# Towards a Primitive Higher Order Calculus of Broadcasting Systems

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 fundamentaly 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 $\lambda$-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) [13]. 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.

- The receiver has to consume whatever is on the net, at any time.
- Only one message can be transmitted at a time.
- Collision detection and resolution are provided by the protocol, 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 will look more closely at how systems such as the Ethernet work.

### 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 that are 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. Therefore, the function pairs must be expressed some computationally rich underlying language.
2. CBS only provides a *static* model of Ethernet architectures. For example, real bridges can change their routing behaviour. CBS provides neither for this 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 constructors 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, this is not useful in the higher order setting. When messages involve processes (non-first order values), such a definition is too discriminating. 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, standard technique for proving that bisimulation is a congruence does not go through. A key difficulty seems to be that standard technique cannot be used directly to show that the substitution of equivalent processes for variables preserves equivalence (This problem 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 $\lambda$-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 $\pi$-calculus [10] are discussed briefly (Full details can be found in the extended version of the paper [11]).

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 [15] and CBS: Roughly, HOBS is to CBS what CHOCS is to CCS.

An extended version of the paper available online [11] gives details of all definitions, proofs and further development.

## 2 Syntax and Semantics

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

- $\mathbf{0}$ 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 $\lambda$-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 [13].

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 [13], 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 $\alpha$-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 (see [11] for details).

The rest of this section discusses some semantics issues. For further discussion and other design issues see the extended version of the paper [11].

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 $p \xrightarrow{m?} p'$

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

**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* | $c \in C ::= [] \mid \mathbf{0} \mid x \mid x?c \mid c!c \mid \langle x?c + c!c \rangle \mid c \mid c \mid c^\frown c \mid c \triangleleft c$ | | |

**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

## 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*

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

**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 | $\dfrac{p_1 \xrightarrow{q?} p_1' \quad p_2 \xrightarrow{q?} p_2'}{p_1 \mid p_2 \xrightarrow{q?} p_1' \mid p_2'}$ | $\dfrac{p_1 \xrightarrow{m!} p_1' \quad p_2 \xrightarrow{m?} p_2'}{p_1 \mid p_2 \xrightarrow{m!} p_1' \mid p_2'}$ <br> $\dfrac{p_1 \xrightarrow{m?} p_1' \quad p_2 \xrightarrow{m!} p_2'}{p_1 \mid p_2 \xrightarrow{m!} p_1' \mid p_2'}$ |
| Link | $\dfrac{p_2 \xrightarrow{q?} p_2'}{p_1 \frown p_2 \xrightarrow{q?} p_1 \frown p_2'}$ | $\dfrac{p_1 \xrightarrow{m!} p_1'}{p_1 \frown p_2 \xrightarrow{m!} p_1' \frown p_2}$ <br> $\dfrac{p_1 \xrightarrow{m?} p_1' \quad p_2 \xrightarrow{m!} p_2'}{p_1 \frown p_2 \xrightarrow{\tau!} p_1' \frown p_2'}$ |
| Feed | $f \xrightarrow{q?} f \triangleleft q$ | $\dfrac{g_1 \xrightarrow{p_2?} p_1'}{g_1 \triangleleft p_2 \xrightarrow{\tau!} p_1'}$ <br> $\dfrac{f_1 \xrightarrow{\tau!} p_1'}{f_1 \triangleleft p_2 \xrightarrow{\tau!} p_1' \triangleleft p_2}$ |

**Fig. 2.** Semantics

Adaptation of Thomsen's definition of applicative higher order simulation [15] 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 $\lambda$-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,*

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

Using standard techniques, we can show that the identity relation on closed processes is a simulation and a bisimulation. The property of being a simulation, a bisimulation respectively is preserved by relational composition and union. Also, the bisimulation property is preserved by converse.

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 4 (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 5.**

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

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

**Proposition 6.** *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. $\langle x?p_1 + p_2!p_3\rangle|x?p_4 \sim \langle x?(p_1|p_4) + p_2!(p_3|p_4[p_2/x])\rangle$

## 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 7 (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 8 (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*

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

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

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

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

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

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 $\sim$ 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*

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

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 [15, 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 (cf. [11]). Extension of a relation to open terms is also the standard one:

**Definition 11 (Open Extension).** *Let $\mathcal{R} \subseteq P \times P$ be a relation on process terms. Its* open extension *is defined by the following rule*

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

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

**Definition 12 (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*

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

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 13.** *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*

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

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

**Corollary 14.** *We have $\mathcal{R}^\bullet \subseteq {\mathcal{R}^\bullet}_\varnothing{}^\circ$ as an immediate consequence of rule* Cand Subst *and rule* Cand Ref.

**Lemma 15.** *Let $\mathcal{R} \subseteq P \times P$ be an equivalence relation. Then $\mathcal{R}^{\bullet *}$ is symmetric.*

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 16 (Closed Middle).** $\forall p, q \in P_\varnothing . p\,\mathcal{R}^\bullet\,q \;\Rightarrow\; \exists r \in P_\varnothing . p\,\widehat{\mathcal{R}^\bullet}\,r\,\mathcal{R}^\circ\,q$

### 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 13 (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 two simulation conditions of Definition 2 hold.

