Characteristic Bisimulations for Session Processes

Dimitrios Kouzapas[1], Jorge A. Pérez[2], and Nobuko Yoshida[3]

1 Imperial College, London
2,3 University of Groningen

Abstract

This work proposes tractable bisimulations for the higher-order π-calculus with session primitives ($\mathsf{HO}\pi$). We develop three typed bisimulations, which are shown to coincide with contextual equivalence. These characterisations demonstrate that observing as inputs only a specific finite set of higher-order values (which inhabit session types) suffices to reason about $\mathsf{HO}\pi$ processes.

1 Introduction

By combining features from the λ-calculus and the π-calculus, in *higher-order process calculi* exchanged values may contain processes. In this paper, we consider higher-order calculi with *session primitives*, thus enabling the specification of reciprocal exchanges (protocols) for higher-order mobile processes, which can be verified via type-checking using *session types* [7]. The study of higher-order concurrency has received significant attention, from untyped and typed perspectives (see, e.g., [26, 23, 22, 9, 19, 16, 15, 11, 27]). Although models of session-typed communication with features of higher-order concurrency exist [18, 6], their *tractable behavioural equivalences* and *relative expressiveness* remain little understood. Clarifying their status is not only useful for, e.g., justifying non-trivial mobile protocol optimisations, but also for transferring key reasoning techniques between (higher-order) session calculi. Our discovery is that *linearity* of session types plays a vital role to offer new equalities and fully abstract encodability, which to our best knowledge have not been proposed before.

The main higher-order language in our work, denoted $\mathsf{HO}\pi$, extends the higher-order π-calculus [23] with session primitives: it contains constructs for synchronisation on shared names, recursion, name abstractions (i.e., functions from name identifiers to processes, denoted $\lambda x.P$) and applications (denoted $(\lambda x.P)a$); and session communication (value passing and labelled choice using linear names). We study two significant subcalculi of $\mathsf{HO}\pi$, which distil higher- and first-order mobility: the HO-calculus, which is $\mathsf{HO}\pi$ without recursion and name passing, and the session π-calculus (here denoted π), which is $\mathsf{HO}\pi$ without abstractions and applications. While π is, in essence, the calculus in [7], this paper shows that HO is a new core calculus for higher-order session concurrency.

In this paper, we address tractable behavioural equivalences for $\mathsf{HO}\pi$. A well-studied behavioural equivalence in the higher-order setting is *context bisimilarity* [21], a labelled characterisation of reduction-closed, barbed congruence, which offers an appropriate discriminative power at the price of heavy universal quantifications in output clauses. Obtaining alternative characterisations is thus a recurring issue in the study of higher-order calculi. Our approach shows that protocol specifications given by session types are essential to limit the behaviour of higher-order session processes. Exploiting elementary processes inhabiting session types, this limitation is formally enforced by a refined (typed) labelled transition system (LTS) that narrows down the spectrum of allowed process behaviours, thus enabling tractable reasoning techniques. Two tractable characterisations of bisimilarity are then introduced. Remarkably, using session types we prove that these bisimilarities coincide with context bisimilarity, without using operators for name-matching.

Outline / Contributions. This paper is structured as follows:

- § 2 overviews key ideas of our tractable bisimulations.
- § 3 presents the higher-order session calculus $\mathsf{HO}\pi$ and its subcalculi HO and π. Then, § 4 gives the session type system and states type soundness for $\mathsf{HO}\pi$ and its variants.
- § 5 develops *higher-order* and *characteristic* bisimilarities, our two tractable characterisations of contextual equivalence which alleviate the issues of context bisimilarity [21]. These relations are shown to coincide in $\mathsf{HO}\pi$ (Thm. 14).
- § 6 concludes with related works. The appendix summarises the typing system.

The paper is self-contained. ***Additional related work, more examples, omitted definitions, and proofs are in [1].***

2 Overview: Tractable Higher-Order Session Bisimulations

We outline our motivations and methods to show how session types are used for formulating two tractable bisimulations.

Overcoming Issues of Context Bisimilarity. Context bisimilarity ($\approx$, Def. 8) is a too demanding relation on processes. To see the issue, we show the following clause for output. Suppose $P \,\Re\, Q$, for some context bisimulation $\Re$. Then:

($\star$) Whenever $P \xrightarrow{(\nu\, \tilde{m_1})n!\langle V\rangle} P'$ there exist Q' and W such that $Q \xrightarrow{(\nu\, \tilde{m_2})n!\langle W\rangle} Q'$ and, ***for all*** R with $\mathtt{fv}(R) = x$, $(\nu\, \tilde{m_1})(P' \mid R\{V/x\}) \,\Re\, (\nu\, \tilde{m_2})(Q' \mid R\{W/x\})$.

Above, $(\nu\, \tilde{m_1})n!\langle V\rangle$ is the output label of value V with extrusion of names in $\tilde{m_1}$. To reduce the burden induced by universal quantification, we introduce *higher-order* and *characteristic* bisimulations, two tractable equivalences denoted $\approx^H$ and $\approx^C$, respectively. As we work with an *early* labelled transition system (LTS), we take the following two steps:

(a) We replace ($\star$) with a clause involving a more tractable process closure.
(b) We refine the transition rule for input in the LTS, to avoid observing infinitely many actions on the same input prefix.

Trigger Processes with Session Communication. Concerning (a), we exploit session types. We first observe that closure $R\{V/x\}$ in ($\star$) is context bisimilar to the process:

$$P = (\nu\, s)((\lambda z.\, z?(x).R)\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{1}$$

In fact, we do have $P \approx R\{V/x\}$, since application and reduction of dual endpoints are deterministic. Now let us consider process T_V below, where t is a fresh name:

$$T_V = t?(x).(\nu\, s)(x\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{2}$$

We write $P \xrightarrow{n?\langle V\rangle} P'$ to denote an input transition along n. If T_V inputs value $\lambda z.\, z?(x).R$ then we can simulate the closure of P:

$$T_V \xrightarrow{t?\langle \lambda z.\, z?(x).R\rangle} P \approx R\{V/x\} \tag{3}$$

Processes such as T_V offer a value at a fresh name; we will use this class of ***trigger processes*** to define a refined bisimilarity without the demanding output clause ($\star$). Given a fresh name t, we write $t \Leftarrow V$ to stand for a trigger process T_V for value V. We note that in contrast to previous approaches [25, 9] our trigger processes do *not* use recursion or replication. This is crucial for preserving linearity of session endpoints.

Characteristic Processes and Values. Concerning (b), we limit the possible input values (such as $\lambda z.\, z?(x).R$ above) by exploiting session types. The key concept is that of

characteristic process/value of a type, the simplest term inhabiting that type (Def. 9). This way, e.g., let $S = ?(S_1{\multimap}\diamond); !\langle S_2\rangle; \mathtt{end}$ be a session type: first input an abstraction, then output a value of type S_2. Then, process $Q = u?(x).(u!\langle s_2\rangle.\mathbf{0} \mid x\, s_1)$ is a characteristic process for S along name u. Given a session type S, we write $[\![S]\!]^u$ for its characteristic process along name u (cf. Def. 9). Also, given value type U, we write $[\![U]\!]_{\mathsf{c}}$ to denote its characteristic value.

We use the characteristic value $[\![U]\!]_{\mathsf{c}}$ to limit input transitions. Following the same reasoning as (1)–(3), we can use an alternative trigger process, called ***characteristic trigger process*** with type U to replace clause $(\star)$:

$$t \Leftarrow V{:}U \overset{\mathsf{def}}{=} t?(x).(\nu\, s)([?(U);\mathtt{end}]^s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \tag{4}$$

Thus, in contrast to the trigger process (2), the characteristic trigger process in (4) does not involve a higher-order communication on fresh name t. To refine the input transition system, we need to observe an additional value, $\lambda x.\, t?(y).(y\, x)$, called the ***trigger value***. This is necessary, because it turns out that a characteristic value alone as the observable input is not enough to define a sound bisimulation. Roughly speaking, the trigger value is used to observe/simulate application processes.

