From Higher-Order π-Calculus to π-Calculus in the Presence of Static Operators

José-Luis Vivas[1]* and Mads Dam[2]**

[1] Dept. of Teleinformatics, Royal Institute of Technology, Stockholm
[2] SICS, Swedish Institute of Computer Science, Stockholm

Abstract. Some applications of higher-order processes require better control of communication capabilities than what is provided by the π-calculus primitives. In particular we have found the dynamic restriction operator of CHOCS, here called blocking, useful. We investigate the consequences of adding static operators such as blocking to the first- and higher-order π-calculus. In the presence of the blocking operator (and static operators in general) the higher-order reduction of Sangiorgi, used to demonstrate the reducibility of higher-order communication features to first-order ones, breaks down. We show, as our main result, that the higher-order reduction can be regained, using an approach by which higher-order communications are replaced, roughly, by the transmission and dynamic interpretation of syntax trees. However, the reduction is very indirect, and not usable in practice. This throws new light on the position that higher-order features in the π-calculus are superfluous and not needed in practice.

1 Introduction

One of the most significant contributions of the π-calculus has been the demonstration that higher-order features in concurrency can be eliminated in favour of first-order ones by means of channel name generation and communication. This issue has been extensively studied in the context of lambda-calculus under various evaluation regimes (cf. [3]), and in his thesis [8] Sangiorgi explored in depth the reduction of higher-order processes to first-order ones. Instead of communicating a higher-order object, a local copy is created, protected by a trigger in the shape of a newly generated channel name. This trigger can then be communicated in place of the higher-order object itself. On the basis of this sort of reduction it has been argued (cf. [8]) that, in the context of the π-calculus, higher-order features are matters of convenience only: No essential descriptive or analytical power is added by the higher-order features.

In this paper we reexamine this position, and find it borne out in principle, but not in practice. We argue the following points:

* Supported by the Swedish National Board for Technical and Industrial Development (NUTEK) under grant no. 94-06164. Email: josev@it.kth.se

** Supported by a Swedish Foundation for Strategic Research Junior Individual Grant. Email: mfd@sics.se

1. Practical applications call for process combinators other than those provided by the basic π-calculus. Specifically we consider the dynamic restriction, or blocking[1] operator of Thomsen's CHOCS [9].
2. Adding blocking to the higher-order π-calculus causes Sangiorgi's reduction to break down.
3. In the presence of blocking it remains possible to reduce the higher-order calculus to the first-order one, even in a compositional manner.
4. The reduction, however, is complicated, and amounts in effect to the communication and interpretation of parse trees. In contrast to Sangiorgi's reduction which is conceptually quite simple this reduction can not be used in practice to reduce non-trivial arguments concerning higher-order processes to arguments concerning first-order ones.

Our reduction is very general and can be applied to a wide range of static process combinators. Our specific interest in the blocking operator stems from some difficulties connected with the representation of cryptographic protocols in the higher-order π-calculus [2].

Application: Cryptographic Protocols Consider a higher-order process of the shape $A = \overline{a}m.A$. The process A is an object which repeatedly outputs m along a to whomever possesses knowledge of a and is willing to listen. In principle m can be any sort of higher-order object, but here it suffices to think of m as a message carried by a. A cryptographic analogy of A is thus the object $\{m\}_a$, m encrypted by the (shared) encryption key a. We might very well want to communicate A as a higher-order object over some other, possibly insecure, channel `xfer`. The sender would simply pass A along `xfer`, and the receiver would first receive some process object X, then immediately activate X while in parallel trying to extract the message m through a. That is, the receiver would have the shape $\texttt{xfer}(X).(X \mid a(y).B(y))$ where $B(y)$ is the continuation processing the extracted y in some suitable way. Observe that we can assume receivers and senders to execute in an environment containing other receivers and senders, along with unknown and possibly hostile intruders.

Here we encounter a first difficulty: We have provided no guarantee that it is really X and $B(y)$ which communicate along a and not some other process which is trying to decrypt using a by accident or because of some protocol flaw. That is, decryption is insecure, contradicting commonly held assumptions in the analysis of key management protocols. When encryption is nested, however, the problem is aggravated. A higher-order representation of $\{\{m\}_a\}_b$ is the object $A' = \overline{b}A.A'$. Extraction of m from A' would follow the pattern

$$\texttt{xfer}(X).(X \mid b(Y).(Y \mid a(z).B(z))).$$

[1] Since it is not completely clear in which senses the restriction operators are really static or dynamic we prefer a more neutral terminology and use "restriction" for the π-calculus restriction operator and "blocking" for the CHOCS dynamic restriction operator.

Now, after extraction of A along b the ensuing process configuration $A' \mid A \mid a(z).B(z)$ has made it possible for an intruder to "snatch" m from A on the basis of knowing a, without necessarily knowing b beforehand. This is clearly unreasonable.

Observe that the π-calculus restriction does not provide an obvious remedy. If we were to replace a receiver of the shape $\mathtt{xfer}(X).(X \mid a(y).B(y))$ by one of the shape $\mathtt{xfer}(X).\nu a.(X \mid a(y).B(y))$ to protect decryption, alpha-conversion would apply to prevent a's in X and a's guarding $B(y)$ from being identified.

