Variations on Mobile Processes [⋆]

Yuxi Fu [1]

Department of Computer Science, Shanghai Jiaotong University
1954 Hua Shan Road, Shanghai 200030, China

Abstract

The paper investigates a concurrent computation model, chi calculus, in which communications resemble cut eliminations for classical proofs. Two bisimilarities, local bisimilarity and barbed bisimilarity, on chi processes are studied and are shown to be congruence relations. The former equivalence turns out to be strictly stronger than the latter. It is shown that chi calculus is capable of modeling sequential computation in that it captures the operational semantics of call-by-name lambda calculus. A translation from pi calculus to chi calculus is given, demonstrating that, practically speaking, pi is a sublanguage of chi. A higher order version of chi calculus is proposed and examined. It combines the communication mechanism of chi calculus and the recursion mechanism of full lambda calculus, and therefore extends both.

Key words: concurrency theory, process algebra, mobile process, bisimulation

1 Introduction

Concurrent computation is currently an open-ended issue. The situation is in contrast with sequential computation whose operational semantics is formalized by, among others, the λ-calculus ([8]) and is well understood. In retrospect, the λ-calculus can be seen as a fallout of proof theory. The Curry-Howard's proposition-as-type principle allows one to code up constructive proofs as typed terms (proof-as-term). At the core of the constructive logic is the minimal logic, whose type theoretical formulation gives rise to, roughly, the simply typed λ-calculus. Now the untyped λ-calculus is obtained from the simply typed λ-calculus by removing all typing information. Of course the λ-calculus did not come into existence this way. But this way of looking at the model emphasizes its connection to proof theory ([59]).

⋆ *Theoretical Computer Science*, **221**(1): 327-368, 1999.
1 E-mail: `fu-yx@cs.sjtu.edu.cn`.

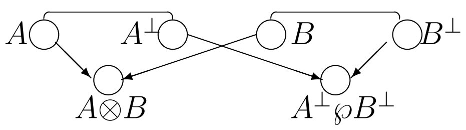
In recent years, classical proofs have been investigated in a computational setting. Girard proposed proof nets ([22,23,63]) as term representations of classical linear proofs, see also [3]. These classical terms are typed. The conclusion of a proof derivation is the type of the proof net corresponding to that proof derivation. The computations of these terms are cut eliminations modeled by rewritings of graphs. As the terms are typed, cuts happen between nodes of correlated types. Abramsky's proof-as-process interpretation ([4,15]) relates proof nets to processes. At operational level, this interpretation is supported by a cut-elimination-as-communication paradigm. It looks like a type-erasing interpretation similar to the one found in the sequential world.

If one intends to push the analogy between constructive proof/sequential computation on one hand and classical proof/concurrent computation on the other, one wonders what the computational aspect of classical proofs would suggest for constructing a model of concurrent computation. This paper investigates a concurrent computation model obtained by reversing the roles of proofs and processes in Abramsky's paradigm. That is to say that we regard communications as cut eliminations. The way to arrive at such a model of communication echoes that in the sequential world. As the 'minimal logic' in a classical scenario, we take the multiplicative linear logic. There is nothing canonical about this choice. As the typed classical terms we take of course the proof nets. Processes are then obtained by, roughly speaking, removing the typing information.

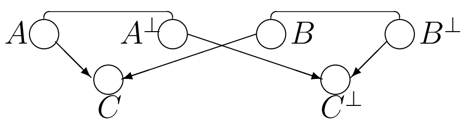
Ignoring units, the multiplicative linear logic has the following rules:

$$\dfrac{}{A, A^{\perp}}\,Axiom \qquad \dfrac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A\otimes B}\,\otimes \qquad \dfrac{\Gamma, A, B}{\Gamma, A\wp B}\,\wp \qquad \dfrac{\Gamma, C \quad \Delta, C^{\perp}}{\Gamma, \Delta}\,Cut$$

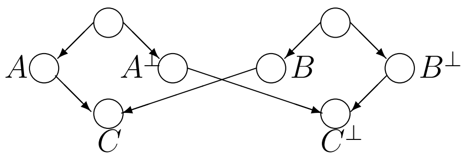
Using these rules, a sequent might have two derivation trees that differ only in inessential orders of applications of rules. A proof net is construed as a canonical representation of proof derivations with inessential differences. The sequent $A\otimes B, A^{\perp}\wp B^{\perp}$, for instance, has a unique proof net as follows:



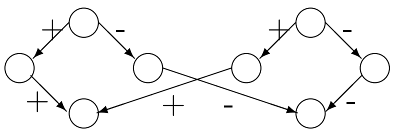
The first step towards a concurrent model is to abstract away the logical aspect of proof nets but keep its proof theoretical content. The above proof net becomes



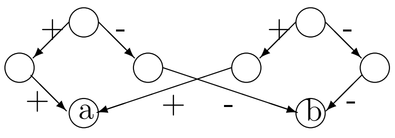
There are two kinds of edges in the net. So the second step is to transform the net into a graph in which all edges are directed arrows:



In the third step, we leave out the typing information while recording the correlation information by labels on arrows. We thus arrive at an untyped graph



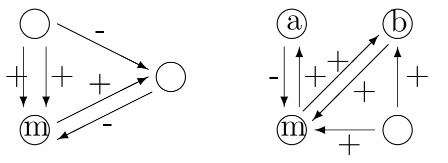
This is the untyped version of the original classical typed term. Notice that there are two kinds of node in the proof net: the internal nodes and the conclusion nodes, the latter representing the conclusions of the corresponding proof. When forming new nets, it is the conclusion nodes that interact with each other. In other words, nodes in a proof net can be classified into internal (local) nodes and external (global) nodes. In order to distinguish the two kinds of node in the untyped graph, we label the conclusion nodes with small letters:



We call graphs of this kind reaction graphs. The formal definition is as follows, where $\mathcal{N}$ is a set of names ranged over by lower case letters.

Definition 1 *A finite directed graph is a quadruple $\langle N, E, d_0, d_1 \rangle$ where both N and E are finite sets, d_0 and d_1 are functions from E to N specifying respectively the source and the target nodes of arrows. A reaction graph is a sextuple $\langle N, E, d_0, d_1, o, e \rangle$ where $\langle N, E, d_0, d_1 \rangle$ is a finite directed graph, o is a partial function from N to $\mathcal{N}$ that is injective on its domain of definition, and e is a function from E to $\{-, +\}$.*

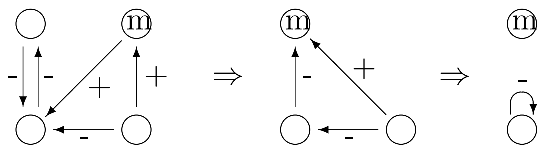
In a reaction graph, a node without (with) a label is called *local* (*global*). Reaction graphs can be seen as the underlying graphs of proof derivations in a generalized and distilled form. The following are two examples of reaction graphs:



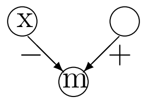
In the above diagrams, a local node is drawn as a cycle while a global node is drawn with its label inside a cycle.

The labels of the arrows in a reaction graphs indicate the polarities of nodes with respect to each other. In the above right graph, the node labeled by a (b) shows up negative (positive) polarity to the node labeled m.