Refined Input Transition Rule. Based on the above discussion, we refine the (early) transition rule for input actions. The transition rule for input roughly becomes (see Def. 11 for details):

$$\frac{P \xrightarrow{n?\langle V\rangle} P' \quad V = m \vee V \equiv \lambda x.\, t?(y).(y\, x) \vee V \equiv [\![U]\!]_{\mathsf{c}} \text{ with } t \text{ fresh}}{P' \overset{n?\langle V\rangle}{\longmapsto} P'}$$

Note the distinction between standard and refined transitions: $\xrightarrow{n?\langle V\rangle}$ vs. $\overset{n?\langle V\rangle}{\longmapsto}$. Using this rule, we define an alternative LTS with refined (higher-order) input. This refined LTS is used for both higher-order ($\approx^H$) and characteristic ($\approx^C$) bisimulations (Defs. 12 and 13), in which the demanding clause $(\star)$ is replaced with more tractable clauses based on trigger processes (cf. (2)) and characteristic trigger processes (cf. (4)), respectively. We show that $\approx^H$ is useful for $\mathsf{HO}\pi$ and HO, and that $\approx^C$ can be uniformly used in all subcalculi, including π.

Our calculus lacks name matching, which is usually crucial to prove completeness of bisimilarity. Instead of matching, we use types: a process trigger embeds a name into a characteristic process so to observe its session behaviour.

3 Higher-Order Session π-Calculi

We introduce the *Higher-Order Session π-Calculus* ($\mathsf{HO}\pi$). $\mathsf{HO}\pi$ includes both name- and abstraction-passing as well as recursion; it is a subcalculus of the language studied in [18]. Following the literature [9], for simplicity of the presentation we concentrate on the second-order call-by-value $\mathsf{HO}\pi$. (In § **??** we consider extensions of $\mathsf{HO}\pi$ with higher-order abstractions and polyadicity in name-passing/abstractions.)

3.1 Syntax of $\mathsf{HO}\pi$

Values The syntax of $\mathsf{HO}\pi$ is defined in Fig. 1 We use $a, b, c, \dots$ (resp. $s, \overline{s}, \dots$) to range over shared (resp. session) names. We use $m, n, t, \dots$ for session or shared names. We define the dual operation over names n as $\overline{n}$ with $\overline{\overline{s}} = s$ and $\overline{a} = a$. Intuitively, names s and $\overline{s}$ are dual (two) *endpoints* while shared names represent shared (non-deterministic) points.

$$u, w ::= n \mid x, y, z \quad n ::= a, b \mid s, \overline{s} \quad V, W ::= \colorbox{lightgray}{u} \mid \lambda x.\, P$$

$$\begin{array}{rl} P, Q ::= & u!\langle V\rangle.P \;\mid\; u?(x).P \;\mid\; u \triangleleft l.P \;\mid\; u \triangleright \{l_i : P_i\}_{i\in I} \\ \mid & \colorbox{lightgray}{$X \;\mid\; \mu X.P$} \;\mid\; V\,u \;\mid\; P \mid Q \;\mid\; (\nu\, n)P \;\mid\; \mathbf{0} \end{array}$$

Figure 1 Syntax of HOπ (HO lacks the constructs in grey).

$$\begin{array}{rcll} (\lambda x.\, P)\, u & \longrightarrow & P\{u/x\} & \text{[App]} \\ n!\langle V\rangle.P \mid \overline{n}?(x).Q & \longrightarrow & P \mid Q\{V/x\} & \text{[Pass]} \\ n \triangleleft l_j.Q \mid \overline{n} \triangleright \{l_i : P_i\}_{i\in I} & \longrightarrow & Q \mid P_j \;\; (j \in I) & \text{[Sel]} \\ P \longrightarrow P' & \Rightarrow & (\nu\, n)P \longrightarrow (\nu\, n)P' & \text{[Res]} \\ P \longrightarrow P' & \Rightarrow & P \mid Q \longrightarrow P' \mid Q & \text{[Par]} \\ P \equiv Q \longrightarrow Q' \equiv P' & \Rightarrow & P \longrightarrow P' & \text{[Cong]} \end{array}$$

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

Figure 2 Operational Semantics of HOπ.

Variables are denoted with $x, y, z, \ldots$, and recursive variables are denoted with $X, Y \ldots$. An abstraction $\lambda x.\, P$ is a process P with name parameter x. Values V, W include identifiers $u, v, \ldots$ and abstractions $\lambda x.\, P$ (first- and higher-order values, resp.).

Terms include the π-calculus prefixes for sending and receiving values V. Process $u!\langle V\rangle.P$ denotes the output of value V over name u, with continuation P; process $u?(x).P$ denotes the input prefix on name u of a value that will substitute variable x in continuation P. Recursion is expressed by $\mu X.P$, which binds the recursive variable X in process P. Process $V\,u$ is the application which substitutes name u on the abstraction V. Typing ensures that V is not a name. Prefix $u \triangleleft l.P$ selects label l on name u and then behaves as P. Process $u \triangleright \{l_i : P_i\}_{i\in I}$ offers a choice on labels l_i with continuation P_i, given that $i \in I$. Constructs for inaction $\mathbf{0}$, parallel composition $P_1 \mid P_2$, and name restriction $(\nu\, n)P$ are standard. Session name restriction $(\nu\, s)P$ simultaneously binds endpoints s and $\overline{s}$ in P. We use $\mathtt{fv}(P)$ and $\mathtt{fn}(P)$ to denote a set of free variables and names; and assume V in $u!\langle V\rangle.P$ does not include free recursive variables X. If $\mathtt{fv}(P) = \emptyset$, we call P *closed*.

3.2 Subcalculi of HOπ

We define two subcalculi of HOπ. The first is the *core higher-order session calculus* (denoted HO), which lacks recursion and name passing; its formal syntax is obtained from Fig. 1 by excluding constructs in grey. The second subcalculus is the *session π-calculus* (denoted π), which lacks the higher-order constructs (i.e., abstraction passing and application), but includes recursion. Let $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$. We write $\mathsf{C}^{-\mathsf{sh}}$ for C without shared names (we delete a, b from n). We shall demonstrate that HOπ, HO, and π have the same expressivity.

3.3 Operational Semantics

Fig. 2 defines the operational semantics of $\mathsf{HO}\pi$. [App] is a name application; [Pass] defines a shared interaction at n (with $\overline{n} = n$) or a session interaction; [Sel] is the standard rule for labelled choice/selection: given an index set I, a process selects label l_j on name n over a set of labels $\{l_i\}_{i \in I}$ offered by a branching on the dual endpoint $\overline{n}$; and other rules are standard. Rules for *structural congruence* are defined in Fig. 2 (bottom). We assume the expected extension of $\equiv$ to values V. We write $\longrightarrow^*$ for a multi-step reduction.

4 Types and Typing

This section defines a session typing system for $\mathsf{HO}\pi$ and states its main properties. Our system distills the key features of [18, 19], thus it is simpler.

Types. The syntax of types of $\mathsf{HO}\pi$ is given below:

$$\begin{array}{llll}
\text{(value)} & U & ::= & \boxed{C} \;\mid\; L \\
\text{(name)} & C & ::= & S \;\mid\; \langle S \rangle \;\mid\; \langle L \rangle \\
\text{(abstr)} & L & ::= & C{\rightarrow}\diamond \;\mid\; C{\multimap}\diamond \\
\text{(session)} & S & ::= & !\langle U \rangle; S \;\mid\; ?(U); S \;\mid\; \oplus\{l_i : S_i\}_{i \in I} \\
 & & \mid & \&\{l_i : S_i\}_{i \in I} \;\mid\; \mu\mathtt{t}.S \;\mid\; \mathtt{t} \;\mid\; \mathtt{end}
\end{array}$$

Value type U includes 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 $\langle L \rangle$. Types $C{\rightarrow}\diamond$ and $C{\multimap}\diamond$ denote *shared* and *linear* higher-order types, respectively.

We write S to denote session types. *Output type* $!\langle U \rangle; S$ first sends a value of type U and then follows the type described by S. Dually, $?(U); S$ denotes an *input type*. The *branching type* $\&\{l_i : S_i\}_{i \in I}$ and the *selection type* $\oplus\{l_i : S_i\}_{i \in I}$ define the labelled choice. We assume the *recursive type* $\mu\mathtt{t}.S$ is guarded, i.e., $\mu\mathtt{t}.\mathtt{t}$ is not allowed. Type $\mathtt{end}$ is the termination type.

