Characteristic Bisimulations for Higher-Order Session Processes

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

1 **Imperial College London, UK**
2 **University of Glasgow, UK**
3 **University of Groningen, The Netherlands**

Abstract

Characterising contextual equivalence is a long-standing issue for higher-order (process) languages. In the setting of a higher-order π-calculus with sessions, we develop *characteristic bisimilarity*, a typed bisimilarity which fully characterises contextual equivalence. To our knowledge, ours is the first characterisation of its kind. Using simple values inhabiting (session) types, our approach distinguishes from untyped methods for characterising contextual equivalence in higher-order processes: we show that observing as inputs only a precise finite set of higher-order values suffices to reason about higher-order session processes. We demonstrate how characteristic bisimilarity can be used to justify optimisations in session protocols with mobile code communication.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs; F.3.2 Semantics of Programming Languages.

Keywords and phrases Behavioural equivalences, session types, higher-order process calculi.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Context. In *higher-order process calculi* communicated values may contain processes. Higher-order concurrency has received significant attention from untyped and typed perspectives (see, e.g., [15, 6, 10, 7, 13]). In this work, we consider $\mathsf{HO}\pi$, a higher-order process calculus with *session primitives*: in addition to functional abstractions/applications (as in the call-by-value λ-calculus), $\mathsf{HO}\pi$ contains constructs for synchronisation on shared names, session communication on linear names, and recursion. Thus, $\mathsf{HO}\pi$ processes may specify protocols for higher-order processes that can be type-checked using *session types* [4]. Although models of session communication with process passing exist [12, 3], their *behavioural equivalences* remain little understood. Since types can limit the contexts (environments) in which processes can interact, typed equivalences usually offer *coarser* semantics than untyped semantics. Hence, clarifying the status of these equivalences is key to, e.g., justify non-trivial optimisations in protocols involving both name- and process-passing.

A well-known behavioural equivalence for higher-order processes is *context bisimilarity* [16]. This characterisation of barbed congruence offers an adequate distinguishing power at the price of heavy universal quantifications in output clauses. Obtaining alternative characterisations of context bisimilarity is thus a recurring, important problem for higher-order calculi—see, e.g., [15, 16, 6, 7]. In particular, Sangiorgi [15, 16] has given characterisations of context bisimilarity for higher-order processes; such characterisations, however, do not scale to calculi with *recursive types*, which are essential to session-based concurrency. A characterisation that solves this limitation was developed by Jeffrey and Rathke in [6].

This Work. Building upon [15, 16, 6], our discovery is that *linearity* of session types plays a vital role in solving the open problem of characterising context bisimilarity for higher-order mobile processes with session communications. Our approach is to exploit the coarser semantics induced by session types to limit the behaviour of higher-order session processes. Formally, we enforce this limitation by defining a *refined* labelled transition system (LTS) which effectively narrows down the spectrum of allowed process behaviours, exploiting elementary processes inhabiting session types. We then introduce *characteristic bisimilarity*: this new notion of typed bisimilarity is *tractable*, in that it relies on the refined LTS for input actions and, more importantly, does not appeal to universal quantifications on output actions. Our main result is that characteristic bisimilarity coincides with context bisimilarity. Besides confirming the value of characteristic bisimilarity as an useful reasoning technique for higher-order processes with sessions, this result is remarkable also from a technical perspective, for associated completeness proofs do not require operators for name-matching, in contrast to untyped methods for higher-order processes with recursive types [6].

We explain how we exploit session types to define characteristic bisimilarity. Key notions are *triggered* and *characteristic processes/values.* Below, we write $s?(x).P$ for an input on endpoint s, and $\overline{s}!\langle V\rangle.Q$ for an output of value V on endpoint $\overline{s}$ (the *dual* of s). Also, $R \xrightarrow{s?\langle V\rangle} R'$ denotes an input transition along n and $R \xrightarrow{(\nu\,\widetilde{m})s!\langle V\rangle} R'$ denotes an output transition along s, sending value V, and extruding names $\widetilde{m}$. Weak transitions are as usual. Throughout the paper, we write $\Re, \Re', \ldots$ to denote binary relations on (typed) processes.

Issues of Context Bisimilarity. Context bisimilarity ($\approx$, Def. 10) is an overly demanding relation on higher-order processes. There are two issues, associated to demanding clauses for output and input actions. A *first issue* is the universal quantification on the output clause of context bisimilarity. Suppose $P \,\Re\, Q$, for some context bisimulation $\Re$. We have:

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

The *second issue* is due to inputs: it follows from the fact that we work with an *early* labelled transition system (LTS). Thus, an input prefix may observe infinitely many different values. To alleviate this burden, in *characteristic bisimilarity* ($\approx^{\mathtt{C}}$) we take two (related) steps:

(a) We replace ($\star$) with a clause involving a *more tractable* process closure; and
(b) We refine inputs to avoid observing infinitely many actions on the same input prefix.

Trigger Processes. To address (a), we exploit session types. We first observe that closure $R\{V/x\}$ in ($\star$) is context bisimilar to the process $P = (\nu\, s)((\lambda z.\, z?(x).R)\, s \mid \overline{s}!\langle V\rangle.\mathbf{0})$. 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. If T_V inputs value $\lambda z.\, z?(x).R$ then we can simulate the closure of P:

$$T_V = t?(x).(\nu\, s)(x\, s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \quad \text{and} \quad T_V \xrightarrow{t?\langle \lambda z.\, z?(x).R\rangle} P \approx R\{V/x\} \qquad (1)$$

Processes such as T_V offer a value at a fresh name; this class of ***trigger processes*** already suggests a tractable formulation of bisimilarity without the demanding clause ($\star$). Process T_V in (1) requires a higher-order communication along t. As we explain below, we can give an alternative trigger process; the key is using *elementary inhabitants* of session types.

Characteristic Processes and Values. To address (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. 11). This way, for instance, 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 $u?(x).(u!\langle s_2\rangle.\mathbf{0} \mid x\, s_1)$ is a characteristic process for S along u. Given a session type S, we write $\llparenthesis S \rrparenthesis^u$ for its characteristic process along u (cf. Def. 11). Also, given value type U, then $\llparenthesis U \rrparenthesis_{\mathsf{c}}$ denotes its characteristic value. As we explain now, we use $\llparenthesis U \rrparenthesis_{\mathsf{c}}$ to limit input transitions.

