceil + 1$$
$$\lfloor X\rceil = 4 \quad \lfloor\mu X.Q\rceil = \lfloor Q\rceil + 4$$

Definition 3.3 (Predicates on Types and Names). Let C be a session type. We write $\mathrm{tr}(C)$ to indicate that C is a tail-recursive session type. Also, given $u : C$, we write $\mathtt{lin}(u)$ if $C = S$ and $\neg\mathrm{tr}(S)$.

Definition 3.4 (Subsequent index substitution). Let n_i be an indexed name. We define $\mathsf{next}(n_i) = (\mathtt{lin}(n_i))\,?\,\{n_{i+1}/n_i\}\colon\{\}$.

We define how to obtain MSTs for π from standard session types:

Definition 3.5 (Decomposing First-Order Types). The decomposition function $\mathcal{H}(\cdot)$ on finite types, obtained by combining the

P	$\mathcal{A}^k_{\tilde{x}}(P)_g$	
$u_i!\langle w_j\rangle.Q$	$c_k?(\tilde{x}).(\nu\, a)\big(u_i!\langle a\rangle.\big(\overline{c_{k+3}}!\langle\tilde{x}\rangle \mid \mathcal{A}^{k+3}_{\tilde{x}}(Q\sigma)_g \mid$ $a?(y).y?(z_1).\overline{c_{k+1}}!\langle z_1\rangle \mid c_{k+1}?(z_1).z_1?(x).\overline{c_{k+2}}!\langle x\rangle \mid$ $c_{k+2}?(x).(\nu\, s)\big(x!\langle s\rangle.\tilde{s}!\langle\tilde{w}\rangle\big)\big)\big)$	$w_j : C$ $\tilde{w} = (w_j, \ldots, w_{j+\lvert\mathcal{H}(C)\rvert-1})$ $\sigma = \mathsf{next}(u_i)$
$u_i?(w).Q$	$c_k?(\tilde{x}).u_i?(y).\overline{c_{k+1}}!\langle\tilde{x}, y\rangle \mid$ $(\nu\, s_1)\big(c_{k+1}?(\tilde{x}, y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\tilde{x}\rangle \mid$ $c_{k+2}?(y).(\nu\, s)\big(y!\langle s\rangle.\tilde{s}!\langle s_1\rangle\big) \mid$ $c_{k+3}?(\tilde{x}).(\nu\, a)\big(\overline{s_1}!\langle a\rangle.\big(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid \widehat{Q}_{\tilde{x}}\big)\big)\big)$ where: $\widehat{Q}_{\tilde{x}} = a?(y').y'?(\tilde{w}).\big(\overline{c_{k+4}}!\langle\tilde{x}\rangle \mid \mathcal{A}^{k+4}_{\tilde{x}}(Q\{w_1/w\}\sigma)_g\big)$	$w : C$ $\tilde{w} = (w_1, \ldots, w_{\lvert\mathcal{H}(C)\rvert})$ $l = \lfloor Q\rceil$ $\sigma = \mathsf{next}(u_i)$
$r_i!\langle w_j\rangle.P$	$c_k?(\tilde{x}).(\nu\, a_1)c^r!\langle a_1\rangle.\big(\mathcal{A}^{k+3}_{\tilde{x}}(P)_g \mid a_1?(y_1).y_1?(\tilde{z}).W\big)$ where: $W = (\nu\, a_2)\big(z_{f(S)}!\langle a_2\rangle.\big(\overline{c_{k+3}}!\langle\tilde{x}\rangle.c^r?(b).(\nu\, s)\big(b!\langle s\rangle.\tilde{s}!\langle\tilde{z}\rangle\big) \mid$ $a_2?(y_2).y_2?(z_1').\overline{c_{k+1}}!\langle\rangle \mid$ $c_{k+1}?().z_1'?(x).\overline{c_{k+2}}!\langle x\rangle \mid c_{k+2}?(x).(\nu\, s')\big(x!\langle s'\rangle.\tilde{s}'!\langle\tilde{w}\rangle\big)\big)\big)\big)$	$r : S \wedge \mathsf{tr}(S)$ $\tilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}'^\star(S)\rvert})$ $\tilde{c} = (c_{k+1}, c_{k+2})$ $w : C \wedge \tilde{w} = (w_j, \ldots, w_{j+\lvert\mathcal{H}(C)\rvert-1})$
$r_i?(w).P$	$c_k?(\tilde{x}).(\nu\, a_1)\big(c^r!\langle a_1\rangle.\big((\nu\, s_1)\big(c_{k+1}?(y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\rangle \mid$ $c_{k+2}?(y).(\nu\, s)\big(y!\langle s\rangle.\tilde{s}!\langle s_1\rangle\big) \mid$ $c_{k+3}?().(\nu\, a_2)\big(s_1!\langle a_2\rangle.\big(\overline{c_{k+l+4}}!\langle\rangle \mid c_{k+l+4}?().\mathbf{0} \mid$ $a_2?(y_2).y_2?(\tilde{w}).\big(\overline{c_{k+4}}!\langle\tilde{x}\rangle \mid \mathcal{A}^{k+4}_{\tilde{x}}(P\{w_1/w\})_g\big)\big)\big) \mid$ $a_1?(y_1).y_1?(\tilde{z}).z_{f(S)}?(y).$ $\overline{c_{k+1}}!\langle y\rangle.c^r?(b).(\nu\, s')\big(b!\langle s'\rangle.\tilde{s}'!\langle\tilde{z}\rangle\big)\big)\big)\big)$	$r : S \wedge \mathsf{tr}(S)$ $\tilde{z} = (z_1, \ldots, z_{\lvert\mathcal{R}'^\star(S)\rvert})$ $l = \lfloor P\rceil$ $w : C \wedge \tilde{w} = (w_1, \ldots, w_{\lvert\mathcal{H}(C)\rvert})$
$(\nu\, s : C)P'$	$(\nu\, \tilde{s} : \mathcal{H}(C))\, \mathcal{A}^k_{\tilde{x}}(P'\sigma)_g$	$\tilde{s} = (s_1, \ldots, s_{\lvert\mathcal{H}(C)\rvert}) \quad \sigma = \{s_1\overline{s_1}/s\overline{s}\}$
$(\nu\, r : \mu\mathsf{t}.S)P'$	$(\nu\, \tilde{r} : \mathcal{R}'(S))\, c^r?(b).(\nu\, s')\big(b!\langle s'\rangle.\tilde{s}'!\langle\tilde{r}\rangle\big) \mid c^{\overline{r}}?(b).(\nu\, s')\big(b!\langle s'\rangle.\tilde{s}'!\langle\tilde{\overline{r}}\rangle\big) \mid \mathcal{A}^k_{\tilde{x}}(P'\sigma)_g$	$\mathsf{tr}(\mu\mathsf{t}.S) \qquad \sigma = \{r_1\overline{r_1}/r\overline{r}\}$ $\tilde{r} = (r_1, \ldots, r_{\lvert\mathcal{R}'(S)\rvert})$ $\tilde{\overline{r}} = (\overline{r}_1, \ldots, \overline{r}_{\lvert\mathcal{R}'(S)\rvert})$
$Q_1 \mid Q_2$	$c_k?(\tilde{x}).\overline{c_{k+1}}!\langle\tilde{y}\rangle.\overline{c_{k+l+1}}!\langle\tilde{z}\rangle \mid \mathcal{A}^{k+1}_{\tilde{y}}(Q_1)_g \mid \mathcal{A}^{k+l+1}_{\tilde{z}}(Q_2)_g$	$\tilde{y} = \mathsf{fv}(Q_1) \quad \tilde{z} = \mathsf{fv}(Q_2) \quad l = \lfloor Q\rceil$
$\mathbf{0}$	$c_k?().\mathbf{0}$	
$\mu X.P$	$(\nu\, s_1)\big(c_k?(\tilde{x}).\overline{c_{k+1}}!\langle\tilde{x}\rangle.\overline{c_{k+3}}!\langle\tilde{x}\rangle \mid$ $c_{k+1}?(\tilde{x}).(\nu\, a_1)\big(s_1!\langle a_1\rangle.\big(\overline{c_{k+2}}!\langle\rangle \mid c_{k+2}?().\mathbf{0} \mid$ $c_{k+3}?(\tilde{x}).s_1?(z_x).\overline{c_{k+4}}!\langle\tilde{x}, z_x\rangle \mid$ $\mathcal{A}^{k+4}_{\tilde{x}, z_x}(P)_{g,\{X\to\tilde{n}\}} \mid$ $* a_1?(y_1').y_1'?(\lVert\tilde{n}^1\rVert, \ldots, \lVert\tilde{n}^m\rVert, y_1).\widehat{P}\big)\big)$ where: $\widehat{P} = (\nu\, \tilde{c})\big(\prod_{0<i\le m} c^{n_i}?(b).(\nu\, s')\big(b!\langle s'\rangle.\tilde{s}'!\langle\lVert\tilde{n}^i\rVert\rangle\big) \mid \overline{c_{k+2}}!\langle\tilde{x}\rangle$ $c_{k+2}?(\tilde{x}).y_1?(z_x).\overline{c_{k+3}}!\langle\tilde{x}, z_x\rangle \mid \big[\!\big[\mathcal{A}^{k+3}_{\tilde{x}, z_x}(P)_{g,\{X\to\tilde{n}\}}\big]\!\big]_{\tilde{c},\tilde{c}_r}\big)$	$\tilde{n} = \mathsf{fn}(P)$ $m = \lvert\tilde{n}\rvert$ $\lVert\tilde{n}\rVert = (\lVert n_1\rVert, \ldots, \lVert n_m\rVert)$ $i \in \{1, \ldots, m\}.$ $\lVert n_i\rVert : S_i$ $\lVert\tilde{n}^i\rVert = (\lVert n_1^i\rVert, \ldots, \lVert n^i_{\lvert\mathcal{H}(S_i)\rvert}\rVert)$ $\tilde{c} = (c_{k+2}, \ldots, c_{k+\lfloor\lVert P\rVert^1_{g,\{X\to\tilde{n}\}}\rceil+1})$ $\tilde{c}_r = \bigcup_{v\in\tilde{n}} c^v$
X	$(\nu\, s_1)\big(c_k?(z_x).\overline{c_{k+1}}!\langle z_x\rangle.\overline{c_{k+2}}!\langle z_x\rangle \mid c_{k+2}?(z_x).\bar{s}_1!\langle z_x\rangle.\overline{c_{k+3}}!\langle\rangle \mid c_{k+3}?().\mathbf{0}$ $c_{k+1}?(z_x).(\nu\, a_1)\big(c^{n_1}!\langle a_1\rangle.\big(\, a_1?(y_1).y_1?(\tilde{z}_1).\ldots.(\nu\, a_j)Q\big)\big)\big)$ where: $Q = \big(c^{n_j}!\langle a_j\rangle.\big(a_j?(y_j).y_j?(\tilde{z}_j).(\nu\, s')\big(z_x!\langle s'\rangle.\tilde{s}'!\langle\tilde{z}_1, \ldots, \tilde{z}_j, s_1\rangle\big)\big)\big)$	$\tilde{n} = g(X)$ $\lvert\tilde{n}\rvert = j$ $i \in \{1, \ldots, j\}$ $n_i : S \wedge \mathsf{tr}(S_i)$ $\tilde{z}_i = (z_1^i, \ldots, z^i_{\lvert\mathcal{R}'^\star(S_i)\rvert})$

Table 2: Decompose by composition: Breakdown function $\mathcal{A}^k_{\tilde{x}}(\cdot)_g$ for π processes (cf. Definition 3.6).

mappings $(\!|\cdot|\!)^1$, $\mathcal{G}(\cdot)$, and $(\!|\cdot|\!)^2$, is defined in Fig. 9 (top, where omitted cases are defined homomorphically). It is extended to account for recursive session types in Fig. 9 (center).

