Towards a Primitive Higher Order Calculus of Broadcasting Systems
(Extended Version)

Karol Ostrovský, K. V. S. Prasad and Walid Taha

Chalmers University of Technology, Gothenburg, Sweden,
{karol,prasad,taha}@cs.chalmers.se

Abstract. Ethernet-style broadcast is a pervasive style of computer communication. In this style, the medium is a single nameless channel. Previous work on modelling such systems proposed CBS. In this paper, we propose a fundamentally different calculus called HOBS. Compared to CBS, HOBS 1) is higher order rather than first order, 2) supports dynamic subsystem encapsulation rather than static, and 3) does not require an underlying language to be Turing-complete. Moving to a higher order calculus is key to increasing the expressivity of the primitive calculus and alleviating the need for an underlying language. The move, however, raises the need for significantly more machinery to establish the basic properties of the new calculus. The main contribution of this paper is adapting Howe's method to HOBS to prove that bisimulation is a congruence. From this result, HOBS is shown to embed the lazy λ-calculus. We conclude with an outline of ongoing and future work.

1 Introduction

Ethernet-style broadcast is a pervasive style of computer communication. The bare medium provided by the Ethernet is a single nameless channel. Typically, more sophisticated programming idioms such as point-to-point communication or named channels are built on top of the Ethernet. But using the Ethernet as-is can allow the programmer to make better use of bandwidth, and exploit broadcast (itself a powerful and natural programming primitive). In this paper, we propose a primitive higher order calculus of broadcasting systems (HOBS) that models many of the important features of the bare Ethernet, and develop some of its basic operational properties.

1.1 Basic Characteristics of the Ethernet

HOBS builds on and extends the calculus of broadcasting systems (CBS) [14]. The basic abstractions of CBS are present in HOBS. These basic abstractions are inspired by the Ethernet protocol:

- The medium is a single nameless channel.

- Any node can broadcast a message, and it is instantaneously delivered to all the other nodes.
- Messages need not specify either transmitter or receiver.
- The transmitter of a message decides what is transmitted and when.
- Any receiver has to consume whatever is on the net, at any time.
- Only one message can be transmitted at a any time.
- Collision detection and resolution are provided by the protocol (for HOBS, the operational semantics), so the abstract view is that if two nodes are trying to transmit simultaneously, one is chosen arbitrarily to do so.
- All nodes are treated equally; their position on the net does not matter.

But while HOBS and CBS share this common basis, the two systems take fundamentally different approaches to the problem of subsystem (and broadcast) encapsulation. To illustrate these differences, we take a closer look at how the Ethernet addresses this issues.

1.2 Modelling Ethernet-Style Encapsulation

Whenever the basic mode of communication is broadcast, encapsulating subsystems is an absolute necessity. In the Ethernet, bridges are used to regulate communication between Ethernet subsystems. A bridge can stop or translate messages crossing it, but local transmission on either side is unaffected by the bridge. Either side of a bridge can be seen as a subsystem of the other.

CBS models bridges by pairs of functions that filter and translate messages going in each direction across the bridge. While this is an intuitively appealing model suitable for many applications, it has limitations:

1. An underlying language is needed. In particular, CBS is a *first order* process calculus, that is, messages are distinct from processes. A separate, computationally rich underlying language is needed to express the function pairs.
2. CBS only provides a *static* model of Ethernet architectures. For example, real bridges can change their routing behaviour. CBS provides neither for such a change nor for mobile systems that might cross bridges.
3. Any broadcast that has to cross a bridge in CBS does so instantly. This is unrealistic; real bridges usually buffer messages.

HOBS addresses these limitations by:

- Supporting first-class, transmittable processes, and
- Providing two novel constructs called Link and Feed.

Combined, these features of HOBS yield a Turing-complete language sufficient for expressing translators (limitation 1 above), and allow us to model dynamic architectures (limitation 2). The Link and Feed constructs allow us to model the buffering of messages that cannot be consumed immediately (limitation 3).

1.3 Problem

Working with a higher order calculus comes at a cost to developing the theory of HOBS. In particular, whereas the definition of behavioural equivalence in a first order language can require an exact match between the transmitted messages, such a definition is too discriminating when messages involve processes (non-first order values). Thus, behavioural equivalence must only require the transmission of equivalent messages.

Unfortunately, this seemingly small change to the notion of equivalence introduces significant complexity to the proofs of the basic properties themselves. In particular, the standard technique for proving that bisimulation is a congruence does not go through. A key difficulty seems to be that the standard technique cannot be used directly to show that the substitution of equivalent processes for variables preserves equivalence (This problem is detailed in Section 4.1.)

1.4 Contributions and Organisation of this Paper

After presenting the syntax and semantics of HOBS (Section 2), we give the formal definition of applicative equivalence for HOBS (Section 3).

The key technical contribution of this paper is using an adaptation of Howe's method [8] to establish that applicative equivalence for HOBS is a bisimulation (Section 4). As is typical in the treatment of concurrent calculi, we also introduce a notion of *weak* equivalence. It turns out that essentially the same development can be used for this notion of equivalence (Section 5).

To illustrate the utility of the these results, we use them to show that HOBS embeds the lazy λ-calculus [1] and therefore any data type (Section 6). The existence of this encoding seems to come directly from the fact that HOBS is higher order. Possible encodings of CBS and the π-calculus [10] are discussed (Also in Section 6).

These results lay the groundwork for further exploration, for example to compare the expressive power of different primitive calculi of concurrency. We conclude the paper by discussing the related calculi and future works (Section 7).

1.5 Remarks

No knowledge is assumed of CBS or any other process calculus; the formal development in this paper is self-contained.

A side comment for readers familiar with CCS [9], CHOCS [16] and CBS: Roughly, HOBS is to CBS what CHOCS is to CCS.

2 Syntax and Semantics

HOBS has eight process constructors, formally defined in the next subsection. Their informal meaning is as follows:

- **0** is a process that says nothing and ignores everything it hears.

- x is a variable name. Scoping of variables and substitution is essentially as in the λ-calculus.
- $x?p_1$ receives any message q and becomes $p_1[q/x]$.
- $p_1!p_2$ can say p_1 and become p_2. It ignores everything it hears.
- $\langle x?p_1 + p_2!p_3\rangle$ says p_2 and becomes p_3 except if it hears something, say q, whereupon it becomes $p_1[q/x]$.
- $p_1|p_2$ is the parallel composition of p_1 and p_2. They interact with each other; with the environment, $p_1|p_2$ interacts as if it were one process.
- $p_1 \curvearrowright p_2$ is p_1 "linked" to p_2. This link is asymmetric; p_2 hears the environment and p_1 speaks to it. A nameless private channel connects p_2 to p_1.
- $p_1 \triangleleft p_2$ consists of p_1 being "fed" p_2 for later consumption as an incoming message. It cannot speak to the environment till p_1 has consumed p_2.

As is characteristic for a higher order calculus, variable names are constructs in themselves. Apart from this and the feed and link constructs, HOBS looks exactly like CBS in both the syntax and the operational rules.

The semantics is given by labelled transitions, the labels being of the form $p!$ or $p?$ where p is a process. The former are broadcast, speech, or transmission, and the latter are hearing or reception. Transmission can consistently be interpreted as an autonomous action, and reception as controlled by the environment. This is because processes are always ready to hear anything, transmission absorbs reception in parallel composition, and encapsulation can stop reception but only hide transmission. For more discussion, see [14].

The link operator $\curvearrowright$ turns out to be associative. Note that $p_1 \curvearrowright p_2 \curvearrowright p_3$ can be seen as p_2 encapsulated by p_3 inwards and by p_1 outwards. Thus the link is half a bridge, a simplifying idea that might be useful even in CBS.

The feed operator $\triangleleft$ is foreshadowed in implementations of CBS, see [14], that achieve apparent synchrony while allowing subsystems to fall behind.

2.1 Formal Syntax and Semantics of HOBS

The syntax of HOBS is defined in Figure 1. Terms are treated as equivalence classes of α-convertible terms. The set P_L is a set of process terms with free variables in the set L, for example $P_\varnothing$ is the set of all closed terms.

Figure 2 defines the semantics in terms of a transition relation $\cdot \xrightarrow{\cdot} \cdot \subseteq P_\varnothing \times A_\varnothing \times P_\varnothing$. *Free-variables*, *substitution* and *context filling* are defined as usual (Appendix A).

The rest of this section discusses some semantics issues.

HOBS is a non-deterministic calculus. The transition relation $\cdot \xrightarrow{\cdot} \cdot \subseteq P_\varnothing \times A_\varnothing \times P_\varnothing$ fails to be a function (in the first two arguments) because the composition rule allows arbitrary exchange of messages between sub-processes. The choice constructor does not introduce non-determinism by itself, since any broadcast collision can be resolved by allowing the left sub-process of a parallel composition to broadcast.

However, the calculus is deterministic on input. Also, it is so-called input enabled, which means that every process can receive any input at any point. In

mathematical terms, we can show the following property by induction on the height of inference of reduction $p \xrightarrow{m?} p'$

$$\forall p, m.\, \exists! p'.\, p \xrightarrow{m?} p' \tag{1}$$

Allowing arbitrary process to buffer its input, that is, changing the receive rule for feed

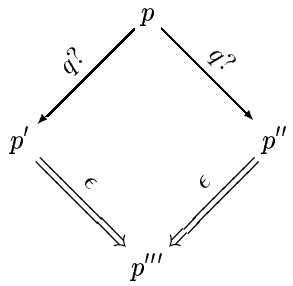
$$\text{from} \quad f \xrightarrow{q?} f \triangleleft q \quad \text{to} \quad p \xrightarrow{q?} p \triangleleft q$$

would change the calculus to input enabled. That is, we could only prove

$$\forall p, m.\, \exists p'.\, p \xrightarrow{m?} p'$$

Moreover, such a calculus would be capable of producing infinite sequences of $\tau!$ transitions, since we could derive for example $\mathbf{0} \triangleleft p \xrightarrow{\tau!} \mathbf{0} \triangleleft p$.

Instead of input determinism, we could prove input confluency, that is that the following diagram commutes:



where $\xRightarrow{\epsilon}$ is the reflexive transitive closure of $\xrightarrow{\tau!}$. In this case, the $\xRightarrow{\epsilon}$ transition would correspond to several β-reduction steps in sequence. The relation to λ-calculus is studied in more detail in Section 5

The split of the syntax into two categories, namely G-Terms and Feed Buffers, allowed us to study different possibilities to define the behaviour of feed constructor. For example in the last rule, the use of f rather than p, that is, having the rule

$$\frac{f_1 \xrightarrow{\tau!} p_1'}{f_1 \triangleleft p_2 \xrightarrow{\tau!} p_1' \triangleleft p_2} \quad \text{rather than} \quad \frac{p_1 \xrightarrow{\tau!} p_1'}{p_1 \triangleleft p_2 \xrightarrow{\tau!} p_1' \triangleleft p_2}$$

is essential. Without the restriction to q the rule could cause an undesired interference between the link and the feed constructs. To illustrate this interference we need some technical development, therefore we defer more detailed discussion to Appendix C.

Also, changing the first transmit rule for link

$$\text{from} \quad \frac{q_1 \xrightarrow{p_2?} p_1'}{q_1 \triangleleft p_2 \xrightarrow{\tau!} p_1'} \quad \text{to} \quad \frac{p_1 \xrightarrow{p_2?} p_1'}{p_1 \triangleleft p_2 \xrightarrow{\tau!} p_1'}$$

would allow a derivation of infinite sequences of $\tau!$ transitions, for example we would have the following derivation:

$$\frac{f_1 \xrightarrow{p_3?} f_1 \triangleleft p_3}{f_1 \triangleleft p_3 \xrightarrow{\tau!} f_1 \triangleleft p_3}$$

3 Applicative Bisimulation for HOBS

This section follows the standard development of simulation and bisimulation relations for concurrent calculi [9]. Also, standard proof techniques are used in all the proofs in this section.

Since the transition system carries processes in labels and higher order simulation/bisimulation has to account for the structure of these processes, message extension defined below extends process relations to message relations.

Definition 1 (Message Extension). *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. Its* message extension *$\mathcal{R}_\tau \subseteq M \times M$ is defined by the following rules*

$$\frac{}{\tau \,\mathcal{R}_\tau\, \tau}\ \textsc{TauExt} \qquad \frac{p \,\mathcal{R}\, q}{p \,\mathcal{R}_\tau\, q}\ \textsc{MsgExt}$$

Adaptation of Thomsen's definition of applicative higher order simulation [16] seems as a most suitable notion of strong simulation for HOBS. There are two reasons for this: first, simulation in a higher order calculus has to account for structure of messages; second, HOBS can be viewed as an extension of the lazy λ-calculus [1] in which applicative bisimulation is widely used.