Refined Input Transitions. To refine input transitions, we need to observe an additional value, $\lambda x.\, t?(y).(y\, x)$, called the ***trigger value***. This is necessary: it turns out that a characteristic value alone as the observable input is not enough to define a sound bisimulation (cf. Ex. 12). Intuitively, the trigger value is used to observe/simulate application processes. Based on the above discussion, we refine the transition rule for input actions (cf. Def. 13). Roughly, the refined rule is:

$$\boxed{P \xrightarrow{s?\langle V\rangle} P' \wedge (V = m \vee V \equiv \lambda x.\, t?(y).(y\, x) \vee V \equiv \llparenthesis U \rrparenthesis_{\mathsf{c}} \text{ with } t \text{ fresh}) \;\; \Rightarrow \;\; P' \overset{s?\langle V\rangle}{\longmapsto} P'}$$

Note the distinction between standard and refined transitions: $\xrightarrow{s?\langle V\rangle}$ vs. $\overset{s?\langle V\rangle}{\longmapsto}$. Our refined rule for (higher-order) input admits only names, trigger values, and characteristic values. Using this rule, we define an alternative, refined LTS on typed processes: we use it to define characteristic bisimulation ($\approx^{\mathsf{C}}$, Def. 14), in which the demanding clause ($\star$) is replaced with a more tractable output clause based on characteristic trigger processes (cf. (2)).

Characteristic Triggers. Following the same reasoning as (1), we can use an alternative trigger process, called ***characteristic trigger process*** with type U to replace clause ($\star$):

$$\boxed{t \Leftarrow V : U \stackrel{\mathsf{def}}{=} t?(x).(\nu\, s)(\llbracket ?(U); \mathtt{end} \rrbracket^s \mid \overline{s}!\langle V\rangle.\mathbf{0})} \qquad (2)$$

This is justified because in (1) $T_V \overset{t?\langle \llparenthesis ?(U);\mathtt{end} \rrparenthesis_{\mathsf{c}}\rangle}{\longmapsto} \approx (\nu\, s)(\llparenthesis ?(U); \mathtt{end} \rrparenthesis^s \mid \overline{s}!\langle V\rangle.\mathbf{0})$. Thus, unlike process (1), the characteristic trigger process in (2) does not involve a higher-order communication on t. In contrast to previous approaches [15, 6] our characteristic trigger processes do *not* use recursion or replication. This is key to preserve linearity of session endpoints.

It is also noteworthy that $\mathsf{HO}\pi$ lacks name matching, which is usually crucial to prove completeness of bisimilarity—see, e.g., [6]. Instead of matching, we use types: a process trigger embeds a name into a characteristic process so to observe its session behaviour.

Outline. Next we present the session calculus $\mathsf{HO}\pi$. § 3 gives the session type system for $\mathsf{HO}\pi$ and states type soundness. § 4 develops *characteristic* bisimilarity and states our main result: characteristic and context bisimilarities coincide for well-typed $\mathsf{HO}\pi$ processes (Thm. 16). § 5 concludes with related works.

2 A Higher-Order Session π-Calculus

We introduce the *Higher-Order Session π-Calculus* ($\mathsf{HO}\pi$). $\mathsf{HO}\pi$ includes both name- and abstraction-passing, shared and session communication, as well as recursion; it is essentially the language proposed in [12] (where tractable bisimilarities are not addressed).

$$
\begin{array}{llllll}
u, w & ::= & n \mid x, y, z & n \; ::= \; a, b \mid s, \overline{s} & V, W \; ::= \; u \mid \lambda x.\, P \\
P, Q & ::= & u!\langle V \rangle.P \mid u?(x).P \mid u \triangleleft l.P \mid u \triangleright \{l_i : P_i\}_{i \in I} \\
& \mid & X \mid \mu X.P \mid V\, W \mid P \mid Q \mid (\nu\, n)P \mid \mathbf{0}
\end{array}
$$

$$
P \mid \mathbf{0} \equiv P \quad P_1 \mid P_2 \equiv P_2 \mid P_1 \quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \quad \mu X.P \equiv P\{^{\mu X.P}/X\}
$$

$$
(\nu\, n)\mathbf{0} \equiv \mathbf{0} \qquad P \mid (\nu\, n)Q \equiv (\nu\, n)(P \mid Q)\ (n \notin \mathtt{fn}(P)) \qquad P \equiv Q \text{ if } P \equiv_\alpha Q
$$

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

Figure 1 HOπ: Syntax and Operational Semantics (Structural Congruence and Reduction).

Syntax. The syntax of HOπ is given in Fig. 1 (upper part). We use $a, b, c, \ldots$ (resp. $s, \overline{s}, \ldots$) to range over shared (resp. session) names. We use $m, n, t, \ldots$ 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 *endpoints* while shared names represent non-deterministic points. 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 π-calculus constructs for sending/receiving values V. Process $u!\langle V \rangle.P$ denotes the output of 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 $\mu X.P$ binds the recursive variable X in process P. Process $V\, W$ is the application which substitutes values W on the abstraction V. Typing ensures that V is not a name. Processes $u \triangleright \{l_i : P_i\}_{i \in I}$ and $u \triangleleft l.P$ define labelled choice: given a finite index set I, process $u \triangleright \{l_i : P_i\}_{i \in I}$ offers a choice among processes with pairwise distinct labels; process $u \triangleleft l.P$ selects label l on name u and then behaves as P. Constructs for inaction $\mathbf{0}$, parallel composition $P_1 \mid P_2$, and name restriction $(\nu\, n)P$ are standard. Session name restriction $(\nu\, s)P$ binds endpoints s and $\overline{s}$ in P. We use $\mathtt{fv}(P)$ and $\mathtt{fn}(P)$ to denote sets of free variables and names; we assume V in $u!\langle V \rangle.P$ does not include free recursive variables. If $\mathtt{fv}(P) = \emptyset$, we call P *closed*.