Another possible alternative is to replace the encrypted message $A = \overline{a}m.A$ by the abstraction $(\lambda a)A$, in order to allow the parameter a to be supplied locally. This, however, does not work, as a hostile receiver can then decrypt this message at will by appropriately instantiating a.

Blocking What is called for is the blocking operator $P\backslash a$ which blocks a without binding it. This provides a kind of firewall preventing communication along the channel a between P and its environment, akin to the CCS restriction operator. It allows $P\backslash a \xrightarrow{\alpha} P'\backslash a$ only if $P \xrightarrow{\alpha} P'$ and a is not the channel on which synchronization of the action α takes place. Thus we can account for reception using a process of the shape $\mathtt{xfer}(X).((X \mid a(y).B(y))\backslash a)$.

We believe quite strongly that the issue of "localized control" which we raise is far from an artificial one. Quite on the contrary, as code transmission capabilities move from the realm of operating systems to become important programming paradigms, issues pertaining to dynamic resource protection and control are getting ever more important.

Higher-Order Reduction In this paper we investigate the consequences of adding the blocking operator to the π-calculus and its higher-order variant. This is less trivial than a first glance might suggest. Consider the higher-order process

$$P = (\overline{\mathtt{xfer}}A.B) \mid \mathtt{xfer}(X).(X \mid C)\backslash a.$$

A compositional (ie. non-global) reduction to first-order will reduce the sender and the receiver separately. Using the approach of [8] we would replace P by a process of the shape

$$\nu b.(\overline{\mathtt{xfer}}b.B \mid b(c).A) \mid \mathtt{xfer}(d).(\overline{d}.0 \mid C)\backslash a.$$

But now A and C are prevented by the blocking operator from communicating along a which is clearly not acceptable.

The solution we suggest is very general and powerful: We replace A by a π-calculus representation of its parse tree. Parse tree information is passed lazily from sender to receiver in terms of first-order information only. The receiver uses this syntax information to emulate the behaviour of the remote agent in the local context. The main part of the paper is devoted to fleshing out this idea and establishing its correctness.

This method works well with static operators, but awkwardly for dynamic ones except prefixing. The reason is that static operators are easily mimicked by

the receiver, which simply applies these operators to itself, whereas dynamic ones cannot always be treated in the same way because actions related to communication between sender and receiver might affect the operator (e.g. the pre-emptive power of internal action in the context of the choice operator). Therefore our method cannot be generalized to involve all kinds of e.g. GSOS-operators.

The organization of this paper is as follows. In section 2 we give the main definitions and present some results on the first-order calculus. We show that some algebraic properties and many laws concerning the restriction operator in CCS, in a slightly modified form, continue to be valid for the blocking operator. We obtain soundness and completeness results similar to the results in [4]. In Section 3 we show, as the first main result, that the blocking operator and mismatching can be expressed in terms of each other. We proceed then to the higher-order calculus, and discuss in section 4 an encoding of the reduced version of the higher-order π-calculus defined in [8] extended with blocking. This calculus is monadic, and only finite sums are considered. We show that in the presence of blocking the higher-order paradigm might not be reducible to first-order in the same straightforward manner as for standard π-calculus, for reasons similar to the case for static scoping, e.g. CHOCS. We show then that by sending an encoding of the process, instead of the process itself, reducibility is still possible for at least a reduced version of the calculus, though without full-abstraction. We finally define this reduction and give a sketch of the proof. In section 5 we present some definitions concerning barbed bisimulation that are necessary to establish the main result of the paper, which is the subject of section 6. Finally, some conclusions are presented in section 7.

A version with more proof details and a full definition of the reduction is available electronically at `ftp://ftp.sics.se/pub/fdt/mfd/fhoptp.ps.Z`.

2 Higher-Order π-Calculus with Blocking

Our work is based on the higher-order π-calculus as introduced by Sangiorgi [8], extended with blocking and mismatching. In this section we introduce the syntax and operational semantics of this calculus.

2.1 Syntactical Matters

Agents are generated according to the following abstract syntax:

$$
\begin{aligned}
P ::= {} & 0 \mid \sum_{i=1}^{n} \alpha_i.P_i \mid P_1 \mid P_2 \mid [x=y]P \mid [x \neq y]P \mid (x)P \mid !P \mid \\
& P\backslash z \mid X\langle F\rangle \mid X\langle x\rangle \\
\alpha ::= {} & \overline{x}\langle F\rangle \mid \overline{x}y \mid x(X) \mid x(y) \\
F ::= {} & (\lambda X)P \mid (\lambda x)P \mid X
\end{aligned}
$$

Here x, y and z are channel names, and X is an agent variable. We will also use $\overline{x}(y)$ to mean $(y)\overline{x}y$.

Most operators (summation, parallel, nil, matching) are familiar from CCS and the π-calculus. $!P$ (the "bang") represents the parallel product of an unbounded number of copies of P, $[x = y]P$ is matching, enabling P only when $x = y$, and $[x \neq y]P$ is mismatching, enabling P only when x and y are distinct. For blocking we use the CCS restriction operator notation: $P\backslash x$ blocks communication between P and its environment along the channel x while allowing communication along other channels to mention x.