Definition 2 (Applicative Simulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is a* (strong, higher order) applicative simulation, *written $S(\mathcal{R})$, when*

$$\forall (p,q) \in \mathcal{R},\ \forall m, p'.$$
$$1.\ p \xrightarrow{m?} p' \ \Rightarrow\ (\exists q'.\, q \xrightarrow{m?} q' \ \wedge\ p' \,\mathcal{R}\, q')$$
$$2.\ p \xrightarrow{m!} p' \ \Rightarrow\ (\exists q', n.\, q \xrightarrow{n!} q' \ \wedge\ p' \,\mathcal{R}\, q' \ \wedge\ m \,\mathcal{R}_\tau\, n)$$

In a standard way, a relation $\mathcal{R}$ is applicative bisimulation if both it and its converse are applicative simulations

Definition 3 (Applicative Bisimulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is an* applicative bisimulation, *written $B(\mathcal{R})$, when both $S(\mathcal{R})$ and $S(\mathcal{R}^{-1})$ hold. That is,*

$$\frac{S(\mathcal{R}) \quad S(\mathcal{R}^{-1})}{B(\mathcal{R})}$$

Syntax

Variables	$x \in X$	$=$	*A countable set of names*
Processes	$p, q, r \in P ::= g \mid f$		
G-terms	$g \in G ::= \mathbf{0}$	–	Nil
	$\mid x$	–	Variable
	$\mid x?p$	–	Input
	$\mid p!p$	–	Output
	$\mid \langle x?p + p!p \rangle$	–	Guarded Choice
	$\mid p \mid p$	–	Parallel Composition
	$\mid p^{\frown} p$	–	Link
Feed Buffers	$f \in F ::= p \triangleleft p$		
Messages	$l, m, n \in M ::= \tau \mid p$		
Actions	$a \in A ::= m! \mid m?$		
Contexts	$c \in C ::= [\,]$	–	Hole
	$\mid \mathbf{0}$	–	Nil
	$\mid x$	–	Variable
	$\mid x?c$	–	Input
	$\mid c!c$	–	Output
	$\mid \langle x?c + c!c \rangle$	–	Guarded Choice
	$\mid c \mid c$	–	Parallel Composition
	$\mid c^{\frown} c$	–	Link
	$\mid c \triangleleft c$	–	Feed Buffer

Syntax indexed by a set of free-variables L

$$\begin{aligned} P_L &= \{p \mid p \in P \wedge FV(p) \subseteq L\} \\ A_L &= \{\tau?, \tau!, p?, p! \mid p \in P_L\} \end{aligned}$$

Notation

The names ranging over each syntactic category are given above. Thus process terms are always p, q or r. Natural number subscripts indicate subterms. So p_1 is a subterm of p. Primes indicate derivatives, as in $p \xrightarrow{q?} p'$.

Fig. 1. Syntax

Semantics

	Receive	Transmit
Silence	$p \xrightarrow{\tau?} p$	
Nil	$\mathbf{0} \xrightarrow{q?} \mathbf{0}$	
Input	$x?p_1 \xrightarrow{q?} p_1[q/x]$	
Output	$p_1!p_2 \xrightarrow{q?} p_1!p_2$	$p_1!p_2 \xrightarrow{p_1!} p_2$
Choice	$\langle x?p_1 + p_2!p_3\rangle \xrightarrow{q?} p_1[q/x]$	$\langle x?p_1 + p_2!p_3\rangle \xrightarrow{p_2!} p_3$
Compose	$\dfrac{p_1 \xrightarrow{q?} p_1' \quad p_2 \xrightarrow{q?} p_2'}{p_1 \vert p_2 \xrightarrow{q?} p_1' \vert p_2'}$	$\dfrac{p_1 \xrightarrow{m!} p_1' \quad p_2 \xrightarrow{m?} p_2'}{p_1 \vert p_2 \xrightarrow{m!} p_1' \vert p_2'}$ $\dfrac{p_1 \xrightarrow{m?} p_1' \quad p_2 \xrightarrow{m!} p_2'}{p_1 \vert p_2 \xrightarrow{m!} p_1' \vert p_2'}$
Link	$\dfrac{p_2 \xrightarrow{q?} p_2'}{p_1 \frown p_2 \xrightarrow{q?} p_1 \frown p_2'}$	$\dfrac{p_1 \xrightarrow{m!} p_1'}{p_1 \frown p_2 \xrightarrow{m!} p_1' \frown p_2}$ $\dfrac{p_1 \xrightarrow{m?} p_1' \quad p_2 \xrightarrow{m!} p_2'}{p_1 \frown p_2 \xrightarrow{\tau!} p_1' \frown p_2'}$
Feed	$f \xrightarrow{q?} f \triangleleft q$	$\dfrac{g_1 \xrightarrow{p_2?} p_1'}{g_1 \triangleleft p_2 \xrightarrow{\tau!} p_1'}$ $\dfrac{f_1 \xrightarrow{\tau!} p_1'}{f_1 \triangleleft p_2 \xrightarrow{\tau!} p_1' \triangleleft p_2}$

Fig. 2. Semantics

The following lemma sums up the basic properties of simulation and bisimulation relations. First, identity relation on closed processes is a simulation and a bisimulation. Second, the property of being a simulation or a bisimulation is preserved by relational composition, union. Also, the bisimulation property is preserved by transpose as well.

Lemma 4. *Let* $\mathcal{R}, \mathcal{S} \subseteq P_\varnothing \times P_\varnothing$ *and* $Id_P = \{(p,p) \mid \forall p \in P_\varnothing\}$ *be relations. Then*

1. $S(Id_P)$, $B(Id_P)$
2. $S(\mathcal{R}) \wedge S(\mathcal{S}) \;\Rightarrow\; S(\mathcal{RS})$, $S(\mathcal{R} \cup \mathcal{S})$
3. $B(\mathcal{R}) \wedge B(\mathcal{S}) \;\Rightarrow\; B(\mathcal{RS})$, $B(\mathcal{R} \cup \mathcal{S})$
4. $B(\mathcal{R}) \;\Rightarrow\; B(\mathcal{R}^{-1})$

Proof. Follows immediately from the definitions (see for example the proof of Proposition 2, in Section 4.2 in [9]). □

Two closed processes p and q are equivalent, written $p \sim q$, if there exist a bisimulation relation $\mathcal{R}$ such that $(p,q) \in \mathcal{R}$. In other words, applicative equivalence is a union of all bisimulation relations:

Definition 5 (Applicative Equivalence). *The* applicative equivalence *is a relation* $\sim \subseteq P_\varnothing \times P_\varnothing$ *defined as a union of all bisimulation relations. That is,*

$$\sim \;=\; \bigcup_{\mathcal{R} \subseteq P_\varnothing \times P_\varnothing} \{\mathcal{R} \mid B(\mathcal{R})\}$$

Proposition 6.

1. $B(\sim)$, *that is,* $\sim$ *is a bisimulation.*
2. $\sim$ *is an equivalence, that is,* $\sim$ *is reflexive, symmetric and transitive.*

Proof. We get the first part of the proof from Proposition 4, namely inductively using $B(\mathcal{R}) \wedge B(\mathcal{S}) \;\Rightarrow\; B(\mathcal{R} \cup \mathcal{S})$.

For the second part, we get reflexivity and transitivity from Proposition 4, namely from $B(Id_P)$ and $B(\mathcal{R}) \wedge B(\mathcal{S}) \;\Rightarrow\; B(\mathcal{RS})$. Finally, $\sim$ is symmetric since it is a bisimulation. □

The calculus has been shown to enjoy the following elementary properties:

Proposition 7. *We have:*

- *Input*
  1. $x?\mathbf{0} \sim \mathbf{0}$
- *Parallel Composition*
  1. $p|\mathbf{0} \sim p$
  2. $p_1|p_2 \sim p_2|p_1$
  3. $p_1|(p_2|p_3) \sim (p_1|p_2)|p_3$
- *Link*
  1. $\mathbf{0}^\frown p \sim \mathbf{0}$
  2. $(p_1{}^\frown p_2)^\frown p_3 \sim p_1{}^\frown(p_2{}^\frown p_3)$
  3. $x?p^\frown\mathbf{0} \sim \mathbf{0}$
  4. $\langle x?p_1 + p_2!p_3\rangle^\frown\mathbf{0} \sim p_2!p_3{}^\frown\mathbf{0}$
- *Choice*
  1. $\langle x?p_1!p_2 + p_1!p_2\rangle \sim p_1!p_2$, $x \notin FV(p_1!p_2)$
  2. $\langle x?p_1 + p_2!p_3\rangle|x?p_4 \sim \langle x?(p_1|p_4) + p_2!(p_3|p_4[p_2/x])\rangle$

Proof. Proof by establishing bisimulations for each of the parts. □

4 Equivalence as a Congruence

In this section we use Howe's method [8] to show that the applicative equivalence relation $\sim$ is a congruence. To motivate the need for Howe's proof method, we start by showing the difficulties with the standard proof technique. Then, we present an adaptation of Howe's basic development for HOBS. And, we conclude by applying this adaptation to applicative equivalence.

An equivalence is a congruence when two equivalent terms are not distinguishable in any context:

Definition 8 (Congruence). *Let $\mathcal{R} \subseteq P \times P$ be an equivalence relation on process terms. $\mathcal{R}$ is a* congruence *when*

$$\forall p, q \in P, c \in C.\, p\,\mathcal{R}\,q \;\Rightarrow\; c[p]\,\mathcal{R}\,c[q]$$

4.1 Difficulties with the Standard Proof method

The standard congruence proof method is to show, by induction on syntax, that equivalence $\sim$ contains its compatible refinement $\widehat{\sim}$, where compatible refinement is defined as:

Definition 9 (Compatible Refinement). *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. Its* compatible refinement *$\widehat{\mathcal{R}}$ is defined by the following rules*

$$\frac{}{\mathbf{0}\,\widehat{\mathcal{R}}\,\mathbf{0}}\ \textsc{Comp Nil} \qquad \frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2 \quad p_3\,\mathcal{R}\,q_3}{\langle x?p_1 + p_2!p_3\rangle\,\widehat{\mathcal{R}}\,\langle x?q_1 + q_2!q_3\rangle}\ \textsc{Comp Choice}$$

$$\frac{}{x\,\widehat{\mathcal{R}}\,x}\ \textsc{Comp Var} \qquad \frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2}{p_1|p_2\,\widehat{\mathcal{R}}\,q_1|q_2}\ \textsc{Comp Comp}$$

$$\frac{p_1\,\mathcal{R}\,q_1}{x?p_1\,\widehat{\mathcal{R}}\,x?q_1}\ \textsc{Comp In} \qquad \frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2}{p_1 \frown p_2\,\widehat{\mathcal{R}}\,q_1 \frown q_2}\ \textsc{Comp Link}$$

$$\frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2}{p_1!p_2\,\widehat{\mathcal{R}}\,q_1!q_2}\ \textsc{Comp Out} \qquad \frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2}{p_1 \triangleleft p_2\,\widehat{\mathcal{R}}\,q_1 \triangleleft q_2}\ \textsc{Comp Feed}$$

Note that substitutivity is preserved by compatible refinement (Lemma 11 below), where substitutivity is defined (in a usual way) as:

Definition 10 (Substitutivity). *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. $\mathcal{R}$ is called* substitutive *when the following rule holds*

$$\frac{p_1\,\mathcal{R}\,q_1 \quad p_2\,\mathcal{R}\,q_2}{p_1[p_2/x]\,\mathcal{R}\,q_1[q_2/x]}\ \textsc{Rel Subst}$$

Lemma 11. *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. If the relation $\mathcal{R}$ is substitutive then so is the compatible refinement $\widehat{\mathcal{R}}$.*

Proof. Straightforward induction on the definition of the compatible refinement $\widehat{\mathcal{R}}$. □

The advantage of using the notion of compatible refinement $\widehat{\mathcal{R}}$ is that it allows to concisely express case analysis on the outermost constructor. And, the following lemma justifies the standard proof method:

Lemma 12. *Let $\mathcal{R} \subseteq P \times P$ be an equivalence relation on process terms. Then $\mathcal{R}$ is a congruence iff $\widehat{\mathcal{R}} \subseteq \mathcal{R}$.*

Proof. A variant of the proof in [7].

"$\Rightarrow$" Suppose that $\mathcal{R}$ is a congruence. We show that $\widehat{\mathcal{R}} \subseteq \mathcal{R}$ using the definition of compatible refinement $\widehat{\mathcal{R}}$.

COMP NIL and COMP VAR hold since $\mathcal{R}$ is reflexive.

COMP IN let $c \equiv x?[]$ and $p \,\mathcal{R}\, q$. Since $\mathcal{R}$ is a congruence we know that $c[p] \,\mathcal{R}\, c[q]$, that is $x?p \,\mathcal{R}\, x?q$.

COMP OUT let $p_1 \,\mathcal{R}\, q_1$ and $p_2 \,\mathcal{R}\, q_2$. Since $\mathcal{R}$ is a congruence we know that for $c_1 \equiv []!p_2$ and $c_2 \equiv q_1![]$ we have

$$c_1[p_1] \,\mathcal{R}\, c_1[q_1] \;\wedge\; c_2[p_2] \,\mathcal{R}\, c_2[q_2]$$

which we can rewrite as

$$p_1!p_2 \,\mathcal{R}\, q_1!p_2 \;\wedge\; q_1!p_2 \,\mathcal{R}\, q_1!q_2$$

Now since $\mathcal{R}$ is transitive we have that $p_1!p_2 \,\mathcal{R}\, q_1!q_2$.

The rest follows the arguments used in COMP OUT .

"$\Leftarrow$" Suppose that $\widehat{\mathcal{R}} \subseteq \mathcal{R}$. We show that $\mathcal{R}$ is a congruence by structural induction on the context. For all the cases suppose that $p \,\mathcal{R}\, q$.

$c \equiv []$ trivially $c[p] \,\mathcal{R}\, c[q]$ since $p \,\mathcal{R}\, q$.

$c \equiv \mathbf{0}$ trivially $c[p] \,\mathcal{R}\, c[q]$ since $\mathcal{R}$ is reflexive.

$c \equiv x$ trivially $c[p] \,\mathcal{R}\, c[q]$ since $\mathcal{R}$ is reflexive.

$c \equiv x?c_1$ since c_1 is structurally smaller, using induction hypothesis we know that $c_1[p] \,\mathcal{R}\, c_1[q]$. Now since $\widehat{\mathcal{R}} \subseteq \mathcal{R}$ we can use rule COMP IN and we get $x?c_1[p] \,\mathcal{R}\, x?c_1[q]$. Using the definition of context filling we get that $c[p] \,\mathcal{R}\, c[q]$.

The rest follows the arguments used for input.

□

The standard proof (to show $\widehat{\sim} \subseteq \sim$) still proceeds by case analysis. Several cases are simple (nil $\mathbf{0}$, variable and output $x?\cdot$). The case of feed $\cdot \triangleleft \cdot$ is slightly more complicated. All the other cases are problematic (especially composition $\cdot|\cdot$), since they require substitutivity of equivalence $\sim$.