Semantics. Fig. 1 (lower part) defines the operational semantics of HOπ, given as a reduction relation that relies on a *structural congruence* $\equiv$. We assume the expected extension of $\equiv$ to values V. Reduction is denoted $\longrightarrow$; some intuitions on the rules in Fig. 1 follow. Rule [App] is a value application; rule [Pass] defines a shared interaction at n (with $\overline{n} = n$) or a session interaction; rule [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. We write $\longrightarrow^*$ for a multi-step reduction.

▶ **Example 1** (Hotel Booking Scenario)**.** To illustrate HOπ and its expressive power, we consider a usecase scenario that adapts the example given by Mostrous and Yoshida [12, 13]. The scenario involves a Client process that wants to book a hotel room. Client narrows the choice down to two hotels, and requires a quote from the two in order to decide. The round-trip time (RTT) required for taking quotes from the two hotels is not optimal, so the client sends mobile processes to both hotels to automatically negotiate and book a room.

We now present two HOπ implementations of this scenario. For convenience, we write

$\texttt{if}\ e\ \texttt{then}\ (P_1\ ;\ P_2)$ to denote a conditional process that executes P_1 or P_2 depending on boolean expression e (encodable using labelled choice). The *first implementation* is as follows:

$$P_{xy} \stackrel{\text{def}}{=} x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}).y!\langle \mathsf{quote}\rangle.y \triangleright \left\{ \begin{array}{l} \mathsf{accept} : x \triangleleft \mathsf{accept}.x!\langle \mathsf{credit}\rangle.\mathbf{0}, \\ \mathsf{reject} : x \triangleleft \mathsf{reject}.\mathbf{0} \end{array} \right\}$$

$$\begin{array}{rcl} \mathsf{Client}_1 & \stackrel{\text{def}}{=} & (\nu\, h_1, h_2)(s_1!\langle \lambda x.\, P_{xy}\{h_1/y\}\rangle.s_2!\langle \lambda x.\, P_{xy}\{h_2/y\}\rangle.\mathbf{0} \mid \\ & & \overline{h_1}?(x).\overline{h_2}?(y).\texttt{if}\ x \leq y\ \texttt{then}\ (\overline{h_1} \triangleleft \mathsf{accept}.\overline{h_2} \triangleleft \mathsf{reject}.\mathbf{0}\ ;\ \overline{h_1} \triangleleft \mathsf{reject}.\overline{h_2} \triangleleft \mathsf{accept}.\mathbf{0})) \end{array}$$

Process Client_1 sends two abstractions with body P_{xy}, one to each hotel, using sessions s_1 and s_2. That is, P_{xy} is the mobile code: while name x is meant to be instantiated by the hotel as the negotiating endpoint, name y is used to interact with Client_1. Intuitively, process P_{xy} (i) sends the room requirements to the hotel; (ii) receives a quote from the hotel; (iii) sends the quote to Client_1; (iv) expects a choice from Client_1 whether to accept or reject the offer; (v) if the choice is accept then it informs the hotel and performs the booking; otherwise, if the choice is reject then it informs the hotel and ends the session. Client_1 instantiates two copies of P_{xy} as abstractions on session x. It uses two fresh endpoints h_1, h_2 to substitute channel y in P_{xy}. This enables communication with the mobile code(s). In fact, Client_1 uses the dual endpoints $\overline{h_1}$ and $\overline{h_2}$ to receive the negotiation result from the two remote instances of P and then inform the two processes for the final booking decision.

Notice that the above implementation does not affect the time needed for the whole protocol to execute, since the two remote processes are used to send/receive data to Client_1.

We present now a *second implementation* in which the two mobile processes are meant to interact with each other (rather than with the client) to reach to an agreement:

$$\begin{array}{rcl} R_x & \stackrel{\text{def}}{=} & \texttt{if}\ \ \mathsf{quote}_1 \leq \mathsf{quote}_2\ \texttt{then}\ (x \triangleleft \mathsf{accept}.x!\langle \mathsf{credit}\rangle.\mathbf{0}\ ;\ x \triangleleft \mathsf{reject}.\mathbf{0}) \\ Q_1 & \stackrel{\text{def}}{=} & x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}_1).y!\langle \mathsf{quote}_1\rangle.y?(\mathsf{quote}_2).R_x \\ Q_2 & \stackrel{\text{def}}{=} & x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}_1).y?(\mathsf{quote}_2).y!\langle \mathsf{quote}_1\rangle.R_x \\ \mathsf{Client}_2 & \stackrel{\text{def}}{=} & (\nu\, h)(s_1!\langle \lambda x.\, Q_1\{h/y\}\rangle.s_2!\langle \lambda x.\, Q_2\{\overline{h}/y\}\rangle.\mathbf{0}) \end{array}$$

Processes Q_1 and Q_2 negotiate a quote from the hotel in the same fashion as process P_{xy} in Client_1. The key difference with respect to P_{xy} is that y is used for interaction between process Q_1 and Q_2. Both processes send their quotes to each other and then internally follow the same logic to reach to a decision. Process Client_2 then uses sessions s_1 and s_2 to send the two instances of Q_1 and Q_2 to the two hotels, using them as abstractions on name x. It further substitutes the two endpoints of a fresh channel h to channels y respectively, in order for the two instances to communicate with each other.

The differences between Client_1 and Client_2 can be seen in the sequence diagrams of Fig. 2. We will assign session types to these client processes in Example 4. Later on, we will show that they are behaviourally equivalent using characteristic bisimilarity; see Prop. 4.3.

3 Types and Typing

We define a session typing system for $\mathsf{HO}\pi$ and state its main properties. Our system distills the key features of [12, 13]. We give selected definitions; see [1] for a full description.