Computations with reaction graphs are cut eliminations. A cut elimination happens between a local node and a global or local node. When they show up opposite polarities to a same node, they can react by removing the two arrows indicating the polarities and coalescing the two nodes. We are not going to define formally cut-eliminations in reaction graphs. The interested reader is referred to [18]. Here we use an example to illustrate the basic idea. The following is an example of two consecutive cut-eliminations:



In the left reaction graph, the two upper nodes show up opposite polarities to the left bottom node. This cut is eliminated in the first reduction. The two arrows are removed and the two upper nodes are coerced with the resulting node labeled by m. In the middle reaction graph, the two bottom nodes with the arrows pointing to the node labeled m form a cut. The second reduction eliminates the cut. In the final reaction graph, we can garbage-collect the detached global node. So a configuration of the form, say,



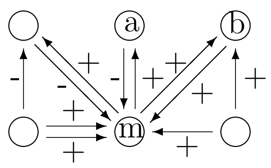
in a reaction graph is a cut. It's elimination deletes the two arrows and coerces the two source nodes, dragging the remaining arrows all the way. The idea of this paper is to think of this cut-elimination as a communication in which x is exported through m to instantiate a local node. To develop the idea, we need a process-like notation for reaction graphs. Let us define graph terms by abstract syntax as follows:

$$G := \mathbf{0} \mid m[x] \mid \overline{m}[x] \mid (x)G \mid G|G'$$

where $\mathbf{0}$ is the empty reaction graph; $m[x]$ and $\overline{m}[x]$ are respectively the following reaction graphs:

$$\begin{tikzcd} x \arrow[r, "+"] & m & & x \arrow[r, "-"] & m \end{tikzcd}$$

$(x)G$ is obtained from the reaction graph G by removing the label x from G; $G|G'$ is the amalgamation of G and G', coercing nodes with same labels. For example the amalgamation of the two reaction graphs given right after Definition 1 is the following reaction graph



In the process-like notation, the two consecutive cut-eliminations in the previous example can be described by the following reductions:

$$(x)(y)(z)(m[x]|\overline{y}[x]|y[m]|\overline{y}[z]|\overline{z}[y]) \to (x)(y)(m[x]|\overline{y}[x]|\overline{m}[y]) \to (x)(\overline{x}[x]).$$

This term representation gives rise to a calculus of reaction graphs.

It is clear from the above example that communications in the calculus of reaction graphs amount to identifications of objects. This deviates from the traditional view that communications are instantiations of formal parameters.

The calculus of graphs only deals with finite step computations. To achieve Turing computability, we extend the language with standard process combinators. More specifically, guarded replication is incorporated to admit infinite computation, whereas sequentiality operator is introduced to enhance the control power of the language. The resultant language will be referred to as χ-calculus, where χ stands for ex*ch*ange of *i*nformation. This paper initiates an investigation of this computation model. In Section 2 and Section 3 we examine the semantics of χ-processes. Various possible bisimilarities on χ-processes are proposed and compared. It is shown that they boil down to two distinguished congruence relations: local bisimilarity and barbed bisimilarity. The former is strictly finer than the latter. The language is related to the π-calculus in Section 4. It is argued that in practice one can regard the π-calculus as a sublanguage of the χ-calculus. In Section 5 it is pointed out that the operational semantics of the call-by-name λ-calculus can be readily captured in the χ-calculus. The investigation of the language is continued in Section 6 by integrating it with λ-calculus. In Section 7 the cut-elimination-as-communication paradigm is recast in an algebraic setting by constructing a $*$-autonomous category of χ-processes. Some final remarks are made in Section 8, where related works are discussed.

Preliminary results of this paper have been announced in [17]. Some results from [16] have also been incorporated.

2 A Model for Concurrent Computation

The χ-calculus is basically the calculus of reaction graphs enriched with the sequentiality operator and recursion found typically in process algebra. The operational semantics of the language can be defined in terms of a labeled transition system ([16]). In this paper we give a reductional semantics for χ-calculus in the style of [11,34].

Let $\mathcal{N}$ be a set of names ranged over by lower case letters and $\overline{\mathcal{N}} \stackrel{\text{def}}{=} \{\overline{a} \mid a \in \mathcal{N}\}$ be the set of conames. The union $\mathcal{N} \cup \overline{\mathcal{N}}$ will be ranged over by α. Define $\overline{\alpha}$ to be m ($\overline{m}$) if α is $\overline{m}$ (m). Let $\mathcal{C}$ be the set of χ-processes defined by the following abstract syntax:

$$P := \mathbf{0} \mid \alpha[x].P \mid P|P' \mid (x)P \mid \alpha(x){*}P.$$

As usual $\mathbf{0}$ is the inactive process. A trailing inactive process will be omitted. $m[x].P$ and $\overline{m}[x].P$ are processes that must first perform a communication through name m before enabling $P[y/x]$, where y is the name received in communication. Here m is in a subject position while x is in an object position. $P|P'$ is a process in parallel composition form, in which P and P' can evolve independently and may communicate during the course of their evolution. $(x)P$ is a process in which x is local to P, meaning that $(x)P$ is not allowed to communicate with another process through name x. The x in this process is called a local name. The guarded recursion $\alpha(x){*}P$ makes a copy of P with instantiated x whenever it is called upon. The name x in this process is also regarded as local. The set of local names appeared in P is denoted by $ln(P)$, whereas the set of global names, or nonlocal names, in P is designated by $gn(P)$. Set $n(P)$ is the union of $ln(P)$ and $gn(P)$. We adopt the well-known α-convention saying that a local name in a process can be replaced by a fresh name without affecting the syntax of the process.

The notation $[y/x]$ will stand for an atomic substitution. The result of substituting y for x throughout P is denoted by $P[y/x]$. Local names in P need be renamed to avoid y being captured. A substitution $[y_1/x_1]\ldots[y_n/x_n]$ is a concatenation of atomic substitutions. The effect of a substitution on a process is defined as follows: $P[] \stackrel{\text{def}}{=} P$; $P[y_1/x_1]\ldots[y_n/x_n] \stackrel{\text{def}}{=} (\ldots P[y_1/x_1]\ldots)[y_n/x_n]$. Here $[]$ is the empty substitution. The set of substitutions will be ranged over by σ.

To simplify the algebraic theory of the language, a structural congruence is imposed on the members of $\mathcal{C}$.

Definition 2 *The structural relation $=$ is the least congruence on χ-processes that contains:*

(i) $P|\mathbf{0} = P$*,* $P_1|P_2 = P_2|P_1$*, and* $P_1|(P_2|P_3) = (P_1|P_2)|P_3$*;*
(ii) $(x)\mathbf{0} = \mathbf{0}$*,* $(x)(y)P = (y)(x)P$*, and* $(x)(P|Q) = P|(x)Q$ *if* $x \notin gn(P)$*;*
(iii) $P = Q$ *if* P *and* Q *are* α*-convertible.*

We regard $=$ as a grammatic equality. So $P = Q$ means that P and Q are syntactically the same. It follows that we can write $P_1|P_2|P_3$ without ambiguity. The reductional semantics for χ-calculus can now be defined as follows:

$$(x)(R|\alpha[x].P|\overline{\alpha}[y].Q) \rightarrow (x)(R[y/x]|P[y/x]|Q[y/x])$$

$$\alpha(x)*P|\overline{\alpha}[y].Q \rightarrow \alpha(x)*P|P[y/x]|Q$$

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \frac{P \rightarrow P'}{(x)P \rightarrow (x)P'}$$

The first reduction rule can be replaced by the following two:

$$(x)(R|\alpha[x].P|\overline{\alpha}[y].Q) \rightarrow R[y/x]|P[y/x]|Q[y/x], \quad \text{where } x \neq y$$

$$(x)(R|\alpha[x].P|\overline{\alpha}[x].Q) \rightarrow (x)(R|P|Q)$$

As we regard the structural congruence $=$ as a syntactical equality, the following rule

$$\frac{P = P' \quad P \rightarrow Q \quad Q = Q'}{P' \rightarrow Q'}$$

comes for free.