In HOBS, the standard inductive proof of substitutivity of equivalence $\sim$ would require equivalence to be a congruence. And, we are stuck. Attempt to prove substitutivity and $\widehat{\sim} \subseteq \sim$ simultaneously does not work either, since term's size can increase and this makes use of induction on syntax impossible. Similar problems seem to be common for higher order calculi (see for example [16, 5, 1]).

4.2 Howe's Basic Development

Howe [8] proposed a general method for proving that some bisimulation-based equivalences are congruences. Following similar adaptations of Howe's method [7, 5] we present the adaptation to HOBS together with few technical lemmas. We use the standard definition of the restriction $\mathcal{R}_{\varnothing}$ of a relation $\mathcal{R}$ to closed processes (Definition 13). Extension of a relation to open terms is also the standard one:

Definition 13 (Restriction to closed processes). *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. Its* restriction *to closed process terms is defined as:*

$$\mathcal{R}_{\varnothing} = \mathcal{R} \cap (P_{\varnothing} \times P_{\varnothing})$$

Definition 14 (Open Extension). *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. Its* open extension *is defined by the following rule*

$$\frac{\forall \sigma.\, (p)\sigma, (q)\sigma \in P_{\varnothing} \;\Rightarrow\; (p)\sigma \,\mathcal{R}\, (q)\sigma}{p \,\mathcal{R}^{\circ}\, q}$$

Note that for $\mathcal{R} \subseteq P_{\varnothing} \times P_{\varnothing}$ we have ${\mathcal{R}^{\circ}}_{\varnothing} = \mathcal{R}$. But if $\mathcal{R} \subseteq P \times P$ we cannot in general prove that $\mathcal{R} = {\mathcal{R}_{\varnothing}}^{\circ}$. As a simple counter example consider $\mathcal{R} = (x, y)$ where x and y are distinct variables.

The key part of Howe's method is the definition of the candidate relation $\mathcal{R}^{\bullet}$:

Definition 15 (Candidate Relation). *Let $R \subseteq P \times P$ be a relation on process terms. The a* candidate relation *is defined as the least relation that satisfies the rule*

$$\frac{p \,\widehat{\mathcal{R}^{\bullet}}\, r \quad r \,\mathcal{R}^{\circ}\, q}{p \,\mathcal{R}^{\bullet}\, q} \textsc{Cand Def}$$

The definition of the candidate relation $\mathcal{R}^{\bullet}$ is designed to facilitate simultaneous inductive proof on syntax and on reductions. Note that the definition of the compatible refinement $\widehat{\mathcal{R}}$ involves only case analysis over syntax. And, inlining the compatible refinement $\widehat{\mathcal{R}}$ in the definition of the candidate relation would reveal inductive use of the candidate relation $\mathcal{R}^{\bullet}$.

The relevant properties of a candidate relation $\mathcal{R}^{\bullet}$ are summed up below:

Lemma 16. *Let $\mathcal{R} \subseteq P_{\varnothing} \times P_{\varnothing}$ be a preorder (reflexive, transitive relation) on closed process terms. Then the following rules are valid*

$$\frac{}{p \,\mathcal{R}^{\bullet}\, p} \textsc{Cand Ref} \qquad \frac{p \,\mathcal{R}^{\circ}\, q}{p \,\mathcal{R}^{\bullet}\, q} \textsc{Cand Sim} \qquad \frac{p \,\widehat{\mathcal{R}^{\bullet}}\, q}{p \,\mathcal{R}^{\bullet}\, q} \textsc{Cand Cong}$$

$$\frac{p \,\mathcal{R}^{\bullet}\, r \quad r \,\mathcal{R}^{\circ}\, q}{p \,\mathcal{R}^{\bullet}\, q} \textsc{Cand Right} \qquad \frac{p_1 \,\mathcal{R}^{\bullet}\, q_1 \quad p_2 \,\mathcal{R}^{\bullet}\, q_2}{p_1[p_2/x] \,\mathcal{R}^{\bullet}\, q_1[q_2/x]} \textsc{Cand Subst}$$

Proof. See Lemma 3.5 in [7] or for a similar proof in more general setting see Lemma 3.1 and Lemma 3.2 in [8]. □

Corollary 17. *We have* $\mathcal{R}^\bullet \subseteq {\mathcal{R}^\bullet}_\varnothing{}^\circ$ *as an immediate consequence of rule* CAND SUBST *and rule* CAND REF.

Lemma 18. *Let* $\mathcal{R} \subseteq P \times P$ *be an equivalence relation. Then* $\mathcal{R}^{\bullet *}$ *is symmetric.*

Proof. It suffices to show that if $p\,\mathcal{R}^\bullet\,q$ then $q\,\mathcal{R}^{\bullet *}\,p$. The proof proceeds by induction over the structure of p. We will prove all the cases at once: let $p\,\mathcal{R}^\bullet\,q$. We know from the definition of the candidate relation $\mathcal{R}^\bullet$ that

$$\exists r.\, p\,\widehat{\mathcal{R}^\bullet}\,r\,\mathcal{R}^\circ\,q \tag{2}$$

Now, to derive $p\,\widehat{\mathcal{R}^\bullet}\,r$ we must use one of the COMP $\cdot$ rules from the definition of the compatible refinement $\widehat{\mathcal{R}}$. In both the base cases ($p \equiv \mathbf{0}$ and $p \equiv x$) we trivially have $r\,\widehat{\mathcal{R}^{\bullet *}}\,p$. In all the other cases we know that subterms of p and r (p_i and r_i) must be related, that is $p_i\,\mathcal{R}^\bullet\,q_i$. This allows us to apply the induction hypothesis to get that $r_i\,\mathcal{R}^{\bullet *}\,p_i$ and to use the appropriate COMP $\cdot$ rule to get that $r\,\widehat{\mathcal{R}^{\bullet *}}\,p$. Applying results of Lemma 16 (CAND CONG) we get

$$r\,\mathcal{R}^{\bullet *}\,p \tag{3}$$

To finish the proof, we have that $r\,\mathcal{R}^\circ\,q$ and by symmetry of $\mathcal{R}$ and by Lemma 16 (CAND SIM) we have that

$$q\,\mathcal{R}^\bullet\,r \tag{4}$$

Combination of equation 4 and equation 3 gives us the result.

For a similar proof in more general setting see Lemma 3.3 in [8] □

The three most important properties of a candidate relation are substitutivity (rule CAND SUBST), the fact that it contains its compatible refinement (rule CAND CONG), and that its reflexive and transitive closure is symmetric (Lemma 18).

The last lemma of this subsection says that if two candidate related processes, $p\,\mathcal{R}^\bullet\,q$, are closed then there always exists a proof tree for this derivation which involves only closed process terms in the last derivation step.

Lemma 19 (Closed Middle).

$$\forall p, q \in P_\varnothing.\, p\,\mathcal{R}^\bullet\,q \;\Rightarrow\; \exists r \in P_\varnothing.\, p\,\widehat{\mathcal{R}^\bullet}\,r\,\mathcal{R}^\circ\,q$$

Proof. From the definition of the candidate relation $\mathcal{R}^\bullet$ we know that

$$\exists r \in P.\, p\,\widehat{\mathcal{R}^\bullet}\,r\,\mathcal{R}^\circ\,q$$

If r is closed we are done. In the rest of the proof we show that if r is not closed then $\exists r' \in P_\varnothing.\, p\,\widehat{\mathcal{R}^\bullet}\,r'\,\mathcal{R}^\circ\,q$. We take $r' = (r)\sigma$, where σ is any closing substitution, that is $(r)\sigma \in P_\varnothing$. Then, using the fact that p and q are closed, we get $p\,\widehat{\mathcal{R}^\bullet}\,r'$ from Lemma 11 and $r'\,\mathcal{R}^\circ\,q$ follows from the definition of the open extension $\mathcal{R}^\circ$. □

4.3 Congruence of the Equivalence Relation

In this subsection, we fix the underlying relation $\mathcal{R}$ to be $\sim$. The goal of this subsection is twofold: first, to show that the candidate relation $\sim^\bullet$ coincides with the open extension $\sim^\circ$, that is $\sim^\bullet = \sim^\circ$; and second, to use this fact to complete the congruence proof.

We already have that $\sim^\circ \subseteq \sim^\bullet$ from Lemma 16 (CAND SIM). To show the converse we begin be proving that the closed restriction of the candidate relation $\sim^\bullet_\varnothing$ is a simulation. This requires showing that the following two simulation conditions hold:

$$\begin{array}{l}\forall p, q \in P_\varnothing . p \sim^\bullet q \;\Rightarrow\; \forall m, p' . \\ 1.\; p \xrightarrow{m?} p' \;\Rightarrow\; (\exists q' . q \xrightarrow{m?} q' \;\wedge\; p' \sim^\bullet q') \\ 2.\; p \xrightarrow{m!} p' \;\Rightarrow\; (\exists q', n . q \xrightarrow{n!} q' \;\wedge\; p' \sim^\bullet q' \;\wedge\; m \sim^\bullet_\tau n)\end{array}$$

We split the proof into two lemmas: Lemma 20 (Receive) and Lemma 21 (Transmit), which prove the respective conditions. Similarly to the standard proof, the parallel composition case is the most difficult and actually requires stronger receive condition to hold. Therefore, the first lemma (Receive) below proves a restriction of the first condition. Then the second lemma (Transmit) below proves the second condition and makes use of the Receive Lemma.

Lemma 20 (Receive). *Let $p, q \in P_\varnothing$ be two closed processes and $p \sim^\bullet q$. Then*

$$\forall m, n, p' . p \xrightarrow{m?} p' \;\wedge\; m \sim^\bullet_\tau n \;\Rightarrow\; (\exists q' . q \xrightarrow{n?} q' \;\wedge\; p' \sim^\bullet q')$$

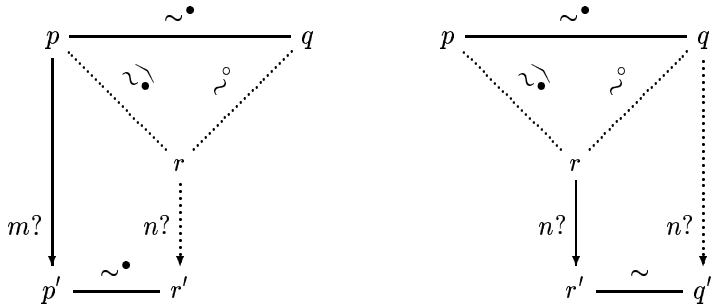
Proof. First note that from Lemma 19 (Closed Middle) we have that

$$\exists r \in P_\varnothing . p \mathrel{\widehat{\sim^\bullet}} r \;\sim^\circ\; q$$

Also, from the definition of equivalence $\sim$ and the fact that r and q are closed processes we know that

$$\forall n, r' . r \xrightarrow{n?} r' \;\Rightarrow\; (\exists q' . q \xrightarrow{n?} q' \;\wedge\; r' \sim q') \tag{5}$$

Pictorially, we have described the situation represented by the right diagram below. In the rest of the proof we show that the statement represented by the left diagram below holds as well. Joining the two diagrams and using Lemma 16 (CAND RIGHT) to infer $p' \sim^\bullet q'$ gives us the result. In the diagram, normal lines, respectively dotted lines are universally, respectively existentially quantified.



To prove the statement in the left diagram we proceed by induction on the height of inference of transition $p \xrightarrow{m?} p'$.

Silence we know that $m \equiv \tau$ and from the definition of the message extension we have that n must also be of the form τ. Since we can apply Silence receive rule to r we trivially get the result.

Nil we know that $p \equiv \mathbf{0}$ and from the definition of the compatible refinement (Comp Nil) we have that $r \equiv \mathbf{0}$. Using the Nil receive rule on r to derive the appropriate transition we get the result

$$\begin{array}{l} \forall m, n, p' \equiv \mathbf{0}.\, p \xrightarrow{m?} p' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow \\ (\exists r' \equiv \mathbf{0}.\, r \xrightarrow{n?} r' \ \wedge\ p' \sim^{\bullet} r') \end{array} \tag{6}$$

where we obtain $p' \sim^{\bullet} r'$ by rule Comp Nil and rule Cand Cong.

Input we know that $p \equiv x?p_1$ and also we know that m must be a process. From the definition of the compatible refinement (Comp In) we have that $r \equiv x?r_1$, and moreover $p_1 \sim^{\bullet} r_1$. Now, we can use Input receive rule on r to derive the appropriate transition and we get the result

$$\begin{array}{l} \forall m, n, p' \equiv p[m/x].\, p \xrightarrow{m?} p' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow \\ (\exists r' \equiv r_1[n/x].\, r \xrightarrow{n?} r' \ \wedge\ p' \sim^{\bullet} r') \end{array} \tag{7}$$

where we know that n has to be a process from the definition of the message extension (Definition 1). We also know that both m and n must be closed because the transition relation is only defined for closed processes. We obtain $p' \sim^{\bullet} r'$ from the following derivation:

$$\dfrac{p_1 \ \sim^{\bullet}\ r_1 \qquad \dfrac{m \sim^{\bullet}_{\tau} n}{m \sim^{\bullet} n}\ \textsc{MsgExt}}{p' \ \equiv\ p_1[m/x] \sim^{\bullet} r_1[n/x] \ \equiv\ r'}\ \textsc{Cand Subst}$$

We also know that p' and r' are closed since p and r were closed and therefore $FV(p_1, r_1) \subseteq \{x\}$ and we substituted a closed processes for variable x.

Output we know that $p \equiv p_1!p_2$ and from the definition of the compatible refinement (Comp Out) we have that $r \equiv p_1!r_2$. Using the Output receive rule on r to derive the appropriate transition and we get the result

$$\begin{array}{l} \forall m, n, p' \equiv (p_1!p_2).\, p \xrightarrow{m?} p' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow \\ (\exists r' \equiv (r_1!r_2).\, r \xrightarrow{n?} r' \ \wedge\ p' \sim^{\bullet} r') \end{array} \tag{8}$$

by using rule Cand Cong to infer $p' \sim^{\bullet} r'$.