The auxiliary function $\mathcal{R}'^\star(\cdot)$, given in Fig. 9 (bottom), is used in Tab. 2 to decompose *guarded* tail-recursive types: it skips session prefixes until a type of form $\mu\mathsf{t}.S$ is encountered; when that occurs, the recursive type is decomposed using $\mathcal{R}'(\cdot)$.

We are finally ready to define the decomposition function $\mathcal{F}(\cdot)$, the analog of Definition 2.3 but for processes in π:

Definition 3.6 (Process Decomposition). Let P be a closed π process with $\tilde{u} = \mathsf{fn}(P)$ and $\tilde{v} = \mathsf{rn}(P)$. Given the breakdown function $\mathcal{A}^k_{\tilde{x}}(\cdot)_g$ in Table 2, the decomposition $\mathcal{F}(P)$ is defined as:

$$\mathcal{F}(P) = (\nu\, \tilde{c})(\nu\, \tilde{c}_r)\Big(\prod_{r\in\tilde{v}} P^r \mid \overline{c_k}!\langle\rangle.\mathbf{0} \mid \mathcal{A}^k_{\epsilon}(P\sigma)_g\Big)$$

$$\mathcal{H}(\langle S\rangle) = \langle \mathcal{H}(S)\rangle$$

$$\mathcal{H}(!\langle S\rangle; S') = \begin{cases} M & \text{if } S' = \mathsf{end} \\ M, \mathcal{H}(S') & \text{otherwise} \end{cases}$$

where $M = !\big\langle \langle ?(?(\langle ?(?(\mathcal{H}(S)); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle \big\rangle; \mathsf{end}$

$$\mathcal{H}(?(S); S') = \begin{cases} M & \text{if } S' = \mathsf{end} \\ M, \mathcal{H}(S') & \text{otherwise} \end{cases}$$

where $M = ?(\langle ?(?(\langle ?(?(\mathcal{H}(S)); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle); \mathsf{end}$

$$\mathcal{H}(\mathsf{end}) = \mathsf{end}$$

$$\mathcal{H}(S_1, \ldots, S_n) = \mathcal{H}(S_1), \ldots, \mathcal{H}(S_n)$$

$$\mathcal{H}(\mu \mathsf{t}.S) = \begin{cases} \mathcal{R}'(S) & \text{if } \mu \mathsf{t}.S \text{ is tail-recursive} \\ \mu \mathsf{t}.\mathcal{H}(S) & \text{otherwise} \end{cases}$$

$$\mathcal{R}'(!\langle S\rangle; S') = \mu \mathsf{t}.!\big\langle \langle ?(?(\langle ?(?(\mathcal{H}(S)); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle \big\rangle; \mathsf{t}, \mathcal{R}'(S')$$

$$\mathcal{R}'(?(S); S') = \mu \mathsf{t}.?\big(\langle ?(?(\langle ?(?(\mathcal{H}(S)); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle \big); \mathsf{t}, \mathcal{R}'(S')$$

$$\mathcal{H}(\mathsf{t}) = \mathsf{t} \qquad \mathcal{R}'(\mathsf{t}) = \epsilon$$

$$\mathcal{R}'^{\star}(?(S); S') = \mathcal{R}'^{\star}(S') \qquad \mathcal{R}'^{\star}(!\langle S\rangle; S') = \mathcal{R}'^{\star}(S')$$

$$\mathcal{R}'^{\star}(\mu \mathsf{t}.S) = \mathcal{R}'^{\star}(S)$$

Figure 9: Decomposition of types $\mathcal{H}(\cdot)$ (cf. Def. 3.5)

where: $k > 0, \widetilde{c} = (c_k, \ldots, c_{k+\lfloor P\rceil-1}); \widetilde{c_r} = \bigcup_{r\in\widetilde{v}} c^r; \sigma = \{\mathsf{init}(\tilde{u})/\widetilde{u}\}$; $P^r = c^r?(b).(\nu\, s)(b!\langle s\rangle.\overline{s}!\langle \overline{r}\rangle)$ with $r : S$ and $\widetilde{r} = r_1, \ldots, r_{|\mathcal{G}(S)|}$.

3.3 Examples

Example 3.7 (A Process with Delegation). Consider a process P that implements channels $\overline{w}$ (with type $T = ?(\mathsf{Int}); !\langle \mathsf{Bool}\rangle; \mathsf{end}$) and u (with type $S = !\langle \overline{T}\rangle; \mathsf{end}$):

$$P = (\nu\, u : S)(\underbrace{u!\langle w\rangle.\overline{w}?(t).\overline{w}!\langle \mathsf{odd}(t)\rangle.\mathbf{0}}_{A} \mid \underbrace{\overline{u}?(x).x!\langle 5\rangle.x?(b).\mathbf{0}}_{B})$$

By Def. 3.2, $\lfloor P\rceil = 25$. Then, the decomposition of P into a collection of first-order processes typed with minimal session types is:

$$\mathcal{F}(P) = (\nu\, c_1, \ldots, c_{25})(\overline{c_1}!\langle\rangle.\mathbf{0} \mid (\nu\, u_1)\mathcal{A}^1_\epsilon((A \mid B)\sigma')),$$

where $\sigma = \mathsf{init}(\mathsf{fn}(P))$ and $\sigma' = \sigma \cdot \{u_1\overline{u_1}/u\overline{u}\}$. We omit parameter g as it is empty. We have:

$$\mathcal{A}^1_\epsilon((A \mid B)\sigma') = c_1?().\overline{c_2}!\langle\rangle.\overline{c_{13}}!\langle\rangle \mid \mathcal{A}^2_\epsilon(A\sigma') \mid \mathcal{A}^{13}_\epsilon(B\sigma')$$

We use the following abbreviations for subprocesses of A and B: $A' = \overline{w}_1?(t).A''$, $A'' = \overline{w}_1!\langle \mathsf{odd}(t)\rangle.\mathbf{0}$, and $B' = x_1!\langle 5\rangle.x_1?(b).\mathbf{0}$. The breakdown of A is in Fig. 10; the breakdown of B follows:

$$\begin{aligned}\mathcal{A}^{13}_\epsilon(B) = {} & c_{13}?().\overline{u}_1?(y_4).\overline{c_{14}}!\langle y_4\rangle \mid (\nu\, s_1)(c_{14}?(y).\overline{c_{15}}!\langle y\rangle.\overline{c_{16}}!\langle\rangle \mid \\ & c_{15}?(y_4).(\nu\, s'')(y_4!\langle s''\rangle.\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid c_{16}?().(\nu\, a_3)(s_1!\langle a_3\rangle. \\ & (\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B')))))\end{aligned}$$

$$\begin{aligned}\mathcal{A}^2_\epsilon(A) = {} & c_2?().(\nu\, a_1)(u_1!\langle a_1\rangle.(\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ & a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ & c_4?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle)))\end{aligned}$$

$$\begin{aligned}\mathcal{A}^5_\epsilon(A') = {} & c_5?().\overline{w_1}?(y_2).\overline{c_6}!\langle y_2\rangle \mid (\nu\, s_1)(c_6?(y_2).\overline{c_7}!\langle y_2\rangle.\overline{c_8}!\big\langle\big\rangle \mid \\ & c_7?(y_2).(\nu\, s')(y_2!\langle s'\rangle.\overline{s'}!\langle s_1\rangle.\mathbf{0})) \mid \\ & c_8?().(\nu\, a_2)(\overline{s_1}!\langle a_2\rangle.(\overline{c_{10}}!\langle\rangle \mid c_{10}?().\mathbf{0} \mid \\ & a_2?(y_3).y_3?(t_1).(\overline{c_9}!\langle\rangle \mid \mathcal{A}^9_\epsilon(A''))))\end{aligned}$$

$$\begin{aligned}\mathcal{A}^9_\epsilon(A'') = {} & c_9?().(\nu\, a)(\overline{w}_2!\langle a\rangle.(\overline{c_{12}}!\langle\rangle \mid c_{12}?().\mathbf{0} \mid \\ & a?(y).y?(z_1).\overline{c_{11}}!\langle z_1\rangle \mid c_{10}?(z_1).z_1?(x).\overline{c_{11}}!\langle x\rangle \mid \\ & c_{11}?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle \mathsf{odd}(t)\rangle)))\end{aligned}$$

Figure 10: Breakdown for process A in Exam. 3.7.

The breakdown of B' is similar. Type S is broken down into MSTs M_1 and M_2, as follows:

$$M_1 = ?\big(\langle ?(?(\langle ?(?(\mathsf{Int}); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle\big); \mathsf{end}$$

$$M_2 = !\big\langle \langle ?(?(\langle ?(?(\mathsf{Bool}); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle \big\rangle; \mathsf{end}$$

Names $\overline{w}_1$ and $\overline{w}_2$ are typed with M_1 and M_2, respectively. Then, name u_1 is typed with M, given by:

$$M = !\big\langle \langle ?(?(\langle ?(?(\overline{M}_1, \overline{M}_2); \mathsf{end}\rangle); \mathsf{end}); \mathsf{end})\rangle \big\rangle; \mathsf{end}$$

Consider the reductions of $\mathcal{F}(P)$ that mimic the exchange of w along u in P. We first have three synchronizations on c_1, c_2, c_{13}:

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^3 {} & (\nu\, \widetilde{c})(\,(\nu\, a_1)(\,u_1!\langle a_1\rangle.\,(\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ & a_1?(y_1).y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ & c_4?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle)))) \mid \overline{u}_1?(y_4).\,\overline{c_{14}}!\langle y_4\rangle \mid \\ & (\nu\, s_1)(c_{14}?(y).\overline{c_{15}}!\langle y\rangle.\overline{c_{16}}!\langle\rangle \mid \\ & c_{15}?(y_4).(\nu\, s'')(y_4!\langle s''\rangle.\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid \\ & c_{16}?().(\nu\, a_3)(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\ & c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))\end{aligned}$$

where $\widetilde{c} = (c_3, \ldots, c_{12}, c_{14}, \ldots, c_{25})$. Then, a synchronization on u_1 sends name a_1 (highlighted above). Name a_1 is further propagated along c_{14} and c_{15}. Another synchronization occurs on c_{16}.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^7 {} & (\nu\, \widetilde{c}_*)(\nu\, a_1)(\,\overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \,a_1?(y_1).\, y_1?(z_1).\overline{c_3}!\langle z_1\rangle \mid \\ & c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid c_4?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ & (\nu\, s_1)((\nu\, s'')(\,a_1!\langle s''\rangle.\,\overline{s''}!\langle s_1\rangle.\mathbf{0}) \mid (\nu\, a_3)(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid \\ & c_{21}?().\mathbf{0} \mid a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))))\end{aligned}$$

where $\widetilde{c}_* = (c_3, \ldots, c_{12}, c_{17}, \ldots, c_{25})$

The next reduction communicates session name s'' along a_1:

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{8} (\nu\, \widetilde{c}_*)(\nu\, s'')(\ \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ s''?(z_1).\ \overline{c_3}!\langle z_1\rangle \mid c_3?(z_1).z_1?(x).\overline{c_4}!\langle x\rangle \mid \\ c_4?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ (\nu\, s_1)(\ \overline{s''}!\langle s_1\rangle.\ \mathbf{0} \mid (\nu\, a_3)(s_1!\langle a_3\rangle.(\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid \\ a_3?(y_5).y_5?(x_1, x_2).(\ \overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))))\end{aligned}$$

After the synchronization on channel s'', name z_1 is further sent to the next parallel process through the propagator c_3:

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{10} (\nu\, \widetilde{c}_{**})(\nu\, s_1)(\ \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ s_1?(x).\ \overline{c_4}!\langle x\rangle \mid c_4?(x).(\nu\, s)(x!\langle s\rangle.\overline{s}!\langle w_1, w_2\rangle) \mid \\ (\nu\, a_3)(\ s_1!\langle a_3\rangle.\ (\overline{c_{21}}!\langle\rangle \mid c_{21}?().\mathbf{0} \mid \\ a_3?(y_5).y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))))) \\ \text{where } \widetilde{c}_{**} = (c_4, \ldots, c_{12}, c_{17}, \ldots, c_{25})\end{aligned}$$

Communication on s_1 leads to variable x being substituted by name a_3, which is then passed on c_4 to the next process. In addition, inaction is simulated by a synchronization on c_{21}.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{13} \\ (\nu\, \widetilde{c}_\bullet)(\nu\, a_3)(\ \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid (\nu\, s)(\ a_3!\langle s\rangle.\ \overline{s}!\langle w_1, w_2\rangle) \mid \\ a_3?(y_5).\ y_5?(x_1, x_2).(\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))) \\ \text{where } \widetilde{c}_\bullet = (c_5, \ldots, c_{12}, c_{17}, \ldots, c_{25})\end{aligned}$$

Now, the passing of the decomposition of w is finally simulated by two reductions: first, a synchronization on a_3 sends the endpoint of session s, which replaces variable y_5; then, the dual endpoint is used to send w_1, w_2, substituting variables x_1, x_2 in $\mathcal{A}^{17}_\epsilon(B')$.

$$\begin{aligned}\mathcal{F}(P) \longrightarrow^{14} (\nu\, \widetilde{c}_{\bullet\bullet})(\nu\, s)(\ \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ \overline{s}!\langle w_1, w_2\rangle \mid s?(x_1, x_2).\ (\overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B'))) \\ \mathcal{F}(P) \longrightarrow^{15} (\nu\, \widetilde{c}_{\bullet\bullet})(\ \overline{c_5}!\langle\rangle \mid \mathcal{A}^5_\epsilon(A') \mid \\ \overline{c_{17}}!\langle\rangle \mid \mathcal{A}^{17}_\epsilon(B')\{w_1 w_2/x_1 x_2\}) = Q\end{aligned}$$

Above, $\widetilde{c}_{\bullet\bullet} = (c_5, \ldots, c_{12}, c_{17}, \ldots, c_{25})$. This is how $\mathcal{F}(P)$ simulates the first action of P. Notice that in Q names w_1, w_2 substitute x_1, x_2 and the first synchronization on w can be simulated on name w_1.

Example 3.8 (A Recursive Process). Let $P = \mu X.P'$ be a process implementing a channel r with the tail-recursive session type $S = \mu \mathrm{t}.?(\mathtt{Int});!\langle\mathtt{Int}\rangle;\mathrm{t}$, with $P' = r?(w).r!\langle -w\rangle.X$. We decompose r using S and obtain two channels typed with MSTs as in Fig. 9:

$$\begin{aligned}r_1 &: \mu \mathrm{t}.?\big(\langle ?(?(\langle ?(\mathtt{Int});\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\rangle\big);\mathrm{t} \\ r_2 &: \mu \mathrm{t}.!\big\langle\langle ?(?(\langle ?(\mathtt{Int});\mathsf{end}\rangle);\mathsf{end});\mathsf{end}\big\rangle\big\rangle;\mathrm{t}\end{aligned}$$

Then, process $\mathcal{F}(P)$ is

$$(\nu\, \widetilde{c})(\nu\, c^r)(c^r?(b).(\nu\, s)(b!\langle s\rangle.\overline{s}!\langle r_1, r_2\rangle) \mid \overline{c_1}!\langle\rangle \mid \mathcal{A}^1_\epsilon(P\{r_1/r\})_\emptyset)$$

where $\widetilde{c} = (c_1, \ldots, c_{\lfloor P'\rceil})$ and $\mathcal{A}^1_\epsilon(P\{r_1/r\})_\emptyset$ is in Fig. 11.

In Fig. 11, $\mathcal{A}^1_\epsilon(P\{r_1/r\})$ simulates recursion in P using replication. Given some index k, process R^k mimics actions of the recursive body. It first gets a decomposition of r by interacting with the process providing recursive names on c^r (for the first instance, this is a top-level process in $\mathcal{F}(P)$). Then, it mimics the first input action on the channel received for z_1 (that is, r_1): the input of actual names for w_1 is delegated through channel redirections to name y_2 (both prefixes are highlighted in Fig. 11). Once the recursive name is used, the decomposition of recursive name is made available for the breakdown of the continuation by a communication on c^r. Similarly, in the continuation, the second action on r, output, is mimicked by r_2 (received for z_2), with the output of actual name w_1 delegated to $\overline{s}'$ (both prefixes are highlighted in Fig. 11).

Subprocess R^5 is a breakdown of the first instance of the recursive body. The replication guarded by a_1 produces a next instance, i.e., process $R^2\{x_{r_1}, x_{r_2}/r_1, r_2\}$ in $\widehat{P}$. By communication on a_1 and a few reductions on propagators, it gets activated: along a_1 it first receives a name for y_1' along which it also receives: (i) recursive names

$$\begin{aligned}\mathcal{A}^1_\epsilon(P\{r_1/r\})_\emptyset = (\nu\, s_1)(c_1?().\overline{c_2}!\langle\rangle.\overline{c_4}!\langle\rangle \mid \\ c_2?().(\nu\, a_1)(s_1!\langle a_1\rangle.(\overline{c_3}!\langle\rangle \mid c_3?().\mathbf{0} \mid \\ c_4?().s_1?(z_x).\overline{c_5}!\langle z_x\rangle \mid \\ R^5 \mid *\ a_1?(y_1').y_1'?(x_{r_1}, x_{r_2}, y_1).\widehat{P})))\end{aligned}$$

where:

$$\begin{aligned}\widehat{P} = (\nu\, \widetilde{c})(c^r?(b).(\nu\, s')(b!\langle s'\rangle.\overline{s}'!\langle x_{r_1}, x_{r_2}\rangle) \mid \overline{c_1}!\langle\rangle \mid \\ c_1?().y_1?(z_x).\overline{c_2}!\langle z_x\rangle \mid R^2\{x_{r_1}, x_{r_2}/r_1, r_2\}) \\ R^k = c_k?(z_x). \\ (\nu\, a_1)(c^r!\langle a_1\rangle.((\nu\, s_1)(c_{k+1}?(y).\overline{c_{k+2}}!\langle y\rangle.\overline{c_{k+3}}!\langle\rangle \mid \\ c_{k+2}?(y).(\nu\, s)(y!\langle s\rangle.\overline{s}!\langle s_1\rangle) \mid \\ c_{k+3}?().(\nu\, a_2)(s_1!\langle a_2\rangle.(\overline{c_{k+l+4}}!\langle\rangle \mid \\ c_{k+l+4}?().\mathbf{0} \mid a_2?(y_2).\ y_2?(w_1). \\ (\nu\, \widetilde{c})(\overline{c_{k+4}}!\langle z_x\rangle \mid \mathcal{A}^{k+4}_{z_x}(r_2!\langle -w_1\rangle.X)_g))) \mid \\ a_1?(y_1).y_1?(z_1, z_2).\ z_1?(y). \\ \overline{c_{k+1}}!\langle y\rangle.c^r?(b).(\nu\, s')(b!\langle s'\rangle.\overline{s}'!\langle z_1, z_2\rangle)))) \\ \mathcal{A}^{k+4}_{z_x}(r_2!\langle -w_1\rangle.X)_g = c_k?(z_x).(\nu\, a_1)c^r!\langle a_1\rangle. \\ (\mathcal{A}^{k+7}_{z_x}(X)_g \mid a_1?(y_1).y_1?(\widetilde{z}).W) \\ W = (\nu\, a_2)(\ z_2!\langle a_2\rangle.\ (\overline{c_{k+7}}!\langle z_x\rangle.c^r?(b).(\nu\, s)(b!\langle s\rangle.\overline{s}!\langle\widetilde{z}\rangle) \mid \\ a_2?(y_2).y_2?(z_1').(\nu\, \widetilde{c})(\overline{c_{k+5}}!\langle\rangle \mid \\ c_{k+5}?().z_1'?(x).\overline{c_{k+6}}!\langle x\rangle \mid \\ c_{k+6}?(x).(\nu\, s')(x!\langle s'\rangle.\ \overline{s}'!\langle -w1\rangle\ ))) \\ \mathcal{A}^{k+7}_{z_x}(X)_g = (\nu\, s_1)(c_{k+7}?(z_x).\overline{c_{k+8}}!\langle z_x\rangle.\overline{c_{k+9}}!\langle z_x\rangle \mid \\ c_{k+8}?(z_x).(\nu\, a_1)(c^r!\langle a_1\rangle.(c_{k+9}?(z_x).\overline{s}_1!\langle z_x\rangle. \mid \\ \overline{c_{k+10}}!\langle\rangle c_{k+10}?().\mathbf{0} \mid \\ a_1?(y_1).y_1?(r_1, r_2). \\ (\nu\, s')(z_x!\langle s'\rangle.\overline{s}'!\langle r_1, r_2, s_1\rangle)))))))\end{aligned}$$

with $g = \{X \mapsto r_1, r_2\}$

Figure 11: Breakdown of recursive process (Exam. 3.8)

r_1, r_2 for variables x_{r_1}, x_{r_2}, and (ii) a name for y_1 along which it will receive a_1 again, for future instances, as it can be seen in $\mathcal{A}^{k+7}_{z_x}(X)_g$.

3.4 Results

We establish the minimality result for π using the typability of $\mathcal{F}(\cdot)$. We need some auxiliary definitions to characterize the propagators required to decompose recursive processes.

Theorems 3.10 and 3.27 in [2] state typability results by introducing two typing environments, denoted Θ and Φ. While environment Θ is used to type linear propagators (e.g., $c_k, c_{k+1}, \ldots$) generated by the breakdown function $\mathcal{B}^{-}(\cdot)$, environment Φ types shared propagators used in trios that propagate breakdown of recursive names (e.g., $c^r, c^v, \ldots$ where r and v are recursive names).

Definition 3.9 (Session environment for propagators). Let Θ be the session environment and Φ be the recursive propagator environment defined in Theorem 3.10 and Theorem 3.27 [2], respectively. Then, by applying the encoding $(\!|\cdot|\!)^2$, we define Θ' and Φ' as follows: $\Theta' = (\!|\Theta|\!)^2, \Phi' = (\!|\Phi|\!)^2$.