To help understand the communication rules, we give some examples. In the following reductions, x and y are distinct.

$$\begin{aligned}
(x)(R|\overline{m}[y].P|m[x].Q) &\rightarrow R[y/x]|P[y/x]|Q[y/x] \\
\overline{m}[y].P|(x)(R|m[x].Q) &\rightarrow P|R[y/x]|Q[y/x] \\
(y)(\overline{m}[y].P|(x)(R|m[x].Q)) &\rightarrow (y)(P|R[y/x]|Q[y/x]) \\
(x)\overline{m}[x].P|(y)m[y].Q &\rightarrow (z)(P[z/x]|Q[z/y]), \text{ where } z \text{ is fresh} \\
(x)(R|\overline{m}[x].P|m[x].Q) &\rightarrow (x)(R|P|Q).
\end{aligned}$$

In the first example, a communication replaces the local name x by the global name y throughout the process over which the localization operator (x) applies. The global name y in P however remains unchanged. In other words, global names overwrite. Notice that $(x)(R[y/x]|P[y/x]|Q[y/x]) =$

$R[y/x]|P[y/x]|Q[y/x]$. In the second example the process $\overline{m}[y].P$ lies outside the scope of the localization operator (x). So the communication through m does not affect P. In order to use the rule, notice that $\overline{m}[y].P|(x)(R|m[x].Q) = \overline{m}[y].P|(z)(R[z/x]|m[z].Q[z/x]) = (z)(\overline{m}[y].P|R[z/x]|m[z].Q[z/x])$ for a fresh z. So $\overline{m}[y].P|(x)(R|m[x].Q) \rightarrow (P|R[z/x]|Q[z/x])[y/z] = P|R[y/x]|Q[y/x]$. The third reduction is obtained from the second by applying the second structural rule. How does the forth reduction come about? Well for a fresh name z, $(x)\overline{m}[x].P|(y)m[y].Q = (z)((x)\overline{m}[x].P|m[z].Q[z/y])$. So the reduction can be deduced as in the previous case. The final reduction is an example of communication where two processes exchange a same local name. This reduction is an instance of the first reduction rule where y is x.

It is clear from these examples that the localization operator in the χ-calculus acts as an effect delimiter. A communication either instantiates a local name by a global name or identifies two local names.

Let $\rightarrow^+$ ($\rightarrow^*$) be the transitive (reflexive and transitive) closure of $\rightarrow$. We will denote by $\vec{x}$ a sequence $x_1, \ldots, x_n$ of names. We will also abbreviate $(x_1)\ldots(x_n)P$ to $(\vec{x})P$. When the length of the sequence $\vec{x}$ is zero, $(\vec{x})P$ is just P. The length of $\vec{x}$ is denoted by $|\vec{x}|$. If $\vec{a} = a_1 \ldots a_n$ then $\vec{a}[\vec{x}]$ denotes $a_1[x_1]|\ldots|a_n[x_n]$.

Before ending this section, we state a technical lemma to be used later on.

Lemma 3 *If $P \rightarrow Q$ then $P[y/x] \rightarrow Q[y/x]$.*

Proof: By structural induction it is easy to show that $P_1 = P_2$ implies $P_1[y/x] = P_2[y/x]$, from which the result follows. $\square$

3 Bisimulation Equivalence

To study the algebraic semantics of χ-processes, it is convenient to have a labeled transition system defined as follows, where δ ranges over $\{\overset{\alpha x}{\rightarrow}, \overset{\alpha[x]}{\rightarrow}, \overset{\alpha(x)}{\rightarrow} \mid \alpha \in \mathcal{N} \cup \overline{\mathcal{N}}, x \in \mathcal{N}\}$:

$$\frac{}{(y)(R|\alpha[y].P) \overset{\alpha x}{\rightarrow} R[x/y]|P[x/y]} \qquad \frac{}{\alpha(y)*P \overset{\alpha x}{\rightarrow} \alpha(y)*P|P[x/y]}$$

$$\frac{}{\alpha[x].P \overset{\alpha[x]}{\rightarrow} P} \qquad \frac{P \overset{\alpha[x]}{\rightarrow} P'}{(x)P \overset{\alpha(x)}{\rightarrow} P'} \qquad \frac{}{\alpha(x)*P \overset{\alpha(x)}{\rightarrow} \alpha(x)*P|P}$$

$$\frac{P \xrightarrow{\delta} P' \quad ln(\delta) \cap gn(Q) = \emptyset}{P|Q \xrightarrow{\delta} P'|Q} \qquad \frac{P \xrightarrow{\delta} P' \quad x \notin n(\delta)}{(x)P \xrightarrow{\delta} (x)P'}$$

In the rules, $ln(\delta)$ is $\{x\}$ when δ is $\alpha(x)$; it is the empty set otherwise; $n(\delta)$ is the set of names in δ. In sequel, $\overset{\delta}{\Rightarrow}$ denotes the relation $\rightarrow^* \xrightarrow{\delta} \rightarrow^*$.

The properties stated in the following lemma can be easily proved by structural induction.

Lemma 4 *(i) If $P \overset{\alpha[x]}{\rightarrow} P'$ then some $\vec{z}$, P_1 and P_2 exist such that $P = (\vec{z})(P_1|\alpha[x].P_2)$, $P' = (\vec{z})(P_1|P_2)$ and $x \notin \{\vec{z}\}$.*
(ii) If $P \overset{\alpha x}{\rightarrow} P'$ then some $\vec{z}$, y, P_1 and P_2 exist such that either

$$P = (\vec{z})(y)(P_1|\alpha[y].P_2) \text{ and } P' = (\vec{z})(P_1[x/y]|P_2[x/y])$$

or

$$P = (\vec{z})(P_1|\alpha(y)*P_2) \text{ and } P' = (\vec{z})(P_1|P_2[x/y]|\alpha(y)*P_2).$$

(iii) If $P \overset{\alpha(x)}{\rightarrow} P'$ then some $\vec{z}$, P_1 and P_2 exist such that either

$$P = (\vec{z})(x)(P_1|\alpha[x].P_2) \text{ and } P' = (\vec{z})(P_1|P_2)$$

or

$$P = (\vec{z})(P_1|\alpha(x)*P_2) \text{ and } P' = (\vec{z})(P_1|P_2|\alpha(x)*P_2).$$

One could have combined the labeled transition system with the reduction relation $\rightarrow$ to form a system defining the operational semantics of χ, following the standard approach.We have however chosen to separate the semantics of communication from that of communicability. In either way, one has to prove a number of bookkeeping lemmas. Here are two such lemmas.

Lemma 5 *Suppose $P, Q \in \mathcal{C}$.*
(i) If $P \overset{\alpha x}{\rightarrow} P'$ and $Q \overset{\overline{\alpha}[x]}{\rightarrow} Q'$ then $P|Q \rightarrow P'|Q'$.
(ii) If $P \overset{\alpha[x]}{\rightarrow} P'$ and $Q \overset{\overline{\alpha}[y]}{\rightarrow} Q'$ then $(x)(P|Q) \rightarrow (x)(P'[y/x]|Q'[y/x])$.

PROOF: Let's see how to prove (i). Suppose $P \overset{\alpha x}{\rightarrow} P'$. By Lemma 4 some $\vec{z}$, y, P_1 and P_2 exist such that either

$$P = (\vec{z})(y)(P_1|\alpha[y].P_2) \text{ and } P' = (\vec{z})(P_1[x/y]|P_2[x/y])$$

or

$$P = (\vec{z})(P_1|\alpha(y)*P_2) \text{ and } P' = (\vec{z})(P_1|P_2[x/y]|\alpha(y)*P_2).$$

Similarly $Q \overset{\overline{\alpha}[x]}{\rightarrow} Q'$ implies that some $\vec{z}$, Q_1 and Q_2 exist such that $Q = (\vec{z})(Q_1|\overline{\alpha}[x].Q_2)$ and $Q' = (\vec{z})(Q_1|Q_2)$. Now either

$$\begin{aligned} P|Q &= (\vec{z})(y)(P_1|\alpha[y].P_2)|(\vec{z})(Q_1|\overline{\alpha}[x].Q_2) \\ &\rightarrow (\vec{z})(P_1[x/y]|P_2[x/y])|(\vec{z})(Q_1|Q_2) \\ &= P'|Q' \end{aligned}$$

or

$$\begin{aligned} P|Q &= (\vec{z})(P_1|\alpha(y)*P_2)|(\vec{z})(Q_1|\overline{\alpha}[x].Q_2) \\ &\rightarrow (\vec{z})(P_1|P_2[x/y]|\alpha(y)*P_2)|(\vec{z})(Q_1|Q_2) \\ &= P'|Q'. \end{aligned}$$

The proof of (ii) is similar. □

The next lemma can be proved by simple induction on derivation.