Choice this case is basically the same as the Input case

Compose we know that $p \equiv p_1|p_2$ and from the definition of the compatible refinement (Comp Comp) we have that $r \equiv r_1|r_2$, moreover $p_1 \sim^{\bullet} r_1$ and $p_2 \sim^{\bullet} r_2$. In fact, since parallel composition does not create new variable bindings, we know that p_i and r_i are closed. This allows us to use induction

hypothesis on $p_1 \xrightarrow{m?} p_1'$ and $p_2 \xrightarrow{m?} p_2'$ (note that p_1 and p_2 can make these transitions since HOBS is input enabled, see equation 1)

$$\forall m, n, p_1'.\, p_1 \xrightarrow{m?} p_1' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow\ (\exists r_1'.\, r_1 \xrightarrow{n?} r_1' \ \wedge\ p_1' \sim^{\bullet} r_1')$$

$$\forall m, n, r_2'.\, p_2 \xrightarrow{m?} r_2' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow\ (\exists r_2'.\, r_2 \xrightarrow{n?} r_2' \ \wedge\ p_2' \sim^{\bullet} r_2')$$

Now, we can show that r can also make a receive transition and moreover

$$\begin{array}{l} \forall m, n, p' \equiv (p_1'|p_2').\, p \xrightarrow{m?} p' \ \wedge\ m \sim^{\bullet}_{\tau} n \ \Rightarrow \\ (\exists r' \equiv (r_1'|r_2').\, r \xrightarrow{n?} r' \ \wedge\ p' \sim^{\bullet} r') \end{array} \tag{9}$$

where we obtain $p' \sim^{\bullet} r'$ by rule COMP COMP and rule CAND CONG.

Link similar argument as for the Compose case.

Feed we know that $p \equiv p_1 \triangleleft p_2$ and we also know that m must be a process. Moreover from the definition of the compatible refinement (COMP FEED) we have that $r \equiv r_1 \triangleleft r_2$. Now we can use the Feed receive rule on r to derive the appropriate transition and we get the result

$$\begin{array}{l} \forall m, n, p' \equiv (p_1 \triangleleft p_2 \triangleleft m).\, p \xrightarrow{m?} p' \wedge \\ m \sim^{\bullet}_{\tau} n \ \Rightarrow\ (\exists r' \equiv (r_1 \triangleleft r_2 \triangleleft n).\, r \xrightarrow{n?} r' \ \wedge\ p' \sim^{\bullet} r') \end{array} \tag{10}$$

where we obtain $p' \sim^{\bullet} r'$ from rule CAND CONG , rule COMP FEED and the definition of the message extension in the following way:

$$\dfrac{\dfrac{\dfrac{p \mathrel{\widehat{\sim^{\bullet}}} r}{p \sim^{\bullet} r}\,\textsc{Cand Cong} \qquad \dfrac{m \sim^{\bullet}_{\tau} n}{m \sim^{\bullet} n}\,\textsc{MsgExt}}{p \triangleleft m \mathrel{\widehat{\sim^{\bullet}}} r \triangleleft n}\,\textsc{Comp Feed}}{p' \ \equiv\ p \triangleleft m \sim^{\bullet} r \triangleleft n \ \equiv\ r'}$$

□

Lemma 21 (Transmit). *Let $p, q \in P_{\varnothing}$ be two closed processes and $p \sim^{\bullet} q$. Then*

$$\forall m, p'.\, p \xrightarrow{m!} p' \ \Rightarrow\ (\exists n, q'.\, q \xrightarrow{n!} q' \ \wedge\ p' \sim^{\bullet} q' \ \wedge\ m \sim^{\bullet}_{\tau} n) \tag{11}$$

Proof. First note that from Lemma 19 (Closed Middle) we have that

$$\exists r \in P_{\varnothing}.\, p \mathrel{\widehat{\sim^{\bullet}}} r \ \sim^{\circ}\ q$$

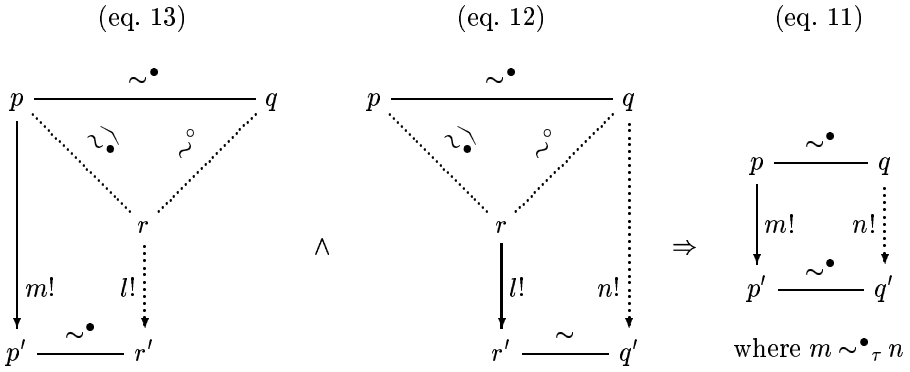
Also, from the definition of equivalence $\sim$ and the fact that r and q are closed processes we know that

$$\forall l, r'.\, r \xrightarrow{l!} r' \ \Rightarrow\ (\exists n, q'.\, q \xrightarrow{n!} q' \ \wedge\ r' \ \sim\ q' \ \wedge\ l \sim_{\tau} n) \tag{12}$$

Is remains to prove the following statement:

$$\forall m, p'.\, p \xrightarrow{m!} p' \ \Rightarrow\ (\exists l, r'.\, r \xrightarrow{l!} r' \ \wedge\ p' \ \sim\ r' \ \wedge\ m \sim_{\tau} l) \tag{13}$$

Joining statements (13) and (12), and and using Lemma 16 (CAND RIGHT) to infer $p' \sim^{\bullet} q'$ and $m \sim^{\bullet}_{\tau} n$ gives us the result. Pictorially (normal lines, respectively dotted lines are universally, respectively existentially quantified):



To prove the statement (eq. 13) we proceed by induction on the structure of p.

$\boldsymbol{p \equiv 0}$ no transmit rule.

$\boldsymbol{p \equiv x?p_1}$ no transmit rule.

$\boldsymbol{p \equiv p_1!p_2}$ from the definition of the compatible refinement (COMP OUT) we have that $r \equiv r_1!r_2$ and $p_1 \sim^\bullet r_1$ and $p_2 \sim^\bullet r_2$. In fact, since output prefix does not create new variable bindings, we know that p_i and r_i are closed processes. When applying the only transmit rule for output to both p and r to infer their appropriate transitions we get:

$$\begin{array}{l} \forall m \equiv p_1, p' \equiv p_2 . p \xrightarrow{m!} p' \ \Rightarrow \\ (\exists l \equiv r_1, r' \equiv r_2 . r \xrightarrow{l!} r' \ \wedge \ p' \sim^\bullet r' \ \wedge \ m \sim^\bullet_\tau l) \end{array} \tag{14}$$

where we obtain $m \sim^\bullet_\tau l$ straight from the definition of the message extension.

$\boldsymbol{p \equiv \langle x?p_1 + p_2!p_3 \rangle}$ this case is basically the same as the case of output.

$\boldsymbol{p \equiv p_1|p_2}$ from the definition of the compatible refinement (COMP COMP) we have that $r \equiv r_1|r_2$, and $p_1 \sim^\bullet r_1$ and $p_2 \sim^\bullet r_2$. In fact, since parallel composition does not create new variable bindings, we know that p_i and r_i are closed processes. Now suppose that m is a process and that the output of a process for p was inferred in the following way:

$$\frac{p_1 \xrightarrow{m!} p_1' \quad p_2 \xrightarrow{m?} p_2'}{p_1|p_2 \xrightarrow{m!} p_1'|p_2'}$$

Using induction hypothesis on p_1 we have that

$$\forall m, p_1' . p_1 \xrightarrow{m!} p_1' \ \Rightarrow \ (\exists l, r_1' . r_1 \xrightarrow{l!} r_1' \ \wedge \ p_1' \sim^\bullet r_1' \ \wedge \ m \sim^\bullet_\tau l)$$

Now, knowing that r_1 can transmit message l and recalling that HOBS is input enabled (equation 1) we can show that that the whole process r can make the following transition:

$$\frac{r_1 \xrightarrow{l!} r_1' \quad r_2 \xrightarrow{l?} r_2'}{r_1|r_2 \xrightarrow{l!} r_1'|r_2'}$$

It remains to show that $p' \sim^\bullet r'$. We already have that $p'_1 \sim^\bullet r'_1$. To show $p'_2 \sim^\bullet r'_2$ we can use of Lemma 20 (Receive):

$$\forall m, n, p'_2.\, p_2 \xrightarrow{m?} p'_2 \ \wedge\ m \sim^\bullet_\tau l \ \Rightarrow\ (\exists r'_2.\, r_2 \xrightarrow{l?} r'_2 \ \wedge\ p'_2 \sim^\bullet r'_2)$$

At this point, it is essential that Lemma 20 (Receive) proved stronger property then the first simulation property. The combination of the two above statements gives us the result

$$\begin{array}{l} \forall m, p' \equiv p'_1 | p'_2.\, p \xrightarrow{m!} p' \ \Rightarrow \\ (\exists l, r' \equiv r'_1 | r'_2.\, r \xrightarrow{l!} r' \ \wedge\ p' \sim^\bullet r' \ \wedge\ m \sim^\bullet_\tau l) \end{array} \tag{15}$$

where we obtain $p' \sim^\bullet r'$ by rule COMP COMP and rule CAND CONG and the fact that all the involved processes are closed.
The case when $m \equiv \tau$ is similar, but a little bit simpler since it does not require the use of Lemma 20 (Receive). Also the case of the symmetric rule is similar.

$\boldsymbol{p_1 \equiv p_{1_1} \frown p_{1_2}}$ this case is similar to the Composition case, since the rules for Link are very similar to those of the Composition.

$\boldsymbol{p_1 \equiv p_{1_1} \lhd p_{1_2}}$ this case is again similar to the Composition case.

□

Having the two above lemmas we have proved that the candidate relation $\sim^\bullet$ is a simulation. Also, using Lemma 18 we get that $\sim^\bullet_\varnothing{}^*$ is symmetric, which means that it is a bisimulation. The two following propositions prove these statements formally. To conclude this section, we then state and prove a main proposition, that the open extension of the equivalence relation $\sim$ is a congruence.

Proposition 22. *$\sim^\bullet_\varnothing$ is a simulation.*

Proof. Note that both Lemma 20 (Receive) and Lemma 21 (Transmit) deal only with closed processes. So the consequence of these lemmas is that the restriction to closed processes of the candidate relation $\sim^\bullet_\varnothing$ is a simulation. □

Proposition 23. *$\sim^\bullet_\varnothing{}^*$ is a bisimulation.*

Proof. Since $\sim^\bullet_\varnothing$ is a simulation (Proposition 22) so is $\sim^\bullet_\varnothing{}^*$ a simulation (Lemma 4). From Lemma 18 we know that $\sim^\bullet_\varnothing{}^*$ is symmetric. Therefore, $\sim^\bullet_\varnothing{}^*$ is a bisimulation. □

Proposition 24. *$\sim^\circ$ is a congruence.*

Proof. First we show that $\sim^\circ$ coincides with the candidate relation $\sim^\bullet$. Note that from Lemma 16 (CAND SIM) we know that $\sim^\circ \subseteq \sim^\bullet$. To prove the converse, from Proposition 23 we know that $\sim^\bullet_\varnothing{}^*$ is a bisimulation. This gives us the following inclusions $\sim^\bullet_\varnothing \subseteq \sim^\bullet_\varnothing{}^* \subseteq \sim$. Since open extension is monotone we have $\sim^\bullet_\varnothing{}^\circ \subseteq \sim^\circ$. Finally, from Corollary 17 we have that $\sim^\bullet \subseteq \sim^\bullet_\varnothing{}^\circ$, which yields $\sim^\bullet \subseteq \sim^\circ$.

To conclude the proof, the relation $\sim^\circ$ is an equivalence, and since it is equal to the candidate relation it contains its compatible refinement ($\widehat{\sim^\circ} \subseteq \sim^\circ$, rule CAND CONG). By Lemma 12 this implies that $\sim^\circ$ is a congruence. □

5 Weak Bisimulation

For many purposes, strong applicative equivalence is too fine as it is sensitive to the number of silent transitions performed by process terms. For example, this means that strong applicative equivalence distinguishes terms that would be β equivalent in the λ-calculus:

$$(x?x) \triangleleft (x?x) \not\sim x?x$$

Equipped with the weak transition relation $\cdot \stackrel{\cdot}{\Longrightarrow} \cdot$ (Definition 25) we define the weak simulation and weak bisimulation in the standard way (Definition 26 and Definition 27 below). Weak equivalence $\approx$ is also defined in the standard way as a union of all weak bisimulation relations (Definition 28). In the same way as for the strong equivalence we prove that $\approx$ is a weak bisimulation and that it is an equivalence (Proposition 29). Moreover, the technique used in proving that strong equivalence is a congruence works also for the weak equivalence (Proposition 30).