We can use $\Theta' = (\!|\Theta|\!)^2$ in the following statement, where we state the typability result for the breakdown function.

Theorem 3.10 (Typability of Breakdown). *Let P be an initialized π process. If $\Gamma; \Delta, \Delta_\mu \vdash P \triangleright \diamond$, then*

$$\mathcal{H}(\Gamma'), \Phi'; \mathcal{H}(\Delta), \Theta' \vdash \mathcal{A}^k_{\tilde{\epsilon}}(P)_g \triangleright \diamond$$

where $k > 0$; $\widetilde{r} = \mathsf{dom}(\Delta_\mu)$; $\Phi' = \prod_{r \in \tilde{r}} c^r : \langle\langle ?(\mathcal{R}'^{\star}(\Delta_\mu(r))); \mathsf{end}\rangle\rangle$; and $\mathsf{balanced}(\Theta')$ with

$$\mathsf{dom}(\Theta') = \{c_k, c_{k+1}, \ldots, c_{k+\lfloor P \rceil - 1}\} \cup \{\overline{c_{k+1}}, \ldots, \overline{c_{k+\lfloor P \rceil - 1}}\}$$

such that $\Theta'(c_k) = ?(\cdot); \mathsf{end}$.

Proof. Directly by using Theorem 5.1 [11], Theorem 3.27 [2], and Theorem 5.2 [11]. See [1] for details. □

We now consider typability for the decomposition function, using $\Phi' = (\!|\Phi|\!)^2$ as in Def. 3.9. The proof follows from Thm. 3.10; see [1].

Theorem 3.11 (Minimality Result for π). *Let P be a closed π process, with $\widetilde{u} = \mathsf{fn}(P)$ and $\widetilde{v} = \mathsf{rn}(P)$. If $\Gamma; \Delta, \Delta_\mu \vdash P \triangleright \diamond$, where Δ_μ only involves recursive session types, then $\mathcal{H}(\Gamma\sigma); \mathcal{H}(\Delta\sigma), \mathcal{H}(\Delta_\mu\sigma) \vdash \mathcal{F}(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

4 OPTIMIZATIONS

Although conceptually simple, the composition approach to decomposition induces redundancies. Here we propose $\mathcal{F}^*(\cdot)$, an optimization of the decomposition $\mathcal{F}(\cdot)$, and establish its static and dynamic correctness, in terms of the minimality result (cf. Thm. 4.14) but also *operational correspondence* (cf. Thm. 4.20), respectively.

4.1 Motivation

To motivate our insights, consider the process $\mathcal{A}^k_{\tilde{x}}(u_i?(w).Q)_g$ as presented in § 3.1 and Tab. 2. We identify some suboptimal features of this decomposition: (i) channel redirections; (ii) redundant synchronizations on propagators; (iii) the structure of trios is lost.

While an original process P receives a name for variable w along u_i, its breakdown does not input a breakdown of w directly, but through a series of channel redirections: u_i receives a name along which it sends restricted name s, along which it sends the restricted name s_1 and so on. Finally, the name received for y' receives $\widetilde{w}$, the breakdown of w. This redundancy is perhaps more evident in Def. 3.5, which gives the translation of types by composition: the mimicked input action is five-level nested for the original name. This is due to the composition of $[\![\cdot]\!]^1_g$ and $[\![\cdot]\!]^2$.

Also, $\mathcal{A}^k_{\tilde{x}}(u_i?(w).Q)_g$ features redundant communications on propagators. For example, the bound name y is locally propagated by c_{k+1} and c_{k+2}. This is the result of breaking down sequential prefixes induced by $[\![\cdot]\!]^1_g$ (not present in the original process). Last but not least, the trio structure is lost as subprocess $\widehat{Q}_{\tilde{x}}$ is guarded and nested, and it inductively invokes the function on continuation Q. This results in an arbitrary level of process nesting, which is induced by the final application of encoding $[\![\cdot]\!]^2$ in the composition.

The non-optimality of $\mathcal{A}^k_{\tilde{x}}(\cdot)_g$ is more prominent in the treatment of recursive processes and recursive names. As HO does not feature recursion constructs, $[\![\cdot]\!]^1_g$ encodes recursive behaviors by relying on abstraction passing and shared abstractions. Then, going back to π via $[\![\cdot]\!]^2_g$, this is translated to a process involving a replicated subprocess. But going through this path, the encoding of recursive process becomes convoluted. On top of that, all non-optimal features of the core fragment (as discussed for the case of input) are also present in the decomposition of recursion.

Here we develop an optimized decomposition function, denoted $\mathcal{F}^*(\cdot)$ (Def. 4.8), that avoids the redundancies described above. The optimized decomposition produces a composition of *trios processes*, with a fixed maximum number of nested prefixes. The decomposed process does not redirect channels and only introduces propagators that codify the sequentiality of the original process.

4.2 Preliminaries

We decompose a session type into a *list* of minimal session types:

Definition 4.1 (Decomposing Types). Let S and C be a session and a channel type, resp. (cf. Fig. 4). The *type decomposition function* $\mathcal{H}^*(\cdot)$ is defined in Figure 12.

Example 4.2 (Decomposing a Recursive Type). Let $S = \mu\mathsf{t}.S'$ be a recursive session type, with $S' = ?(\mathsf{Int}); ?(\mathsf{Bool}); !\langle\mathsf{Bool}\rangle; \mathsf{t}$. By Fig. 12, since S is tail-recursive, $\mathcal{H}^*(S) = \mathcal{R}(S')$. Further,

$$\mathcal{R}(S') = \mu\mathsf{t}.?(\mathcal{H}^*(\mathsf{Int})); \mathsf{t}, \mathcal{R}(?(\mathsf{Bool}); !\langle\mathsf{Bool}\rangle; \mathsf{t})$$

By definition of $\mathcal{R}(\cdot)$, we obtain

$$\mathcal{H}^*(S) = \mu\mathsf{t}.?(\mathsf{Int}); \mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool}); \mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle; \mathsf{t}, \mathcal{R}(\mathsf{t})$$

(using $\mathcal{H}^*(\mathsf{Int}) = \mathsf{Int}$ and $\mathcal{H}^*(\mathsf{Bool}) = \mathsf{Bool}$). Since $\mathcal{R}(\mathsf{t}) = \epsilon$, we have

$$\mathcal{H}^*(S) = \mu\mathsf{t}.?(\mathsf{Int}); \mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool}); \mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle; \mathsf{t}$$

Example 4.3 (Decomposing an Unfolded Recursive Type). Let $T = ?(\mathsf{Bool}); !\langle\mathsf{Bool}\rangle; S$ be a derived unfolding of S from Exam. 4.2. Then, by Fig. 12, $\mathcal{R}^\star(T)$ is the list of minimal recursive types obtained as follows: first, $\mathcal{R}^\star(T) = \mathcal{R}^\star(!\langle\mathsf{Bool}\rangle; \mu\mathsf{t}.S')$ and after one more step, $\mathcal{R}^\star(!\langle\mathsf{Bool}\rangle; \mu\mathsf{t}.S') = \mathcal{R}^\star(\mu\mathsf{t}.S')$. Finally, we have $\mathcal{R}^\star(\mu\mathsf{t}.S') = \mathcal{R}(S')$. We get the same list of minimal types as in Exam. 4.2:

$$\mathcal{R}^\star(T) = \mu\mathsf{t}.?(\mathsf{Int}); \mathsf{t}, \mu\mathsf{t}.?(\mathsf{Bool}); \mathsf{t}, \mu\mathsf{t}.!\langle\mathsf{Bool}\rangle; \mathsf{t}$$

$$
\begin{aligned}
\mathcal{H}^*(\mathsf{end}) &= \mathsf{end}\\
\mathcal{H}^*(\langle S\rangle) &= \langle \mathcal{H}^*(S)\rangle\\
\mathcal{H}^*(S_1,\ldots,S_n) &= \mathcal{H}^*(S_1),\ldots,\mathcal{H}^*(S_n)\\
\mathcal{H}^*(!\langle C\rangle;S) &= \begin{cases} !\langle \mathcal{H}^*(C)\rangle;\mathsf{end} & \text{if } S=\mathsf{end}\\ !\langle \mathcal{H}^*(C)\rangle;\mathsf{end}\,,\mathcal{H}^*(S) & \text{otherwise}\end{cases}\\
\mathcal{H}^*(?(C);S) &= \begin{cases} ?(\mathcal{H}^*(C));\mathsf{end} & \text{if } S=\mathsf{end}\\ ?(\mathcal{H}^*(C));\mathsf{end}\,,\mathcal{H}^*(S) & \text{otherwise}\end{cases}\\
\mathcal{H}^*(\mu \mathsf{t}.S') &= \mathcal{R}(S')\\
\mathcal{H}^*(S) &= \mathcal{R}^\star(S) \quad \text{where } S \neq \mu \mathsf{t}.S'\\
\mathcal{R}(\mathsf{t}) &= \epsilon\\
\mathcal{R}(!\langle C\rangle;S) &= \mu \mathsf{t}.!\langle \mathcal{H}^*(C)\rangle;\mathsf{t},\mathcal{R}(S)\\
\mathcal{R}(?(C);S) &= \mu \mathsf{t}.?(\mathcal{H}^*(C));\mathsf{t},\mathcal{R}(S)\\
\mathcal{R}^\star(?(C);S) &= \mathcal{R}^\star(!\langle C\rangle;S) = \mathcal{R}^\star(S)\\
\mathcal{R}^\star(\mu \mathsf{t}.S) &= \mathcal{R}(S)
\end{aligned}
$$

Figure 12: Decomposition of types $\mathcal{H}^*(\cdot)$ (cf. Def. 4.1)

Definition 4.4 (Decomposing Environments). Given environments Γ and Δ, we define $\mathcal{H}^*(\Gamma)$ and $\mathcal{H}^*(\Delta)$ inductively as $\mathcal{H}^*(\emptyset)=\emptyset$ and

$$
\begin{aligned}
\mathcal{H}^*(\Delta, u_i : S) &= \mathcal{H}^*(\Delta),(u_i,\ldots,u_{i+|\mathcal{H}^*(S)|-1}) : \mathcal{H}^*(S)\\
\mathcal{H}^*(\Gamma, u_i : \langle S\rangle) &= \mathcal{H}^*(\Gamma), u_i : \mathcal{H}^*(\langle S\rangle)
\end{aligned}
$$

Definition 4.5 (Degree of a Process). The *optimized degree* of a process P, denoted $\lfloor P\rceil^*$, is inductively defined as follows:

$$
\begin{cases}
\lfloor Q\rceil^* + 1 & \text{if } P = u_i!\langle y\rangle.Q \text{ or } P = u_i?(y).Q\\
\lfloor Q\rceil^* & \text{if } P = (\nu\, s : S)Q\\
\lfloor Q\rceil^* + 1 & \text{if } P = (\nu\, r : S)Q \text{ and } \mathrm{tr}(S)\\
\lfloor Q\rceil^* + \lfloor R\rceil^* + 1 & \text{if } P = Q \mid R\\
1 & \text{if } P = \mathbf{0} \text{ or } P = X\\
\lfloor Q\rceil^* + 1 & \text{if } P = \mu X.Q
\end{cases}
$$

As before, given a finite tuple of names $\widetilde{u} = (a,b,s,s',\ldots)$, we write $\mathsf{init}(\widetilde{u})$ to denote the tuple $(a_1,b_1,s_1,s_1',\ldots)$; also, we say that a process is initialized if all of its names have some index.