The higher-order nature of the calculus is brought out by the actions α. Sending actions (of the shape $\bar{x}\langle F\rangle$ or $\bar{x}y$) can pass names as well as general agent abstractions F.

π-calculus restrictions $(x)P$, and input action prefixes of the shape $x(X)$ or $x(y)$ are binding, of x, X, and y, respectively. There are no other operators with binding power. Terms are identified up to alpha-conversion. $s(\alpha)$ is the singleton set containing the subject x of an action of one the actions α above. $n(P)$ ($n(\alpha)$ is the set of names occurring in the agent P (the action α), and $fn(P)$ ($fn(\alpha)$) is the set of free names in P (α). Similarly $bn(P)$ ($bn(\alpha)$) is the set of bound names.

We identify a number of sublanguages:

- Π is the sublanguage not containing mismatching or blocking.
- π is the sublanguage of Π not containing higher-order parameters (and hence agent variables).
- For any of the languages L, LB is the language obtained by adding blocking, and LM is the language obtained by adding mismatching. A *first-order agent* is an agent in πBM.

2.2 Operational Semantics

Transitions have the general shape $P \xrightarrow{\alpha} Q$, where α is a first-order input or output, or the silent action. The operational semantics is given in the appendix. Here it suffices to present the semantical rules governing the operators with which familiarity can not be assumed:

$$\textbf{BLOCK}: \frac{P \xrightarrow{\alpha} P'}{P\backslash z \xrightarrow{\alpha} P'\backslash z}\, s(\alpha) \cap \{z, \bar{z}\} = \emptyset$$

$$\textbf{MISMATCH}: \frac{P \xrightarrow{\alpha} P'}{[x \neq y]P \xrightarrow{\alpha} P'}\, (x \neq y)$$

2.3 Equivalences

Appropriate behavioral equivalences depend on the nature of the calculus being investigated. For the first-order calculi one important notion of equivalence is bisimulation.

Definition 1 (Strong Bisimulation). A binary relation $\mathcal{S}$ on first-order agents is a *(strong) simulation* if $P\mathcal{S}Q$ implies:

1. If $P \xrightarrow{\alpha} P'$ and α is a free action, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.
2. If $P \xrightarrow{x(y)} P'$ and $y \notin n(P,Q)$, then for some Q', $Q \xrightarrow{x(y)} Q'$ and for all w, $P'\{w/y\}\mathcal{S}Q'\{w/y\}$.
3. If $P \xrightarrow{\bar{x}(y)} P'$ and y $\notin n(P,Q)$, then for some Q', $Q \xrightarrow{\bar{x}(y)} Q'$ and $P'\mathcal{S}Q'$.

A binary relation $\mathcal{S}$ is a *(strong) bisimulation* if both $\mathcal{S}$ and its inverse are simulations. Two agents P and Q, are *strongly equivalent*, $P \simeq Q$, if there is some strong bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$.

For higher-order agents a more convenient approach is that of barbed equivalence [5], as this permits us to direct primary attention at the channels along which communication takes place, rather than the parameters.

So, let $P\downarrow_a$ hold just in case $P \xrightarrow{\alpha} Q$ for some Q and α such that $s(\alpha) = \{a\}$. Let also $\longrightarrow = \xrightarrow{\tau}$, let $\Longrightarrow$ be the reflexive and transitive closure of $\longrightarrow$, and let $P\Downarrow_a$ mean that $P \Longrightarrow P'\downarrow_a$, for some P'.

We further need the notion of static contexts. A *static context* is a term $C[\cdot]$ with a "hole" $[\cdot]$ in it, as generated by the following grammar:

$$C ::= P \mid [\cdot] \mid C[\cdot]|C[\cdot] \mid (x)C[\cdot] \mid !C[\cdot] \mid C[\cdot]\backslash z.$$

Here P ranges over the agent language under consideration. We write $C[P]$ for $C[\cdot]$ with P substituted for every occurrence of $[\cdot]$.

Definition 2 (Barbed Bisimulation). A binary relation $\mathcal{R}$ on processes is a *strong (weak) barbed simulation* if $P\mathcal{R}Q$ implies:

1. Whenever $P \longrightarrow P'$ then $Q \longrightarrow Q'$ ($Q \Longrightarrow Q'$) for some Q' such that $P'\mathcal{R}Q'$.
2. For each a, if $P\downarrow_a$ then $Q\downarrow_a$ ($Q\Downarrow_a$).

A relation $\mathcal{R}$ is a *barbed bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are barbed simulations. The agents P and Q are *strong (weak) barbed-bisimilar*, written $P \dot{\sim} Q$ ($P \dot{\approx} Q$), if $P\mathcal{S}Q$ for some strong (weak) barbed bisimulation $\mathcal{S}$. If the agents P and Q belong to the same language, then they are *strong (weak) barbed equivalent*, written $P \sim Q$ ($P \approx Q$), if for each static context $C[\cdot]$ in the language it holds that $C[P] \dot{\sim} C[Q]$ ($C[P] \dot{\approx} C[Q]$).