Types of HO exclude $\boxed{C}$ from value types of $\mathsf{HO}\pi$; the types of π exclude L. From each $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$, $\mathsf{C}^{-\mathsf{sh}}$ excludes shared name types ($\langle S \rangle$ and $\langle L \rangle$), from name type C.

Following [3], we write S_1 dual S_2 if S_1 is the dual of S_2. Intuitively, the duality of types is obtained by dualising ! by ?, ? by !, $\oplus$ by & and & by $\oplus$, incorporating the fixed point construction (see Def. 17 in the Appendix).

Typing Judgements of $\mathsf{HO}\pi$. 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 first judgement states that under environments $\Gamma; \Lambda; \Delta$ value V has type U, whereas the second judgement states that under environments $\Gamma; \Lambda; \Delta$ process P has the process type $\diamond$. The environments are defined below:

$$\begin{array}{lll}
\Lambda & ::= & \emptyset \;\mid\; \Lambda \cdot x{:}C{\multimap}\diamond \qquad \Delta \;::=\; \emptyset \;\mid\; \Delta \cdot u{:}S \\
\Gamma & ::= & \emptyset \;\mid\; \Gamma \cdot x : C{\rightarrow}\diamond \;\mid\; \Gamma \cdot u : \langle S \rangle \;\mid\; \Gamma \cdot u : \langle L \rangle \;\mid\; \Gamma \cdot X : \Delta
\end{array}$$

Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. Λ is a mapping from variables to linear higher-order types; and Δ is a mapping from session names to session types. Both Λ and Δ are only subject to exchange. We require that the domains of Γ, Λ and Δ are pairwise distinct. $\Delta_1 \cdot \Delta_2$ means a disjoint union of Δ_1 and Δ_2. We are interested in *balanced* session environments:

▶ **Definition 1** (Balanced). We say that a session environment Δ is *balanced* if whenever $s : S_1, \overline{s} : S_2 \in \Delta$ then S_1 dual S_2.

The Typing System of HOπ. Since the typing system is similar to [18, 19], we fully describe it in App. A. The paper can be read without knowing the details of the typing system. Type soundness relies on the following auxiliary notion.

▶ **Definition 2** (Reduction of Session Environment). We define the relation $\longrightarrow$ on session environments as:

$$\Delta \cdot s :!\langle U\rangle; S_1 \cdot \overline{s} :?(U); S_2 \longrightarrow \Delta \cdot s : S_1 \cdot \overline{s} : S_2$$
$$\Delta \cdot s : \oplus\{l_i : S_i\}_{i\in I} \cdot \overline{s} : \&\{l_i : S_i'\}_{i\in I} \longrightarrow \Delta \cdot s : S_k \cdot \overline{s} : S_k' \ (k \in I)$$

We state the type soundness results of HOπ; it implies the type soundness of the sub-calculi HO, π, and C$^{-\mathsf{sh}}$.

▶ **Theorem 3** (Type Soundness). *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.*

5 Higher-Order Session Bisimulation

We develop a theory for observational equivalences over session typed HOπ processes. The theory follows the principles laid by the previous work of the authors [14, 13]. We introduce three different bisimulations and prove that all of them coincide with typed, reduction-closed, barbed congruence (Thm. 14).

5.1 Labelled Transition System for Processes

Labels. We define an (early) labelled transition system (LTS) over untyped processes. Later on, using the *environmental* transition semantics, we can define a typed transition relation to formalise how a typed process interacts with a typed observer. The interaction is defined on action ℓ:

$$\ell \ ::= \ \tau \ \mid \ n?\langle V\rangle \ \mid \ (\nu\, \tilde{m})n!\langle V\rangle \ \mid \ n \oplus l \ \mid \ n\&l$$

The internal action is defined by label τ. Action $(\nu\, \tilde{m})n!\langle V\rangle$ denotes the sending of value V over channel n with a possible empty set of names $\tilde{m}$ being restricted (we may write $n!\langle V\rangle$ when $\tilde{m}$ is empty). Dually, the action for value reception is $n?\langle V\rangle$. We also have actions for select and branch on a label l; these are denoted $n \oplus l$ and $n\&l$, resp. We write $\mathtt{fn}(\ell)$ and $\mathtt{bn}(\ell)$ to denote the sets of free/bound names in ℓ, resp. Given a label $\ell \neq \tau$, we write $\mathtt{subj}(\ell)$ to denote the *subject* of the action.

Dual actions occur on subjects that are dual between them and carry the same object; thus, output is dual to input and selection is dual to branching. Formally, duality on actions is the symmetric relation $\asymp$ that satisfies: (i) $n \oplus l \asymp \overline{n}\&l$ and (ii) $(\nu\, \tilde{m})n!\langle V\rangle \asymp \overline{n}?\langle V\rangle$.

LTS over Untyped Processes. The labelled transition system (LTS) over untyped processes is given in Fig. 3. We write $P_1 \xrightarrow{\ell} P_2$ with the usual meaning. The rules are standard [14, 13]. A process with an output prefix can interact with the environment with an output action that carries a value V (rule $\langle$Snd$\rangle$). Dually, in rule $\langle$Rv$\rangle$ a receiver process can observe an input of an arbitrary value V. Select and branch processes observe the select and branch actions in rules $\langle$Sel$\rangle$ and $\langle$Bra$\rangle$, resp. Rule $\langle$Res$\rangle$ closes the LTS under restriction

$$\langle\mathrm{App}\rangle \quad (\lambda x.\, P)\, u \xrightarrow{\tau} P\{u/x\}$$

$$\langle\mathrm{Snd}\rangle\ n!\langle V\rangle.P \xrightarrow{n!\langle V\rangle} P \qquad \langle\mathrm{Rv}\rangle\ n?(x).P \xrightarrow{n?\langle V\rangle} P\{V/x\}$$

$$\langle\mathrm{Sel}\rangle\ s \triangleleft l.P \xrightarrow{s\oplus l} P \qquad \langle\mathrm{Bra}\rangle\ s \triangleright \{l_i : P_i\}_{i\in I} \xrightarrow{s\& l_j} P_j \quad (j \in I)$$

$$\langle\mathrm{Alpha}\rangle \frac{P \equiv_\alpha Q \quad Q \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P'} \qquad \langle\mathrm{Res}\rangle \frac{P \xrightarrow{\ell} P' \quad n \notin \mathtt{fn}(\ell)}{(\nu\, n)P \xrightarrow{\ell} (\nu\, n)P'}$$

$$\langle\mathrm{New}\rangle \frac{P \xrightarrow{(\nu\, \tilde{m})n!\langle V\rangle} P' \quad m \in \mathtt{fn}(V)}{(\nu\, m)P \xrightarrow{(\nu\, m\cdot\tilde{m}')n!\langle V\rangle} P'} \qquad \langle\mathrm{Rec}\rangle \frac{P\{\mu X.P/X\} \xrightarrow{\ell} P'}{\mu X.P \xrightarrow{\ell} P'}$$

$$\langle\mathrm{Tau}\rangle \quad \frac{P \xrightarrow{\ell_1} P' \qquad Q \xrightarrow{\ell_2} Q' \qquad \ell_1 \asymp \ell_2}{P \mid Q \xrightarrow{\tau} (\nu\, \mathtt{bn}(\ell_1) \cup \mathtt{bn}(\ell_2))(P' \mid Q')}$$

$$\langle\mathrm{Par}_L\rangle \quad \frac{P \xrightarrow{\ell} P' \quad \mathtt{bn}(\ell) \cap \mathtt{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q}$$

Figure 3 The Untyped LTS. We omit rule $\langle\mathrm{Par}_R\rangle$.

if the restricted name does not occur free in the observable action. If a restricted name occurs free in the carried value of an output action, the process performs scope opening (rule $\langle\mathrm{New}\rangle$). Rule $\langle\mathrm{Rec}\rangle$ handles recursion unfolding. Rule $\langle\mathrm{Tau}\rangle$ states that two parallel processes which perform dual actions can synchronise by an internal transition. Rules $\langle\mathrm{Par}_L\rangle/\langle\mathrm{Par}_R\rangle$ and $\langle\mathrm{Alpha}\rangle$ close the LTS under parallel composition and α-renaming.

5.2 Environmental Labelled Transition System

Fig. 4 defines a labelled transition relation between a triple of environments, denoted $(\Gamma_1, \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma_2, \Lambda_2, \Delta_2)$. It extends the transition systems in [14, 13] to higher-order sessions.