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

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

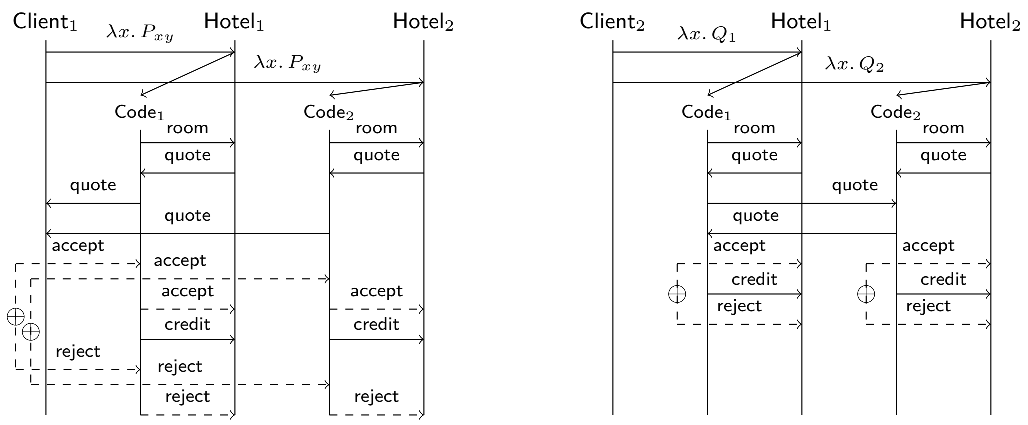

Figure 2 Sequence diagrams for Client$_1$ and Client$_2$ as in Example 1.

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 $U{\rightarrow}\diamond$ and $U{\multimap}\diamond$ denote *shared* and *linear* higher-order types, respectively. As for session types, the *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* $\mu\mathsf{t}.S$ is guarded, i.e., $\mu\mathsf{t}.\mathsf{t}$ is not allowed. Type `end` is the termination type.

Following [2], we write S_1 dual S_2 if S_1 is the *dual* of S_2. Intuitively, duality converts ! into ? and $\oplus$ into & (and viceversa).

Typing Environments and Judgements. Typing *environments* are defined below:

$$\Gamma \;::=\; \emptyset \;\mid\; \Gamma\cdot x : U{\rightarrow}\diamond \;\mid\; \Gamma\cdot u:\langle S\rangle \;\mid\; \Gamma\cdot u:\langle L\rangle \;\mid\; \Gamma\cdot X:\Delta$$
$$\Lambda \;::=\; \emptyset \;\mid\; \Lambda\cdot x : U{\multimap}\diamond \qquad \Delta \;::=\; \emptyset \;\mid\; \Delta\cdot u : S$$

Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. Λ maps variables to linear higher-order types, and Δ maps session names to session types. Both Λ and Δ are only subject to exchange. The domains of Γ, Λ and Δ are assumed pairwise distinct. $\Delta_1\cdot\Delta_2$ is the disjoint union of Δ_1 and Δ_2. We define *typing judgements* for values and processes:

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

The first judgement says that under environments $\Gamma;\Lambda;\Delta$ value V has type U; the second judgement says that under environments $\Gamma;\Lambda;\Delta$ process P has the process type $\diamond$. The type soundness result for HOπ (Thm. 3) relies on two auxiliary notions on session environments:

▶ **Definition 2** (Session Environments: Balanced/Reduction)**.** Let Δ be a session environment.

- 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 the reduction 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 rely on a typing system that is similar to the one developed in [12, 13]. We state the type soundness result for HOπ processes; see [1] for details of the associated proofs.

▶ **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.*

▶ **Example 4** (Hotel Booking Revisited). Assume $S = !\langle \mathsf{quote} \rangle; \&\{\mathsf{accept} : \mathsf{end}, \mathsf{reject} : \mathsf{end}\}$ and $U = !\langle \mathsf{room} \rangle; ?(\mathsf{quote}); \oplus\{\mathsf{accept} : !\langle \mathsf{credit} \rangle; \mathsf{end}, \mathsf{reject} : \mathsf{end}\}$. We give types to the client processes of Ex. 1:

$$\begin{array}{rcl} \emptyset; \emptyset; y : S & \vdash & \lambda x. P_{xy} \triangleright U \multimap \diamond \\ \emptyset; \emptyset; s_1 : !\langle U \multimap \diamond \rangle; \mathsf{end} \cdot s_2 : !\langle U \multimap \diamond \rangle; \mathsf{end} & \vdash & \mathsf{Client}_1 \triangleright \diamond \\ \emptyset; \emptyset; y : !\langle \mathsf{quote} \rangle; ?(\mathsf{quote}); \mathsf{end} & \vdash & \lambda x. Q_i \triangleright U \multimap \diamond \quad (i = 1, 2) \\ \emptyset; \emptyset; s_1 : !\langle U \multimap \diamond \rangle; \mathsf{end} \cdot s_2 : !\langle U \multimap \diamond \rangle; \mathsf{end} & \vdash & \mathsf{Client}_2 \triangleright \diamond \end{array}$$

4 Characteristic Session Bisimulation

We develop a theory for observational equivalence over session typed $\mathsf{HO}\pi$ processes that follows the principles laid in our previous works [9, 8]. We introduce *characteristic bisimulation* (Def. 14) and prove that it coincides with reduction-closed, barbed congruence (Thm. 16).

We begin by defining an (early) labelled transition system (LTS) on untyped processes (§ 4.1). Then, using the *environmental* transition semantics (§ 4.2), we define a typed LTS to formalise how a typed process interacts with a typed observer.

4.1 Labelled Transition System for Processes

Interaction is defined on action labels ℓ:

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

Label τ defines internal actions. Action $(\nu\, \tilde{m})n!\langle V \rangle$ denotes the sending of value V over channel n with a possible empty set of restricted names $\tilde{m}$ (we may write $n!\langle V \rangle$ when $\tilde{m}$ is empty). Dually, the action for value reception is $n?\langle V \rangle$. Actions for select and branch on a label l 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 $\ell \neq \tau$, we write $\mathtt{subj}(\ell)$ to denote the *subject* of ℓ.

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) $(\nu\, \tilde{m})n!\langle V \rangle \asymp \overline{n}?\langle V \rangle$.

The 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 [9, 8]. A process with an output prefix can interact with the environment with an output action that carries a value V (rule $\langle \textsc{Snd} \rangle$). Dually, in rule $\langle \textsc{Rv} \rangle$ a receiver process can observe an input of an arbitrary value V. Select and branch processes observe the select and branch actions in rules $\langle \textsc{Sel} \rangle$ and $\langle \textsc{Bra} \rangle$, resp. Rule $\langle \textsc{Res} \rangle$ closes the LTS under restriction 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 $\langle \textsc{New} \rangle$). Rule $\langle \textsc{Rec} \rangle$ handles recursion unfolding. Rule $\langle \textsc{Tau} \rangle$ states that two parallel processes which perform dual actions can synchronise by an internal transition. Rules $\langle \textsc{Par}_L \rangle / \langle \textsc{Par}_R \rangle$ and $\langle \textsc{Alpha} \rangle$ close the LTS under parallel composition and α-renaming.