Definition 25 (Weak Transition). *Let $\stackrel{\epsilon}{\Longrightarrow}$ be the reflexive transitive closure of $\stackrel{\tau!}{\longrightarrow}$. Then the weak transition is defined as*

$$\stackrel{a}{\Longrightarrow} \quad = \quad \begin{cases} \stackrel{\epsilon}{\Longrightarrow} & \text{if } a \equiv \tau! \\ \stackrel{\epsilon}{\Longrightarrow} \stackrel{a}{\longrightarrow} & \text{otherwise} \end{cases}$$

Definition 26 (Weak simulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is a* (higher order, applicative) weak simulation*, written $S_w(\mathcal{R})$ when*

$$\begin{array}{l} \forall (p_1, p_2) \in \mathcal{R}, \forall m_1, p_3. \\ 1.\ p_1 \stackrel{m_1?}{\longrightarrow} p_3 \ \Rightarrow\ (\exists p_4 . p_2 \stackrel{m_1?}{\Longrightarrow} p_4 \ \wedge\ p_3 \,\mathcal{R}\, p_4) \\ 2.\ p_1 \stackrel{m_1!}{\longrightarrow} p_3 \ \Rightarrow\ (\exists m_2, p_4 . p_2 \stackrel{m_2!}{\Longrightarrow} p_4 \ \wedge\ p_3 \,\mathcal{R}\, p_4 \ \wedge\ m_1 \,\mathcal{R}_\tau\, m_2) \end{array}$$

Definition 27 (Weak Bisimulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is a* weak bisimulation*, written $B_w(\mathcal{R})$ when*

$$\frac{S_w(\mathcal{R}) \quad S_w(\mathcal{R}^{-1})}{B_w(\mathcal{R})}$$

Definition 28 (Weak Equivalence). *The* weak equivalence *is a relation $\approx \subseteq P_\varnothing \times P_\varnothing$ defined as:*

$$\approx \ = \bigcup_{\mathcal{R} \subseteq P_\varnothing \times P_\varnothing} \{\mathcal{R} \mid B_w(\mathcal{R})\}$$

Proposition 29.

1. *$B_w(\approx)$, that is, $\approx$ is a weak bisimulation*
2. *$\approx$ is an equivalence.*

Proposition 30. $\approx$ *is a congruence.*

Proof. The proof follows that of Proposition 24. The only significant difference is when induction hypothesis is used (for example in the case of parallel composition). When using the induction hypothesis we get that the appropriate subterms can perform weak transition $\stackrel{a}{\Longrightarrow}$. Then we have to interleave the $\tau!$ transitions before possibly performing the action a. This interleaving is possible since every process can receive τ without any change to the process itself (see rule Silence in Figure 2). □

Now we can prove that β equivalence holds in HOBS when using weak equivalence $\approx$ (Proposition 33 below). To simplify the proof we split it into three parts.

Lemma 31. *Let* p_1, p_2, p_3 *be closed processes and* $p_1 \stackrel{p_2?}{\longrightarrow} p_1'$. *Then*

$$p_1 \lhd p_2 \lhd p_3 \approx p_1' \lhd p_3$$

Proof. We will show that the relation R defined below is a weak bisimulation. This relation also contains the two processes we wanted to show bisimilar.

$$\begin{aligned} R_0 &= \{(p_1 \lhd p_2 \lhd p_3, p_1' \lhd p_3) \mid \forall p_1, p_2, p_3 \in P_\varnothing \wedge p_1 \stackrel{p_2?}{\longrightarrow} p_1'\} \\ R_{i+1} &= \{(p \lhd r, q \lhd r) \mid \forall r \in P_\varnothing \wedge (p, q) \in R_i\} \\ R &= \bigcup_{i \in N} R_i \cup Id_p \end{aligned}$$

Let $p \mathcal{R} q$. This implies $p \mathcal{R}_i q$ for some i. If p or q make a receive transition, then we get $p' \mathcal{R}_{i+1} q'$ straight from the definition of $\mathcal{R}$. Since p and q can only transmit τ we only have to check two cases, which we do in the rest of the proof.

In case $p \stackrel{\tau!}{\longrightarrow} p'$ we know that

$$\begin{aligned} p' &\equiv p_1' \lhd q \lhd r_1 \lhd \cdots \lhd r_i \\ q &\equiv p_1' \lhd q \lhd r_1 \lhd \cdots \lhd r_i \end{aligned}$$

which automatically gives us the result $p' \mathcal{R} q$ since $\mathcal{R}$ contains the identity relation Id_p.

In case $q \stackrel{\tau!}{\longrightarrow} q'$, p can make two $\tau!$ transitions and similar to above we have

$$\begin{aligned} q' &\equiv q' \lhd r_1 \lhd \cdots \lhd r_i \\ p \stackrel{\tau!}{\longrightarrow}{}^2 p' &\equiv q' \lhd r_1 \lhd \cdots \lhd r_i \end{aligned}$$

□

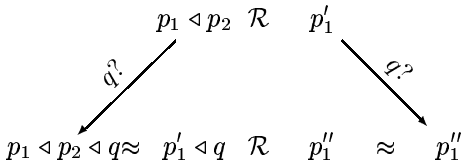
Lemma 32. *Let* p_1, p_2 *be closed processes and* $p_1 \stackrel{p_2?}{\longrightarrow} p_1'$. *Then*

$$p_1 \lhd p_2 \approx p_1'$$

Proof. We will show that the relation R defined below is a weak bisimulation up to $\approx$. This relation also contains the two processes we wanted to show bisimilar.

$$R = \{(p_1 \triangleleft p_2, p_1') \mid \forall p_1, p_2 \in P_\varnothing \ \wedge \ p_1 \xrightarrow{p_2?} p_1'\} \ \cup \ Id_p$$

We examine the single difficult case, other cases are trivial. Let $p_1 \triangleleft p_2 \,\mathcal{R}\, p_1'$ and let $p_1 \triangleleft p_2 \xrightarrow{q?} p_1 \triangleleft p_2 \triangleleft q$ then the diagram below shows how $\mathcal{R}$ satisfies the receive simulation condition up to $\approx$.



where we use the previous lemma to get $p_1 \triangleleft p_2 \triangleleft q \ \approx \ p_1' \triangleleft q$. □

Proposition 33. $(x?p_1) \triangleleft p_2 \ \approx \ p_1[p_2/x]$

Proof. Consequence of previous lemmas. □

6 Embeddings and Encodings

While a driving criterion in the design of HOBS is simplicity and resemblance to the Ethernet, one long term technical goal is to use this simple and uniform calculus to interpret other (more sophisticated) proposals for broadcasting calculi, such as $b\pi$-calculus [4] and CBS. In addition, interpretations in HOBS of hand-shake calculi, such as π-calculus, π-calculus with groups [2], and CCS, can bring insights to the expressive power of these different calculi.

6.1 Embedding of the λ-calculus

The syntax and semantics of the lazy λ-calculus is presented in Figure 3. The function $\cdot \longrightarrow \cdot \subseteq E \times E$ is the one-step reduction semantics. The translation of λ-expressions, which is basically a change of syntax, is also given in Figure 3.

What makes lazy λ-calculus an embedded calculus is: first, the restriction of HOBS syntax to variables, input and feed buffers is basically the λ-expressions syntax; second, four transition rules (Figure 2) for these three constructs can be compressed to exactly match the two rules of the lazy λ-calculus. Also, both notions of substitution directly correspond. Therefore, we can easily prove operational correspondence and adequacy, from which we can derive soundness, by using the fact that weak equivalence is a congruence and that the translation is compositional.

Lemma 34. *Let* $[\cdot := \cdot]$ *be the standard* λ*-calculus notion of substitution. Then*

$$[\![e_1[x := e_2]]\!]_\lambda \ \equiv \ [\![e_1]\!]_\lambda[[\![e_2]\!]_\lambda/x]$$

Syntax

$$\textit{Expressions}\ \ e \in E\ ::=\ x \mid \lambda x.e \mid e_1\ e_2$$

Semantics

$$\frac{}{(\lambda x.e_1)\ e_2 \longrightarrow e_1[x := e_2]}\,\beta \qquad \frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\,\mu$$

Embedding

$$\begin{array}{rcl} [\![x]\!]_\lambda & \equiv & x \\ [\![\lambda x.e]\!]_\lambda & \equiv & x?[\![e]\!]_\lambda \\ [\![e_1\ e_2]\!]_\lambda & \equiv & [\![e_1]\!]_\lambda \triangleleft [\![e_2]\!]_\lambda \end{array}$$

Fig. 3. Syntax, semantics and embedding of lazy λ-calculus.

Proof. By structural induction on e_1. □

Lemma 35.

$$[\![(\lambda x.e_1)e_2]\!]_\lambda \approx [\![e_1[x := e_2]]\!]_\lambda$$

Proof. Consequence of the previous lemma and Proposition 33. □

Proposition 36. *(Soundness) Let* $\simeq_\lambda$ *be the standard* λ*-calculus notion of observation equivalence. Then*

$$[\![e_1]\!]_\lambda \approx [\![e_2]\!]_\lambda \Rightarrow e_1 \simeq_\lambda e_2$$

Proof. Consequence of the previous lemma and the fact that weak bisimulation $\approx$ is a congruence. □

As is common for higher order calculi, syntax of HOBS does not include constructor for recursion. And, presented λ-calculus embedding justifies this. Just as in the λ-calculus there is a recursive Y combinator, HOBS can express recursion by derived constructor rec defined below. This recursive constructor has the expected behaviour as stated in the proposition that follows.

$$\begin{array}{rcl} \mathsf{W}_x(p) & \equiv & y?(x?p \triangleleft (y \triangleleft y)) \\ \mathsf{rec}\ x.p & \equiv & \mathsf{W}_x(p) \triangleleft \mathsf{W}_x(p) \end{array}$$

Proposition 37 (Correctness).

$$\mathsf{rec}\ x.p \approx p[\mathsf{rec}\ x.p/x]$$

Proof. Using Proposition 33 twice. □

Truth Values, Conditional Statement

$$\mathsf{T} \equiv t?f?t \qquad \mathsf{F} \equiv t?f?f$$

$$\mathsf{if}\ p_1\ \mathsf{then}\ p_2\ \mathsf{else}\ p_3 \equiv p_1 \triangleleft p_2 \triangleleft p_3 \qquad \mathsf{not}\ p \equiv t?f?(p \triangleleft f \triangleleft t) \qquad \mathsf{Eq}\ p_1\ p_2 \equiv p_1 \triangleleft p_2 \triangleleft (\mathsf{not}\ p_2)$$

Numerals

$$\overline{0} \equiv f?z?z \qquad \overline{n+1} \equiv f?z?(f \triangleleft (\overline{n} \triangleleft f \triangleleft z))$$

$$\mathsf{Succ}\ p \equiv f?z?(f \triangleleft (p \triangleleft f \triangleleft z)) \qquad \mathsf{isZero}\ p \equiv p \triangleleft (x?\mathsf{F}) \triangleleft \mathsf{T}$$

Pairs

$$(\mathsf{Pair}\ p_1\ p_2) \equiv t?(t \triangleleft p_1 \triangleleft p_2)$$

$$\mathsf{fst}\ p \equiv p \triangleleft \mathsf{T} \qquad \mathsf{snd}\ p \equiv p \triangleleft \mathsf{F}$$

Fig. 4. Data structures

6.2 Church Data Encodings

Since we have λ-calculus as a sub-calculus of HOBS we can use the standard λ-calculus encodings of data structures. We will use several data structures in the later sections so we present the Church style encoding of truth values together with conditional statement and some operations, and numerals and pairs together with some operations in Figure 4. We will also need equality for numerals and pairs, but these two function, both denoted as $\mathsf{Eq}\ p_1\ p_2$, are easily definable in terms of functions we have already defined and function $\mathsf{Pred}\ p$ which is also relatively simple.

6.3 Nil

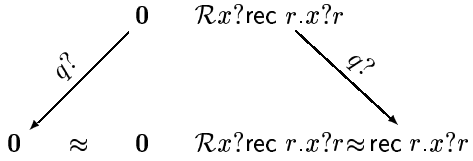
The following example shows that $\mathbf{0}$ is in some sense redundant process constructor since we can easily encode it using recursion. Its syntactic importance is that it marks deadlocked process that can be garbage-collected.

Proposition 38. *We have:* $\mathbf{0} \approx \mathsf{rec}\ r.x?r$

Proof. First note that $\mathsf{rec}\ r.x?r \approx x?\mathsf{rec}\ r.x?r$. Now it suffices to show that the following relation is a weak bisimulation up to $\approx$

$$R = \{(\mathbf{0}, x?\mathsf{rec}\ r.x?r)\}$$

Since both processes in the relation can only make a receive transition the following diagram shows all the cases



□

6.4 Buffer – Queue

$$\begin{aligned} \mathsf{buf} &\equiv \mathsf{rec}\, r.x?c?\langle y?(r \triangleleft x \triangleleft (c \triangleleft y)) + x!c\rangle \\ \mathsf{id} &\equiv \mathsf{rec}\, r.x?(\mathsf{buf} \triangleleft x \triangleleft r) \end{aligned}$$

Conjecture 39 (Correctness).

$$(1)\ p \lessapprox \mathsf{id} \frown p$$
$$(2)\ p \lessapprox p \frown \mathsf{id}$$

6.5 The Dot

We can construct a process which can simulate the dot construct from the older version of HOBS [11].

$$\mathsf{dot}\ p \equiv p_1 \frown (p_2 \frown p \frown p_3 \mid d) \frown p_4$$

$$\begin{aligned} In\ p &\equiv i?m?o?(i \triangleleft p) \\ Msg\ p &\equiv i?m?o?(m \triangleleft p) \\ Out\ p &\equiv i?m?o?(o \triangleleft p) \end{aligned}$$

$$\begin{aligned} p_1 &\equiv \mathsf{rec}\, r.x?(x \triangleleft (\_?\_?r) \triangleleft (\_?\_?r) \triangleleft \mathsf{buf} \triangleleft r) \\ p_2 &\equiv x?(\mathsf{buf} \triangleleft (Msg\ x) \triangleleft p_2') \\ p_2' &\equiv \mathsf{rec}\, r.x?(\mathsf{buf} \triangleleft (Out\ x) \triangleleft r) \\ p_3 &\equiv \mathsf{rec}\, r.x?(x \triangleleft (\_?\_?r) \triangleleft \mathsf{buf} \triangleleft (\_?\_?r) \triangleleft r) \\ p_4 &\equiv \mathsf{rec}\, r.x?(\mathsf{buf} \triangleleft (In\ x) \triangleleft r) \end{aligned}$$

$$\begin{aligned} d &\equiv d_1 \triangleleft d_2 \\ d_1 &\equiv \mathsf{rec}\, r.c?x?(x \triangleleft (x?(r \triangleleft (c \triangleleft (In\ x)))) \triangleleft (\_?c) \triangleleft (\_?r)) \\ d_2 &\equiv \mathsf{rec}\, r.(x?(x \triangleleft x?(\mathsf{buf} \triangleleft (Msg\ x) \triangleleft r) \triangleleft (\_?r) \triangleleft (\_?r))) \end{aligned}$$

Conjecture 40 (Correctness).

$$\bullet p \lessapprox \mathsf{dot}\ p$$

6.6 CBS Embedding

Having CBS as a precursor in the development of HOBS, it is natural to ask whether HOBS can interpret CBS. First, CBS assumes an underlying language with data types, which HOBS provides in the form of an embedded lazy λ-calculus, using the standard Church data encodings. Second, the CBS translator is the only construct that is not present in HOBS. To interpret it, we use a special parametrised queue construct. The queue construct together with the parameter (the translating function) is used as a one-way translator. Linking two of these to a process gives a CBS-style translator with decoupled translation.