Input Actions are defined by rules [SRv] and [ShRv]. In rule [SRv] the type of value V and the type of the object associated to the session type on s should coincide. Moreover, the resulting type tuple must contain the environments associated to V. The rule requires that the dual endpoint $\overline{s}$ is not present in the session environment, since if it were present the only communication that could take place is the interaction between the two endpoints (using rule [Tau] below). Rule [ShRv] is for shared names and follows similar principles.

Output Actions are defined by rules [SSnd] and [ShSnd]. Rule [SSnd] states the conditions for observing action $(\nu\, \tilde{m})s!\langle V\rangle$ on a type tuple $(\Gamma, \Lambda, \Delta \cdot s{:}S)$. The session environment Δ with $s{:}S$ should include the session environment of the sent value V, *excluding* the session environments of names m_j in $\tilde{m}$ which restrict the scope of value V. Similarly, the linear variable environment Λ' of V should be included in Λ. Scope extrusion of session names in $\tilde{m}$ requires that the dual endpoints of $\tilde{m}$ should appear in the resulting session environment. Similarly for shared names in $\tilde{m}$ that are extruded. All free values used for typing V are subtracted from the resulting type tuple. The prefix of session s is consumed by the action. Rule [ShSnd] is for output actions on shared names: the name must be typed with $\langle U\rangle$; conditions on V are identical to those on rule [SSnd].

Other Actions Rules [Sel] and [Bra] describe actions for select and branch. Hidden

$$[\text{SRv}] \quad \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta \cdot s :?(U); S) \xrightarrow{s?\langle V \rangle} (\Gamma; \Lambda \cdot \Lambda'; \Delta \cdot \Delta' \cdot s : S)}$$

$$[\text{ShRv}] \quad \frac{\Gamma; \emptyset; \emptyset \vdash a \triangleright \langle U \rangle \quad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta) \xrightarrow{a?\langle V \rangle} (\Gamma; \Lambda \cdot \Lambda'; \Delta \cdot \Delta')}$$

$$[\text{SSnd}] \quad \frac{\begin{array}{lll} \Gamma \cdot \Gamma'; \Lambda'; \Delta' \vdash V \triangleright U & \Gamma'; \emptyset; \Delta_j \vdash m_j \triangleright U_j & \overline{s} \notin \mathtt{dom}(\Delta) \\ \Delta' \backslash \cup_j \Delta_j \subseteq (\Delta \cdot s : S) & \Gamma'; \emptyset; \Delta_j' \vdash \overline{m}_j \triangleright U_j' & \Lambda' \subseteq \Lambda \end{array}}{(\Gamma; \Lambda; \Delta \cdot s :!\langle U \rangle; S) \xrightarrow{(\nu\, \tilde{m}) s!\langle V \rangle} (\Gamma \cdot \Gamma'; \Lambda \backslash \Lambda'; (\Delta \cdot s : S \cdot \cup_j \Delta_j') \backslash \Delta')}$$

$$[\text{ShSnd}] \quad \frac{\begin{array}{lll} \Gamma \cdot \Gamma'; \Lambda'; \Delta' \vdash V \triangleright U & \Gamma'; \emptyset; \Delta_j \vdash m_j \triangleright U_j & \Gamma; \emptyset; \emptyset \vdash a \triangleright \langle U \rangle \\ \Delta' \backslash \cup_j \Delta_j \subseteq \Delta & \Gamma'; \emptyset; \Delta_j' \vdash \overline{m}_j \triangleright U_j' & \Lambda' \subseteq \Lambda \end{array}}{(\Gamma; \Lambda; \Delta) \xrightarrow{(\nu\, \tilde{m}) a!\langle V \rangle} (\Gamma \cdot \Gamma'; \Lambda \backslash \Lambda'; (\Delta \cdot \cup_j \Delta_j') \backslash \Delta')}$$

$$[\text{Sel}] \quad \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma; \Lambda; \Delta \cdot s : \oplus\{l_i : S_i\}_{i \in I}) \xrightarrow{s \oplus l_j} (\Gamma; \Lambda; \Delta \cdot s : S_j)}$$

$$[\text{Bra}] \quad \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma; \Lambda; \Delta \cdot s : \&\{l_i : T_i\}_{i \in I}) \xrightarrow{s \& l_j} (\Gamma; \Lambda; \Delta \cdot s : S_j)}$$

$$[\text{Tau}] \quad \frac{\Delta_1 \longrightarrow \Delta_2 \vee \Delta_1 = \Delta_2}{(\Gamma; \Lambda; \Delta_1) \xrightarrow{\tau} (\Gamma; \Lambda; \Delta_2)}$$

Figure 4 Labelled Transition System for Typed Environments.

transitions defined by rule [Tau] do not change the session environment or they follow the reduction on session environments (Def. 2).
The weakening property on shared environments ensures that $(\Gamma_2, \Lambda_1, \Delta_1) \overset{\ell}{\longmapsto} (\Gamma_2, \Lambda_2, \Delta_2)$ if $(\Gamma_1, \Lambda_1, \Delta_1) \overset{\ell}{\longmapsto} (\Gamma_2, \Lambda_2, \Delta_2)$. We define the typed LTS combining the LTSs in Fig. 3 and Fig. 4.

▶ **Definition 4** (Typed Transition System). A *typed transition relation* is a typed relation $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$ where: (1) $P_1 \xrightarrow{\ell} P_2$ and (2) $(\Gamma, \emptyset, \Delta_1) \xrightarrow{\ell} (\Gamma, \emptyset, \Delta_2)$ with $\Gamma; \emptyset; \Delta_i \vdash P_i \triangleright \diamond$ $(i = 1, 2)$. We extend to $\Longrightarrow$ and $\overset{\hat{\ell}}{\Longrightarrow}$ where we write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$, $\overset{\ell}{\Longrightarrow}$ for the transitions $\Longrightarrow \xrightarrow{\ell} \Longrightarrow$, and $\overset{\hat{\ell}}{\Longrightarrow}$ for $\overset{\ell}{\Longrightarrow}$ if $\ell \neq \tau$ otherwise $\Longrightarrow$.

5.3 Reduction-Closed, Barbed Congruence

We define *typed relations* and *contextual equivalence*. We first define *confluence* over session environments Δ: we denote $\Delta_1 \rightleftharpoons \Delta_2$ if there exists Δ such that $\Delta_1 \longrightarrow^* \Delta$ and $\Delta_2 \longrightarrow^* \Delta$ (here we write $\longrightarrow^*$ for the multi-step reduction in Def. 2).

▶ **Definition 5.** We say that $\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \ \Re \ \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$ is a *typed relation* whenever P_1 and P_2 are closed; Δ_1 and Δ_2 are balanced; and $\Delta_1 \rightleftharpoons \Delta_2$. We write $\Gamma; \Delta_1 \vdash P_1 \ \Re \ \Delta_2 \vdash P_2$

for the typed relation $\Gamma; \emptyset; \Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma; \emptyset; \Delta_2 \vdash P_2 \triangleright \diamond$.

Typed relations relate only closed terms whose session environments are balanced and confluent. Next we define *barbs* [17] with respect to types.

▶ **Definition 6** (Barbs)**.** Let P be a closed process. We define:

1. $P \downarrow_n$ if $P \equiv (\nu\, \tilde{m})(n!\langle V\rangle.P_2 \mid P_3), n \notin \tilde{m}$.
2. $\Gamma; \Delta \vdash P \downarrow_n$ if $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$ with $P \downarrow_n$ and $\overline{n} \notin \mathtt{dom}(\Delta)$.
   $\Gamma; \Delta \vdash P \Downarrow_n$ if $P \longrightarrow^* P'$ and $\Gamma; \Delta' \vdash P' \downarrow_n$.

A barb $\downarrow_n$ is an observable on an output prefix with subject n. Similarly a weak barb $\Downarrow_n$ is a barb after a number of reduction steps. Typed barbs $\downarrow_n$ (resp. $\Downarrow_n$) occur on typed processes $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$. When n is a session name we require that its dual endpoint $\overline{n}$ is not present in the session environment Δ.