Lemma 6 *If $P \overset{\alpha x}{\rightarrow} P'$ and $x \notin gn(P)$ then $P \overset{\alpha(x)}{\rightarrow} P'$.*

3.1 Local Bisimilarity

A bisimulation equivalence ([44,33]) should be neither too strong nor too weak. From an algebraic point of view, one looks for a congruence relation. However a bisimulation congruence is not always the best equivalence from an observational viewpoint. A bisimulation equivalence for χ-processes should take into account the distinguished feature of the localization operators of the language. The equivalence we introduce in this section is based upon the familiar idea that two pieces of program are considered observationally equivalent if and only if placing them in a same context results in two pieces of observationally equivalent program. Working explicitly with contexts is unnecessary in our setting due to the presence of the structural equality =.

Definition 7 *Let $\mathcal{R}$ be a subset of $\mathcal{C}\times\mathcal{C}$. The relation $\mathcal{R}$ is a local simulation if $P\mathcal{R}Q$ implies that for any process R and any sequence $\vec{x}$ of names it holds that*

if $(\vec{x})(P|R) \overset{\delta}{\Rightarrow} P'$ then Q' exists such that $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$.

The relation $\mathcal{R}$ is a local bisimulation if both $\mathcal{R}$ and its inverse are local simulations. The local bisimilarity $\approx$ is the largest local bisimulation.

Local bisimulations are closed under localization and composition operations at each bisimulation step. This is not at all a strong requirement. Any useful bisimulation equivalence should at least be closed under these operations; and if a bisimulation equivalence is closed under a particular operation, it is closed under that operation at each bisimulation step.

It is clear that $\approx$ is an equivalence relation. Let's say that a binary relation $\mathcal{R}$ on $\mathcal{C}$ is locally closed if $P\mathcal{R}Q$ implies $(\vec{x})(P|R)\mathcal{R}(\vec{x})(Q|R)$ for any $R \in \mathcal{C}$ and any sequence $\vec{x}$ of names. To show that a locally closed relation $\mathcal{R}$ is a local simulation, one only has to show that if $P\mathcal{R}Q$ then

there exists some Q' such that $Q \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$ whenever $P \overset{\delta}{\Rightarrow} P'$.

Local (bi)simulation can be defined in a more familiar way as given in the next lemma.

Lemma 8 *$\mathcal{R}$ is a local simulation if and only if $P\mathcal{R}Q$ implies that for any process R and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(P|R) \rightarrow P'$ then Q' exists such that $(\vec{x})(Q|R) \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(P|R) \xrightarrow{\delta} P'$ then Q' exists such that $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$.

The alternative definition is more useful in practice. The next lemma follows trivially from the definition. But it will be used again and again.

Lemma 9 *The following properties hold:*
(i) If $P \approx Q$ and $P \rightarrow P'$ then Q' exists such that $Q \rightarrow^* Q'$ and $P' \approx Q'$.
(ii) If $P \approx Q$ and $P \xrightarrow{\delta} P'$ then Q' exists such that $Q \overset{\delta}{\Rightarrow} Q'$ and $P' \approx Q'$.

As usual, local bisimulation up to $\approx$ is a useful tool for proving two χ-processes being locally bisimilar ([57]).

Definition 10 *Let $\mathcal{R}$ be a subset of $\mathcal{C}\times\mathcal{C}$. The relation $\mathcal{R}$ is a local simulation up to $\approx$ if $P\mathcal{R}Q$ implies that for any process R and any sequence $\vec{x}$ of names it holds that*

if $(\vec{x})(P|R) \overset{\delta}{\Rightarrow} P'$ then there exists some Q' such that $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q'$ and $P' \approx \mathcal{R} \approx Q'$.

$\mathcal{R}$ is a local bisimulation up to $\approx$ if both $\mathcal{R}$ and its inverse are local simulations up to $\approx$.

A local bisimulation $\mathcal{R}$ up to $\approx$ satisfies the standard property of being contained in $\approx$.

Lemma 11 *Suppose that $\mathcal{R} \subset \mathcal{C} \times \mathcal{C}$ satisfies the following property: if $P\mathcal{R}Q$, then for any process R and any sequence $\vec{x}$ of names, it holds that*
(i) if $(\vec{x})(P|R) \rightarrow P'$ then Q' and Q'' exist such that $(\vec{x})(Q|R) \rightarrow^* Q' \approx Q''$ and $P'\mathcal{R}Q''$;
(ii) if $(\vec{x})(Q|R) \rightarrow Q'$ then P' and P'' exist such that $(\vec{x})(P|R) \rightarrow^* P' \approx P''$ and $P''\mathcal{R}Q'$;
(iii) if $(\vec{x})(P|R) \xrightarrow{\delta} P'$ then Q' and Q'' exist such that $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q' \approx Q''$ and $P'\mathcal{R}Q''$;
(iv) if $(\vec{x})(Q|R) \xrightarrow{\delta} Q'$ then P' and P'' exist such that $(\vec{x})(P|R) \overset{\delta}{\Rightarrow} P' \approx P''$ and $P''\mathcal{R}Q'$.
Then $\mathcal{R}$ is a local bisimulation up to $\approx$.

Proof: Using lemma 9, it is easy to show that if $P\mathcal{R}Q$ and $(\vec{x})(P|R) \rightarrow^* Q$ (or $(\vec{x})(P|R) \overset{\delta}{\Rightarrow} Q$) then Q' exists such that $(\vec{x})(Q|R) \rightarrow^* Q'$ (or $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q'$) and $P' \approx \mathcal{R} \approx Q'$. $\square$

In the rest of this section, we prove that $\approx$ is a congruence relation. We establish a few technical lemmas first.

Lemma 12 *If $P \rightarrow^* P_1 \approx Q$ and $Q \rightarrow^* Q_1 \approx P$ then $P \approx Q$.*

Proof: Suppose $\vec{x}$ are names and R a χ-process. If for example $(\vec{x})(P|R) \xrightarrow{\delta} P'$, then some Q' exists such that $(\vec{x})(Q_1|R) \overset{\delta}{\Rightarrow} Q'$ and $P' \approx Q'$. But then $(\vec{x})(Q|R) \overset{\delta}{\Rightarrow} Q'$. $\square$

To show that local bisimilarity $\approx$ is closed under substitution, we need the following auxiliary result.

Lemma 13 *Suppose a does not appear in P. If $(\vec{x})(P|R) \rightarrow (\vec{x'})(P'|R')$ is induced by a communication within P and $(\vec{x'})(P'|R') \xrightarrow{ay} (\vec{x''})(P''|R'')$ is induced by an action from R', then $\vec{x_1}, P_1, R_1$ exist such that $(\vec{x})(P|R) \xrightarrow{ay} (\vec{x_1})(P_1|R_1)$ is induced by the same action and $(\vec{x_1})(P_1|R_1) \rightarrow (\vec{x''})(P''|R'')$ is induced by the same communication.*

Proof: This is an easy proof using Lemma 4 and Lemma 3. $\square$

The next lemma is crucial in showing that $\approx$ is a congruence relation. It is the first indication that local bisimilarity is algebraically appropriate.