Given a tuple of initialized names $\widetilde{u}$ and a tuple of indexed names $\widetilde{x}$, it is useful to collect those names in $\widetilde{x}$ that appear in $\widetilde{u}$.

Definition 4.6 (Free indexed names). Let $\widetilde{u}$ and $\widetilde{x}$ be two tuples of names. We define the set $\mathsf{fnb}(\widetilde{u},\widetilde{x})$ as $\{z_k : z_i \in \widetilde{u} \wedge z_k \in \widetilde{x}\}$.

As usual, we treat sets of names as tuples (and vice-versa). By abusing notation, given a process P, we shall write $\mathsf{fnb}(P,\widetilde{y})$ to stand for $\mathsf{fnb}(\mathsf{fn}(P),\widetilde{y})$. Then, we have that $\mathsf{fnb}(P,\widetilde{x}) \subseteq \widetilde{x}$. In the definition of the breakdown function, this notion allows us to conveniently determine a *context* for a subsequent trio.

Remark 1. *Whenever $c_k?(\widetilde{y})$ (resp. $\overline{c_k}!\langle\widetilde{y}\rangle$) with $\widetilde{y}=\epsilon$, we shall write $c_k?()$ (resp. $\overline{c_k}!\langle\rangle$) to stand for $c_k?(y)$ (resp. $\overline{c_k}!\langle y\rangle$) such that $c_k : ?(\langle\mathsf{end}\rangle);\mathsf{end}$ (resp. $\overline{c_k} : !\langle\langle\mathsf{end}\rangle\rangle;\mathsf{end}$).*

Definition 4.7 (Index function). Let S be an (unfolded) recursive session type. The function $f(S)$ is defined as follows:

$$
f(S) = \begin{cases} f_0'(S'\{S/\mathsf{t}\}) & \text{if } S = \mu \mathsf{t}.S'\\ f_0'(S) & \text{otherwise}\end{cases}
$$

where: $f_l'(!\langle U\rangle;S) = f_{l+1}'(S)$, $f_l'(?(U);S) = f_{l+1}'(S)$, and $f_l'(\mu \mathsf{t}.S) = |\mathcal{R}(S)| - l + 1$.

Given a process P, we write $\mathrm{frv}(P)$ to denote that P has a free recursive variable.

4.3 The Optimized Decomposition

We define the optimized decomposition $\mathcal{F}^*(\cdot)$ by relying on the revised breakdown function $\mathbb{A}^k_{\widetilde{x}}(\cdot)$ (cf. § 4.3.1). Given a context $\widetilde{x}$ and a $k>0$, $\mathbb{A}^k_{\widetilde{x}}(\cdot)$ is defined on initialized processes. Table 3 gives the definition: we use an auxiliary function for recursive processes, denoted $\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(\cdot)_g$ (cf. § 4.3.2), where parameter g is a mapping from recursive variables to a list of name variables.

In the following, to keep presentation simple, we assume processes $\mu X.P$ in which P does not contain a subprocess of shape $\mu Y.P'$. The generalization of our decomposition without this assumption is not difficult, but is notationally heavy.

4.3.1 The Breakdown Function. We describe entries 1-7 in Table 3.

1. Input Process $\mathbb{A}^k_{\widetilde{x}}(u_i?(y).Q)$ consists of a leading trio that mimics the input and runs in parallel with the breakdown of Q. In the trio, a context $\widetilde{x}$ is expected along c_k. Then, an input on u_l mimics the input action: it expects the *decomposition* of name y, denoted $\widetilde{y}$. To decompose y we use its type: if $y : S$ then $\widetilde{y} = (y_1,\ldots,y_{|\mathcal{H}^*(S)|})$. The index of u_l depends on the type of u_i. Intuitively, if u_i is tail-recursive then $l = f(S)$ (Def. 4.7) as index and we do not increment it, as the same decomposition of u_i should be used to mimic a new instance in the continuation. Otherwise, if u_i is linear then we use the substitution $\sigma = \{u_{i+1}/u_i\}$ to increment it in Q. Next, the context $\widetilde{z} = \mathsf{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ is propagated, where $\widetilde{w} = (u_i)$ or $\widetilde{w} = \epsilon$.

2. Output Process $\mathbb{A}^k_{\widetilde{x}}(u_i!\langle y_j\rangle.Q)$ sends the decomposition of y on u_l, with l as in the input case. We decompose name y_j based on its type S: $\widetilde{y} = (y_j,\ldots,y_{j+|\mathcal{H}^*(S)|-1})$. The context to be propagated is $\widetilde{z} = \mathsf{fnb}(P, \widetilde{x} \setminus \widetilde{w})$, where $\widetilde{w}$ and σ are as in the input case.

3. Restriction (Non-recursive name) The breakdown of process $(\nu\, s : C)Q$ is $(\nu\, \widetilde{s} : \mathcal{H}^*(C))\,\mathcal{B}^k_{\widetilde{x}}(Q\sigma)$, where s is decomposed using C: $\widetilde{s} = (s_1,\ldots,s_{|\mathcal{H}^*(C)|})$. Since $(\nu\, s)$ binds s and its dual $\overline{s}$ (or only s if C is a shared type) the substitution σ is simply $\{s_1\overline{s}_1/s\overline{s}\}$ and initializes indexes in Q.

4. Restriction (Recursive name) As in the previous case, in the breakdown of $(\nu\, s : \mu \mathsf{t}.S)Q$ the name s is decomposed into $\widetilde{s}$ by relying on $\mu \mathsf{t}.S$. Here the breakdown consists of the breakdown of Q running in parallel with a control trio, which appends restricted (recursive) names $\widetilde{s}$ and $\widetilde{\overline{s}}$ to the context, i.e., $\widetilde{z} = \widetilde{x},\widetilde{s},\widetilde{\overline{s}}$.

5. Composition The breakdown of process $Q_1 \mid Q_2$ uses a control trio to trigger the breakdowns of Q_1 and Q_2, similarly as before.

6. Inaction The breakdown of $\mathbf{0}$ is simply an input prefix that receives an empty context (i.e., $\widetilde{x} = \epsilon$).

7. Recursion The breakdown of $\mu X.P$ is as follows:

$$
(\nu\, c_X^r)(c_k?(\widetilde{x}).\overline{c_{k+1}^r}!\langle\widetilde{z}\rangle.\mu X.c_X^r?(\widetilde{y}).\overline{c_{k+1}^r}!\langle\widetilde{y}\rangle.X \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(P)_g)
$$

	P	$\mathbb{A}^k_{\tilde{x}}(P)$	
1	$u_i?(y).Q$	$c_k?(\widetilde{x}).u_l?(\widetilde{y}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle \mid \mathbb{A}^{k+1}_{\widetilde{z}}(Q\sigma)$	$y_j : S \wedge \widetilde{y} = (y_1,\ldots,y_{\lvert S\rvert})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,f(S)\colon i$ $\sigma = \mathsf{next}(u_i)\cdot\{y_1/y\}$
2	$u_i!\langle y_j\rangle.Q$	$c_k?(\widetilde{x}).u_l!\langle\widetilde{y}\rangle.\overline{c_{k+1}}!\langle\widetilde{z}\rangle \mid \mathbb{A}^{k+1}_{\widetilde{z}}(Q\sigma)$	$y_j : S \wedge \widetilde{y} = (y_j,\ldots,y_{j+\lvert\mathcal{H}^*(S)\rvert-1})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,f(S)\colon i$ $\sigma = \mathsf{next}(u_i)$
3	$(\nu\, s : C)Q$	$(\nu\,\widetilde{s} : \mathcal{H}^*(C))\,\mathbb{A}^k_{\tilde{x}}(Q\sigma)$	$\widetilde{s} = (s_1,\ldots,s_{\lvert\mathcal{H}^*(C)\rvert})$ $\sigma = \{s_1\overline{s_1}/s\overline{s}\}$
4	$(\nu\, s : \mu\mathrm{t}.S)Q$	$(\nu\,\widetilde{s} : \mathcal{R}(S))\,\big(c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{z}\rangle.\mathbf{0} \mid \mathbb{A}^{k+1}_{\widetilde{z}}(Q)\big)$	$\mathsf{tr}(\mu\mathrm{t}.S)$ $\quad \widetilde{s} = (s_1,\ldots,s_{\lvert\mathcal{R}(S)\rvert})$ $\widetilde{z} = \widetilde{x},\widetilde{s},\widetilde{\overline{s}}$ $\quad \widetilde{\overline{s}} = (\overline{s_1},\ldots,\overline{s_{\lvert\mathcal{R}(S)\rvert}})$
5	$Q_1 \mid Q_2$	$c_k?(\widetilde{x}).\overline{c_{k+1}}!\langle\widetilde{y}\rangle.\overline{c_{k+l+1}}!\langle\widetilde{z}\rangle \mid \mathbb{A}^{k+1}_{\tilde{y}}(Q_1) \mid \mathbb{A}^{k+l+1}_{\tilde{z}}(Q_2)$	$\widetilde{y} = \mathtt{fnb}(Q_1,\widetilde{x})$ $\quad \widetilde{z} = \mathtt{fnb}(Q_2,\widetilde{x})$ $l = \lvert Q_1\rvert$
6	$\mathbf{0}$	$c_k?().\mathbf{0}$	
7	$\mu X.P$	$(\nu\, c^r_X)(c_k?(\widetilde{x}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.\mu X.c^r_X?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{y}\rangle.X \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(P)_g)$	$\widetilde{n} = \mathtt{fs}(P)$ $\quad \widetilde{n} : \widetilde{C} \wedge \widetilde{z} = \mathsf{bn}(\widetilde{n} : \widetilde{C})$ $\lvert\widetilde{z}\rvert = \lvert\widetilde{y}\rvert$ $\quad g = \{X \mapsto \widetilde{z}\}$