3 Blocking and Mismatching in the First-Order Case

We first direct attention to the first-order calculus, considering the equational properties of blocking, and the relative expressiveness of blocking and mismatching.

3.1 Algebraic Properties

Most properties of bisimulation for π-calculus carry over to πB with minimal changes. In particular one easily shows that blocking preserves strong bisimulation equivalence, and thus $\simeq$ is a congruence over all operators with the exception of input prefix. The following laws govern the blocking operator:

$$
\begin{array}{ll}
\textbf{H0} & 0\backslash z \simeq 0 \\
\textbf{H1} & P\backslash x\backslash y \simeq P\backslash y\backslash x \\
\textbf{H2} & (P+Q)\backslash z \simeq P\backslash z + Q\backslash z \\
\textbf{H3} & (\alpha.P)\backslash z \simeq \alpha.(P\backslash z), \quad \text{if } z \notin s(\alpha) \cup bn(\alpha) \\
\textbf{H4} & (\alpha.P)\backslash z \simeq 0, \quad \text{if } z \in s(\alpha) \\
\textbf{HR} & ((y)P)\backslash z \simeq (y)(P\backslash z), \quad \text{if } y \neq z \\
\textbf{HM} & ([x=y]P)\backslash z \simeq [x=y](P\backslash z)
\end{array}
$$

By adding laws **H0** to **H4**, **HR** and **HM** to the other algebraic laws in [8], with $=$ substituted for $\simeq$, we get:

Theorem 1 (Soundness). *If* $\vdash P = Q$ *then* $P \simeq Q$. □

Using the same techniques as [4] we obtain:

Lemma 1. *For any* P*, there is a head normal form* H *such that* $\vdash P = H$. □

Theorem 2 (Completeness for finite agents). *For all finite first-order agents* P *and* Q*, if* $P \simeq Q$ *then* $\vdash P = Q$ *is provable.* □

3.2 Expressiveness of the blocking operator

We now proceed to show that, in the case of first-order agents, blocking has the same expressive power as matching and mismatching.

Mismatching may be expressed up to weak ground equivalence using blocking in the following way. Consider the agent $[x \neq y]P$. This agent is equivalent to P if $x \neq y$, and otherwise it is equivalent to 0. We let the agent P be guarded by a restricted channel w, $w.P$, and be executed only if $x \neq y$ by letting the channel $\overline{x}.0$ under blocking by y synchronize with $x.\overline{w}.0$. This synchronization takes place only if $x \neq y$. To avoid additional communication capabilities, we block x too. Thus we obtain

Proposition 1. $[x \neq y]P \cong (w)((\bar{x}.0\backslash y \mid x.\bar{w}.0)\backslash x \mid w.P)$ □

Matching may be expressed similarly:

Proposition 2. $[x = y]P \cong (w)((\bar{x}.0 \mid y.\bar{w}.0)\backslash x\backslash y \mid w.P)$ □

By application of a simple transformation $\mathcal{T}$, defined below, on agents, any agent containing occurrences of the blocking operator may be expressed up to strong equivalence by an agent with no occurrences of blocking. The basic idea is to eliminate occurrences of a blocked channel by replacing it by a fresh channel

under the restriction operator. Since channels may be bound by an input prefix, we have to test them for equality with the blocked channel.

In this way, the transformation $\mathcal{T}$ is a homomorphism for all operators but blocking. For blocking it is defined in terms of an ancillary transformation $\mathcal{T}_{wz}$ thus:

$$\mathcal{T}(P\backslash z) = (w)\mathcal{T}_{wz}(P), w \notin n(P\backslash z).$$

Intuitively $\mathcal{T}_{wz}(P)$ performs the task of testing the subject of an action prefix for equality with z and in this case replace it by w. Consequently $\mathcal{T}_{wz}$ is a homomorphism for the operators $|$, $+$, $!$, matching, mismatching and silent prefix. For the other operators it is defined thus:

$$\begin{aligned}
\mathcal{T}_{wz}(\bar{x}y.P) &= [x = z]\bar{w}y.\mathcal{T}_{wz}(P) + [x \neq z]\bar{x}y.\mathcal{T}_{wz}(P)) \\
\mathcal{T}_{wz}(x(y).P) &= [x = z]w(y').\mathcal{T}_{wz}(P\{y'/y\}) + [x \neq z]x(y').\mathcal{T}_{wz}(P\{y'/y\}) \\
&\quad\ y' \notin fn((y)P) \cup \{w, z\} \\
\mathcal{T}_{wz}((x)P) &= (x')\mathcal{T}_{wz}(P\{x'/x\}), x' \notin fn((x)P) \cup \{w, z\} \\
\mathcal{T}_{wz}(P\backslash z') &= \mathcal{T}_{wz}(\mathcal{T}(P\backslash z'))
\end{aligned}$$

Theorem 3 (Correctness of $\mathcal{T}$). *$P \simeq \mathcal{T}(P)$ for any agent P.*

Proof. It may be shown that $\mathcal{S} = \{(P,\ T(P)) \mid P \text{ agent}\}$ is a strong bisimulation (up-to strong equivalence). □

4 The Higher-Order Case

Having shown that blocking can be eliminated in favour of mismatching in the case of the first-order calculus we now ask if this continues to hold when higher-order communication is added, i.e. we want to know whether ΠB is representable within πB. As we have explained, this problem is much harder than for Π (without blocking), because blockings give rise to dynamically changing "run-time" process environments of a nature drastically different from those of the pure calculus.