Lemma 14 *If $P \approx Q$ then $P\sigma \approx Q\sigma$ for an arbitrary substitution σ.*

Proof: Suppose $P \approx Q$. We only have to show that for two arbitrary names x, y one has that $P[y/x] \approx Q[y/x]$. Let a be a fresh name. Then $(x)(P|a[x]) \xrightarrow{ay} P[y/x]$. So Q_1 exists such that $(x)(Q|a[x]) \overset{ay}{\Rightarrow} Q_1 \approx P[y/x]$, which can be factorized as

$$\begin{aligned}(x)(Q|a[x]) &\rightarrow^* (x)(Q_2|a[x]) \\ &\xrightarrow{ay} Q_2[y/x] \\ &\rightarrow^* Q_1.\end{aligned}$$

By Lemma 13, this sequence of actions can be reorganized as follows:

$$\begin{aligned}(x)(Q|a[x]) &\xrightarrow{ay} Q[y/x] \\ &\rightarrow^* Q_3 \\ &\rightarrow^* Q_1.\end{aligned}$$

Similarly some P_1 exists such that $P[y/x] \rightarrow^* P_1 \approx Q[y/x]$. By Lemma 12, $P[y/x] \approx Q[y/x]$. $\square$

We now come to the main result of the section.

Theorem 15 *$\approx$ is a congruence equivalence: if $P \approx Q$ and $O \in \mathcal{C}$ then*
(i) $\alpha[x].P \approx \alpha[x].Q$;
(ii) $P|O \approx Q|O$;
(iii) $(x)P \approx (x)Q$;
(iv) $\alpha(x)*P \approx \alpha(x)*Q$.

Proof: We prove only (ii) and (iv). The other two can be proved similarly. (ii) We show that $\{(P|O, Q|O) \mid P \approx Q \wedge O \in \mathcal{C}\}$ is a local bisimulation. Suppose $R \in \mathcal{C}$ and $\vec{x}$ is a sequence of names. Then clearly $(\vec{x})((P|O)|R) = (\vec{x})(P|(O|R))$ and $(\vec{x})((Q|O)|R) = (\vec{x})(Q|(O|R))$. So this case follows immediately from definition. (iv) Let $\mathcal{R}$ be the following locally closed relation

$$\{((\vec{x})(m(y)*P|R), (\vec{x})(m(y)*Q|R)) \mid P \approx Q,\ R \in \mathcal{C},\ m, \vec{x} \text{ names}\}.$$

Suppose $(\vec{x})(m(y)*P|R)\mathcal{R}(\vec{x})(m(y)*Q|R)$ and $(\vec{x})(m(y)*P|R) \rightarrow P'$. There are these cases:

- $(\vec{x})(m(y)*P|R) \rightarrow P'$ is induced by a communication within R. Then $P' =$

$(\vec{x'})(m\sigma(y)*P\sigma|R')$ for some substitution σ. But then $(\vec{x})(m(y)*Q|R) \rightarrow (\vec{x'})(m\sigma(y)*Q\sigma|R')$. By Lemma 14, $P\sigma \approx Q\sigma$. Therefore

$$(\vec{x'})(m\sigma(y)*P\sigma|R')\mathcal{R}(\vec{x'})(m\sigma(y)*Q\sigma|R').$$

- $(\vec{x})(m(y)*P|R) \rightarrow P'$ is induced by a communication between $m(y)*P$ and R. Then P' is of the form $(\vec{x})(m(y)*P|P[a/y]|R')$. Similarly

$$(\vec{x})(m(y)*Q|R) \rightarrow (\vec{x})(m(y)*Q|Q[a/y]|R').$$

  By Lemma 14 $P[a/y] \approx Q[a/y]$. By (ii) and (iii), $(\vec{x})(m(y)*Q|P[a/y]|R') \approx (\vec{x})(m(y)*Q|Q[a/y]|R')$.
- Similarly, if $(\vec{x})(m(y)*P|R) \xrightarrow{\delta} P'$, then $(\vec{x})(m(y)*Q|R) \xrightarrow{\delta} Q'$ and $P'\mathcal{R}Q'' \approx Q'$ for some Q' and Q''.

So the conditions of Lemma 11 are satisfied. It then follows that $\mathcal{R}$ is a local bisimulation up to $\approx$. $\square$

The actions of the χ-calculus can be classified into three groups: the input actions of the form $\xrightarrow{\alpha x}$, the (free) output actions of the form $\xrightarrow{\alpha[x]}$ and the restricted output actions of the form $\xrightarrow{\alpha(x)}$. In defining local bisimilarity, we assume that all these actions are observable. What if we ignore actions of a certain type. In other words, what bisimilarities do we obtain if only actions in one or two of the three groups are declared observable? We now look into two bisimulation equivalence relations which distinguish two processes only when they fail to simulate each other's free, respectively restricted, output actions.

Definition 16 *Suppose $\mathcal{R}$ is a binary relation on χ-processes. It is called an output bisimulation if $P\mathcal{R}Q$ implies that for any R and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(P|R) \overset{\alpha[y]}{\Longrightarrow} P'$ then Q' exists such that $(\vec{x})(Q|R) \overset{\alpha[y]}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(Q|R) \overset{\alpha[y]}{\Longrightarrow} Q'$ then P' exists such that $(\vec{x})(P|R) \overset{\alpha[y]}{\Longrightarrow} P'$ and $P'\mathcal{R}Q'$.
The output bisimilarity $\approx_o$ is the largest output bisimulation.

For $\approx_o$ the observables are free output actions. A seemingly different bisimilarity is obtained if the observables are confined to restricted output actions.

Definition 17 *Suppose $\mathcal{R}$ is a binary relation on χ-processes. It is called a restricted output bisimulation if $P\mathcal{R}Q$ implies that for any R and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(P|R) \overset{\alpha(y)}{\Longrightarrow} P'$ then Q' exists such that $(\vec{x})(Q|R) \overset{\alpha(y)}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(Q|R) \overset{\alpha(y)}{\Longrightarrow} Q'$ then P' exists such that $(\vec{x})(P|R) \overset{\alpha(y)}{\Longrightarrow} P'$ and $P'\mathcal{R}Q'$.
The restricted output bisimilarity $\approx_{ro}$ is the largest restricted output bisimulation.

Definition 16 and Definition 17 suggest immediately another bisimilarity which equates two processes if no environment can tell them apart under the assumption that the only observables are input actions. We will come back to it later on.

The next theorem renders redundant any further study of the two equivalence relations just defined. It also implies that the algebraic theory of χ-calculus would not be affected if restricted output actions are ignored.

Theorem 18 *$\approx$, $\approx_o$ and $\approx_{ro}$ are one and the same relation.*

PROOF: The proof consists of following parts:

- It is clear that $\approx \subseteq \approx_o$ and $\approx \subseteq \approx_{ro}$.
- Suppose $P \approx_o Q$ and $P \xrightarrow{\alpha x} P'$. Let a be a fresh name throughout the rest of the proof. Now $P|\overline{\alpha}[x].a[a] \overset{a[a]}{\Rightarrow} P'$ must be matched up by $Q|\overline{\alpha}[x].a[a] \overset{a[a]}{\Rightarrow} Q'$ for some Q' such that $P' \approx_o Q'$. It follows that $Q \overset{\alpha x}{\Rightarrow} Q'$.
- Suppose $P \approx_o Q$ and $P \xrightarrow{\alpha(x)} P'$. As in the previous case one obtains some Q' such that $Q \overset{\alpha x}{\Rightarrow} Q'$ and $P' \approx_o Q'$. By Lemma 6, $Q \overset{\alpha(x)}{\Rightarrow} Q'$.
- Suppose $P \approx_{ro} Q$ and $P \xrightarrow{\alpha[x]} P'$. Then $(x)(P|a[x]) \xrightarrow{\alpha(x)} P'|a[x]$. So Q_1 exists such that $(x)(Q|a[x]) \overset{\alpha(x)}{\Rightarrow} Q_1$ and $P'|a[x] \approx_{ro} Q_1$. Moreover $(x)(P'|a[x]) \xrightarrow{a(x)} P'$ must be simulated by $(x)Q_1 \overset{a(x)}{\Rightarrow} Q' \approx_{ro} P'$ for some Q'. It follows that Q_1 must be of the form $Q_1'|a[x]$ and $Q \overset{\alpha[x]}{\Rightarrow} Q_1' \Rightarrow Q'$.