4.2 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 LTSs in [9, 8] to higher-order sessions. Notice that due to weakening we have $(\Gamma', \Lambda_1, \Delta_1) \stackrel{\ell}{\longmapsto} (\Gamma', \Lambda_2, \Delta_2)$ if $(\Gamma, \Lambda_1, \Delta_1) \stackrel{\ell}{\longmapsto} (\Gamma', \Lambda_2, \Delta_2)$.

$$\langle\textsc{App}\rangle\ \frac{}{(\lambda x.\,P)\,V \xrightarrow{\tau} P\{^{V}\!/_{x}\}} \qquad \langle\textsc{Snd}\rangle\ \frac{}{n!\langle V\rangle.P \xrightarrow{n!\langle V\rangle} P} \qquad \langle\textsc{Rv}\rangle\ \frac{}{n?(x).P \xrightarrow{n?\langle V\rangle} P\{^{V}\!/_{x}\}}$$

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

$$\langle\textsc{Alpha}\rangle\ \frac{P \equiv_\alpha Q \quad Q \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P'} \qquad \langle\textsc{Res}\rangle\ \frac{P \xrightarrow{\ell} P' \quad n \notin \mathtt{fn}(\ell)}{(\nu\, n)P \xrightarrow{\ell} (\nu\, n)P'} \qquad \langle\textsc{New}\rangle\ \frac{P \xrightarrow{(\nu\,\widetilde{m})n!\langle V\rangle} P' \quad m \in \mathtt{fn}(V)}{(\nu\, m)P \xrightarrow{(\nu\, m\cdot\widetilde{m'})n!\langle V\rangle} P'}$$

$$\langle\textsc{Par}_L\rangle\ \frac{P \xrightarrow{\ell} P' \quad \mathtt{bn}(\ell)\cap\mathtt{fn}(Q)=\emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \langle\textsc{Tau}\rangle\ \frac{P \xrightarrow{\ell_1} P' \qquad Q \xrightarrow{\ell_2} Q' \qquad \ell_1 \asymp \ell_2}{P \mid Q \xrightarrow{\tau} (\nu\, \mathtt{bn}(\ell_1)\cup\mathtt{bn}(\ell_2))(P' \mid Q')} \qquad \langle\textsc{Rec}\rangle\ \frac{P\{^{\mu X.P}\!/_{X}\} \xrightarrow{\ell} P'}{\mu X.P \xrightarrow{\ell} P'}$$

Figure 3 The Untyped LTS for HOπ processes. We omit rule $\langle\textsc{Par}_R\rangle$.

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. The resulting type tuple must contain the environments associated to V. The dual endpoint $\overline{s}$ cannot be present in the session environment: if it were present the only possible communication would be the interaction between the two endpoints (cf. rule [Tau]). 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 $(\nu\,\widetilde{m})s!\langle V\rangle$ 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 $\widetilde{m}$ which restrict the scope of value V. Analogously, the linear variable environment Λ' of V should be included in Λ. Scope extrusion of session names in $\widetilde{m}$ requires that the dual endpoints of $\widetilde{m}$ should appear in the resulting session environment. Similarly for shared names in $\widetilde{m}$ 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. Rule [Tau] defines internal transitions: it keeps the session environment unchanged or reduces it (Def. 2).

▶ **Example 5.** Consider environment $(\Gamma; \emptyset; s :!\langle !\langle S\rangle;\mathtt{end}\multimap\diamond\rangle;\mathtt{end}\cdot s' : S)$ and typed value

$$\Gamma; \emptyset; s' : S\cdot m :?(\mathtt{end});\mathtt{end} \vdash V \triangleright !\langle S\rangle;\mathtt{end}\multimap\diamond \quad \text{with} \quad V = \lambda x.\, x!\langle s'\rangle.m?(z).\mathbf{0}$$

We illustrate rule [SSnd] in Fig. 4. Let $\Delta_1' = \{\overline{m} :!\langle\mathtt{end}\rangle;\mathtt{end}\}$ and $U =!\langle S\rangle;\mathtt{end}\multimap\diamond$. Then we can derive:

$$(\Gamma; \emptyset; s :!\langle !\langle S\rangle;\mathtt{end}\multimap\diamond\rangle;\mathtt{end}\cdot s' : S) \xrightarrow{(\nu\, m)s!\langle V\rangle} (\Gamma; \emptyset; s : \mathtt{end})$$

Our typed LTS combines the LTSs in Fig. 3 and Fig. 4.

▶ **Definition 6** (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) $P_1 \xrightarrow{\ell} P_2$; (2) $(\Gamma, \emptyset, \Delta_1) \xrightarrow{\ell} (\Gamma, \emptyset, \Delta_2)$ with $\Gamma; \emptyset; \Delta_i \vdash P_i \triangleright \diamond$

$$[\textsc{SRv}]\ \dfrac{\overline{s} \notin \mathtt{dom}(\Delta) \quad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta \cdot s :?(U); S) \xrightarrow{s?\langle V \rangle} (\Gamma; \Lambda \cdot \Lambda'; \Delta \cdot \Delta' \cdot s : S)} \qquad [\textsc{ShRv}]\ \dfrac{\Gamma; \emptyset; \emptyset \vdash a \triangleright \langle U \rangle \quad \Gamma; \Lambda'; \Delta' \vdash V \triangleright U}{(\Gamma; \Lambda; \Delta) \xrightarrow{a?\langle V \rangle} (\Gamma; \Lambda \cdot \Lambda'; \Delta \cdot \Delta')}$$

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

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

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

$$[\textsc{Bra}]\ \dfrac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma; \Lambda; \Delta \cdot s : \&\{l_i : T_i\}_{i \in I}) \xrightarrow{s \& l_j} (\Gamma; \Lambda; \Delta \cdot s : S_j)} \qquad [\textsc{Tau}]\ \dfrac{\Delta_1 \longrightarrow \Delta_2 \vee \Delta_1 = \Delta_2}{(\Gamma; \Lambda; \Delta_1) \xrightarrow{\tau} (\Gamma; \Lambda; \Delta_2)}$$

Figure 4 Labelled Transition System for Typed Environments.