4.1 The Reduction

In order to represent ΠB in πB, we apply a transformation $\mathcal{H}$, which is a function from ΠB to πB. We will show that P and $\mathcal{H}(P)$ are weakly equivalent in a sense that has yet to be defined, if P is closed and well-sorted.

We assume that for each name x in ΠB there corresponds a unique name x in πB, and also that for each process variable Y in ΠB there corresponds a unique channel y in the target calculus πB.

The basic idea is that pointers to abstractions, instead of abstractions themselves, are objects of communication. To each abstraction $(\lambda X)P$ that is the object of a communication there corresponds a spawning process $spawn_w(F)$. This process can continuously receive pointers y to abstractions instantiating X,

upon which it will launch a process of type $send_v(P\{Y/X\})$, whose task is the transmission of an encoding of $P\{Y/X\}$. Concurrently, a process of type $rec\langle v\rangle$, which we call receiver processes, receives the encoding of a process $P\{Y/X\}$ and dynamically emulates it. Receiver processes arise in connection with applications of type $Y\langle F\rangle$, where Y is not instantiated directly by an abstraction, but by a pointer to an abstraction, and must emulate the behavior of $Y\langle F\rangle$.

The three agents $spawn_w(F)$, $send_v(P)$ and $rec\langle v\rangle$ form the the core of the transformation $\mathcal{H}$, and are explained in detail below.

Higher-Order Output In the central case of higher-order output we get:

$$\mathcal{H}(\overline{x}\langle F\rangle.P) = \overline{x}(w).(\mathcal{H}(P)|spawn_w(F))$$

provided F is not a process variable. Here, instead of communicating the abstraction F, a "pointer" w to F, or rather to a process responsible for spawning encodings of F, $spawn_w(F)$, is sent instead. If the abstraction F is a process variable Y, what is communicated is its corresponding pointer y:

$$\mathcal{H}(\overline{x}\langle Y\rangle.P) = \overline{x}y.\mathcal{H}(P).$$

For closed processes, this situation can arise only after the input of the pointer of some process which instantiates Y, for example in $a(Y).\overline{x}\langle Y\rangle.P$. In this case the spawning process for the agent F associated with y, $spawn_y(F)$, must have been declared elsewhere.

Application Since we are dealing only with closed processes, an application $Y\langle F\rangle$ may be invoked only after instantiation of Y with some abstraction G through a previous input. The execution of $\mathcal{H}(Y\langle F\rangle)$ runs in parallel with the spawning process for G, $spawn_y(G)$, which must have been defined elsewhere:

$$\mathcal{H}(Y\langle F\rangle) = \overline{y}(u).\overline{u}(v).\overline{u}(w).(rec\langle v\rangle|spawn_w(F)).$$

The application of the process G to its argument F, here represented by a pointer w, is executed by the "receiver" process $rec\langle v\rangle$, an agent whose function is to enact a copy of $G\langle F\rangle$ in the environment where it occurs (possibly within the scope of some blocking operators) by means of the reception and execution of an encoding of $G\langle F\rangle$ through v, a fresh channel sent by the spawning process $spawn_y(G)$ through channel u for the this purpose. The task of sending an encoding of a process is performed by a sender process which needs to know the pointer to the agent being applied, in this case w. For this purpose the pointer w is also communicated to $spawn_y(G)$.

If the argument of Y is a name x, then it is communicated to the spawning process $spawn_y(G)$:

$$\mathcal{H}(Y\langle x\rangle) = \overline{y}(u).\overline{u}(w).\overline{u}x.rec\langle w\rangle.$$

If the argument is a process variable X a similar construction is used.

Example 1. The higher-order process

$$\overline{x}\langle(\lambda X)P\rangle.Q|x(Y).Y\langle G\rangle$$

is represented as the first-order process

$$\overline{x}(w).(\mathcal{H}(Q)|!w(u).u(v).u(x).send_v(P))|x(y).\overline{y}(u).\overline{u}(v).\overline{u}(w').(rec\langle v\rangle|spawn_{w'}(G)$$

where *spawn*, *send*, and *rec* are defined below.

4.2 Senders

The task of the process $spawn_w(F)$, assuming $F = (\lambda X)P$ or $(\lambda x)P$, is to spawn, for any v, "sender" processes $send_v(P)$, whose task is the transmission through v of encodings of P with X or x instantiated to a pointer to the process instantiating X resp a channel instantiating x:

$$spawn_w((\lambda x)P) = !w(u).u(v).u(x).send_v(P))$$
$$spawn_w((\lambda Y)P) = !w(u).u(v).u(y).send_v(P))$$

In order to perform its task, the sender process $send_v(P)$ must make use of special channels indicating the nature of P's head operator, and which should not be used for other purposes. These are: z, c, s, m, n, r, b and i. They represent the process *0* (z), composition (c), sum (s), matching (m), restriction (n), bang (r), blocking (b), input (i) and output (o). We give just a few examples to explain this.