Syntax

Names $x, y \in \mathcal{X} \;=\; A\ countable\ set\ of\ names$

Prefixes $\pi \in \Pi ::= x!y \mid x?y \mid x?*y$

Processes $P \in \mathcal{P} ::= \mathbf{0} \mid \pi.P \mid \langle x?y.P + x!y.P \rangle \mid P|P \mid (\nu x)P$

Fig. 5. Syntax of the π-calculus.

6.7 π-calculus encoding

Using Church numerals to encode channel names we can easily interpret the π-calculus without the new operator. Devising a sound encoding of the full π-calculus is more challenging, since there are several technical difficulties, for example explicit α-conversion, that have to be solved.

Figure 5 defines the syntax of the π-calculus. The intuitive semantics for the π-calculus constructs is as follows:

- $\mathbf{0}$ is a deadlocked process unable to send or receive.
- $x!y.P$ can send a name y on channel x.
- $x?y.P$ can receive a name on channel x and bind it in P
- $x?*y.P$ is a replicated input process which can receive a name on channel x and spawn a copy of process P in parallel with itself. The received name is bound in the spawned copy of P.
- $\langle x_1?y_1.P_1 + x_2!y_2.P_2 \rangle$ is a restricted choice process which can either receive a name on channel x_1, bind it in P_1 and become P_1 or it can send a name y_2 on channel x_2 an become P_2.
- $P_1|P_2$ is a parallel composition of two processes in which they can either act autonomously or they can communicate with each other.
- $(\nu x)P$ creates a new name x and binds it in P.

Remark 41. This is a slightly modified version of π-calculus, with restricted replication and choice constructors. It has been shown (for example [17]), that this restricted form of replication has the full power of normal replication. We present a translation of restricted choice, because it directly corresponds to our choice constructor. We expect that a translation of full choice is possible, but adding it would complicate the translation.

The encoding of the π-calculus is in Figure 6.

Conjecture 42 (Soundness). Let $\sim_\pi$ be the strong equivalence for the π-calculus processes and $\approx_{\pi\alpha}$ be the weak-bisimulation up to $\pi\alpha$-convertibility $=_{\pi\alpha}$. Then

$$\{\![P]\!\}_\pi \approx_{\pi\alpha} \{\![Q]\!\}_\pi \;\Rightarrow\; P \sim_\pi Q$$

1. Encoding of names

Let $|\cdot| \in \mathcal{X} \to N$ be the Gödel numbering of names.

3. Failure servers

$$
\begin{array}{rcl}
F_1(r,c) & \equiv & \langle \_?r \triangleleft (\mathsf{Succ}\ c) + \mathsf{Fail}!r \triangleleft (\mathsf{Succ}\ c)\rangle \\
F_2(m,r,c) & \equiv & \mathsf{case}\ m\ \mathsf{of} \\
 & & \quad (\mathsf{Msg}\ n\ v) \to F_1(r,c) \\
 & & \quad \mathsf{otherwise}\ \to r \triangleleft (\mathsf{Succ}\ c)
\end{array}
$$

4. Prefixes

$$
\begin{array}{rcl}
[\![x!y]\!]_\Pi(p,r,c) & \equiv & (\mathsf{Msg}\ x\ y)!f?\,\mathsf{case}\ f\ \mathsf{of} \\
 & & \qquad\qquad (\mathsf{Fail}) \to r \triangleleft (\mathsf{Succ}\ c) \\
 & & \qquad\qquad \mathsf{otherwise} \to p \triangleleft (\mathsf{Succ}\ c) \\
[\![x?y]\!]_\Pi(p,r,c) & \equiv & m?\,\mathsf{case}\ m\ \mathsf{of} \\
 & & \quad (\mathsf{Msg}\ n\ v) \to \mathsf{if}\ (\mathsf{Eq}\ n\ x)\ \mathsf{then} \\
 & & \qquad\qquad \langle \_?r \triangleleft (\mathsf{Succ}\ c) \\
 & & \qquad\qquad +m!((y?p) \triangleleft v \triangleleft (\mathsf{Succ}\ c))\rangle \\
 & & \qquad\quad \mathsf{else}\ F_1(r,c) \\
 & & \quad \mathsf{otherwise} \to r \triangleleft (\mathsf{Succ}\ c) \\
[\![x?*y]\!]_\Pi(p,r,c) & \equiv & m?\,\mathsf{case}\ m\ \mathsf{of} \\
 & & \quad (\mathsf{Msg}\ n\ v) \to \mathsf{if}\ (\mathsf{Eq}\ n\ x)\ \mathsf{then} \\
 & & \qquad\qquad \langle \_?r \triangleleft (\mathsf{Succ}\ c) \\
 & & \qquad\qquad +m!(((y?p) \triangleleft v \mid r) \triangleleft (\mathsf{Succ}\ c))\rangle \\
 & & \qquad\quad \mathsf{else}\ F_1(r,c) \\
 & & \quad \mathsf{otherwise} \to r \triangleleft (\mathsf{Succ}\ c)
\end{array}
$$

5. Sub-processes

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!]_\pi & \equiv & \mathbf{0} \\
[\![x!y.P]\!]_\pi & \equiv & \mathsf{rec}\ r.c?\,\langle m?F_2(m,r,c) \\
 & & \qquad\qquad +[\![x!y]\!]_\Pi([\![P]\!]_\pi,r,c)\rangle \\
[\![x?y.P]\!]_\pi & \equiv & \mathsf{rec}\ r.c?[\![x?y]\!]_\Pi([\![P]\!]_\pi,r,c) \\
[\![x?*y.P]\!]_\pi & \equiv & \mathsf{rec}\ r.c?[\![x?*y]\!]_\Pi([\![P]\!]_\pi,r,c) \\
[\![\langle x_1?y_1.P_1 + x_2!y_2.P_2\rangle]\!]_\pi & \equiv & \mathsf{rec}\ r.c?\,\langle [\![x_1?y_1]\!]_\Pi([\![P_1]\!]_\pi,r,c) \\
 & & \qquad\qquad +[\![x_2!y_2]\!]_\Pi([\![P_2]\!]_\pi,r,c)\rangle \\
[\![P_1|P_2]\!]_\pi & \equiv & [\![P_1]\!]_\pi \mid [\![P_2]\!]_\pi \\
[\![(\nu x)P]\!]_\pi & \equiv & \mathsf{rec}\ r.c?\,\langle m?F_2(m,r,c) \\
 & & \qquad\qquad +\mathsf{Fail}!(x?[\![P]\!]_\pi \triangleleft c \triangleleft (\mathsf{Succ}\ c))\rangle
\end{array}
$$

5. Process

$$
\{\!|P|\!\}_\pi \equiv [\![P]\!]_\pi \sigma \triangleleft (\mathsf{Succ}\ \overline{m})
$$

where $\sigma = \{\,[\,\overline{|x|}/x\,] \mid x \in \mathcal{X}\,\}$

$m = max\{\,|x| \mid x \in fn(P)\,\}$

Fig. 6. Encoding of the π-calculus.

7 Related Work

In this section we review works related to our basic design choices and the central proof technique used in the paper.

7.1 Alternatives Approaches to Modelling Dynamic Connectivity

One way to achieve dynamic broadcast architectures is to include messages that can change bridge behaviour; this would correspond to the transmission of channel names in the π-calculus [10]. Another is to let processes be transmitted, so that copies can be run elsewhere in the system. This makes the calculus *higher order*, like CHOCS [16]. This is the approach taken in this paper. A preliminary variant of HOBS sketched in [13] retains the underlying language of messages and functions. The resulting calculus seems to be unnecessarily complex, and having the underlying language seems redundant.

Processes are the only entities in HOBS, and are used to characterise bridges. (Since these can be broadcast, HOBS may also turn out to be able to mimic some features of the π-calculus). Arrival at a bridge and delivery across it happen in sequence. HOBS is thus free of the CBS insistence that these actions be simultaneous, but at the cost of less powerful synchronisation between subsystems.

To underline the independence in these choices, note that a quite conceivable design for (first order non-primitive) CBS is to characterise bridges by processes.

7.2 Related Calculi

The $b\pi$-calculus [4] can be seen as a version of the π-calculus with broadcast communication instead of point-to-point. In particular, the $b\pi$-calculus borrows the whole channel name machinery of the π-calculus, including the operator for creation of new names. Thus the $b\pi$-calculus does not model the Ethernet directly, and is not obviously a mobile version of CBS. Reusing ideas from the sketched π-calculus encoding can yield a simple $b\pi$-calculus encoding. Using links to model scopes of new names seems promising. Moreover, such an encoding might be compositional. Even though there are no published studies of equivalences in the $b\pi$-calculus, we expect that with an appropriate type system we can achieve a fully abstract encoding of the $b\pi$-calculus. The type system would be a mixture of Hindley/Milner and polymorphic π-calculus [17] type systems.

The Ambient calculus [3] is a calculus of mobile processes with computation based on a notion of movement. It is equipped with intra-ambient asynchronous communication similar to the asynchronous π-calculus. Since the choice of communication mechanism is independent from the mobility primitives, it may be interesting to study a broadcasting version of Ambient calculus. Also, broadcasting Ambient calculus might have simple encoding in HOBS.

Both HOBS and the join calculus [6] can be viewed as extensions of the λ-calculus. HOBS adds parallel composition and broadcast communication on top of the bare λ-calculus. The join calculus adds parallel composition and a parallel pattern on top of the λ-calculus with explicit let. The relationship between these two calculi remains to be studied.

7.3 Other Congruence Proofs

Ferreira *et al.* [5] use Howe's proof to show that weak bisimulation is a congruence for CML. They use late bisimulation. They leave open the question whether Howe's method can be applied to early bisimulation; this paper does not directly answer their question since late and early semantics and bisimulations coincide for HOBS. A proof for late semantics for HOBS is more elegant than the one here, and can be found in the extended version of the paper [12].

Thomsen proves congruence for CHOCS [16] by adapting the standard proof, but with non-well founded induction. That proof is in effect a similar proof to Howe's technique, but tailored specifically to CHOCS.

Sangiorgi [15] abandons higher order bisimulation for reasons specific to point-to-point communication with private channels, and uses context bisimulation where he adapts the standard proof. His proof is of similar difficulty to the proof presented here, especially in that the case of process application, involving substitution, is difficult.

Acknowledgements We thank Dave Sands for bringing Howe's method to our attention. We thank Jörgen Gustavsson, Martin Weichert and Gordon Pace for many discussions.

A Definitions

Definition 43 (Free Variables). *Let $\mathcal{P}_f(A)$ be the finite power set of any set A. The* free-variable *function $FV(\cdot) : P \to \mathcal{P}_f(X)$ is defined as:*

$$\begin{aligned} FV(\mathbf{0}) &= \{\} \\ FV(x) &= \{x\} \\ FV(x?p) &= FV(p) \setminus \{x\} \\ FV(p_1!p_2) &= FV(p_1) \cup FV(p_2) \\ FV(\langle x?p_1 + p_2!p_3 \rangle) &= FV(x?p_1) \cup FV(p_2!p_3) \\ FV(p_1|p_2) &= FV(p_1) \cup FV(p_2) \\ FV(p_1 \frown p_2) &= FV(p_1) \cup FV(p_2) \\ FV(p_1 \triangleleft p_2) &= FV(p_1) \cup FV(p_2) \end{aligned}$$

We say a term p is closed *when $FV(p) = \{\}$. We write $FV(p_1, \cdots, p_n)$ for $\bigcup_{i=1}^{n} FV(p_i)$.*

Definition 44 (Substitution). *The* substitution function $\cdot[\cdot/\cdot] : P \times P \times X \to P$ *is defined as:*

$$
\begin{array}{rcll}
(\mathbf{0})[p'/x] & \equiv & \mathbf{0} & \\
(x)[p'/x] & \equiv & p' & \\
(y)[p'/x] & \equiv & y & y \not\equiv x \\
(x?p)[p'/x] & \equiv & x?p & \\
(y?p)[p'/x] & \equiv & z?(p[z/y][p'/x]) & y \not\equiv x \text{ and} \\
 & & & z \notin FV(p, p', x, y) \\
(p_1!p_2)[p'/x] & \equiv & (p_1)[p'/x]!(p_2)[p'/x] & \\
(\langle x?p_1 + p_2!p_3\rangle)[p'/x] & \equiv & \langle (x?p_1)[p'/x] + (p_2!p_3)[p'/x]\rangle & \\
(p_1|p_2)[p'/x] & \equiv & (p_1)[p'/x]|(p_2)[p'/x] & \\
(p_1 \frown p_2)[p'/x] & \equiv & (p_1)[p'/x] \frown (p_2)[p'/x] & \\
(p_1 \triangleleft p_2)[p'/x] & \equiv & (p_1)[p'/x] \triangleleft (p_2)[p'/x] &
\end{array}
$$

We often write $(p)\sigma$ for parallel substitutions of several variables.