We split the proof into two lemmas: Lemma 17 (Receive) and Lemma 18 (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 17 (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.* Relatively straightforward proof by induction on the height of inference of transition $p \xrightarrow{m?} p'$. The only interesting case is the case of rule $x?p_1 \xrightarrow{q?} p_1[q/x]$, which makes use of the substitutivity of the candidate relation $\sim^\bullet$. □

**Lemma 18 (Transmit).** *Let $p, q \in P_\varnothing$ be two closed processes and $p \sim^\bullet q$. Then*

$$\forall m, p'.\, p \xrightarrow{m!} p' \;\Rightarrow\; (\exists n, q'.\, q \xrightarrow{n!} q' \;\wedge\; p' \sim^\bullet q' \;\wedge\; m \sim^\bullet_\tau n) \tag{2}$$

*Proof.* First note that from Lemma 16 (Closed Middle) we have that

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

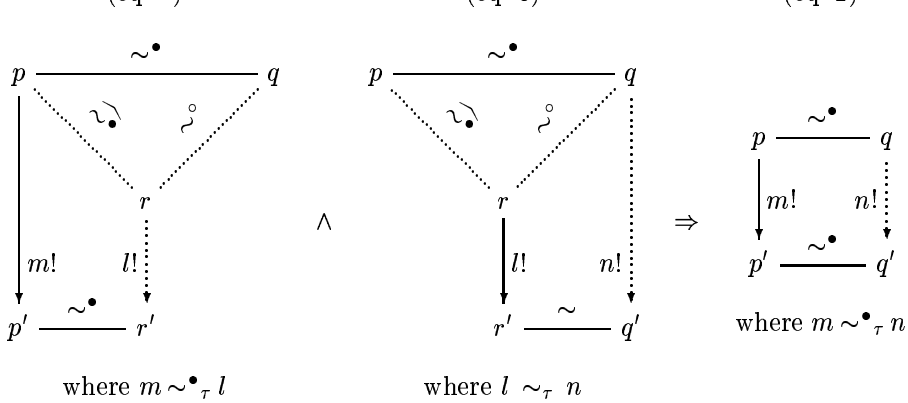
Also, from the definition of equivalence $\sim$ and the fact that $r$ and $q$ are closed processes we know that

$$\forall l, r'.\, r \xrightarrow{l!} r' \;\Rightarrow\; (\exists n, q'.\, q \xrightarrow{n!} q' \;\wedge\; r' \sim q' \;\wedge\; l \sim_\tau n) \tag{3}$$

Is remains to prove the following statement:

$$\forall m, p'.\, p \xrightarrow{m!} p' \;\Rightarrow\; (\exists l, r'.\, r \xrightarrow{l!} r' \;\wedge\; p' \sim r' \;\wedge\; m \sim_\tau l) \tag{4}$$

Joining statements (4) and (3), and and using Lemma 13 (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. 4) we proceed by induction on the structure of $p$. We only describe the most difficult case – parallel composition.

**$p \equiv p_1|p_2$** we have four related sub-processes: $p_1 \sim^\bullet r_1$ and $p_2 \sim^\bullet r_2$. Now suppose that $m$ is a process, and that $p$ made the following transition:

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

Since the candidate relation $\sim^\bullet$ contains its compatible refinement, it is enough to show that each sub-process of $r$ can mimic the corresponding sub-process of $p$. For $r_1$, using the induction hypothesis we get that $r_1 \xrightarrow{l!} r_1'$ and $p_1' \sim^\bullet r_1'$, $m \sim^\bullet_\tau l$. For $r_2$, if we would use only the simulationcondition,

we would get $r_2 \xrightarrow{m?} r_2'$, and this would not allow us to show that $r$ has a corresponding transition, since the inference of a transition requires both labels to be the same. At this point, we can use the stronger receive condition of Lemma 17 and we get precisely what we need, that is: $r_2 \xrightarrow{l?} r_2'$ and $p_2' \sim^\bullet r_2'$. The symmetric case and the case when $m$ is $\tau$ are similar.

□

Having the two above lemmas we have proved that the restriction of the candidate relation $\sim^\bullet{}_\varnothing$ is a simulation. Also, using Lemma 15 we get that $\sim^\bullet{}_\varnothing{}^*$ is symmetric, which means that it is a bisimulation. To conclude this section, we are now ready to state and prove the main proposition.