Parallel Composition For instance, if $P = P_1|P_2$, then c, representing composition, is communicated through v, followed by the exchange of a couple of fresh pointers to both components of P, P_1 and P_2, upon which two new sender processes are created for providing an encoding of P_1 resp. P_2:

$$send_v(P_1 \mid P_2) = \overline{v}c.\overline{v}(v_1).\overline{v}(v_2).(send_{v_1}(P_1) \mid send_{v_2}(P_2)).$$

Input Communicating an input is slightly more complicated. In this case, the channel through which the input occurs is communicated to the receiver, which is supposed to dynamically enact such input synchronization before sending back to the sender the actual parameter, which is the name exchanged in the communication. The sender will thus wait for the communication of this channel which instantiates y, whereupon it goes on sending an encoding the continuation of the prefixed agent:

$$send_v(x(y).P) = \overline{v}i.\overline{v}x.v(y).send_v(P)$$

No distinction is made for higher-order inputs, since in this case what is communicated by the sender is a "pointer" to a process.

Summation The definition of $send_v(\sum_{i=1}^{n} P_i)$, $n > 1$, includes the exchange of a couple of fresh "pointers", one, v_1, for P_1, and the second, v_2, for $\sum_{i=2}^{n} P_i$, which in case $n = 2$ must be a prefixed agent. This scheme works because only well-guarded agents are allowed in summations.

4.3 Receivers

The task of the "receiver" process $rec\langle v\rangle$ is to receive from a sender $send_v(P)$, through the channel v the encoding of a process P, and at the same time to interpret this encoding.

Parallel Composition For process composition $P = P_1|P_2$, the receiver requires a pair of fresh pointers to each of these processes, whereupon it gives rise to a composition of two new receiver processes, $rec\langle v_1\rangle|rec\langle v_2\rangle$, whose task is to receive an encoding of P_1 resp P_2 and execute them.

Summation The most difficult part of the receiver, and illustrating the difficulties in extending generality beyond static operators, is the encoding of summation. We use a protocol similar to that of Pierce and Nestmann in [6]. The details are left out of this version of the paper.

Input For input the task of the receiver is to emulate any of these actions by dynamically offering the subject of the action for communication. In case of name or process inputs the the situation is only slightly more complicated. In this case, the receiver offers a synchronization through the same channel x, whereupon it communicates to the sender the channel exchanged in the synchronization.

5 Barbed Bisimulation

For correctness we use the notion of barbed bisimulation [5]. Full abstraction, that is, the requirement that two terms in ΠB be equivalent if and only if their translations in πB are equivalent, is not fulfilled by the translation $\mathcal{H}$. Sending a process P is like sending object code, protected in a way such that it can only be executed, but not modified. Sending an encoding of P, on the other hand, is like sending the source code: the receiver may change the code at will and also its own behaviour in accordance with the nature of any of the components of P. As an example, for any process $P \in \Pi B$, $\overline{a}\langle(\lambda x)P\rangle.0$ and $\overline{a}\langle(\lambda x)P|0\rangle.0$ are certainly equivalent. Nevertheless, their translations are quite distinct. In the former case an agent $send_w(P)$ will eventually be activated, whereas in the latter case the agent activated will be $send_w(P|0)$. The latter provides an encoding of $P|0$, not P, and it does so by first sending an indication that the main operator is the composition operator, $\overline{w}c$. Any process in πB synchronizing with $send_w(P|0)$ may choose to act according to the nature of this synchronization, for example $w(x).([x = c]0+[x = i]Q)$. Thus, the translations of $\overline{a}\langle(\lambda x)P\rangle.0$ and $\overline{a}\langle(\lambda x)P|0\rangle.0$ cannot possibly be equivalent in any sensible sense. Nevertheless, a restricted

form of completeness is achieved by the translation $\mathcal{H}$ if we limit testing on terms in πB to encodings of source terms. In this restricted form the translation proposed here is both sound and complete.

We then set out the basic definitions to flesh out this idea. First, let $\mathcal{H}$ be the translation described above for processes in ΠB, extended with the rule $\mathcal{H}[\cdot] = [\cdot]$ for contexts.

Definition 3 (Reduced Composition, Reduced Context).

1. A *reduced composition* $\prod$ is a composition in πB of agents of type $spawn_w(F)$, $send_v(P)$, $rec\langle v\rangle$, $\mathcal{H}(P)$, or any of the derivatives of such agents, for any agents F and $P \in \Pi B$, and such that (i) if $spawn_w(F)$ and $spawn_{w'}(G)$ occur in $\prod$, and $w = w'$, then $F \equiv G$; (ii) if $send_v(P)$ and $send'_v(Q)$ occur in $\prod$, and $v = v'$, then $P \equiv Q$.
2. A context $C[\cdot] \in \pi B$ is called a *reduced context* if $C[\cdot] = (\widetilde{y})(\prod | [\cdot])$ for some channel vector $\widetilde{y}$ and some reduced composition $\prod$ with no occurrence of the restriction operator.

Definition 4 (Reduced Equivalence). Two processes P and $Q \in \pi B$ are *strong (weak) reduced equivalent*, written $P \sim_r Q$ ($P \approx_r Q$), if for each reduced context $C[\cdot] \in \pi B$, it holds that $C[P] \dot{\sim} C[Q]$ ($C[P] \dot{\approx} C[Q]$).