$(i = 1, 2)$. We extend to $\Longrightarrow$ and $\overset{\hat{\ell}}{\Longrightarrow}$ where we write $\Longrightarrow$ for the reflexive and transitive closure of $\rightarrow$, $\overset{\ell}{\Longrightarrow}$ for the transitions $\Longrightarrow\overset{\ell}{\rightarrow}\Longrightarrow$, and $\overset{\hat{\ell}}{\Longrightarrow}$ for $\overset{\ell}{\Longrightarrow}$ if $\ell \neq \tau$ otherwise $\Longrightarrow$.

4.3 Reduction-Closed, Barbed Congruence ($\cong$)

We now define *typed relations* and *contextual equivalence* (i.e., barbed congruence). 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 7.** 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* [11] with respect to types.

▶ **Definition 8** (Barbs). Let P be a closed process. We write $P \downarrow_n$ if $P \equiv (\nu\, \tilde{m})(n!\langle V \rangle.P_2 \mid P_3)$, with $n \notin \tilde{m}$. Also: $P \Downarrow_n$ if $P \longrightarrow^* \downarrow_n$. Similarly, we write $\Gamma; \emptyset; \Delta \vdash P \downarrow_n$ if $\Gamma; \emptyset; \Delta \vdash P \triangleright \diamond$ with $P \downarrow_n$ and $\overline{n} \notin \Delta$. Also: $\Gamma; \emptyset; \Delta \vdash P \Downarrow_n$ if $P \longrightarrow^* P'$ and $\Gamma; \emptyset; \Delta' \vdash P' \downarrow_n$.

A barb $\downarrow_n$ is an observable on an output prefix with subject n; 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 in Δ.

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

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

Notation $\mathbb{C}[P]$ denotes the result of substituting the hole $-$ in $\mathbb{C}$ with process P. The first behavioural relation we define is reduction-closed, barbed congruence [5].

▶ **Definition 9** (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 $\mathbb{C}$, Δ_1'', Δ_2'' we have: $\Gamma; \Delta_1'' \vdash \mathbb{C}[P_1] \,\Re\, \Delta_2'' \vdash \mathbb{C}[P_2]$;
4) The symmetric cases of 1 and 2.

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

4.4 Context Bisimilarity ($\approx$)

Following Sangiorgi [16], we now define the standard (weak) context bisimilarity.

▶ **Definition 10** (Context Bisimilarity). 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 $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{(\nu\, \widetilde{m_1})n!\langle V_1\rangle} \Delta_1' \vdash P_2$, there exist Q_2, V_2, Δ_2' such that $\Gamma; \Delta_2 \vdash Q_1 \overset{(\nu\, \widetilde{m_2})n!\langle V_2\rangle}{\Longrightarrow} \Delta_2' \vdash Q_2$ and for all R with $\mathtt{fv}(R) = x$:
$$\Gamma; \Delta_1'' \vdash (\nu\, \widetilde{m_1})(P_2 \mid R\{V_1/x\}) \,\Re\, \Delta_2'' \vdash (\nu\, \widetilde{m_2})(Q_2 \mid R\{V_2/x\});$$
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 hinted at in the Introduction, in the general case, context bisimilarity is hard to compute. Below we introduce *characteristic bisimulations*, which are meant to be a *tractable* proof technique over session typed processes with higher-order communication.

4.5 Characteristic Bisimilarity ($\approx^{\mathsf{C}}$)

We formalise the ideas given in the introduction. We define characteristic processes/values:

▶ **Definition 11** (Characteristic Process and Values). Let u and U be a name and a type, respectively. Fig. 5 defines the *characteristic process* $[\![U]\!]^u$ and the *characteristic value* $[\![U]\!]_{\mathsf{c}}$.

▶ **Proposition 4.1.** Let S be a session type. Then $\Gamma; \emptyset; \Delta \cdot s : S \vdash [\![S]\!]^s \triangleright \diamond$. Also, let $\langle U\rangle$ be a first-order (channel) type. Then $\Gamma \cdot a : \langle U\rangle; \emptyset; \Delta \vdash [\![\langle U\rangle]\!]^a \triangleright \diamond$.

The following example motivates the refined LTS explained in the introduction.

▶ **Example 12** (The Need for Refined Typed LTS). We show that observing a characteristic value input alone is not enough to define a sound bisimulation closure. Consider processes

$$P_1 = s?(x).(x\, s_1 \mid x\, s_2) \qquad\qquad P_2 = s?(x).(x\, s_1 \mid s_2?(y).\mathbf{0}) \tag{3}$$

where $\Gamma; \emptyset; \Delta \cdot s : ?((?(C); \mathtt{end}) \multimap \diamond); \mathtt{end} \vdash P_i \triangleright \diamond$ $(i \in \{1, 2\})$. If P_1 and P_2 input and substitute over x the characteristic value $[\![(?(C); \mathtt{end}) \multimap \diamond]\!]_{\mathsf{c}} = \lambda x.\, x?(y).\mathbf{0}$, then they evolve into:

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

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

$$\llparenthesis S \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} s \ \ (s \text{ fresh}) \quad \llparenthesis \langle S\rangle \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} \llparenthesis \langle L\rangle \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} a \ \ (a \text{ fresh}) \quad \llparenthesis U \rightarrow \diamond \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} \llparenthesis U \multimap \diamond \rrparenthesis_{\mathsf{c}} \stackrel{\mathsf{def}}{=} \lambda x.\, \llparenthesis U \rrparenthesis^x$$

Figure 5 Characteristic Processes (top) and Values (bottom) as in Def. 11. For $\llparenthesis S \rrparenthesis_{\mathsf{c}}$, $\llparenthesis \langle S\rangle \rrparenthesis_{\mathsf{c}}$, and $\llparenthesis \langle L\rangle \rrparenthesis_{\mathsf{c}}$ freshness is assumed with respect to any names in their contexts.