	P	$\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(P)_g$	
8	$u_i!\langle y_j\rangle.Q$	$\mu X.c^r_k?(\widetilde{x}).u_l!\langle\widetilde{y}\rangle.\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g \neq \emptyset$) $\mu X.c^r_k?(\widetilde{x}).\big(u_l!\langle\widetilde{y}\rangle.c^r_{k+1}!\langle\widetilde{z}\rangle \mid X\big) \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g = \emptyset$)	$y : T \wedge (y_j,\ldots,y_{j+\lvert\mathcal{H}^*(T)\rvert-1})$ $\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,f(S)\colon i$ $\sigma = \mathsf{next}(u_i)$
9	$u_i?(y).Q$	$\mu X.c^r_k?(\widetilde{x}).u_l?(\widetilde{y}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g \neq \emptyset$) $\mu X.c^r_k?(\widetilde{x}).\big(u_l?(\widetilde{y}).c^r_{k+1}!\langle\widetilde{z}\rangle \mid X\big) \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g = \emptyset$)	$\widetilde{w} = (\mathtt{lin}(u_i))\,?\,\{u_i\}\colon\epsilon$ $\widetilde{z} = g(X) \cup \mathtt{fnb}(Q, \widetilde{x}\widetilde{y} \setminus \widetilde{w})$ $l = (\mathtt{tr}(u_i))\,?\,f(S)\colon i$ $\sigma = \mathsf{next}(u_i)\cdot\{y_1/y\}$
10	$Q_1 \mid Q_2$	$\mu X.c^r_k?(\widetilde{x}).\big(\overline{c^r_{k+1}}!\langle\widetilde{y_1}\rangle.X \mid c^r_{k+l+1}!\langle\widetilde{y_2}\rangle\big) \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{y}_1}(Q_1)_g \mid \mathbb{A}^{k+l+1}_{\mathsf{rec}\ \tilde{y}_2}(Q_2)_\emptyset$ (if $g \neq \emptyset$) $\mu X.c^r_k?(\widetilde{x}).\big(c^r_{k+1}!\langle\widetilde{y_1}\rangle \mid c^r_{k+l+1}!\langle\widetilde{y_2}\rangle \mid X\big) \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{y}_1}(Q_1)_\emptyset \mid \mathbb{A}^{k+l+1}_{\mathsf{rec}\ \tilde{y}_2}(Q_2)_\emptyset$ (if $g = \emptyset$)	$\mathsf{frv}(Q_1)$ $\widetilde{y_1} = g(X) \cup \mathtt{fnb}(Q_1,\widetilde{x})$ $\widetilde{y_2} = \mathtt{fnb}(Q_2,\widetilde{x})$ $l = \lvert Q_1\rvert$
11	$(\nu\, s : C)Q$	$\mu X.(\nu\,\widetilde{s} : \mathcal{H}^*(C))c^r_k?(\widetilde{x}).\overline{c^r_{k+1}}!\langle\widetilde{z}\rangle.X \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g \neq \emptyset$) $\mu X.(\nu\,\widetilde{s} : \mathcal{H}^*(C))c^r_k?(\widetilde{x}).\big(c^r_{k+1}!\langle\widetilde{z}\rangle \mid X\big) \mid \mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{z}}(Q\sigma)_g$ (if $g = \emptyset$)	$\widetilde{s} = (s_1,\ldots,s_{\lvert\mathcal{H}^*(S)\rvert})$ $\widetilde{\overline{s}} = (\mathtt{lin}(S))\,?\,(\overline{s_1},\ldots,\overline{s_{\lvert\mathcal{H}^*(S)\rvert}})\colon\epsilon$ $\widetilde{z} = \widetilde{x},\widetilde{s},\widetilde{\overline{s}}$ $\quad \sigma = \{s_1\overline{s_1}/s\overline{s}\}$
12	X	$\mu X.c^r_k?(\widetilde{x}).c^r_X!\langle\widetilde{x}\rangle X$ (if $g \neq \emptyset$) $\mu X.c^r_k?().\big(c^r_X!\langle\rangle \mid X\big)$ (if $g = \emptyset$)	
13	$\mathbf{0}$	$c^r_k?().\mathbf{0}$	

Table 3: Optimized breakdown function $\mathbb{A}^k_{\tilde{x}}(\,\cdot\,)$ for processes, and auxiliary function for recursive processes $\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(\,\cdot\,)_g$.

We have a control trio and the breakdown of P, obtained using $\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(\,\cdot\,)_g$ (§ 4.3.2). The trio receives the context $\widetilde{x}$ on c_k and propagates it further. To ensure typability, we bind all session free names of P using the context $\widetilde{z}$, which contains the decomposition of those free names. This context is needed to break down P, and so we record it as $g = \{X \mapsto \widetilde{z}\}$ in the definition of $\mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{x}}(P)_g$. This way, $\widetilde{z}$ will be propagated all the way until reaching X.

Next, the recursive trio is enabled, and receives $\widetilde{y}$ along c^r_X, with $\lvert\widetilde{z}\rvert = \lvert\widetilde{y}\rvert$ and $l = \lvert P\rvert$. The tuple $\widetilde{y}$ is propagated to the first trio of $\mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{x}}(P)_g$. By definition of $\mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{x}}(P)_g$, its propagator c^r_X will

send the same context as received by the first trio. Hence, the recursive part of the control trio keeps sending this context to the next instances of recursive trios of $\mathbb{A}^{k+1}_{\mathsf{rec}\ \tilde{x}}(P)_g$.

Notice that the leading trio actually has four prefixes. This simplifies our presentation: this trio can be broken down into two trios by introducing an extra propagator c_{k+1} to send over c^r_{k+2}.

4.3.2 Handling P in $\mu X.P$. As already mentioned, we use the auxiliary function $\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(\,\cdot\,)_g$ to generate *recursive trios*.

We discuss entries 8-11 in Tab. 3 (other entries are similar as before). A key observation is that parameter g can be empty. To see this, consider a process like $P = \mu X.(Q_1 \mid Q_2)$ where X occurs free in Q_1 but not in Q_2. If X occurs free in Q_1 then its decomposition will have a non-empty g, whereas the g for Q_2 will be empty. In the recursive trios of Tab. 3, the difference between $g \neq \emptyset$ and $g = \emptyset$ is subtle: in the former case, X appears guarded by a propagator; in the latter case, it appears unguarded in a parallel composition. This way, trios in the breakdown of Q_2 replicate themselves on a trigger from the breakdown of Q_1.

Given this difference, we only describe the cases when $g \neq \emptyset$:

8 / 9. Output and Input The breakdown of $u_i!\langle y_j\rangle.Q$ consists of the breakdown of Q in parallel with a leading trio, a recursive process whose body is defined as in $\mathcal{B}(\cdot)$. As names $g(X)$ may not appear free in Q, we must ensure that a context $\widetilde{z}$ for the recursive body is propagated. The breakdown of $r?(y).Q$ is defined similarly.

10. Parallel Composition We discuss the breakdown of $Q_1 \mid Q_2$ assuming $\mathsf{frv}(Q_1)$. We take $\widetilde{y}_1 = g(X) \cup \mathsf{fnb}(Q_1, \widetilde{x})$ to ensure that $g(X)$ is propagated to the breakdown of X. The role of c^r_{k+l+1} is to enact a new instance of the breakdown of Q_2; it has a shared type to enable replication. In a running process, the number of these triggers in parallel denotes the number of available instances of Q_2.

11. Recursive Variable In this case, the breakdown is a control trio that receives the context $\widetilde{x}$ from a preceding trio and propagates it again to the first control trio of the breakdown of a recursive process along c^r_X. Notice that by construction we have $\widetilde{x} = g(X)$.

We may now define the optimized process decomposition:

Definition 4.8 (Decomposing Processes, Optimized). Let P be a π process with $\widetilde{u} = \mathsf{fn}(P)$ and $\widetilde{v} = \mathsf{rn}(P)$. Given the breakdown function $\mathbb{A}^k_{\widetilde{x}}(\cdot)$ in Table 3, the *optimized decomposition* of P, denoted $\mathcal{F}^*(P)$, is defined as

$$\mathcal{F}^*(P) = (\nu\,\widetilde{c})(\overline{c_k}!\langle\widetilde{r}\rangle.\mathbf{0} \mid \mathbb{A}^k_{\widetilde{r}}(P\sigma))$$

where: $k > 0$; $\widetilde{c} = (c_k, \ldots, c_{k+\lfloor P\rceil^*-1})$; $\widetilde{r}$ such that for $v \in \widetilde{v}$ and $v : S\ (v_1, \ldots, v_{|\mathcal{R}(S)|}) \subseteq \widetilde{r}$; and $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.

4.4 Examples

We now illustrate $\mathcal{F}^*(\,\cdot\,)$, $\mathbb{A}^k_{\widetilde{x}}(\,\cdot\,)$, $\mathbb{A}^k_{\mathsf{rec}\ \tilde{x}}(\,\cdot\,)_g$, and $\mathcal{H}^*(\,\cdot\,)$.

Example 4.9 (Exam. 3.7, Revisited). Consider again the process $P = (\nu\,u)(A \mid B)$ as in Exam. 3.7. Recall that P implements session types $S = !\langle\overline{T}\rangle;\mathsf{end}$ and $T = ?(\mathsf{Int});!\langle\mathsf{Bool}\rangle;\mathsf{end}$.

By Def. 4.5, $\lfloor P\rceil^* = 9$. The optimized decomposition of P is:

$$\mathcal{F}^*(P) = (\nu\,\widetilde{c})(\overline{c_1}!\langle\rangle \mid (\nu\,u_1)\mathbb{A}^1_\epsilon((A \mid B)\sigma'))$$

where $\sigma' = \mathsf{init}(\mathsf{fn}(P)) \cdot \{u_1\overline{u_1}/u\overline{u}\}$ and $\widetilde{c} = (c_1, \ldots, c_9)$. We have:

$$\mathbb{A}^1_\epsilon((A \mid B)\sigma')) = c_1?().\overline{c_2}!\langle\rangle.\overline{c_6}!\langle\rangle \mid \mathbb{A}^2_\epsilon(A\sigma') \mid \mathbb{A}^6_\epsilon(B\sigma')$$

The breakdowns of sub-processes A and B are as follows:

$$\mathbb{A}^2_\epsilon(A\sigma') = c_2?().u_1!\langle w_1, w_2\rangle.\overline{c_3}!\langle\rangle \mid c_3?().\overline{w}_1?(t).\overline{c_4}!\langle\rangle \mid c_4?().\overline{w}_2!\langle\mathsf{odd}(t)\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\mathbf{0}$$

$$\mathbb{A}^6_\epsilon(B\sigma') = c_6?().\overline{u}_1?(x_1, x_2).\overline{c_7}!\langle x_1, x_2\rangle \mid c_7?(x_1, x_2).x_1!\langle 5\rangle.\overline{c_8}!\langle x_2\rangle \mid c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0}$$

Name w is decomposed as indexed names $\overline{w}_1, \overline{w}_2$; by using $\mathcal{H}^*(\,\cdot\,)$ (Def. 4.1) on T, their MSTs are $M_1 = !\langle\mathsf{Int}\rangle;\mathsf{end}$ and $M_2 = ?(\mathsf{Bool});\mathsf{end}$, respectively. Name u_1 is the decomposition of name u and it is typed with $!\langle\overline{M}_1, \overline{M}_2\rangle;\mathsf{end}$. After a few administrative reductions on c_1, c_2, and c_6 , $\mathcal{F}^*(P)$ mimics the first source communication:

$$\begin{aligned}\mathcal{F}^*(P) &\longrightarrow^3 (\nu\,\widetilde{c}_*)(\,u_1!\langle w_1, w_2\rangle.\ \overline{c_3}!\langle\rangle \mid \mathbb{A}^3_\epsilon(\overline{w}?(t).\overline{w}!\langle\mathsf{odd}(t)\rangle.\mathbf{0}) \mid \\ &\qquad \overline{u}_1?(x_1, x_2).\ \overline{c_7}!\langle x_1, x_2\rangle \mid \mathbb{A}^7_{x_1,x_2}(x!\langle 5\rangle.x?(b).\mathbf{0})) \\ &\longrightarrow (\nu\,\widetilde{c}_*)(\overline{c_3}!\langle\rangle \mid \mathbb{A}^2_\epsilon(\overline{w}?(t).\overline{w}!\langle\mathsf{odd}(t)\rangle.\mathbf{0}) \mid \overline{c_7}!\langle w_1, w_2\rangle \mid \\ &\qquad \mathbb{A}^7_{x_1,x_2}(x!\langle 5\rangle.x?(b).\mathbf{0}))\end{aligned}$$

Above, $\widetilde{c}_* = (c_3, c_4, c_5, c_7, c_8, c_9)$. After reductions on c_3 and c_7, name w_1 substitutes x_1 and the communication along w_1 can be mimicked:

$$\begin{aligned}\mathcal{F}^*(P) &\longrightarrow^6 (\nu\,\widetilde{c}_{**})\ \overline{w}_1?(t).\ \overline{c_4}!\langle\rangle \mid c_4?().\overline{w}_2!\langle\mathsf{odd}(t)\rangle.\overline{c_5}!\langle\rangle \mid \\ &\qquad c_5?().\mathbf{0} \mid\ w_1!\langle 5\rangle.\ \overline{c_8}!\langle w_2\rangle \mid \\ &\qquad c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0} \\ &\longrightarrow (\nu\,\widetilde{c}_{**})(\overline{c_4}!\langle 5\rangle \mid c_4?(t).\overline{w}_2!\langle\mathsf{odd}(t)\rangle.\overline{c_5}!\langle\rangle \mid c_5?().\mathbf{0} \mid \\ &\qquad \overline{c_8}!\langle w_2\rangle \mid c_8?(x_2).x_2?(b_1).\overline{c_9}!\langle\rangle \mid c_9?().\mathbf{0})\end{aligned}$$

Above, $\widetilde{c}_{**} = (c_4, c_5, c_8, c_9)$. Further reductions follow similarly.