We are done by noticing that both $\approx_o$ and $\approx_{ro}$ are by definition closed under composition and localization operations. $\square$

3.2 Incremental Bisimilarity

Local bisimulation as defined in Definition 7 is well-motivated. But they have the obvious problem of being highly intractable. In many circumstances, a much more manageable description is desirable. This section provides a sharper characterization of local bisimilarity. Let $go(P)$ be the set of global objective names appeared in P.

Definition 19 *Let $\mathcal{R}$ be a subset of $\mathcal{C}\times\mathcal{C}$. The relation $\mathcal{R}$ is an incremental simulation if $P\mathcal{R}Q$ implies that*
(i) if $P \rightarrow P'$ then there exists some Q' such that $Q \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $P \xrightarrow{\delta} P'$ then there exists some Q' such that $Q \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
and if $go(P|Q) \neq \emptyset$ then for some fresh name a and each name x in $go(P|Q)$
(iii) if $(x)(P|a[x]) \rightarrow P'$ then Q' exists such that $(x)(Q|a[x]) \rightarrow^* Q'$ and

$P'\mathcal{R}Q'$;
(iv) if $(x)(P|a[x]) \xrightarrow{\delta} P'$ then Q' exists such that $(x)(Q|a[x]) \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$.
The relation $\mathcal{R}$ is an incremental bisimulation if both $\mathcal{R}$ and its inverse are incremental simulations. The incremental bisimilarity $\approx_r$ is the largest incremental bisimulation.

It can be easily seen that Lemma 12 and Lemma 14 hold for $\approx_r$ as well. The following is a simple yet useful technical lemma.

Lemma 20 *Suppose $a \notin n(P|Q)$. Then*
(i) $(x)(P|a[x]) \approx_r (x)(Q|a[x])$ implies $P \approx_r Q$;
(ii) $P|a[x] \approx_r Q|a[x]$ implies $P \approx_r Q$.

Proof: (i) As $(x)(P|a[x]) \xrightarrow{a(x)} P$, Q_1 exists such that $(x)(Q|a[x]) \overset{a(x)}{\Rightarrow} Q_1 \approx_r P$. Now $(x)(Q|a[x]) \overset{a(x)}{\Rightarrow} Q_1$ implies that $Q \rightarrow^* Q_1$. Similarly P_1 exists such that $P \rightarrow^* P_1 \approx_r Q$. By Lemma 12, $P \approx_r Q$. (ii) is proved similarly. □

Lemma 21 *Suppose P, Q and O are χ-processes. Let $\vec{x}$, $\vec{y}$ and $\vec{a}$ be names such that $\vec{a}$ are pairwise distinct, $|\vec{a}| = |\vec{y}|$ and $\{\vec{a}\} \cap (\{\vec{x}, \vec{y}\} \cup gn(P|Q)) = \emptyset$. If $P \approx_r Q$ then $(\vec{x})(P|O|\vec{a}[\vec{y}]) \approx_r (\vec{x})(Q|O|\vec{a}[\vec{y}])$.*

Proof: Let $\mathcal{R}$ be the following relation

$$\left\{ ((\vec{x})(P|O|\vec{a}[\vec{y}]), (\vec{x})(Q|O|\vec{a}[\vec{y}])) \;\middle|\; \begin{array}{l} P \approx_r Q \text{ and } O \in \mathcal{C} \\ \vec{x}, \vec{y} \text{ and } \vec{a} \text{ satisfy the} \\ \text{condition of the lemma} \end{array} \right\}.$$

Suppose $(\vec{x})(P|O|\vec{a}[\vec{y}])\mathcal{R}(\vec{x})(Q|O|\vec{a}[\vec{y}])$. We examine some major cases:

- $(\vec{x})(P|O|\vec{a}[\vec{y}]) \xrightarrow{\alpha z} (\vec{x'})(P_1|O[z/x_1]|\vec{a}[\vec{y'}])$ is induced by an action from P that substitutes z for some x_1 in $\{\vec{x}\}$. Then $(x_1)(P|b[x_1]) \xrightarrow{\alpha z} P_1|b[z]$ for some new b. So some Q_1 exists such that $(x_1)(Q|b[x_1]) \overset{\alpha z}{\Rightarrow} Q_1|b[z]$ and $P_1|b[z] \approx_r Q_1|b[z]$. The former implies $(\vec{x})(Q|O|\vec{a}[\vec{y}]) \overset{\alpha z}{\Rightarrow} (\vec{x'})(Q_1|O[z/x_1]|\vec{a}[\vec{y'}])$ and the latter implies $P_1 \approx_r Q_1$ by Lemma 20.
- $(\vec{x})(P|O|\vec{a}[\vec{y}]) \rightarrow (\vec{x'})(P_1|O_1|\vec{a}[\vec{y'}])$ is induced by a communication within P in which some $x_1 \in \{\vec{x}\}$ participates and is replaced by some global z. Let b be a fresh name. Then $(x_1)(P|b[x_1]) \rightarrow P_1|b[z]$. So Q_1 exists such that $(x_1)(Q|b[x_1]) \rightarrow^* Q_1|b[z] \approx_r P_1|b[z]$. It follows by Lemma 20 that $P_1 \approx_r Q_1$. Also $(\vec{x})(Q|O|\vec{a}[\vec{y}]) \rightarrow^* (\vec{x'})(Q_1|O_1|\vec{a}[\vec{y'}])$.
- $(\vec{x})(P|O|\vec{a}[\vec{y}]) \rightarrow (\vec{x'})(P_1|O|\vec{a}[\vec{y}])$ is induced by a communication in P between some $\alpha[x_1].A$ and $\overline{\alpha}[x_1].B$ where $x_1 \in \{\vec{x}\}$. Then $(x_1)(P|b[x_1]) \rightarrow$

$(x_1)(P_1|b[x_1])$ for some fresh b. Therefore $(x_1)(Q|b[x_1]) \rightarrow^* (x_1)(Q_1|b[x_1])$ and $(x_1)(P_1|b[x_1]) \approx_r (x_1)(Q_1|b[x_1])$. By Lemma 20, $P_1 \approx_r Q_1$. It is also clear that $(\vec{x})(Q|O|\vec{a}[\vec{y}]) \rightarrow^* (\vec{x'})(Q_1|O|\vec{a}[\vec{y}])$.
- $(\vec{x})(P|O|\vec{a}[\vec{y}]) \stackrel{\alpha z}{\rightarrow} (\vec{x'})(P[z/x_1]|O_1|\vec{a}[\vec{y'}])$ is induced by an action in O that replaces some $x_1 \in \{\vec{x}\}$ by z. Then $(\vec{x})(Q|O|\vec{a}[\vec{y}]) \stackrel{\alpha z}{\rightarrow} (\vec{x'})(Q[z/x_1]|O_1|\vec{a}[\vec{y'}])$. By Lemma 14, $P[z/x_1] \approx_r Q[z/x_1]$.
- $(\vec{x})(P|O|\vec{a}[\vec{y}]) \rightarrow (\vec{x'})(P_1|O_1|\vec{a}[\vec{y'}])$ is induced by a communication between some $\alpha[x_1].A$ in P and some $\overline{\alpha}[z].B$ in O where $x_1 \in \{\vec{x}\}$. Then for some fresh b, $(x_1)(P|b[x_1]) \stackrel{\alpha z}{\rightarrow} P_1|b[z]$. So Q_1 exists such that $(x_1)(Q|b[x_1]) \stackrel{\alpha z}{\Rightarrow} Q_1|b[z]$. Now $P_1 \approx_r Q_1$ follows from Lemma 20 and also $(\vec{x})(Q|O|\vec{a}[\vec{y}]) \rightarrow^* (\vec{x'})(Q_1|O_1|\vec{a}[\vec{y'}])$.