To define a congruence relation, we introduce the family $\mathbb{C}$ of contexts:

$$\begin{array}{rl}\mathbb{C} ::= & - \;\mid\; u!\langle V\rangle.\mathbb{C} \;\mid\; u?(x).\mathbb{C} \;\mid\; u!\langle \lambda x.\mathbb{C}\rangle.P \;\mid\; (\nu\, n)\mathbb{C} \\ \mid & (\lambda x.\mathbb{C})u \;\mid\; \mu X.\mathbb{C} \;\mid\; \mathbb{C} \mid P \;\mid\; P \mid \mathbb{C} \\ \mid & u \triangleleft l.\mathbb{C} \;\mid\; u \triangleright \{l_1 : P_1, \cdots, l_i : \mathbb{C}, \cdots, l_n : P_n\}\end{array}$$

Notation $\mathbb{C}[P]$ replaces the hole $-$ in $\mathbb{C}$ with P.

The first behavioural relation we define is reduction-closed, barbed congruence [8].

▶ **Definition 7** (Reduction-Closed, Barbed Congruence)**.** Typed relation $\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash P_2$ is a *reduction-closed, barbed congruence* whenever:

1) If $P_1 \longrightarrow P_1'$ then there exist P_2', Δ_2' such that $P_2 \longrightarrow^* P_2'$ and $\Gamma; \Delta_1' \vdash P_1' \;\Re\; \Delta_2' \vdash P_2'$;
2) If $\Gamma; \Delta_1 \vdash P_1 \downarrow_n$ then $\Gamma; \Delta_2 \vdash P_2 \Downarrow_n$;
3) For all $\mathbb{C}$, there exist Δ_1'', Δ_2'': $\Gamma; \Delta_1'' \vdash \mathbb{C}[P_1] \;\Re\; \Delta_2'' \vdash \mathbb{C}[P_2]$;
4) The symmetric cases of 1 and 2.

The largest such relation is denoted with $\cong$.

5.4 Context Bisimulation ($\approx$)

The first bisimulation which we define is the standard (weak) context bisimulation [21].

▶ **Definition 8** (Context Bisimulation)**.** A typed relation $\Re$ is *a context bisimulation* if for all $\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{(\nu\, \tilde{m}_1)n!\langle V_1\rangle} \Delta_1' \vdash P_2$, there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \tilde{m}_2)n!\langle V_2\rangle}{\Longrightarrow} \Delta_2' \vdash Q_2$ and for all R with $\mathtt{fv}(R) = x$:

$$\Gamma; \Delta_1'' \vdash (\nu\, \tilde{m}_1)(P_2 \mid R\{V_1/x\}) \;\Re\; \Delta_2'' \vdash (\nu\, \tilde{m}_2)(Q_2 \mid R\{V_2/x\});$$

2) For all $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Longrightarrow} \Delta_2' \vdash Q_2$ and $\Gamma; \Delta_1' \vdash P_2 \;\Re\; \Delta_2' \vdash Q_2$; and
3) The symmetric cases of 1 and 2.

The largest such bisimulation is called context bisimilarity and denoted by $\approx$.

As explained in § 2, in the general case, context bisimulation is hard to compute. Below we introduce $\approx^H$ and $\approx^C$.

$$
\begin{array}{rcl}
\llparenthesis ?(U);S \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u?(x).(\llparenthesis S \rrparenthesis^{u} \mid \llparenthesis U \rrparenthesis^{x}) \\
\llparenthesis !\langle U\rangle;S \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u!\langle \llparenthesis U \rrparenthesis_{\mathsf{c}} \rangle.\llparenthesis S \rrparenthesis^{u} \\
\llparenthesis \oplus\{l:S\} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u \triangleleft l.\llparenthesis S \rrparenthesis^{u} \\
\llparenthesis \&\{l_i:S_i\}_{i\in I} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & u \triangleright \{l_i : \llparenthesis S_i \rrparenthesis^{u}\}_{i\in I} \\
\llparenthesis \mathsf{t} \rrparenthesis^{u} \stackrel{\mathsf{def}}{=} X_{\mathsf{t}} & & \llparenthesis \mu \mathsf{t}.S \rrparenthesis^{u} \stackrel{\mathsf{def}}{=} \mu X_{\mathsf{t}}.\llparenthesis S \rrparenthesis^{u} \\
\llparenthesis \mathsf{end} \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & \mathbf{0} \\
\llparenthesis \langle S\rangle \rrparenthesis^{u} \stackrel{\mathsf{def}}{=} u!\langle \llparenthesis S \rrparenthesis_{\mathsf{c}} \rangle.\mathbf{0} & & \llparenthesis \langle L\rangle \rrparenthesis^{u} \stackrel{\mathsf{def}}{=} u!\langle \llparenthesis L \rrparenthesis_{\mathsf{c}} \rangle.\mathbf{0} \\
\llparenthesis C \!\rightarrow\!\diamond \rrparenthesis^{u} & \stackrel{\mathsf{def}}{=} & \llparenthesis C \!\multimap\!\diamond \rrparenthesis^{u} \stackrel{\mathsf{def}}{=} u\, \llparenthesis C \rrparenthesis_{\mathsf{c}}
\end{array}
$$

$$
\begin{array}{rclr}
\llparenthesis S \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & s & (s \text{ fresh}) \\
\llparenthesis \langle S\rangle \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} \llparenthesis \langle L\rangle \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & a & (a \text{ fresh}) \\
\llparenthesis C \!\rightarrow\!\diamond \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} \llparenthesis C \!\multimap\!\diamond \rrparenthesis_{\mathsf{c}} & \stackrel{\mathsf{def}}{=} & \lambda x.\, \llparenthesis C \rrparenthesis^{x} &
\end{array}
$$

Figure 5 Characteristic Processes (top) and Values (bottom).

5.5 Higher-Order and Characteristic Bisimulations ($\approx^{H}/\approx^{C}$)

We formalise the novelties motivated in § 2. Our main result is Thm. 14. We define characteristic processes/values:

▶ **Definition 9** (Characteristic Process and Values)**.** Let u and U be a name and a type, respectively. Fig. 5 defines the *characteristic process* $\llparenthesis U \rrparenthesis^{u}$ and the *characteristic value* $\llparenthesis U \rrparenthesis_{\mathsf{c}}$.

We can easily verify that characteristic processes/values are inhabitants of their associated type. The following example motivates the refined LTS explained in § 2.

▶ **Example 10** (The Need for Refined Typed LTS)**.** First we demonstrate that observing a characteristic value input alone is not sufficient to define a sound bisimulation closure. Consider typed processes P_1, P_2:

$$P_1 = s?(x).(x\, s_1 \mid x\, s_2) \qquad P_2 = s?(x).(x\, s_1 \mid s_2?(y).\mathbf{0}) \tag{5}$$

where $\Gamma;\emptyset;\Delta\cdot s :?((?(C);\mathsf{end})\!\rightarrow\!\diamond);\mathsf{end} \vdash P_i \triangleright \diamond$ $(i \in \{1,2\})$. If the above processes input and substitute over x the characteristic value $\llparenthesis (?(C);\mathsf{end})\!\rightarrow\!\diamond \rrparenthesis_{\mathsf{c}} = \lambda x.\, x?(y).\mathbf{0}$, then they evolve into:

$$\Gamma;\emptyset;\Delta \vdash s_1?(y).\mathbf{0} \mid s_2?(y).\mathbf{0} \triangleright \diamond$$

therefore becoming context bisimilar. However, the processes in (5) are clearly *not* context bisimilar: there are many input actions which may be used to distinguish them. For example, if P_1 and P_2 input $\lambda x.\, (\nu\, s)(a!\langle s\rangle.x?(y).\mathbf{0})$ with $\Gamma;\emptyset;\Delta \vdash s \triangleright \mathsf{end}$, then their derivatives are not bisimilar.