Example 4.10 (Example 3.8, Revisited). Consider again the tail-recursive session type $S = \mu\mathsf{t}.?(\mathsf{Int});!\langle\mathsf{Int}\rangle;\mathsf{t}$. Also, let R be a process implementing a channel r with type with S as follows:

$$R = \mu X.R' \qquad R' = r?(z).r!\langle -z\rangle.X$$

We decompose name r using S and obtain two channels typed with MSTs as in Fig. 12. We have: $r_1 : \mu\mathsf{t}.?(\mathsf{Int});\mathsf{t}$ and $r_2 : \mu\mathsf{t}.!\langle\mathsf{Int}\rangle;\mathsf{t}$.

The trios produced by $\mathbb{A}^k_\epsilon(R)$ satisfy two properties: they (1) mimic the recursive behavior of R and (2) use the same decomposition of channel r (i.e., r_1,r_2) in every instance.

To accomplish (1), each trio of the breakdown of the recursion body is a recursive trio. For (2), we need two things. First, we expect to receive all recursive names in the context $\widetilde{x}$ when entering the decomposition of the recursion body; further, each trio should use one recursive name from the names received and propagate *all* of them to subsequent trio. Second, we need an extra control trio when breaking down prefix μX: this trio (i) receives recursive names from the last trio in the breakdown of the recursion body and (ii) activates another instance with these recursive names.

Using these ideas, we have the decomposed process $\mathbb{A}^1_{r_1,r_2}(R)$:

$$c_1?(r_1, r_2).\overline{c^r_2}!\langle r_1, r_2\rangle.\mu X.c^r_X?(y_1, y_2).\overline{c^r_2}!\langle y_1, y_2\rangle.X \mid \mathbb{A}^2_{\mathsf{rec}\ r_1,r_2}(R')$$

where $\mathbb{A}^2_{\mathsf{rec}\ r_1,r_2}(R')$ is the composition of three recursive trios:

$$\mu X.c_2^r?(y_1,y_2).r_1?(z_1).\overline{c_3^r}!\langle y_1,y_2,z_1\rangle.X \mid$$
$$\mu X.c_3^r?(y_1,y_2,z_1).r_2?(-z_1).\overline{c_4^r}!\langle y_1,y_2\rangle.X \mid$$
$$\mu X.c_4^r?(y_1,y_2).\overline{c_X^r}!\langle y_1,y_2\rangle.X$$

c_2^r will first activate the recursive trios with context (r_1,r_2). Next, each trio uses one of r_1,r_2 and propagate them both mimicking the recursion body. The last recursive trio sends r_1,r_2 to the top-level control trio, so it can enact another instance of the decomposition of the recursion body by activating the first recursive trio.

4.5 Measuring the Optimization

Here we measure the improvements of $\mathcal{F}^*(\cdot)$ over $\mathcal{F}(\cdot)$. A key metric for comparison is the number of prefixes/sychronizations induced by each decomposition. This includes (1) the number of prefixes involved in *channel redirections* and (2) the *number of propagators*; both can be counted by already defined notions:

(1) Channel redirections can be counted by the levels of nesting in the decompositions of types (cf. Fig. 9 and Fig. 12)
(2) The number of propagators is determined by the degree of a process (cf. Def. 3.2 and Def. 4.5)

These two metrics are related; let us discuss them in detail.

Channel redirections. The decompositions of types for $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$ abstractly describe the respective channel redirections. The type decomposition for $\mathcal{F}(\cdot)$ (Fig. 9) defines 5 levels of nesting for the translation of input/output types. Then, at the level of (decomposed) processes, channels with these types implement redirections: the nesting levels correspond to 5 additional prefixes in the decomposed process that mimic a source input/output action. In contrast, the type decomposition for $\mathcal{F}^*(\cdot)$ (Fig. 12) induces no nesting, and so at the level of processes there are no additional prefixes.

Number of propagators. We define auxiliary functions to count the number of propagators induced by $\mathcal{F}(\cdot)$ and $\mathcal{F}^*(\cdot)$. These functions, denoted $\#(\cdot)$ and $\#^*(\cdot)$, respectively, are defined using the degree functions ($\lfloor\cdot\rceil$ and $\lfloor\cdot\rceil^*$) given by Def. 3.2 and Def. 4.5.

Remarkably, $\lfloor\cdot\rceil$ and $\#(\cdot)$ are not equal. The difference lies in the number of tail-recursive names in a process. In $\mathcal{F}(\cdot)$ there are propagators c_k but also c^r, used for recursive names. Def. 3.2, however, only counts propagators of form c_k. For any P, the number of propagators c^r in $\mathcal{F}(P)$ is the number of free and bound tail-recursive names in P. We remark that, by definition, there may be more than one occurrence of a propagator c^r in $\mathcal{F}(P)$: there is at least one prefix with subject c^r; further occurrences depend on the sequencing structure of the (recursive) type assigned to r. On the other hand, in $\mathcal{F}^*(P)$ there are propagators c_k and propagators c^r_X, whose number corresponds to the number of recursive variables in the process. To define $\#(\cdot)$ and $\#^*(\cdot)$, we write $\mathsf{brn}(P)$ to denote bound occurrences of recursive names and $\#_X(P)$ to denote the number of occurrences of recursive variables.

Definition 4.11 (Propagators in $\mathcal{F}(P)$ and $\mathcal{F}^*(P)$). Given a process P, the number of propagators in each decomposition is given by

$$\#(P) = \lfloor P\rceil + 2\cdot|\mathsf{brn}(P)| + |\mathsf{rn}(P)| \qquad \#^*(P) = \lfloor P\rceil^* + \#_X(P)$$

Notice that $\#^*(P)$ gives the exact number of actions induced by propagators in $\mathcal{F}^*(P)$; in contrast, due to propagators c^r, $\#(P)$ gives the *least* number of such actions in $\mathcal{F}(P)$.

In general, we have $\#(P) \geq \#^*(P)$, but we can be more precise for a broad class of processes. We say that a π process $P \not\equiv \mathbf{0}$ is in *normal form* if $P = (\nu\,\widetilde{n})(Q_1 \mid \ldots \mid Q_n)$, where each Q_i (with $i \in \{1,\ldots,n\}$) is not $\mathbf{0}$ and does not contain restriction nor parallel composition at top-level. We have the following result; see [1] for details.

Proposition 1. *If P is in normal form then $\#(P) \geq \frac{5}{3}\cdot\#^*(P)$.*

This result implies that the number of (extra) synchronizations induced by propagators in $\mathcal{F}(P)$ is larger than in $\mathcal{F}^*(P)$.

4.6 Results

4.6.1 Static Correctness. We first state Thm. 4.13, which ensures the typability of $\mathbb{A}^k_{\widetilde{x}}(\cdot)$ under MSTs. We rely on an auxiliary predicate:

Definition 4.12 (Indexed Names). Suppose some typing environments Γ,Δ. Let $\widetilde{x}$, $\widetilde{y}$ be two tuples of indexed names. We write $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y},\widetilde{x})$ for the predicate

$$\forall z_i.(z_i \in \widetilde{x} \Leftrightarrow ((z_i,\ldots,z_{i+m-1}) \subseteq \widetilde{y}) \wedge m = |\mathcal{H}^*((\Gamma,\Delta)(z_i))|)$$

Theorem 4.13 (Typability of Breakdown). *Let P be an initialized process. If $\Gamma;\Delta \vdash P \triangleright \diamond$ then*

$$\mathcal{H}^*(\Gamma \setminus \widetilde{x});\mathcal{H}^*(\Delta \setminus \widetilde{x}),\Theta \vdash \mathbb{A}^k_{\widetilde{y}}(P) \triangleright \diamond \quad (k>0)$$

where $\widetilde{x} \subseteq \mathsf{fn}(P)$ and $\widetilde{y}$ such that $\mathtt{indexed}_{\Gamma,\Delta}(\widetilde{y},\widetilde{x})$. Also, $\mathsf{balanced}(\Theta)$ with

$$dom(\Theta) = \{c_k, c_{k+1},\ldots,c_{k+|P|-1}\} \cup \{\overline{c_{k+1}},\ldots,\overline{c_{k+|P|-1}}\}$$

and $\Theta(c_k) = ?(\widetilde{M});\mathsf{end}$, where $\widetilde{M} = (\mathcal{H}^*(\Gamma),\mathcal{H}^*(\Delta))(\widetilde{y})$.

Proof. By induction of the structure of P; see [1] for details. □

We now (re)state the minimality result, now based on the decomposition $\mathcal{F}^*(\cdot)$. The proof follows from Thm. 4.13; see [1].

Theorem 4.14 (Minimality Result for π, Optimized). *Let P be a π process with $\widetilde{u} = \mathsf{fn}(P)$. If $\Gamma;\Delta \vdash P \triangleright \diamond$ then $\mathcal{H}^*(\Gamma\sigma);\mathcal{H}^*(\Delta\sigma) \vdash \mathcal{F}^*(P) \triangleright \diamond$, where $\sigma = \{\mathsf{init}(\widetilde{u})/\widetilde{u}\}$.*

4.6.2 Dynamic Correctness. As a complement to the minimality result, we have established that P and $\mathcal{F}^*(P)$ are *behaviorally equivalent* (Thm. 4.20). We overview this result and its required notions.

Thm. 4.20 relies on *MST-bisimilarity* (cf. Def. 4.19, $\approx^{\mathsf{M}}$), a variant of the *characteristic bisimilarity* in [10]. We discuss key differences between the two notions. First, we let an action along name n to be mimicked by an action on a possibly indexed name n_i, for some i.

Definition 4.15 (Indexed name). Given a name n, we write $\check{n}$ to either denote n or any indexed name n_i, with $i>0$.