**Proposition 19.** *$\sim^\circ$ is a congruence.*

*Proof.* First we show that $\sim^\circ$ coincides with the candidate relation $\sim^\bullet$. Note that from Lemma 13 (CAND SIM) we know that $\sim^\circ \subseteq \sim^\bullet$. To prove the converse, from the two above lemmas and Lemma 15 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 14 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 9 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 $\beta$ equivalent in the $\lambda$-calculus:

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

Equipped with the weak transition relation $\cdot \stackrel{\cdot}{\Longrightarrow} \cdot$ (Definition 20) we define the *weak simulation* and *weak bisimulation* in the standard way. Weak equivalence $\approx$ is also defined in the standard way as a union of all weak bisimulation relations. In the same way as for the strong equivalence we prove that $\approx$ is a weak bisimulation and that it is an equivalence (Proposition 21). Moreover, the technique used in proving that strong equivalence is a congruence works also for the weak equivalence (Proposition 22).

**Definition 20 (Weak Transition).** *Let $\stackrel{\epsilon}{\Longrightarrow}$ be the reflexive transitive closure of $\xrightarrow{\tau!}$. Then the weak transition is defined as*

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

**Proposition 21.**

1. *$\approx$ is a weak bisimulation*
2. *$\approx$ is an equivalence.*

**Proposition 22.** $\approx$ *is a congruence.*

*Proof.* The proof follows that of Proposition 19. 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 $\tau$ without any change to the process itself (see rule Silence in Figure 2). □

Now we can prove that $\beta$ equivalence holds in HOBS when using weak equivalence $\approx$. A simple proof of this proposition using weak bisimulation up to $\approx$ technique, and can be found in the extended version of the paper [11].

**Proposition 23.** $(x?p_1) \triangleleft p_2 \approx p_1[p_2/x]$

## 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 $\pi$-calculus, $\pi$-calculus with groups [2], and CCS, can bring insights to the expressive power of these different calculi.

While, in this section, we mainly concentrate on the lazy $\lambda$-calculus embedding and its consequences we also briefly cover possible encodings of CBS and the $\pi$-calculus.

### 6.1 λ-calculus embedding

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

What makes lazy $\lambda$-calculus an embedded calculus is: first, the restriction of HOBS syntax to variables, input and feed buffers is basically the $\lambda$-expressions syntax; second, four transition rules (Figure 2) for these three constructs can be compressed to exactly match the two rules of the lazy $\lambda$-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.

**Proposition 24.** *(Soundness) Let* $\simeq_\lambda$ *be the standard* $\lambda$*-calculus notion of observation equivalence. Then*

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

As is common for higher order calculi, syntax of HOBS does not include constructor for recursion. And, presented $\lambda$-calculus embedding justifies this. Just as in the $\lambda$-calculus there is a recursive $Y$ combinator, HOBS can expressrecursion

**Syntax**

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

**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 $\lambda$-calculus.

by derived constructor rec defined on the left below. This recursive constructor has the expected behaviour as stated on the right below.

$$\begin{array}{rcl} \mathsf{W}_x(p) & \equiv & y?(x?p \triangleleft (y \triangleleft y)) \\ \mathsf{rec}\ x.p & \equiv & \mathsf{W}_x(p) \triangleleft \mathsf{W}_x(p) \end{array} \qquad\qquad \mathsf{rec}\ x.p \;\approx\; p[\mathsf{rec}\ x.p/x]$$

### 6.2 Concurrent Calculi

Having CBS as a precursor in the development of HOBS, it is natural to ask whether HOBS can interpret CBS. First, CBS assumed an underlying language with data types, which HOBS provides in form of embedded lazy $\lambda$-calculus, using standard Church data encodings. Second, all the process constructors in HOBS, apart from link and feed, have the same syntax and semantics as in CBS. It remains to show the encoding of the CBS translator constructor. For this 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.

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

For further discussion and proposed solutions see the extended version of the paper [11]. The soundness proof for these encodings is ongoing work.

## 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 chan-

nel names in the $\pi$-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 [15]. This is the approach taken in this paper. A preliminary variant of HOBS sketched in [12] 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 $\pi$-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 $\pi$-calculus with broadcast communication instead of handshake. In particular, the $b\pi$-calculus borrows the whole channel name machinery of the $\pi$-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 $\pi$-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 appropriate type system we can achieve a fully abstract encoding of $b\pi$-calculus. The type system would be a mixture of Hindley/Milner and polymorphic $\pi$-calculus [16] 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 $\pi$-calculus. Since the choice of communication mechanism is independent from the mobility primitives, it might be interesting to study a broadcasting version of Ambient calculus. Also, broadcasting Ambient calculus might have simpler encoding in HOBS.

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

### 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 ask whether the method can be applied to early bisimulation; this paper does not really 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 [11].

Thomsen proves congruence for CHOCS [15] 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 [14] 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, especially, the case of process application, involving substitution, is difficult.

## 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 $\pi$-Calculus*. Cambridge University Press, 1999.
11. 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/.
12. K. V. S. Prasad. Status report on ongoing work: Higher order broadcasting systems and reasoning about broadcasts. Unpublished manuscript, September 1994.
13. K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.
14. 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.
15. Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, January 1993.
16. David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis CST–126–96, Dept. of Computer Science, University of Edinburgh, 1996.