Definition 45 (Context filling). *The* context filling function $\cdot[\cdot] : C \times C \to C$ *is defined as:*

$$
\begin{array}{rcl}
([])[c'] & \equiv & c' \\
(\mathbf{0})[c'] & \equiv & \mathbf{0} \\
(x)[c'] & \equiv & x \\
(x?c)[c'] & \equiv & x?(c[c']) \\
(c_1!c_2)[c'] & \equiv & (c_1)[c']!(c_2)[c'] \\
(\langle x?c_1 + c_2!c_3\rangle)[c'] & \equiv & \langle (x?c_1)[c'] + (c_2!c_3)[c']\rangle \\
(c_1|c_2)[c'] & \equiv & (c_1)[c'/x]|(c_2)[c'] \\
(c_1 \frown c_2)[c'] & \equiv & (c_1)[c'] \frown (c_2)[c'] \\
(c_1 \triangleleft c_2)[c'] & \equiv & (c_1)[c'] \triangleleft (c_2)[c']
\end{array}
$$

B Congruence Proof for Late Semantics

In this section we present an alternative semantics and alternative bisimulation, and present a proof that this new formulation of the bisimulation is a congruence. Again, we use Howe's method for this proof. Also, we show that the two bisimulations we defined coincide, which gives us two relatively independent proofs of congruence.

The semantics defined in Section 2 uses early instantiation scheme, therefore in this section, we will refer to it as the early semantics.

Keeping the same syntax of HOBS we can define alternative semantics, which is defined in terms of a transition relation $\cdot \stackrel{\cdot}{\longmapsto} \cdot \subseteq P_\varnothing \times A_\varnothing \times P$. This so-called late semantics is presented in Figure 7.

Since the receive rules are almost the same, we can also show the same input properties of the late semantics.

Proposition 46 (Input enabling and determinism).

$$\forall p, x.\, \exists! p'.\, p \stackrel{x?}{\longmapsto} p'$$

	Receive	Transmit
$\forall p$	$p \stackrel{\tau?}{\longmapsto} p$	
Nil	$\mathbf{0} \stackrel{x?}{\longmapsto} \mathbf{0}$	
Input	$x?p_1 \stackrel{x?}{\longmapsto} p_1$	
Output	$p_1!p_2 \stackrel{x?}{\longmapsto} p_1!p_2$	$p_1!p_2 \stackrel{p_1!}{\longmapsto} p_2$
Choice	$\langle x?p_1 + p_2!p_3\rangle \stackrel{x?}{\longmapsto} p_1$	$\langle x?p_1 + p_2!p_3\rangle \stackrel{p_2!}{\longmapsto} p_3$
Compose	$\dfrac{p_1 \stackrel{x?}{\longmapsto} p_1' \quad p_2 \stackrel{x?}{\longmapsto} p_2'}{p_1\|p_2 \stackrel{x?}{\longmapsto} p_1'\|p_2'}$	$\dfrac{p_1 \stackrel{q!}{\longmapsto} p_1' \quad p_2 \stackrel{x?}{\longmapsto} p_2'}{p_1\|p_2 \stackrel{q!}{\longmapsto} p_1'\|p_2'[q/x]}$ $\quad$ $\dfrac{p_1 \stackrel{\tau!}{\longmapsto} p_1' \quad p_2 \stackrel{\tau?}{\longmapsto} p_2'}{p_1\|p_2 \stackrel{\tau!}{\longmapsto} p_1'\|p_2'}$ $\dfrac{p_1 \stackrel{x?}{\longmapsto} p_1' \quad p_2 \stackrel{q!}{\longmapsto} p_2'}{p_1\|p_2 \stackrel{q!}{\longmapsto} p_1'[q/x]\|p_2'}$ $\quad$ $\dfrac{p_1 \stackrel{\tau?}{\longmapsto} p_1' \quad p_2 \stackrel{\tau!}{\longmapsto} p_2'}{p_1\|p_2 \stackrel{p!}{\longmapsto} p_1'\|p_2'}$
Link	$\dfrac{p_2 \stackrel{x?}{\longmapsto} p_2'}{p_1 \frown p_2 \stackrel{x?}{\longmapsto} p_1 \frown p_2'}$	$\dfrac{p_1 \stackrel{m!}{\longmapsto} p_1'}{p_1 \frown p_2 \stackrel{m!}{\longmapsto} p_1' \frown p_2}$ $\dfrac{p_1 \stackrel{x?}{\longmapsto} p_1' \quad p_2 \stackrel{q!}{\longmapsto} p_2'}{p_1 \frown p_2 \stackrel{\tau!}{\longmapsto} p_1'[q/x] \frown p_2'}$ $\quad$ $\dfrac{p_1 \stackrel{\tau?}{\longmapsto} p_1' \quad p_2 \stackrel{\tau!}{\longmapsto} p_2'}{p_1 \frown p_2 \stackrel{\tau!}{\longmapsto} p_1' \frown p_2'}$
Feed	$f \stackrel{x?}{\longmapsto} f \triangleleft x$	$\dfrac{g_1 \stackrel{x?}{\longmapsto} p_1'}{g_1 \triangleleft p_2 \stackrel{\tau!}{\longmapsto} p_1'[p_2/x]}$ $\dfrac{f_1 \stackrel{\tau!}{\longmapsto} p_1'}{f_1 \triangleleft p_2 \stackrel{\tau!}{\longmapsto} p_1' \triangleleft p_2}$

Fig. 7. Late Semantics

The following proposition shows that the early and late semantics are very closely related.

Proposition 47 (Operational correspondence).

1. $\forall p, m.\, p \xrightarrow{m!} p' \iff p \overset{m!}{\longmapsto} p'$
2. $\forall p, m.\, p \xrightarrow{\tau?} p \iff p \overset{\tau?}{\longmapsto} p$
3. $\forall p_1, p_2.\, p_1 \xrightarrow{p_2?} p_1' \iff (p_1 \overset{x?}{\longmapsto} p_1'' \wedge p_1''[p_2/x] \equiv p_1')$

Naturally, we can define a late form of bisimulation, which is more natural for the late semantics. The following definition misses the input condition for the input of τ, but since there is only one general rule for all the processes this omission makes no difference to the relation but it simplifies the definition and the proofs as well.

Definition 48 (Late simulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is a* (strong, higher order) late simulation, *written $S_l(\mathcal{R})$ when*

$\forall (p, q) \in \mathcal{R}.$

1. $\forall x, p'.\, p \overset{x?}{\longmapsto} p' \;\Rightarrow\; (\exists q'.\, q \overset{x?}{\longmapsto} q' \;\wedge\; p'\, \mathcal{R}^\circ\, q'))$
2. $\forall m, p'.\, p \overset{m!}{\longmapsto} p' \;\Rightarrow\; (\exists n, q'.\, q \overset{n!}{\longmapsto} q' \;\wedge\; p'\, \mathcal{R}\, q' \;\wedge\; m\, \mathcal{R}_\tau\, n)$

Remark 49. Note the use of the open extension $\mathcal{R}^\circ$. If we expand the definition of the open extension this means, that

$$\forall r \in P_\varnothing.\, (p'[r/x])\, \mathcal{R}\, (q'[r/x])$$

Definition 50 (Late Bisimulation). *A relation $\mathcal{R} \subseteq P_\varnothing \times P_\varnothing$ on closed process terms is an* applicative bisimulation, *written $B_l(\mathcal{R})$, when*

$$\frac{S_l(\mathcal{R}) \quad S_l(\mathcal{R}^{-1})}{B_l(\mathcal{R})}$$

Definition 51 (Late Equivalence). *The* applicative equivalence *is a relation $\sim_l \subseteq P_\varnothing \times P_\varnothing$ defined as:*

$$\sim_l \;=\; \bigcup_{\mathcal{R} \subseteq P_\varnothing \times P_\varnothing} \{\mathcal{R} \mid B_l(\mathcal{R})\}$$

Proposition 52.

1. *$B_l(\sim_l)$*
2. *$\sim_l$ is an equivalence.*

The late and early systems are strongly connected as is shown in the next proposition.

Proposition 53. $\sim \;=\; \sim_l$

Proof. Using input enabling and input determinism from equation (1 and Proposition 46, and the operational correspondence from Proposition 47 we can see that the first clause of the simulation definitions is basically the same apart from the instantiation scheme. In addition, the fact that we left out the input of τ in the definition of late simulation does not matter since $\forall p.\, p \xrightarrow{\tau?} p \;\wedge\; p \overset{\tau?}{\longmapsto} p$

Using operational correspondence from Proposition 47 we can see that the second clause from the simulation definitions is the same in both cases. □

B.1 Congruence of $\sim_l$

Recall the subsection 4.2, in which we summed up the general results used in Howe's method and we also sketched the general proof strategy used in this method. In this subsection we apply Howe's method to show that $\sim_l$ is a congruence.

Since the whole subsection deals only with the late bisimulation we will write $\sim$ instead of $\sim_l$.

In this subsection we take the equivalence relation $\sim$ as the underlying relation $\mathcal{R}$ and we will work with the candidate relation $\sim^\bullet$. The goal of this subsection is: first to show that the candidate relation $\sim^\bullet$ coincides with the open extension $\sim^\circ$, that is $\sim^\bullet = \sim^\circ$; and second to use this fact as a key part of the congruence proof.

To start with the first part, we already have that $\sim^\circ \subseteq \sim^\bullet$ from Lemma 16 (Cand Sim). To show the converse we begin be proving that the closed restriction of the candidate relation $\sim^\bullet{}_\varnothing$ is a simulation. This requires to show that the following two simulation conditions hold

$$\begin{array}{l}\forall (p,q) \in P_\varnothing.\, p \sim^\bullet q \;\Rightarrow \\ 1.\; \forall x,p'.\, p \overset{x?}{\longmapsto} p' \;\Rightarrow\; (\exists q'.\, q \overset{x?}{\longmapsto} q' \;\wedge\; p' \sim^{\bullet\circ} q')) \\ 2.\; \forall m,p'.\, p \overset{m!}{\longmapsto} p' \;\Rightarrow\; (\exists n,q'.\, q \overset{n!}{\longmapsto} q' \;\wedge\; p' \sim^\bullet q' \;\wedge\; m \sim^\bullet{}_\tau n)\end{array}$$

Again, we spit the proof into two lemmas – Receive and Transmit lemmas – Lemma 54 and Lemma 55 below. First, we prove a strengthening of the first condition. Since these conditions have to hold only for closed processes, from Corollary 17 we know that $\sim^\bullet \subseteq \sim^\bullet{}_\varnothing{}^\circ$, which allows us to replace $\sim^\bullet{}_\varnothing{}^\circ$ with $\sim^\bullet$ in the first condition and obtain a stronger condition.

Lemma 54 (Receive). *Let $p, q \in P_\varnothing$ be two closed processes and $p \sim^\bullet q$. Then*

$$\forall x,p'.\, p \overset{x?}{\longmapsto} p' \;\Rightarrow\; (\exists q'.\, q \overset{x?}{\longmapsto} q' \;\wedge\; p' \;\sim^\bullet\; q')$$

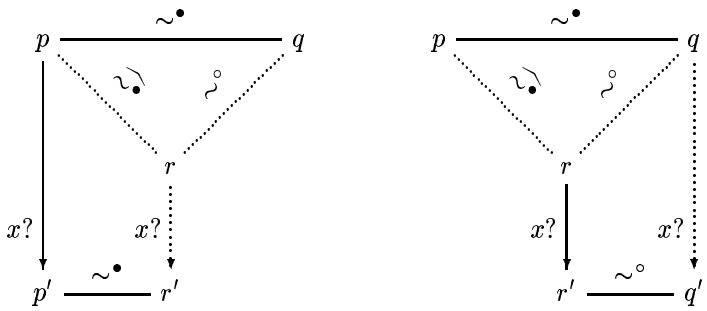
Proof. This proof has similar structure to proof of the Receive lemma in the case of the early transition system. First note that from Lemma 19 (Closed Middle) we have that

$$\exists r \in P_\varnothing.\, p \mathrel{\widehat{\sim^\bullet}} r \;\sim^\circ\; q$$

Also, from the definition of equivalence $\sim$ and the fact that r and q are closed processes we know that

$$\forall r'.\, r \xrightarrow{x?} r' \;\Rightarrow\; (\exists q'.\, q \xrightarrow{x?} q' \;\wedge\; r' \sim^{\circ} q') \tag{16}$$

Pictorially, we have described the situation represented by the right diagram below. In the rest of the proof we show that the statement represented by the left diagram below holds as well. Joining the two diagrams and using Lemma 16 (CAND RIGHT) to infer $p' \sim^{\bullet} q'$ gives us the result. In the diagram, normal lines, respectively dotted lines are universally, respectively existentially quantified.