Observing only the characteristic value results in an under-discriminating bisimulation. However, if a trigger value, $\lambda x.\, t?(y).(y\, x)$ is received on s, then we can distinguish processes in (5):

$$P_1 \stackrel{\ell}{\Longrightarrow} t?(x).(x\, s_1) \mid t?(x).(x\, s_2) \quad \text{and} \quad P_2 \stackrel{\ell}{\Longrightarrow} t?(x).(x\, s_1) \mid s_2?(y).\mathbf{0}$$

where $\ell = s?\langle \lambda x.\, t?(y).(y\, x)\rangle$. One question that arises here is whether the trigger value is enough to distinguish two processes, hence no need of characteristic values as the input. This is not the case since the trigger value alone also results in an over-discriminating bisimulation relation. In fact the trigger value can be observed on any input prefix of *any type*. For example, consider the following processes:

$$\Gamma; \emptyset; \Delta \vdash (\nu\, s)(n?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, P\rangle.\mathbf{0}) \triangleright \diamond \tag{6}$$
$$\Gamma; \emptyset; \Delta \vdash (\nu\, s)(n?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, Q\rangle.\mathbf{0}) \triangleright \diamond \tag{7}$$

if processes in (6)/(7) input the trigger value, we obtain:

$$(\nu\, s)(t?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, P\rangle.\mathbf{0}) \text{ and } (\nu\, s)(t?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, Q\rangle.\mathbf{0})$$

thus we can easily derive a bisimulation closure if we assume a bisimulation definition that allows only trigger value input. But if processes in (6)/(7) input the characteristic value $\lambda z.\, z?(x).(x\, m)$, then they would become:

$$\Gamma; \emptyset; \Delta \vdash (\nu\, s)(s?(x).(x\, m) \mid \overline{s}!\langle \lambda x.\, P\rangle.\mathbf{0}) \approx \Delta \vdash P\{m/x\}$$
$$\Gamma; \emptyset; \Delta \vdash (\nu\, s)(s?(x).(x\, m) \mid \overline{s}!\langle \lambda x.\, Q\rangle.\mathbf{0}) \approx \Delta \vdash Q\{m/x\}$$

which are not bisimilar if $P\{m/x\} \not\approx Q\{m/x\}$.

As explained in § 2, we define the *refined* typed LTS by considering a transition rule for input in which admitted values are trigger or characteristic values or names:

▶ **Definition 11** (Refined Typed Labelled Transition Relation). We define the environment transition rule for input actions using the input rules in Fig. 4:

$$[\text{RRcv}]\quad \frac{(\Gamma_1; \Lambda_1; \Delta_1) \xrightarrow{n?\langle V\rangle} (\Gamma_2; \Lambda_2; \Delta_2) \qquad V = m \vee V \equiv \lambda x.\, t?(y).(y\, x) \vee V \equiv \{\!\!\{U\}\!\!\}_{\mathsf{c}} \text{ with } t \text{ fresh}}{(\Gamma_1; \Lambda_1; \Delta_1) \overset{n?\langle V\rangle}{\longmapsto} (\Gamma_2; \Lambda_2; \Delta_2)}$$

[RRcv] is defined on top of rules [SRv] and [ShRv] in Fig. 4. We then use the non-receiving rules in Fig. 4 together with rule [RRcv] to define $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\longmapsto} \Delta_2 \vdash P_2$ as in Def. 4.

Note $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\longmapsto} \Delta_2 \vdash P_2$ implies $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$. Below we sometimes write $\overset{(\nu\, \tilde{m})n!\langle V:U\rangle}{\longmapsto}$ when the type of V is U.

The Two Bisimulations. We define *higher-order* and *characteristic bisimulations*, two tractable bisimulations for HO and $\mathsf{HO}\pi$. As explained in § 2, the two bisimulations use different trigger processes:

$$\begin{aligned} t \Leftarrow V_1 &\overset{\text{def}}{=} t?(x).(\nu\, s)(x\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \\ t \Leftarrow V : U &\overset{\text{def}}{=} t?(x).(\nu\, s)([?(U); \mathsf{end}]^s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \end{aligned}$$

▶ **Definition 12** (Higher-Order Bisimilarity). A typed relation $\Re$ is a *HO bisimulation* if for all $\Gamma; \Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash Q_1$

1) Whenever $\Gamma; \Delta_1 \vdash P_1 \overset{(\nu\, \tilde{m}_1)n!\langle V_1\rangle}{\longmapsto} \Delta_1' \vdash P_2$, there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \tilde{m}_2)n!\langle V_2\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for fresh t,

$$\Gamma; \Delta_1'' \vdash (\nu\, \tilde{m}_1)(P_2 \mid t \Leftarrow V_1) \;\Re\; \Delta_2'' \vdash (\nu\, \tilde{m}_2)(Q_2 \mid t \Leftarrow V_2)$$

2) For all $\Gamma; \Delta_1 \vdash P_1 \overset{\ell}{\longmapsto} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{\hat{\ell}}{\Longmapsto} \Delta_2' \vdash Q_2$ and $\Gamma; \Delta_1' \vdash P_2 \Re \Delta_2' \vdash Q_2$; and
3) The symmetric cases of 1 and 2.

The largest such bisimulation is called *higher-order bisimilarity* and denoted by $\approx^H$.

We define characteristic bisimilarity:

▶ **Definition 13** (Characteristic Bisimilarity). Characteristic bisimilarity, denoted by $\approx^C$, is defined by replacing Clause 1) in Def. 12 with the following clause:

Whenever $\Gamma; \Delta_1 \vdash P_1 \overset{(\nu\, \tilde{m}_1)n!\langle V_1:U\rangle}{\longmapsto} \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \tilde{m}_2)n!\langle V_2:U\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for fresh t,
$\Gamma; \Delta_1'' \vdash (\nu\, \tilde{m}_1)(P_2 \mid t \Leftarrow V_1 : U_1) \Re \Delta_2'' \vdash (\nu\, \tilde{m}_2)(Q_2 \mid t \Leftarrow V_2 : U_2)$

We now state our main theorem: typed bisimilarities collapse.

▶ **Theorem 14** (Coincidence). $\cong$, $\approx$, $\approx^H$ and $\approx^C$ coincide in $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}\}$ and $\cong$, $\approx$ and $\approx^C$ coincide in $\mathsf{C} \in \{\mathsf{HO}\pi, \mathsf{HO}, \pi\}$.

Thus, we may use $\approx^H$ for tractable reasoning in the higher-order setting; in the first-order setting of π we can still use $\approx^C$.

Processes that do not use shared names are inherently deterministic. The following determinacy property will be useful for proving our expressiveness results (§ ??). We require an auxiliary definition. A transition $\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ is said *deterministic* if it is derived using $\langle \text{App} \rangle$ or $\langle \text{Tau} \rangle$ (where $\mathtt{subj}(\ell_1)$ and $\mathtt{subj}(\ell_2)$ in the premise are dual endpoints), possibly followed by uses of $\langle \text{Alpha} \rangle$, $\langle \text{Res} \rangle$, $\langle \text{Rec} \rangle$, or $\langle \text{Par}_L \rangle / \langle \text{Par}_R \rangle$.

▶ **Lemma 15** (τ-Inertness). 1) Let $\Gamma; \Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ be a deterministic transition, with balanced Δ. Then $\Gamma; \Delta \vdash P \cong \Delta' \vdash P'$ with $\Delta \longrightarrow^* \Delta'$ balanced.
2) Let P be an $\mathsf{HO}\pi^{-\mathsf{sh}}$ process. Assume $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$. Then $P \longrightarrow^* P'$ implies $\Gamma; \Delta \vdash P \cong \Delta' \vdash P'$ with $\Delta \longrightarrow^* \Delta'$.

6 Related Work

Session Typed Processes. The works [5, 4] study encodings of binary session calculi into a linearly typed π-calculus. While [5] gives a precise encoding of π into a linear calculus (an extension of [2]), the work [4] gives operational correspondence (without full abstraction, cf. Def. ??-4) for the first- and higher-order π-calculi into [10]. They investigate embeddability of two different typing systems; by the result of [5], $\mathsf{HO}\pi^+$ is encodable into the linearly typed π-calculi.

The syntax of $\mathsf{HO}\pi$ is a subset of that in [18, 19]. The work [18] develops a full higher-order session calculus with process abstractions and applications; it admits the type $U = U_1 \to U_2 \ldots U_n \to \diamond$ and its linear type U^1 which corresponds to $\tilde{U} \!\to\! \diamond$ and $\tilde{U} \multimap \diamond$ in a super-calculus of $\mathsf{HO}\pi^+$ and $\mathsf{HO}\vec{\pi}$. Our results show that the calculus in [18] is not only expressed but also reasoned in HO (with limited form of arrow types, $C \!\to\! \diamond$ and $C \multimap \diamond$), via precise encodings. None of the above works proposes tractable bisimulations for higher-order processes.