therefore becoming context bisimilar. However, the processes in (3) are clearly *not* context bisimilar: many input actions 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 $\Gamma;\emptyset;\Delta \vdash s \triangleright \mathtt{end}$, 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, we can distinguish P_1, P_2 in (3):

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

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

$$(\nu\, s)(n?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, R_1\rangle.\mathbf{0}) \text{ and } (\nu\, s)(n?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, R_2\rangle.\mathbf{0}) \tag{4}$$

If these processes input the trigger value, we obtain:

$$(\nu\, s)(t?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, R_1\rangle.\mathbf{0}) \text{ and } (\nu\, s)(t?(x).(x\, s) \mid \overline{s}!\langle \lambda x.\, R_2\rangle.\mathbf{0})$$

thus we can easily derive a bisimulation closure if we assume a bisimulation definition that allows only trigger value input. But if processes in (4) input the characteristic value $\lambda z.\, z?(x).(x\, m)$, then they would become, under appropriate Γ and Δ:

$$\Gamma;\emptyset;\Delta \vdash (\nu\, s)(s?(x).(x\, m) \mid \overline{s}!\langle \lambda x.\, R_i\rangle.\mathbf{0}) \approx \Delta \vdash R_i\{m/x\} \quad (i = 1,2)$$

which are not bisimilar if $R_1\{m/x\} \not\approx R_2\{m/x\}$.

As explained in the introduction, 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 13** (Refined Typed Labelled Transition Relation)**.** We define the environment transition rule for input actions using the input rules in Fig. 4:

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

Rule [RRcv] is defined on top of rules [SRv] and [ShRv] in Fig. 4. We use the non-receiving rules in Fig. 4 together with rule [RRcv] to define $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longmapsto} \Delta_2 \vdash P_2$ as in Def. 6.

Notice that $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longmapsto} \Delta_2 \vdash P_2$ (refined transition) implies $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longrightarrow} \Delta_2 \vdash P_2$ (ordinary transition). Below we sometimes write $\stackrel{(\nu\, \widetilde{m})n!\langle V:U\rangle}{\longmapsto}$ when the type of V is U.

Characteristic Bisimulations. We define *characteristic bisimulations*, a tractable bisimulation for $\mathsf{HO}\pi$. As hinted at above, their definition uses trigger processes (cf. (2)):

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

▶ **Definition 14** (Characteristic Bisimilarity)**.** A typed relation $\Re$ is a *characteristic bisimulation* if for all $\Gamma;\Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash Q_1$,

1) Whenever $\Gamma;\Delta_1 \vdash P_1 \overset{(\nu\,\widetilde{m_1})n!\langle V_1:U\rangle}{\longmapsto} \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2' such that $\Gamma;\Delta_2 \vdash Q_1 \overset{(\nu\,\widetilde{m_2})n!\langle V_2:U\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for fresh t,
$\Gamma;\Delta_1'' \vdash (\nu\,\widetilde{m_1})(P_2 \mid t \Leftarrow V_1 : U_1) \,\Re\, \Delta_2'' \vdash (\nu\,\widetilde{m_2})(Q_2 \mid t \Leftarrow V_2 : U_2)$
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 *characteristic bisimilarity* and denoted by $\approx^{\mathsf{C}}$.

Internal transitions associated to session interactions or β-reductions are deterministic.

▶ **Definition 15** (Deterministic Transition)**.** Let $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ be a balanced $\mathsf{HO}\pi$ process. Transition $\Gamma;\Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ is called *session transition* whenever the transition $P \overset{\tau}{\longrightarrow} P'$ is derived using rule $\langle\textsc{Tau}\rangle$ (where $\mathtt{subj}(\ell_1)$ and $\mathtt{subj}(\ell_2)$ in the premise are dual endpoints), possibly followed by uses of $\langle\textsc{Alpha}\rangle$, $\langle\textsc{Res}\rangle$, $\langle\textsc{Rec}\rangle$, or $\langle\textsc{Par}_L\rangle/\langle\textsc{Par}_R\rangle$.

Transition $\Gamma;\Delta \vdash P \overset{\tau}{\longmapsto} \Delta' \vdash P'$ is called *β-transition* whenever the transition $P \overset{\tau}{\longrightarrow} P'$ is derived using rule $\langle\textsc{App}\rangle$, possibly followed by uses of $\langle\textsc{Alpha}\rangle$, $\langle\textsc{Res}\rangle$, $\langle\textsc{Rec}\rangle$, or $\langle\textsc{Par}_L\rangle/\langle\textsc{Par}_R\rangle$. $\Gamma;\Delta \vdash P \overset{\tau_{\mathsf{d}}}{\longmapsto} \Delta' \vdash P'$ denotes either a session transition or a β-transition.

▶ **Proposition 4.2** (τ-inertness)**.** Let $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ be a balanced $\mathsf{HO}\pi$ process. Then $\Gamma;\Delta \vdash P \overset{\tau_{\mathsf{d}}}{\longmapsto} \Delta' \vdash P'$ implies $\Gamma;\Delta \vdash P \approx^{\mathsf{C}} \Delta' \vdash P'$.

See [1] for associated proofs. Our main theorem follows: it allows us to use $\approx^{\mathsf{C}}$ as a tractable reasoning technique for higher-order processes with sessions.

▶ **Theorem 16** (Coincidence)**.** $\cong$, $\approx$, and $\approx^{\mathsf{C}}$ coincide in $\mathsf{HO}\pi$.

Proof (Sketch). We use *higher-order bisimilarity* ($\approx^{\mathtt{H}}$), an auxiliary equivalence that is defined as $\approx^{\mathsf{C}}$ but by using trigger processes with higher-order communication (cf. (1)). We first show that $\approx^{\mathsf{C}}$ and $\approx^{\mathtt{H}}$ coincide by using Prop. 4.2; then, we show that $\approx^{\mathtt{H}}$ coincides with $\approx$ and $\cong$. A key result is a substitution lemma which simplifies reasoning for $\approx^{\mathtt{H}}$ by exploiting characteristic processes/values. See [1] for full details. ◀

Now we prove that processes Client_1 and Client_2 in Example 1 are behaviourally equivalent.