Conclude that $\mathcal{R}$ is an incremental bisimulation. □

A consequence of this lemma is that $\approx_r$ is closed under parallel composition and localization operation.

Corollary 22 *Suppose P, Q, O are χ-processes. If $P \approx_r Q$ then $P|O \approx_r Q|O$ and $(x)P \approx_r (x)Q$.*

Theorem 23 *$\approx_r$ is the same as $\approx$.*

Proof: $\approx \subseteq \approx_r$ is obvious. The reverse inclusion holds by Corollary 22 and the definitions of $\approx$ and $\approx_r$. □

It follows from Theorem 15 and Theorem 23 that $\approx_r$ is closed under all combinators.

Another immediate consequence of Theorem 23 is that one can confine one's attention to finite R in Definition 7. This is a handy property when it comes to proving conservativity results. Thus the χ-calculus is a conservative extension over the subcalculus of finite χ-processes, which is in turn conservative over the calculus of reaction graphs.

3.3 Barbed Bisimilarity

To study the algebraic property of the χ-calculus, we have introduced local bisimulations. They are not very tractable technical tool, but seem to be the most reasonable one for χ-like process algebra. Another sensible class of bisimulations is that of barbed bisimulations introduced in [42]. This section takes a look at barbed bisimilarity on χ-processes.

Definition 24 *A process P is strongly barbed at a, notation $P{\downarrow}a$, if $P \xrightarrow{\delta} P'$ for some P' such that the subject name of δ is a. P is barbed at a, notation $P{\Downarrow}a$, if some P' exists such that $P \rightarrow^* P'{\downarrow}a$. A binary relation $\mathcal{R}$ is barbed if $\forall a \in \mathcal{N}.P{\Downarrow}a \Leftrightarrow Q{\Downarrow}a$ whenever $P\mathcal{R}Q$.*

Our definition of barbed bisimulation is more similar to Honda and Yoshida's ([27]) than to Milner and Sangiorgi's ([42]).

Definition 25 *Suppose $\mathcal{R}$ is a barbed relation on χ-processes. It is called a barbed bisimulation if $P\mathcal{R}Q$ implies that for any R and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(P|R) \rightarrow P'$ then Q' exists such that $(\vec{x})(Q|R) \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(Q|R) \rightarrow Q'$ then P' exists such that $(\vec{x})(P|R) \rightarrow^* P'$ and $P'\mathcal{R}Q'$.
The barbed bisimilarity $\approx_b$ is the largest barbed bisimulation.

In the standard definition of barbed bisimulation, contexts of certain type are added at the beginning. The resulting barbed bisimilarity is closed under contexts of that type. In our definition, closure of contexts of certain type is required at each bisimulation step. The resulting barbed bisimilarity is also closed under contexts of that type. It is then obvious that the two definitions give rise to the same barbed bisimilarity, the largest barbed bisimulation, the reason being that if a bisimulation is closed under contexts of certain type it is closed under contexts of that type at each bisimulation step. Since we are only interested in the largest barbed bisimulation, it makes no difference which definition is adopted.

The barbed bisimilarity can be defined in a way that does not mention explicitly the notion of barb. We now define another bisimilarity congruence which turns out to be the same as barbed bisimilarity.

Definition 26 *Suppose $\mathcal{R}$ is a binary relation on χ-processes. It is called an input bisimulation if $P\mathcal{R}Q$ implies that for any R and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(P|R) \overset{\alpha y}{\Rightarrow} P'$ then Q' exists such that $(\vec{x})(Q|R) \overset{\alpha y}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
(ii) if $(\vec{x})(Q|R) \overset{\alpha y}{\Rightarrow} Q'$ then P' exists such that $(\vec{x})(P|R) \overset{\alpha y}{\Rightarrow} P'$ and $P'\mathcal{R}Q'$.
The input bisimilarity $\approx_i$ is the largest input bisimulation.

Theorem 27 *$\approx_i$ is a congruence equivalence.*

Proof: If we replace every occurrence of $\approx$ by $\approx_i$ in the proof of Lemma 14, we obtain a proof of the following fact: If $P \approx_i Q$ then $P\sigma \approx_i Q\sigma$ for every substitution σ. So the proof of Theorem 15 actually establishes the congruence property for $\approx_i$. $\square$

The barbed bisimilarity is a congruence relation. The proof is similar to that of Theorem 15. Here we give an indirect proof.

Theorem 28 $\approx_b$ *is the same as* $\approx_i$.

PROOF: Clearly $\approx_i$ is barbed. So $\approx_i \subseteq \approx_b$. Conversely suppose that $P \approx_b Q$ and $P \overset{\alpha x}{\Rightarrow} P'$. Then $P|\overline{\alpha}[x].(y)b[y] \overset{by}{\Rightarrow} P'$ for some fresh b. So $Q|\overline{\alpha}[x].(y)b[y] \overset{by}{\Rightarrow} Q'$ for some Q' such that $P' \approx_b Q'$. It follows that $Q \overset{\alpha x}{\Rightarrow} Q'$. That is $\approx_b$ is an input bisimulation. $\square$

For CCS processes, barbed bisimilarity coincides with bisimilarity. For π-processes with *binary* choice operator, the problem of whether the two equivalences are the same is still open at the time of writing. The picture in χ-calculus is different. Local bisimilarity has strictly stronger distinguishing power than barbed bisimilarity. Using the choice combinator, we have

$$(x)a[x].(b)(\bar{b}[x]|b[z]) \not\approx a[z]+(x)a[x].(b)(\bar{b}[x]|b[z])$$

but

$$(x)a[x].(b)(\bar{b}[x]|b[z]) \approx_b a[z]+(x)a[x].(b)(\bar{b}[x]|b[z]).$$

This counter example can be couched in present calculus without the choice operator:

$$a(x)*(b)(\bar{b}[x]|b[z]) \not\approx a[z]|a(x)*(b)(\bar{b}[x]|b[z])$$

yet

$$a(x)*(b)(\bar{b}[x]|b[z]) \approx_b a[z]|a(x)*(b)(\bar{b}[x]|b[z]).$$

On the other hand $\approx$ is clearly barbed. Therefore one has

Theorem 29 *The inclusion* $\approx \subset \approx_b$ *is strict.*

We have not adopted barbed bisimilarity for a number of reasons. First barbed bisimulations deal with communications rather than observable actions of processes, which implies that the bisimulation argument is necessary more involved. For one thing, one has to prove that a barbed bisimulation is barbed; and that is sometimes messy. Second the barbed bisimilarity could be too weak when comparing χ to other process calculi.

4 Pi Processes as Chi Processes

The χ-calculus can be seen as obtained from the π-calculus ([41]) by replacing the variable names by local names. A question naturally arises as to the relationship between the two languages. We give our answers to the question in this section.