Typed Behavioural Equivalences. This work follows the session type behavioural semantics in [14, 13, 20] where a bisimulation is defined on a LTS that assumes a session typed observer. Our theory for higher-order session types differentiates from the work in [14, 13], which considers the first-order binary and multiparty session types, respectively. The work [20] gives a behavioural theory for a logically motivated language of binary sessions without shared names.

Our approach to typed equivalences builds upon techniques by Sangiorgi [23, 21] and Jeffrey and Rathke [9]. The work [23] introduced the first fully-abstract encoding from the higher-order π-calculus into the π-calculus. Sangiorgi's encoding is based on the idea of a replicated input-guarded process (a trigger process). We use a similar replicated triggered process to encode $\mathsf{HO}\pi$ into π (Def. **??**). Operational correspondence for the triggered encoding is shown using a context bisimulation with first-order labels. To deal with the issue of context bisimilarity, Sangiorgi proposes *normal bisimilarity*, a tractable equivalence without universal quantification. To prove that context and normal bisimilarities coincide, [23] uses triggered processes. Triggered bisimulation is also defined on first-order labels where the context bisimulation is restricted to arbitrary trigger substitution. This characterisation of context bisimilarity was refined in [9] for calculi with recursive types, not addressed in [21, 23] and quite relevant in our work (cf. Def. **??**). The bisimulation in [9] is based on an LTS which is extended with trigger meta-notation. As in [21, 23], the LTS in [9] observes first-order triggered values instead of higher-order values, offering a more direct characterisation of contextual equivalence and lifting the restriction to finite types. We briefly contrast the approach in [9] and ours based on higher-order ($\approx^H$) and characteristic ($\approx^C$) bisimilarities:

- The LTS in [9] is enriched with extra labels for triggers; an output action transition emits a trigger and introduces a parallel replicated trigger. Our approach retains usual labels/transitions; in case of output, $\approx^H$ and $\approx^C$ introduce a parallel non-replicated trigger.
- Higher-order input in [9] involves the input of a trigger which reduces after substitution. Rather than a trigger name, $\approx^H$ and $\approx^C$ decree the input of a triggered value $\lambda z.\, t?(x).x\, z$.
- Unlike [9], $\approx^C$ treats first- and higher-order values uniformly. As the typed LTS distinguishes linear and shared values, replicated closures are used only for shared values.
- In [9] a matching construct is crucial to prove completeness of bisimilarity, while our calculi lack matching. In contrast, we use the characteristic process interaction with the environment, exploiting session type structures, i.e., instead of matching a name is embedded into a process and then observe its behaviour.

The *environmental bisimulations* given in [24] use a higher-order LTS to define a bisimulation that stores the observer's knowledge; hence, observed actions are based on this knowledge at any given time. This approach is enhanced in [12, 11] where a mapping from constants to higher-order values is introduced. This allows to observe first-order values instead of higher-order values. It differs from [21, 9] in that the mapping between higher- and first-order values is no longer implicit.

References

1 Full version of this paper. Technical report, Imperial College / Univ. of Groningen, 2015. `http://www.jorgeaperez.net/publications/coresessions`.

2 Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, volume 2044 of *LNCS*, pages 29–45, 2001.

3 Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *TGC*, LNCS, 2014. To appear.

4 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. PPDP, pages 139–150. ACM, 2012.

5 Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.

6 Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.

7 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

8 Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *TCS*, 151(2):437–486, 1995.

9 Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. *LMCS*, 1(1), 2005.

10 Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the Pi-Calculus. *TOPLAS*, 21(5):914–947, September 1999.

11 Vasileios Koutavas and Matthew Hennessy. A testing theory for a higher-order cryptographic language. In *ESOP*, volume 6602 of *LNCS*, pages 358–377, 2011.

12 Vasileios Koutavas and Matthew Hennessy. First-order reasoning for higher-order concurrency. *Computer Languages, Systems & Structures*, 38(3):242–277, 2012.

13 Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. *LMCS*, 10(4), 2014.

14 Dimitrios Kouzapas, Nobuko Yoshida, Raymond Hu, and Kohei Honda. On asynchronous eventful session semantics. *MSCS*, 2015.

15 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In *ICALP*, volume 6199, pages 442–453, 2010.

16 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

17 Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

18 Dimitris Mostrous and Nobuko Yoshida. Two session typing systems for higher-order mobile processes. In *TLCA*, volume 4583 of *LNCS*, pages 321–335. Springer, 2007.

19 Dimitris Mostrous and Nobuko Yoshida. Session Typing and Asynchronous Subtying for Higher-Order π-Calculus. *Info.& Comp.*, 2015. To appear.

20 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014.

21 D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Inf. & Comp.*, 131(2):141–178, 1996.

22 D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *TCS*, 167(2):235–274, 1996.

23 Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

24 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *LICS*, pages 293–302. IEEE, 2007.

25 Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.

26 Bent Thomsen. Plain CHOCS: A Second Generation Calculus for Higher Order Processes. *Acta Informatica*, 30(1):1–59, 1993.

27 Xian Xu. Distinguishing and relating higher-order and first-order processes by expressiveness. *Acta Informatica*, 49(7-8):445–484, 2012.

A Appendix: the Typing System of HOπ

In this appendix we formally define *type equivalence* and *duality*. We also present and describe our typing rules, given in Fig. 6.

▶ **Definition 16** (Type Equivalence). Let ST a set of closed session types. Two types S and S' are said to be *isomorphic* if a pair (S, S') is in the largest fixed point of the monotone function $F : \mathcal{P}(\mathsf{ST} \times \mathsf{ST}) \to \mathcal{P}(\mathsf{ST} \times \mathsf{ST})$ defined by:

$$\begin{array}{rcl}
F(\Re) & = & \{(\mathtt{end}, \mathtt{end})\} \\
& \cup & \{(!\langle U_1\rangle; S_1, !\langle U_2\rangle; S_2) \mid (S_1, S_2), (U_1, U_2) \in \Re\} \\
& \cup & \{(?(U_1); S_1, ?(U_2); S_2) \mid (S_1, S_2), (U_1, U_2) \in \Re\} \\
& \cup & \{(\&\{l_i : S_i\}_{i\in I}\, , \&\{l_i : S_i'\}_{i\in I}) \mid \forall i \in I.(S_i, S_i') \in \Re\} \\
& \cup & \{(\oplus\{l_i : S_i\}_{i\in I}\, , \oplus\{l_i : S_i'\}_{i\in I}) \mid \forall i \in I.(S_i, S_i') \in \Re\} \\
& \cup & \{(\mu \mathtt{t}.S, S') \mid (S\{^{\mu \mathtt{t}.S}/_{\mathtt{t}}\}, S') \in \Re\} \\
& \cup & \{(S, \mu \mathtt{t}.S') \mid (S, S'\{^{\mu \mathtt{t}.S'}/_{\mathtt{t}}\}) \in \Re\}
\end{array}$$

Standard arguments ensure that F is monotone, thus the greatest fixed point of F exists. We write $S_1 \sim S_2$ if $(S_1, S_2) \in \Re$.

▶ **Definition 17** (Duality). Let ST a set of closed session types. Two types S and S' are said to be *dual* if a pair (S, S') is in the largest fixed point of the monotone function $F : \mathcal{P}(\mathsf{ST} \times \mathsf{ST}) \to \mathcal{P}(\mathsf{ST} \times \mathsf{ST})$ defined by:

$$\begin{array}{rcl}
F(\Re) & = & \{(\mathtt{end}, \mathtt{end})\} \\
& \cup & \{(!\langle U_1\rangle; S_1, ?(U_2); S_2) \mid (S_1, S_2) \in \Re,\ U_1 \sim U_2\} \\
& \cup & \{(?(U_1); S_1, !\langle U_2\rangle; S_2) \mid (S_1, S_2) \in \Re,\ U_1 \sim U_2\} \\
& \cup & \{(\oplus\{l_i : S_i\}_{i\in I}\, , \&\{l_i : S_i'\}_{i\in I}) \mid \forall i \in I.(S_i, S_i') \in \Re\} \\
& \cup & \{(\&\{l_i : S_i\}_{i\in I}\, , \oplus\{l_i : S_i'\}_{i\in I}) \mid \forall i \in I.(S_i, S_i') \in \Re\} \\
& \cup & \{(\mu \mathtt{t}.S, S') \mid (S\{^{\mu \mathtt{t}.S}/_{\mathtt{t}}\}, S') \in \Re\} \\
& \cup & \{(S, \mu \mathtt{t}.S') \mid (S, S'\{^{\mu \mathtt{t}.S'}/_{\mathtt{t}}\}) \in \Re\}
\end{array}$$

Standard arguments ensure that F is monotone, thus the greatest fixed point of F exists. We write S_1 dual S_2 if $(S_1, S_2) \in \Re$.