Suppose we wish to relate P and Q using $\approx^{\mathsf{M}}$, and that P performs an output action involving name v. In our setting, Q should send a *tuple* of names: the decomposition of v. The second difference is that output objects should be related by the relation $\bowtie_{\mathsf{c}}$:

Definition 4.16 (Relating names). Let ϵ denote the empty list. We define $\bowtie_{\mathsf{c}}$ as the relation on names defined as

$$\frac{}{\epsilon \bowtie_{\mathsf{c}} \epsilon} \qquad \frac{\Gamma;\Delta \vdash n_i \triangleright C}{n_i \bowtie_{\mathsf{c}} (n_i,\ldots,n_{i+|\mathcal{H}^*(C)|-1})} \qquad \frac{\tilde{n} \bowtie_{\mathsf{c}} \tilde{m}_1 \quad n_i \bowtie_{\mathsf{c}} \tilde{m}_2}{\tilde{n}, n_i \bowtie_{\mathsf{c}} \tilde{m}_1, \tilde{m}_2}$$

Characteristic bisimilarity equates typed processes by relying on *characteristic processes* and *trigger processes*. These notions, which we recall below, need to be adjusted for them to work with MSTs.

Definition 4.17 (Characteristic trigger process [10]). The characteristic trigger process for type C is

$$t \Leftarrow_{\mathsf{C}} v : C \stackrel{\mathsf{def}}{=} t?(x).(\nu\, s)(s?(y).[\![C]\!]^y \mid \overline{s}!\langle v\rangle.\mathbf{0})$$

where $[\![C]\!]^y$ is the characteristic process for C on name y [10].

Our variant of trigger processes is defined as follows:

Definition 4.18 (Minimal characteristic trigger process). Given a type C, the trigger process is

$$t \Leftarrow_{\mathtt{m}} v_i : C \stackrel{\mathsf{def}}{=} t_1?(x).(\nu\, s_1)(s_1?(\widetilde{y}).\langle C\rangle_i^y \mid \overline{s_1}!\langle\tilde{v}\rangle.\mathbf{0})$$

where $v_i \bowtie_{\mathsf{c}} \widetilde{v}$, $y_i \bowtie_{\mathsf{c}} \widetilde{y}$, and $\langle C\rangle_i^y$ is a minimal characteristic process for type C on name y (see [1] for a definition).

We are now ready to define MST-bisimilarity:

Definition 4.19 (MST-Bisimilarity). A typed relation $\mathfrak{R}$ is an *MST bisimulation* if for all $\Gamma_1;\Delta_1 \vdash P_1\ \mathfrak{R}\ \Gamma_2;\Delta_2 \vdash Q_1$,

(1) Whenever $\Gamma_1;\Delta_1 \vdash P_1 \stackrel{(\nu\,\widetilde{m_1})n!\langle v:C_1\rangle}{\longrightarrow} \Delta_1';\Lambda_1' \vdash P_2$ then there exist Q_2, Δ_2', and σ_v such that $\Gamma_2;\Delta_2 \vdash Q_1 \stackrel{(\nu\,\widetilde{m_2})\check{n}!\langle\tilde{v}:\mathcal{H}^*(C)\rangle}{\Longrightarrow} \Delta_2' \vdash Q_2$ where $v\sigma_v \bowtie_{\mathsf{c}} \widetilde{v}$ and, for a fresh t,

$$\Gamma;\Delta_1'' \vdash (\nu\,\widetilde{m_1})(P_2 \mid t \Leftarrow_{\mathsf{C}} v : C_1)\,\mathfrak{R}$$
$$\Delta_2'' \vdash (\nu\,\widetilde{m_2})(Q_2 \mid t \Leftarrow_{\mathtt{m}} v\sigma : C_1)$$

(2) Whenever $\Gamma_1;\Delta_1 \vdash P_1 \stackrel{n?(v)}{\longrightarrow} \Delta_1' \vdash P_2$ then there exist Q_2, Δ_2', and σ_v such that $\Gamma_2;\Delta_2 \vdash Q_1 \stackrel{\check{n}?(\tilde{v})}{\Longrightarrow} \Delta_2' \vdash Q_2$ where $v\sigma_v \bowtie_{\mathsf{c}} \widetilde{v}$ and $\Gamma_1;\Delta_1' \vdash P_2\ \mathfrak{R}\ \Gamma_2;\Delta_2' \vdash Q_2$,

(3) Whenever $\Gamma_1;\Delta_1 \vdash P_1 \stackrel{\ell}{\longrightarrow} \Delta_1' \vdash P_2$, with ℓ not an output or input, then there exist Q_2 and Δ_2' such that $\Gamma_2;\Delta_2 \vdash Q_1 \stackrel{\hat{\ell}}{\Longrightarrow} \Delta_2' \vdash Q_2$ and $\Gamma_1;\Delta_1' \vdash P_2\ \mathfrak{R}\ \Gamma_2;\Delta_2' \vdash Q_2$ and $\mathrm{sub}(\ell) = n$ implies $\mathrm{sub}(\hat{\ell}) = \check{n}$.

(4) The symmetric cases of 1, 2, and 3.

The largest such bisimulation is called *MST bisimilarity* ($\approx^{\mathsf{M}}$).

We can now state our dynamic correctness result:

Theorem 4.20 (Operational Correspondence). *Let P be a π process such that $\Gamma_1;\Delta_1 \vdash P_1$. We have*

$$\Gamma;\Delta \vdash P \ \approx^{\mathsf{M}}\ \mathcal{H}^*(\Gamma);\mathcal{H}^*(\Delta) \vdash \mathcal{F}^*(P)$$

Proof. By coinduction: we exhibit a binary relation $\mathcal{S}$ that contains $(P, \mathcal{F}^*(P))$ and prove that it is an MST bisimulation. See [1] for the full account. □

5 CONCLUDING REMARKS

Concluding Remarks. We showed a minimality result for π, a session-typed π-calculus. This result says that sequentiality in session types is a convenient but not indispensable feature. Following [2], we introduced minimal session types (MSTs) for π and defined two decompositions, which transform processes typable with standard session types into processes typable with MSTs. The first decomposition composes existing encodability results and the minimality result for HO; the second decomposition optimizes the first one by (i) removing redundant synchronizations and (ii) using the native support of recursion in π. For this optimized decomposition, we proved also an operational correspondence result. This way, our work shows that the minimality result is independent from the kind of communicated objects (names or abstractions).

Sequentiality is the key distinguishing feature in the specification of message-passing programs using session types. We remark that by our minimality results do not mean that sequentiality in session types is redundant in *modeling and specifying* processes; rather, we claim that it is not an indispensable notion to *type-checking* them. Because we can type-check *session typed* processes using type systems that do not directly support sequencing in types, our decomposition defines a technique for implementing session types into languages whose type systems do not support sequentiality.

For the sake of space, we have not considered choice constructs (selection and branching). There is no fundamental obstacle in treating them, apart from a very minor caveat: the decomposition in [2] assumes typed processes in which every selection construct comes with a corresponding branching (see [1] for an example).

All in all, besides settling a question left open in [2], our work deepens our understanding about session-based concurrency and the connection between the first-order and higher-order paradigms.

Related Work. We use the trios decomposition by Parrow [12], which he studied for an untyped π-calculus with replication; in contrast, π processes feature recursion. We stress that our goal is to clarify the role of sequentiality in session types by using processes with MSTs, which lack sequentiality. While Parrow's approach elegantly induces processes typable with MSTs, defining trios decompositions for π is just one path towards our goal.

Our work differs significantly with respect to [2]. The source language in [2] is HO, based on abstraction-passing, whereas here we focus on the name-passing calculus π. While in HO propagators carry abstractions, in our case propagators are binding and carry names. Also, names must be decomposed and propagating them requires care. Further novelties appear when decomposing processes with recursion, which require a dedicated collection of *recursive trios* (not supported in HO).

Prior works have related session types with *different* type systems [3–5]. Loosely related is the work by Dardha et al. [3]. They compile a session π-calculus down into a π-calculus with the linear type system of [8] extended with variant types. They represent sequentiality using a continuation-passing style: a session type is interpreted as a linear type carrying a pair consisting of the original payload type and a new linear channel type, to be used for ensuing interactions. The differences are also technical: the approach in [3] thus involves translations connecting *two* different π-calculi and

two different type systems. In contrast, our approach based on MSTs justifies sequentiality using a single typed process framework.

ACKNOWLEDGMENTS

This work has been partially supported by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).

We are grateful to the anonymous reviewers for their careful reading and constructive remarks.

REFERENCES

[1] Alen Arslanagic, Anda-Amelia Palamariuc, and Jorge A. Pérez. 2021. Minimal Session Types for the π-calculus (Extended Version). *CoRR* abs/2107.10936 (2021). arXiv:2107.10936 http://arxiv.org/abs/2107.10936

[2] Alen Arslanagic, Jorge A. Pérez, and Erik Voogd. 2019. Minimal Session Types (Pearl). In *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom (LIPIcs, Vol. 134)*, Alastair F. Donaldson (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 23:1–23:28. https://doi.org/10.4230/LIPIcs.ECOOP.2019.23

[3] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. 2012. Session types revisited. In *Proc. of PPDP 2012*. ACM, 139–150. https://doi.org/10.1145/2370776.2370794

[4] Romain Demangeon and Kohei Honda. 2011. Full Abstraction in a Subtyped pi-Calculus with Linear Types. In *Proc. of CONCUR 2011 (LNCS, Vol. 6901)*. Springer, 280–296. https://doi.org/10.1007/978-3-642-23217-6_19

[5] Simon J. Gay, Nils Gesbert, and António Ravara. 2014. Session Types as Generic Process Types. In *Proceedings Combined 21st International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics, EXPRESS 2014, and 11th Workshop on Structural Operational Semantics, SOS 2014, Rome, Italy, 1st September 2014. (EPTCS, Vol. 160)*, Johannes Borgström and Silvia Crafa (Eds.). 94–110. https://doi.org/10.4204/EPTCS.160.9

[6] Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR'93 (LNCS, Vol. 715)*, Eike Best (Ed.). Springer-Verlag, 509–523.

[7] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98 (LNCS, Vol. 1381)*. Springer, 22–138.

[8] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. 1999. Linearity and the Pi-Calculus. *TOPLAS* 21, 5 (Sept. 1999), 914–947.

[9] Dimitrios Kouzapas, Jorge A. Pérez, and Nobuko Yoshida. 2016. On the Relative Expressiveness of Higher-Order Session Processes. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9632)*, Peter Thiemann (Ed.). Springer, 446–475. https://doi.org/10.1007/978-3-662-49498-1_18

[10] Dimitrios Kouzapas, Jorge A. Pérez, and Nobuko Yoshida. 2017. Characteristic bisimulation for higher-order session processes. *Acta Inf.* 54, 3 (2017), 271–341. https://doi.org/10.1007/s00236-016-0289-7

[11] Dimitrios Kouzapas, Jorge A. Pérez, and Nobuko Yoshida. 2019. On the relative expressiveness of higher-order session processes. *Inf. Comput.* 268 (2019). https://doi.org/10.1016/j.ic.2019.06.002

[12] Joachim Parrow. 2000. Trios in concert. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, Gordon D. Plotkin, Colin Stirling, and Mads Tofte (Eds.). The MIT Press, 623–638. Online version, dated July 22, 1996, available at http://user.it.uu.se/~joachim/trios.pdf.

[13] Davide Sangiorgi. 1992. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms*. Ph.D. Dissertation. University of Edinburgh.