▶ **Proposition 4.3.** Let $S = !\langle\mathsf{room}\rangle; ?(\mathsf{quote}); \oplus\{\mathsf{accept} : !\langle\mathsf{credit}\rangle; \mathsf{end}, \mathsf{reject} : \mathsf{end}\}$ and $\Delta = s_1 : !\langle S{\multimap}\diamond\rangle; \mathsf{end} \cdot s_2 : !\langle S{\multimap}\diamond\rangle; \mathsf{end}$. Then $\emptyset;\Delta \vdash \mathsf{Client}_1 \approx^{\mathsf{C}} \Delta \vdash \mathsf{Client}_2$.

Proof (Sketch). We show a bisimulation closure by following transitions on each Client. See [1] for details. First, the characteristic process is given as: $[\![?(S{\multimap}\diamond);\mathsf{end}]\!]^s = s?(x).(x\, k)$. We show that the clients can simulate each other on the first two output transitions, that also generate the trigger processes:

$$\begin{array}{ll}
\emptyset; \emptyset; \Delta \vdash \mathsf{Client}_1 & \xrightarrow{s_1!\langle \lambda x.\, P_{xy}\{^{h_1}\!/y\}\rangle} \xrightarrow{s_2!\langle \lambda x.\, P_{xy}\{^{h_2}\!/y\}\rangle} \\
\emptyset; \emptyset; k_1 : S \cdot k_2 : S \vdash & (\nu\, h_1, h_2)(\overline{h_1}?(x).\overline{h_2}?(y). \\
& \mathtt{if}\ \ x \leq y\ \mathtt{then}\ (\overline{h_1} \triangleleft \mathsf{accept}.\overline{h_2} \triangleleft \mathsf{reject}.\mathbf{0}\mathbf{;}\overline{h_1} \triangleleft \mathsf{reject}.\overline{h_2} \triangleleft \mathsf{accept}.\mathbf{0}) \\
& \mid t_1 \Leftarrow \lambda x.\, P_{xy}\{^{h_1}\!/y\} : S\multimapdotboth \mid t_2 \Leftarrow \lambda x.\, P_{xy}\{^{h_2}\!/y\} : S\multimapdotboth)
\end{array}$$

$$\begin{array}{ll}
\emptyset; \emptyset; \Delta \vdash \mathsf{Client}_2 & \xrightarrow{s_1!\langle \lambda x.\, Q_1\{^{h}\!/y\}\rangle} \xrightarrow{s_2!\langle \lambda x.\, Q_2\{^{\overline{h}}\!/y\}\rangle} \\
\emptyset; \emptyset; k_1 : S \cdot k_2 : S \vdash & (\nu\, h)(t_1 \Leftarrow \lambda x.\, Q_1\{^{h}\!/y\} : S\multimapdotboth \mid t_2 \Leftarrow \lambda x.\, Q_2\{^{\overline{h}}\!/y\} : S\multimapdotboth)
\end{array}$$

After these transitions, we can analyse that the resulting processes are behaviourally equivalent since they have the same visible transitions; the rest is internal deterministic transitions. ◀

5 Related Work

As in this work, the bisimulations in [9, 8] (binary and multiparty sessions, respectively) are defined and characterised on an LTS which combines an untyped LTS for processes and an LTS on session type environments. The work [14] studies typed equivalences for a theory of binary sessions based on linear logic, without shared names. None of [9, 8, 14] consider session processes with higher-order communication, as we do here. Our results have important consequences in the relative expressivity of higher-order sessions; see [1] for details.

Our approach to typed equivalences builds upon techniques developed by Sangiorgi [15, 16] and Jeffrey and Rathke [6]. As we have discussed, although contextual bisimilarity has a satisfactory discriminative power, its use is hindered by the universal quantification on output. To deal with this, Sangiorgi proposes *normal bisimilarity*, a tractable equivalence without universal quantification. To prove that context and normal bisimilarities coincide, [15] 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 [6] for calculi with recursive types, not addressed in [16, 15] and quite relevant in session-based concurrency. The bisimulation in [6] is based on an LTS extended with trigger meta-notation. As in [16, 15], the LTS in [6] 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 [6] and ours based on characteristic bisimilarity ($\approx^{\mathsf{C}}$):

- The LTS in [6] 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, $\approx^{\mathsf{C}}$ introduces a parallel *non-replicated* trigger.
- Higher-order input in [6] involves the input of a trigger which reduces after substitution. Rather than a trigger name, $\approx^{\mathsf{C}}$ decrees the input of a trigger value $\lambda z.\, t?(x).(x\, z)$.
- Unlike [6], $\approx^{\mathsf{C}}$ 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 [6] name matching is crucial to prove completeness of bisimilarity. In our case, $\mathsf{HO}\pi$ lacks name matching and we use session types: a characteristic value inhabiting a type enables the simplest form of interactions with the environment.

We have compared our approach to that in [6] using a representative example. We considered the transitions and resulting processes involved in checking bisimilarity of process $n!\langle \lambda x.\, x\, (\lambda y.\, y!\langle m\rangle.\mathbf{0})\rangle.\mathbf{0}$ with itself. This comparison, detailed in [1], reveals that our approach requires less visible transitions and replicated processes. Therefore, linearity information does simplify analyses, as it enables simpler witnesses in coinductive proofs.

Environmental bisimulations [17] 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 [7] with a mapping from constants to higher-order values. This allows to observe first-order values instead of higher-order values. It differs from [16, 6] in that the mapping between higher- and first-order values is no longer implicit.

Acknowledgments This work has been partially sponsored by the The Doctoral Prize Fellowship, EPSRC EP/K011715/1, EPSRC EP/K034413/1, and EPSRC EP/L00058X/1, EU project FP7-612985 UpScale, and EU COST Action IC1201 BETTY. Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Universidade Nova de Lisboa, Portugal.

References

1 Full version of this paper. Technical report, 2015. `http://arxiv.org/abs/1502.02585`.

2 Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *TGC 2014*, volume 8902 of *LNCS*, pages 51–66. Springer, 2014.

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

4 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, 1998.

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

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

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

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

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

10 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.

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

12 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.

13 Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.

14 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.

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

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

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