Typing System of HOπ The typing system is defined in Fig. 6. Rules [Sess, Sh, LVar] are name and variable introduction rules. The type $C{\rightarrow}\diamond$ is derived using rule [Prom], where we require a value with linear type to be typed without a linear environment to be typed as a shared type. Rule [EProm] allows to freely use a linear type variable as shared.

Abstraction values are typed with rule [Abs]. The dual of abstraction typing is application typing governed by rule [App], where we expect the type C of an application name u to match the type $C{\multimap}\diamond$ or $C{\rightarrow}\diamond$ of the application variable x.

In [Send], the type U of a send value V should appear as a prefix on the session type $!\langle U\rangle; S$ of u. [Rcv] is its dual. We use a similar approach with session prefixes to type interaction between shared names as defined in rules [Req] and [Acc], where the type of the sent/received object (S and L, respectively) should match the type of the sent/received subject ($\langle S\rangle$ and $\langle L\rangle$, respectively). Rules for selection and branching, [Sel] and [Bra], are standard.

A shared name creation a creates and restricts a in environment Γ as defined in rule [Res]. Creation of a session name s creates and restricts two endpoints with dual types in rule [ResS]. Rule [Par], combines the environments Λ and Δ of the parallel components of a parallel process. The disjointness of environments Λ and Δ is implied. Rule [End] adds the names with type $\mathtt{end}$ in Δ. The recursion requires that the body process matches the type of the recursive variable as in rule [Rec]. The recursive variable is typed directly from the shared environment Γ as in rule [RVar]. The inactive process $\mathbf{0}$ is typed with no linear environments as in [Nil].

$$\text{[Sess]}\quad \Gamma; \emptyset; \{u : S\} \vdash u \triangleright S \qquad \text{[Sh]}\quad \Gamma \cdot u : U; \emptyset; \emptyset \vdash u \triangleright U$$

$$\text{[LVar]}\quad \Gamma; \{x : C{\multimap}\diamond\}; \emptyset \vdash x \triangleright C{\multimap}\diamond$$

$$\text{[Prom]}\quad \frac{\Gamma; \emptyset; \emptyset \vdash V \triangleright C{\multimap}\diamond}{\Gamma; \emptyset; \emptyset \vdash V \triangleright C{\rightarrow}\diamond} \qquad \text{[EProm]}\quad \frac{\Gamma; \Lambda \cdot x : C{\multimap}\diamond; \Delta \vdash P \triangleright \diamond}{\Gamma \cdot x : C{\rightarrow}\diamond; \Lambda; \Delta \vdash P \triangleright \diamond}$$

$$\text{[Abs]}\quad \frac{\Gamma; \Lambda; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \Delta_2 \vdash x \triangleright C}{\Gamma \backslash x; \Lambda; \Delta_1 \backslash \Delta_2 \vdash \lambda x.\, P \triangleright C{\multimap}\diamond}$$

$$\text{[App]}\quad \frac{U = C{\multimap}\diamond \vee C{\rightarrow}\diamond \quad \Gamma; \Lambda; \Delta_1 \vdash V \triangleright U \quad \Gamma; \emptyset; \Delta_2 \vdash u \triangleright C}{\Gamma; \Lambda; \Delta_1 \cdot \Delta_2 \vdash V\, u \triangleright \diamond}$$

$$\text{[Send]}\quad \frac{\Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash V \triangleright U \quad u : S \in \Delta_1 \cdot \Delta_2}{\Gamma; \Lambda_1 \cdot \Lambda_2; ((\Delta_1 \cdot \Delta_2) \setminus u : S) \cdot u :!\langle U \rangle; S \vdash u!\langle V \rangle.P \triangleright \diamond}$$

$$\text{[Rcv]}\quad \frac{\Gamma; \Lambda_1; \Delta_1 \cdot u : S \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash x \triangleright U}{\Gamma \backslash x; \Lambda_1 \cdot \Lambda_2; \Delta_1 \backslash \Delta_2 \cdot u :?(U); S \vdash u?(x).P \triangleright \diamond}$$

$$\text{[Req]}\quad \frac{\begin{array}{c}\Gamma; \emptyset; \emptyset \vdash u \triangleright U_1 \quad \Gamma; \Lambda; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \emptyset; \Delta_2 \vdash V \triangleright U_2 \\ (U_1 = \langle S \rangle \wedge U_2 = S) \vee (U_1 = \langle L \rangle \wedge U_2 = L)\end{array}}{\Gamma; \Lambda; \Delta_1 \cdot \Delta_2 \vdash u!\langle V \rangle.P \triangleright \diamond}$$

$$\text{[Acc]}\quad \frac{\begin{array}{c}\Gamma; \emptyset; \emptyset \vdash u \triangleright U_1 \quad \Gamma; \Lambda_1; \Delta_1 \vdash P \triangleright \diamond \quad \Gamma; \Lambda_2; \Delta_2 \vdash x \triangleright U_2 \\ (U_1 = \langle S \rangle \wedge U_2 = S) \vee (U_1 = \langle L \rangle \wedge U_2 = L)\end{array}}{\Gamma \backslash x; \Lambda_1 \backslash \Lambda_2; \Delta_1 \backslash \Delta_2 \vdash u?(x).P \triangleright \diamond}$$

$$\text{[Bra]}\quad \frac{\forall i \in I \quad \Gamma; \Lambda; \Delta \cdot u : S_i \vdash P_i \triangleright \diamond}{\Gamma; \Lambda; \Delta \cdot u : \&\{l_i : S_i\}_{i \in I} \vdash u \triangleright \{l_i : P_i\}_{i \in I} \triangleright \diamond}$$

$$\text{[Sel]}\quad \frac{\Gamma; \Lambda; \Delta \cdot u : S_j \vdash P \triangleright \diamond \quad j \in I}{\Gamma; \Lambda; \Delta \cdot u : \oplus\{l_i : S_i\}_{i \in I} \vdash u \triangleleft l_j.P \triangleright \diamond}$$

$$\text{[ResS]}\quad \frac{\Gamma; \Lambda; \Delta \cdot s : S_1 \cdot \overline{s} : S_2 \vdash P \triangleright \diamond \quad S_1 \ \mathsf{dual}\ S_2}{\Gamma; \Lambda; \Delta \vdash (\nu\, s)P \triangleright \diamond}$$

$$\text{[Res]}\quad \frac{\Gamma \cdot a : \langle S \rangle; \Lambda; \Delta \vdash P \triangleright \diamond}{\Gamma; \Lambda; \Delta \vdash (\nu\, a)P \triangleright \diamond} \qquad \text{[Par]}\quad \frac{\Gamma; \Lambda_i; \Delta_i \vdash P_i \triangleright \diamond \quad i = 1, 2}{\Gamma; \Lambda_1 \cdot \Lambda_2; \Delta_1 \cdot \Delta_2 \vdash P_1 \mid P_2 \triangleright \diamond}$$

$$\text{[End]}\quad \frac{\Gamma; \Lambda; \Delta \vdash P \triangleright T \quad u \notin \mathtt{dom}(\Gamma, \Lambda, \Delta)}{\Gamma; \Lambda; \Delta \cdot u : \mathtt{end} \vdash P \triangleright \diamond} \qquad \text{[Rec]}\quad \frac{\Gamma \cdot X : \Delta; \emptyset; \Delta \vdash P \triangleright \diamond}{\Gamma; \emptyset; \Delta \vdash \mu X.P \triangleright \diamond}$$

$$\text{[RVar]}\quad \Gamma \cdot X : \Delta; \emptyset; \Delta \vdash X \triangleright \diamond \qquad \text{[Nil]}\quad \Gamma; \emptyset; \emptyset \vdash \mathbf{0} \triangleright \diamond$$

Figure 6 Typing Rules for $\mathsf{HO}\pi$.