Reduced equivalence is an equivalence relation. Moreover, from the definition we get immediately that for any processes P and Q in πB, $P \sim_{\pi B} Q$ implies $P \sim_r Q$, and $P \approx Q$ implies $P \approx_r Q$. Also we obtain the following congruence properties:

Proposition 3. *Strong and weak reduced equivalence are congruences under output prefix, bang (!), restriction, and blocking.* □

6 The Correctness Proof

The next definitions follow closely [7]. We use $P \stackrel{\wedge}{\longrightarrow} P'$ to mean $P \longrightarrow P'$ or $P \equiv P'$, and $\Longrightarrow^+$ to mean the transitive closure of $\longrightarrow$.

Definition 5 (Expansion). $\mathcal{E}$ is an *expansion* if $P\mathcal{E}Q$ implies:

1. Whenever $P \longrightarrow P'$, then Q' exists s.t. $Q \Longrightarrow^+ Q'$ and $P'\mathcal{E}Q'$, and for each a, if $P' \downarrow a$ then $Q' \Downarrow a$;
2. Whenever $Q \longrightarrow Q'$, then P' exists s.t. $P \stackrel{\wedge}{\longrightarrow} P'$ and $P'\mathcal{E}Q'$, and for each a, if $Q' \downarrow a$ then $P' \downarrow a$;

We say that Q *expands* P, written $P \preceq Q$, if $P\mathcal{E}Q$, for some expansion $\mathcal{E}$.

Definition 6 (Weak Barbed Bisimulation up-to $\preceq$). S is a *weak barbed bisimulation up-to* $\preceq$ if:

1. Whenever $P \longrightarrow P'$, then Q' exists s.t. $Q \Longrightarrow Q'$ and $P \succeq \mathcal{S} \dot{\approx} Q'$.

2. Whenever $Q \longrightarrow Q'$, then P' exists s.t. $P \Longrightarrow P'$ and $P \dot{\approx} \mathcal{S} \preceq Q'$.

Lemma 2. *If $\mathcal{S}$ is a weak barbed bisimulation up-to $\preceq$, then $\mathcal{S} \subseteq \dot{\approx}$.*

Proof. By diagram chasing. □

The main result by which we prove correctness is the following:

Theorem 4. *$\mathcal{S} = \{(P, \mathcal{H}(P)) : P \in \Pi B\}$ is a weak barbed bisimulation up-to $\preceq$.*

An outline proof of this theorem is included in the electronic version of this paper. The details are quite complex, though the approach in most cases is non-controversial. One difficulty, however, deserves highlighting. The key difference between a higher-order parameter, $\lambda X.P$ and its representation in the first-order calculus is that in the higher-order case the parameter X is available as a first-class entity and can, for instance, freely be copied into different contexts. For the representation, on the other hand, information regarding the parameter X resides elsewhere, in one (replicatable) copy which needs to service all possible receivers, in all possible contexts. To adequately handle this, the proof of Theorem 4, calls upon the following lemma:

Lemma 3. *Let P, $Q \in \pi B$ be transformations of agents in ΠB or any derivatives of such agents, and such that $spawn_w(G)$ does not occur in either P or Q for any agent $G \in \Pi B$. Then*

1. $(w)(spawn_w(F)|P)|spawn_w(F) \sim_r P|spawn_w(F)$.
2. $(w)(spawn_w(F)|P|Q) \sim_r (w)(spawn_w(F)|P)|(w)(spawn_w(F)|Q)$.
3. $(w)(spawn_w(F)|!P) \sim_r !(w)(spawn_w(F)|P)$.
4. $(w)(P|spawn_w(F)) + (w)(Q|spawn_w(F)) \sim_r (w)((P+Q)|spawn_w(F))$. □

Now, to prove correctness using Theorem 4 the following lemma is proved in a straightforward manner by induction in C's formation.

Lemma 4. *For any process $P \in \Pi B$ and any static context $[\cdot]$,*

$$\mathcal{H}(C[P]) = \mathcal{H}(C)[\mathcal{H}(P)].$$

Now we obtain:

Corollary 1. *$\mathcal{H}$ restricted to static contexts in πB that are encodings of static contexts in ΠB is sound and complete.*

Proof. Soundness: Assume $\mathcal{H}(C)[\mathcal{H}(P)] \dot{\approx} \mathcal{H}(C)[\mathcal{H}(Q)]$ for every static context $C[\cdot] \in \Pi B$. By Lemma 4, this implies that $\mathcal{H}(C[P]) \dot{\approx} \mathcal{H}(C[Q])$. Then by Theorem 4 and transitivity of weak equivalence $C[P] \dot{\approx} C[Q]$. Since this is true for every static context $C \in \Pi B$, then $P \approx_r Q$.