To prove the statement in the left diagram we proceed by induction on the height of inference of transition $p \xrightarrow{x?} p'$. All the cases are very similar to proof of the Receive lemma in the case of the early transition system. The only significant difference is the case of Input. For late semantics, this case is simpler then for early semantics since it does not need the substitutivity of the candidate relation $\sim^{\bullet}$:

Input we know that $p \equiv x?p_1$. From the definition of the compatible refinement (COMP IN) we have that $r \equiv x?r_1$, and moreover $p_1 \sim^{\bullet} r_1$. Now, we can use Input receive rule on r to derive the appropriate transition and we get the result

$$\forall x, p' \equiv p_1.\, p \xrightarrow{x?} p' \;\Rightarrow\; (\exists r' \equiv r_1.\, r \xrightarrow{x?} r' \;\wedge\; p' \sim^{\bullet} r') \tag{17}$$

□

Lemma 55 (Transmit). *Let $p, q \in P_{\varnothing}$ be two closed process terms and let $p \sim^{\bullet} q$ hold. Then*

$$\forall m, p'.\, p \xrightarrow{m!} p' \;\Rightarrow\; (\exists n, q'.\, q \xrightarrow{n!} q' \;\wedge\; p' \sim^{\bullet} q' \;\wedge\; m \sim^{\bullet}_{\tau} n)$$

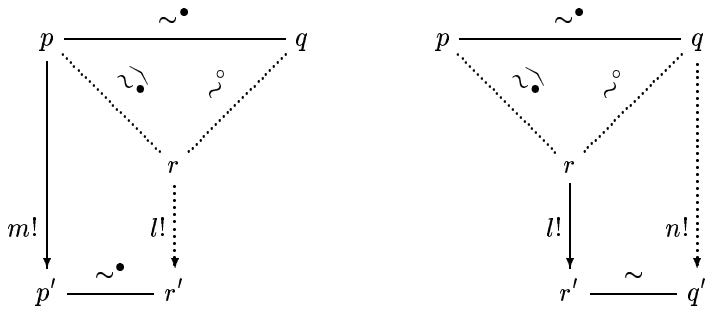
Proof. This proof has similar structure to proof of the Transmit lemma in the case of the early transition system. First note that from Lemma 19 (Closed Middle) we have that

$$\exists r \in P_{\varnothing}.\, p \,\widehat{\sim^{\bullet}}\, r \;\sim^{\circ} q$$

Also, from the definition of equivalence $\sim$ and the fact that r and q are closed processes we know that

$$\forall l, r'.\, r \xrightarrow{l!} r' \;\Rightarrow\; (\exists n, q'.\, q \xrightarrow{n!} q' \;\wedge\; r' \sim q' \;\wedge\; l \sim_\tau n) \tag{18}$$

Pictorially, we have described the situation represented by the right diagram below. In the rest of the proof we show that the statement represented by the left diagram below holds as well. Joining the two diagrams and using Lemma 16 (CAND RIGHT) to infer $p' \sim^\bullet q'$ gives us the result. In the diagram, normal lines, respectively dotted lines are universally, respectively existentially quantified.



To prove the statement in the left diagram we proceed by induction on the structure of p.

The first four cases, namely when p is Nil, Input, Output or Choice are basically the same as in the proof of the Transmit lemma for the early equivalence (Lemma 21). Also just as in the "early" Transmit Lemma the cases when p is Link or Feed are very similar to the case of the Composition. Therefore, we only show the most difficult and the most interesting case.

$p \equiv p_1|p_2$ from COMP COMP we have that $r \equiv r_1|r_2$ and $p_1 \sim^\bullet r_1$ and $p_2 \sim^\bullet r_2$. In fact, since parallel composition does not create new variable bindings, we know that p_i and r_i are closed processes. Now suppose that m is a process and a transmit transition was inferred in the following way:

$$\frac{p_1 \xmapsto{m!} p_1' \quad p_2 \xmapsto{x?} p_2'}{p_1|p_2 \xmapsto{m!} p_1'|p_2'[m/x]}$$

We can use the induction hypothesis on p_1 and we have that

$$\forall m, p_1'.\, p_1 \xmapsto{m!} p_1' \;\Rightarrow\; (\exists l, r_1'.\, r_1 \xmapsto{l!} r_1' \;\wedge\; p_1' \sim^\bullet r_1' \;\wedge\; m \sim^\bullet{}_\tau n)$$

Now, using the stronger condition of Receive Lemma (Lemma 54) on p_2 we have:

$$\forall x, p_2'.\, p_2 \xmapsto{x?} p_2' \;\Rightarrow\; (\exists r_2'.\, r_2 \xmapsto{x?} r_2' \;\wedge\; p_2' \sim^\bullet r_2')$$

Again, the stronger receive condition is essential in order the prove this composition case. Now, we can show that r can also make a transmit transition

and moreover

$$\begin{array}{l}\forall m, p' \equiv p'_1|p'_2[m/x].\\ p \xmapsto{m!} p' \;\Rightarrow\; (\exists l, r' \equiv r'_1|r'_2[l/x].\, r \xmapsto{l!} r' \;\wedge\; p' \sim^\bullet r' \;\wedge\; m \sim^\bullet_\tau l)\end{array} \tag{19}$$

where we obtain $p' \sim^\bullet r'$ by Cand Subst , Comp Comp and Cand Cong and the fact that all the involved processes are closed or become closed after substitution – in more detail:

$$\dfrac{\dfrac{p'_1 \sim^\bullet r'_1 \qquad \dfrac{p'_2 \sim^\bullet r'_2 \qquad \dfrac{m \sim^\bullet_\tau l}{m \sim^\bullet l}\ \textsc{MsgExt}}{p'_2[m/x] \sim^\bullet r'_2[l/x]}\ \textsc{Cand Subst}}{p'_1|p'_2[m/x] \mathrel{\widehat{\sim^\bullet}} r'_1|r'_2[l/x]}\ \textsc{Comp Comp}}{p' \;\equiv\; p'_1|p'_2[m/x] \sim^\bullet r'_1|r'_2[l/x] \;\equiv\; r'}$$

Using the definition of the message extension and the fact that all the involved processes are closed, we can infer $m \sim^\bullet_\tau n$ by and $p' \sim^\bullet q'$ by rule Cand Right .
The case of the symmetric rule is similar. The cases of the two remaining rules follow the same pattern and do not require the use of the Input Lemma since a process does not change upon reception of τ.

□

Proposition 56. *$\sim^\bullet$ is a simulation.*

Proof. Consequence of Lemma 54 (Receive) and Lemma 55 (Transmit). □

Proposition 57. *$\sim^{\bullet *}$ is a bisimulation.*

Proof. Consequence of Proposition 56 and Lemma 18
The argument is the same as for the early congruence candidate in Proposition 23. □

Proposition 58. *$\sim^\circ$ is a congruence.*

Proof. Consequence of Proposition 57 and Lemma 12.
The argument is the same as for the early congruence candidate in Proposition 24. □

C Link-Feed Interference

Using the rule $\dfrac{p_1 \xrightarrow{\tau!} p'_1}{p_1 \triangleleft p_2 \xrightarrow{\tau!} p'_1 \triangleleft p_2}$ instead of $\dfrac{f_1 \xrightarrow{\tau!} p'_1}{f_1 \triangleleft p_2 \xrightarrow{\tau!} p'_1 \triangleleft p_2}$ we get:

Proposition 59. *We do not have:*

1. $(x?p_1) \triangleleft p_2 \;\approx\; p_1[p_2/x]$

Proof (Counter Example).

$$\begin{aligned} p_1 &= (x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0}) \\ p_2 &= \text{any } p \end{aligned}$$

To see that $(x?p_1) \triangleleft p_2 \not\approx p_1[p_2/x]$ we can show the following:

$$(x?p_1) \triangleleft p_2 \xrightarrow{p_3?} (x?p_1) \triangleleft p_2 \triangleleft p_3$$

Now $p_1[p_2/x] \equiv p_1$ has to make a weak transition $p_1 \stackrel{p_3?}{\Longrightarrow} \cdot$ to arrive at a weakly bisimilar state. By looking at the definition of the transition relation we have only two possible weak transitions:

1. $p_1 \stackrel{p_3?}{\Longrightarrow} p_1$
2. $p_1 \stackrel{p_3?}{\Longrightarrow} (x?x!\mathbf{0})\frown(p_3!\mathbf{0})$

But, if we take the first possible transition, then

$$(x?p_1) \triangleleft p_2 \triangleleft p_3 \not\approx p_1$$

since

$$\begin{aligned} & (x?p_1) \triangleleft p_2 \triangleleft p_3 \\ \equiv\ & (x?((x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0}))) \triangleleft p_2 \triangleleft p_3 \\ \xrightarrow{\tau!}\ & ((x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0})) \triangleleft p_3 \\ \xrightarrow{\tau!}\ & ((x?x!\mathbf{0})\frown(x?x!\mathbf{0})) \triangleleft p_3 \\ \xrightarrow{\tau!}\ & (x?x!\mathbf{0})\frown(p_3!\mathbf{0}) \end{aligned}$$

and p_1 cannot mimic these transitions since for p_1 we only have:

$$p_1 \xrightarrow{\tau!} (x?x!\mathbf{0})\frown(x?x!\mathbf{0})$$

where the last term is not capable of any further $\tau!$ transitions. In case we try the second possible transition, then again

$$(x?p_1) \triangleleft p_2 \triangleleft p_3 \not\approx (x?x!\mathbf{0})\frown(p_3!\mathbf{0})$$

since

$$\begin{aligned} & (x?p_1) \triangleleft p_2 \triangleleft p_3 \\ \equiv\ & (x?((x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0}))) \triangleleft p_2 \triangleleft p_3 \\ \xrightarrow{\tau!}\ & ((x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0})) \triangleleft p_3 \\ \xrightarrow{\tau!}\ & (x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0}) \end{aligned}$$

and $(x?x!\mathbf{0})\frown(p_3!\mathbf{0})$ cannot mimic these transitions since we only have:

$$(x?x!\mathbf{0})\frown(p_3!\mathbf{0}) \xrightarrow{\tau!} (p_3!\mathbf{0})\frown\mathbf{0}$$

where clearly $(p_3!\mathbf{0})\frown\mathbf{0}$ is not weakly bisimilar to $(x?x?x!\mathbf{0})\frown(\mathbf{0}!x?x!\mathbf{0})$. □

D Abstract Machine

Figure 8 presents another possible semantics. What can be called abstract machine is a pair of relations: reduction relation $\cdot \longrightarrow \cdot \subseteq P_\varnothing \times P_\varnothing$ and a transmit relation $\cdot \xrightarrow{\cdot} \cdot \subseteq P_\varnothing \times P_\varnothing \times P_\varnothing$. We leave the study of the relationship between the labelled transition semantics and the abstract machine semantics for future work.

Abstract Machine

Axioms	$\mathbf{0} \triangleleft p \longrightarrow \mathbf{0}$ $x?p_1 \triangleleft p \longrightarrow p_1[p/x]$ $p_1!p_2 \triangleleft p \longrightarrow p_1!p_2$ $\langle x?p_1 + p_2!p_3\rangle \triangleleft p \longrightarrow p_1[p/x]$ $(p_1 \mid p_2) \triangleleft p \longrightarrow (p_1 \triangleleft p) \mid (p_2 \triangleleft p)$ $(p_1 \frown p_2) \triangleleft p \longrightarrow p_1 \frown (p_2 \triangleleft p)$
Inference Rules	$\dfrac{p_1 \longrightarrow p_1'}{p_1 \triangleleft p_2 \longrightarrow p_1' \triangleleft p_2}$ $\dfrac{p_1 \longrightarrow p_1'}{p_1 \mid p_2 \longrightarrow p_1' \mid p_2} \quad \dfrac{p_2 \longrightarrow p_2'}{p_1 \mid p_2 \longrightarrow p_1 \mid p_2'}$ $\dfrac{p_1 \longrightarrow p_1'}{p_1 \frown p_2 \longrightarrow p_1' \frown p_2} \quad \dfrac{p_2 \longrightarrow p_2'}{p_1 \frown p_2 \longrightarrow p_1 \frown p_2'}$
Axioms	$p_1!p_2 \xrightarrow{p_1} p_2$ $\langle x?p_1 + p_2!p_3\rangle \xrightarrow{p_2} p_3$
Inference Rules	$\dfrac{p_1 \xrightarrow{p} p_1'}{p_1 \mid p_2 \xrightarrow{p} p_1' \mid (p_2 \triangleleft p)} \quad \dfrac{p_2 \xrightarrow{p} p_2'}{p_1 \mid p_2 \xrightarrow{p} (p_1 \triangleleft p) \mid p_2'}$ $\dfrac{p_1 \xrightarrow{p} p_1'}{p_1 \frown p_2 \xrightarrow{p} p_1' \frown p_2} \quad \dfrac{p_2 \xrightarrow{p} p_2'}{p_1 \frown p_2 \xrightarrow{\tau} (p_1 \triangleleft p) \frown p_2'}$

Fig. 8. Abstract Machine

References

1. Samson Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research topics in Functional Programming*. Addison-Wesley, 1990.
2. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. In Catuscia Palamidessi, editor, *CONCUR 2000*, volume 1877 of *LNCS*, University Park, PA, USA, August 2000. Springer.
3. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, Jun 2000.
4. Cristian Ene and Traian Muntean. Expressivness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory*, volume 1684 of *LNCS*, September 1999.
5. William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core cml. *Journal of Functional Programming*, 8(5):447–491, 1998.
6. Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385. ACM, January 1996.
7. Andrew D. Gordon. Bisimilarity as a theory of functional programming: minicourse. Notes Series BRICS-NS-95-3, BRICS, Department of Computer Science, University of Aarhus, July 1995.
8. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1 February 1996.
9. Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.
10. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
11. Karol Ostrovský, K. V. S. Prasad, and Walid Taha. A preliminary account of Primitive HOBS: A calculus for higher order broadcasting. In *Proceedings of Winter Meeting*, January 2000. Deptartment of Computing Science, Chalmers University of Technology.
12. Karol Ostrovský, K. V. S. Prasad, and Walid Taha. Towards a primitive higher order calculus of broadcasting systems, October 2000. extended version; http://www.cs.chalmers.se/~karol/Papers/, http://www.cs.chalmers.se/~prasad/.
13. K. V. S. Prasad. Status report on ongoing work: Higher order broadcasting systems and reasoning about broadcasts. Unpublished manuscript, September 1994.
14. K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.
15. Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.
16. Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, January 1993.
17. David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis CST–126–96, Dept. of Computer Science, University of Edinburgh, 1996.