Completeness: If $P \approx Q$ then $C[P] \dot{\approx} C[Q]$ for all static contexts $C \in \Pi B$. Then by the theorem and transitivity of weak equivalence, $\mathcal{H}(C[P]) \dot{\approx} \mathcal{H}(C[Q])$. By Lemma 4 we get $\mathcal{H}(C)[\mathcal{H}(P)] \dot{\approx} \mathcal{H}(C)[\mathcal{H}(Q)]$, and thus the transformation is complete with regard to those agents and contexts in πB that are transformations of agents and contexts in ΠB. □

7 Conclusion

We have investigated the consequences of adding dynamic restriction in the style of CHOCS [9] to the higher-order π-calculus. On grounds of practical modelling power we believe very strongly that this is a reasonable thing to do. Higher-order features are useful as programming and modelling abstractions. This applies in the context of the π-calculus too (cf. [2]). But higher-order features entail the need of mechanisms to provide local control of communication, analogous to firewalling, as we have shown. CHOCS dynamic restriction, or, as we call it, *blocking*, appears to do the job well. Whether, at the end of the day, other operators are more appropriate, remains to be seen.

The upshot, however, is that any operator that provides local control of communication in a higher-order setting is likely, as the blocking operator, to interact badly with Sangiorgi's basic result [8] showing that higher-order features in the π-calculus are reducible to first-order ones. We have resolved this by providing a very general and powerful higher-order reduction, based on the idea of communicating and dynamically interpreting parse trees in place of the processes themselves. We conjecture that any "reasonable" static operator can be handled in this way. It would be interesting to prove such a statement in terms of an extension of one of the well-known formats for operational semantics such as GSOS [1], adapted to the π-calculus.

While our results in principle substantiate the claim that, for the π-calculus, higher-order features are matters of convenience only, in practice this does not at all appear to be the case. This issue, or rather the more general issue of what the role of higher-order features in calculi for concurrent and distributed systems should be, needs to be investigated much more deeply in the future.

Acknowledgement

We thank Lars-Åke Fredlund for comments and suggestions.

References

1. Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.
2. M. Dam. Proving trust in systems of second-order processes. In *Proc. HICSS'31* IEEE Comp. Soc., VII:255–264, 1998. Available electronically at `ftp://ftp.sics.se/pub/fdt/mfd/ptssop.ps.Z`.
3. Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
4. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, September 1992.
5. Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. of 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *lncs*, pages 685–695. sv, 1992.

6. Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. pages 179–194. Revised full version as report ERCIM-10/97-R051, European Research Consortium for Informatics and Mathematics, 1997.
7. D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". *Lecture Notes in Computer Science*, 630:32–??, 1992.
8. Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, LFCS, University of Edinburgh, 1993.
9. Bent Thomsen. *Calculi for Higher Order Communicating Systems.* PhD thesis, Imperial College, University of London, September 1990.

Appendix: Operational Semantics of ΠB

In the operational semantics we assume agents are well-sorted according to the definition in [8]. That two names x and y resp. two agents variables X and Y, are of the same sort is denoted by $x : y$ resp $X : Y$.

The operational semantics below uses an early instantiation scheme. There, K stands for an abstraction or a name, and U for a variable or a name. Also, $\widetilde{y}$ stands ambiguously for a name vector or the set containing exactly the names in the vector, and if $\widetilde{y} = (y_1, .., y_n)$, then $(\widetilde{y})$ stands for $(y_1)...(y_n)$.

Rules of Action

ALP: $\dfrac{P' \xrightarrow{\mu} Q, \;\; P \text{ and } P' \text{ are } \alpha-\text{convertible}}{P \xrightarrow{\mu} Q}$
OUT: $\overline{x}\langle K\rangle.P \xrightarrow{\overline{x}\langle K\rangle} P$ **INP:** $x(U).P \xrightarrow{x(K)} P\{K/U\}$, if $K : U$
SUM: $\dfrac{P_k \xrightarrow{\mu} P'}{\sum_{i=1}^{n} P_i \xrightarrow{\mu} P'}\, 1 \le k \le n$ **PAR:** $\dfrac{P \xrightarrow{\mu} P'}{P\|Q \xrightarrow{\mu} P'\|Q}, bn(\mu) \cap fn(Q) = \emptyset$
COM: $\dfrac{P \xrightarrow{(\widetilde{y})\overline{x}\langle K\rangle} P' \;\; Q \xrightarrow{x\langle K\rangle} Q'}{P\|Q \xrightarrow{\tau} (\widetilde{y})(P'\|Q')}, \widetilde{y} \cap fn(Q) = \emptyset$
MATCH: $\dfrac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'}$ **REP:** $\dfrac{P\| \; !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$
RES: $\dfrac{P \xrightarrow{\mu} P'}{(x)P \xrightarrow{\mu} (x)P'}, x \notin n(\mu)$ **OPEN:** $\dfrac{P \xrightarrow{(\widetilde{y})\overline{z}\langle K\rangle} P'}{(x)P \xrightarrow{(x\widetilde{y})\overline{z}\langle K\rangle} P'}\; \begin{array}{l} x \neq z, \\ x \in fn(K) - \widetilde{y} \end{array}$
BLOCK: $\dfrac{P \xrightarrow{\mu} P'}{P\backslash z \xrightarrow{\mu} P'\backslash z}, s(\mu) \notin \{z, \overline{z}\}$

Obs: Symmetric forms for operators + and — have been omitted