4.1 Theoretical Result

The χ-calculus we consider in this paper is the minimal χ in the sense that we have omitted choice and match operators. Consequently we will confine our attention to π-calculus without these operators. Let $\mathcal{P}$ be the set of π-processes defined as follows:

$$P := \mathbf{0} \mid m(x).P \mid \overline{m}[x].P \mid P|P' \mid (x)P \mid m(x)*P.$$

Here $m(x).P$ and $\overline{m}[x].P$ are processes of input, respectively output prefix form. We depart from the standard syntax for output prefix operator of π-calculus purely for the purpose of comparison to the prefix operator in χ-calculus. In $m(x).P$, $(x)P$ and $m(x)*P$, the name x is bound. A name is free if it is not bound. $bn(P)$, respectively $fn(P)$, denotes the set of bound, respectively free, names occurred in P. The union of $bn(P)$ and $fn(P)$ is denoted by $n(P)$. The α-convention is adopted and a structural equality is imposed on the π-processes in the same way as is done with the χ-processes. The operational semantics of π-calculus is defined by the following reduction rules

$$m(x).P|\overline{m}[y].Q \rightarrow P[y/x]|Q$$

$$m(x)*P|\overline{m}[y].Q \rightarrow m(x)*P|P[y/x]|Q$$

together with the structural rules given in Section 2. Let $\{\overset{mx}{\rightarrow}, \overset{\overline{m}[x]}{\rightarrow}, \overset{\overline{m}(x)}{\rightarrow} |m, x \in \mathcal{N}\}$ be ranged over by $\overset{\delta}{\rightarrow}$. $bn(\delta)$ is $\{x\}$ if $\delta = \overline{a}(x)$ for some $a \in \mathcal{N}$; it is the empty set otherwise. $n(\delta)$ is the set of names in δ. The labeled transition system for π-processes is defined as follows:

$$\frac{}{m(y).P \overset{mx}{\rightarrow} P[x/y]} \qquad \frac{}{m(y)*P \overset{mx}{\rightarrow} m(y)*P|P[x/y]}$$

$$\frac{}{\overline{m}[x].P \overset{\overline{m}[x]}{\rightarrow} P} \qquad \frac{P \overset{\overline{m}[x]}{\rightarrow} P'}{(x)P \overset{\overline{m}(x)}{\rightarrow} P'} \qquad \frac{}{m(x)*P \overset{m(x)}{\rightarrow} m(x)*P|P}$$

$$\frac{P \xrightarrow{\delta} P' \quad bn(\beta) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{\delta} P'|Q} \qquad \frac{P \xrightarrow{\delta} P' \quad x \notin n(\beta)}{(x)P \xrightarrow{\delta} (x)P'}$$

Many bisimulation equivalences on π-processes have been proposed in literature ([41,51,56,12,24]). What is most relevant in this section is the open bisimilarity defined in [56]. Actually we will use a version of open bisimilarity stronger than Sangiorgi's in that it does not have a separate treatment to localization operator.

Definition 30 *Let $\mathcal{R}$ be a binary relation on the set of π-processes. The relation $\mathcal{R}$ is an open bisimulation if $P\mathcal{R}Q$ implies that for any substitution σ it holds that*
(i) if $P\sigma \rightarrow P'$ then there exists some Q' such that $Q\sigma \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $Q\sigma \rightarrow Q'$ then there exists some P' such that $P\sigma \rightarrow^* P'$ and $P'\mathcal{R}Q'$;
(iii) if $P\sigma \xrightarrow{\delta} P'$ then there exists some Q' such that $Q\sigma \overset{\delta}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
(iv) if $Q\sigma \xrightarrow{\delta} Q'$ then there exists some P' such that $P\sigma \overset{\delta}{\Rightarrow} P'$ and $P'\mathcal{R}Q'$.
The open bisimilarity $\approx^o$ is the largest open bisimulation.

This open bisimilarity is a congruence equivalence and closed under substitution.

A structural translation from π-processes to χ-processes can be defined as follows:

$$\begin{aligned}
(\mathbf{0})^\circ &\stackrel{\text{def}}{=} \mathbf{0}, \\
(m(x).P)^\circ &\stackrel{\text{def}}{=} (x)m[x].P^\circ, \\
(\overline{m}[x].P)^\circ &\stackrel{\text{def}}{=} \overline{m}[x].P^\circ, \\
(P|Q)^\circ &\stackrel{\text{def}}{=} P^\circ|Q^\circ, \\
((x)P)^\circ &\stackrel{\text{def}}{=} (x)P^\circ, \\
(m(x){*}P)^\circ &\stackrel{\text{def}}{=} m(x){*}P^\circ.
\end{aligned}$$

The following property can be easily verified.

Lemma 31 *Suppose P is a π-process. If $P^\circ \rightarrow P'$ ($P^\circ \xrightarrow{\delta} P'$) then P_1 exists such that $P' = P_1^\circ$ and $P^\circ \rightarrow P_1^\circ$ ($P^\circ \xrightarrow{\delta} P_1^\circ$).*

The next theorem shows that the translation is faithful operationally. It is proved by induction on derivation.

Theorem 32 *For $P, Q \in \mathcal{P}$, it holds that*
(i) $P \rightarrow Q$ if and only if $P^\circ \rightarrow Q^\circ$;

(ii) $P \overset{mx}{\rightarrow} Q$ *if and only if* $P^\circ \overset{mx}{\rightarrow} Q^\circ$*;*
(iii) $P \overset{\overline{m}[x]}{\rightarrow} Q$ *if and only if* $P^\circ \overset{\overline{m}[x]}{\rightarrow} Q^\circ$*;*
(iv) $P \overset{\overline{m}(x)}{\rightarrow} Q$ *if and only if* $P^\circ \overset{\overline{m}(x)}{\rightarrow} Q^\circ$*.*

PROOF: We will prove (ii). If $P \overset{mx}{\rightarrow} Q$ then by structural induction one can show that some $\vec{z}$, y, P_1 and P_2 exist such that either

$$P = (\vec{z})(P_1|m(y).P_2) \text{ and } Q = (\vec{z})(P_1|P_2[x/y])$$

or

$$P = (\vec{z})(P_1|m(y)*P_2) \text{ and } Q = (\vec{z})(P_1|P_2[x/y]|m(y)*P_2).$$

In the former case

$$P^\circ = (\vec{z})(y)(P_1^\circ|m[y].P_2^\circ) \overset{mx}{\rightarrow} (\vec{z})(P_1^\circ|P_2^\circ[x/y]) = Q^\circ$$

and in the latter case

$$P^\circ = (\vec{z})(P_1^\circ|m(y)*P_2^\circ) \overset{mx}{\rightarrow} (\vec{z})(P_1^\circ|P_2^\circ[x/y]|m(y)*P_2^\circ) = Q^\circ.$$

Suppose now $P^\circ \overset{mx}{\rightarrow} Q^\circ$. If P is of the form $P_1|P_2$ and $P_1^\circ \overset{mx}{\rightarrow} Q_1^\circ$ then by induction on derivation $P_1 \overset{mx}{\rightarrow} Q_1$. So $P = P_1|P_2 \overset{mx}{\rightarrow} Q_1|P_2 = Q$. Other cases are equally simple. Conclude that $P^\circ \overset{mx}{\rightarrow} Q^\circ$ always implies $P \overset{mx}{\rightarrow} Q$. □

The translation is also faithful algebraically. The next two lemmas are used in proving this fact.

Lemma 33 *Suppose P is a π-process, R is a χ-process and $\vec{x}$ is a sequence of names. If $(\vec{x})(P^\circ|R) \rightarrow P'$ is induced by a communication within P° then $P' = (\vec{x})(P_1^\circ|R)$ and $P \rightarrow P_1$.*

PROOF: If $(\vec{x})(P^\circ|R) \rightarrow P'$ is induced by a communication within P° then $P^\circ \rightarrow P''$ for some P''. This is because all object names in input prefixes in P° are local in P°. By Lemma 31, $P^\circ \rightarrow P_1^\circ$ for some P_1 such that $P_1^\circ = P''$. By Theorem 32, $P \rightarrow P_1$. □

Lemma 34 *Suppose $P, Q \in \mathcal{P}$, $P \approx^o Q$, $R \in \mathcal{C}$ and $\vec{x}$ is a sequence of names. The following properties hold:*
(i) If $(\vec{x})(R|P^\circ) \rightarrow P'$, then $\vec{x'}$, R', P_1 and Q_1 exist such that $P' = (\vec{x'})(R'|P_1^\circ)$,

$Q \rightarrow^* (\vec{x'})(R'|Q_1^\circ)$ and $P_1 \approx^o Q_1$.
(ii) If $(\vec{x})(R|P^\circ) \xrightarrow{\delta} P'$, then $\vec{x'}$, R', P_1 and Q_1 exist such that $P' = (\vec{x'})(R'|P_1^\circ)$, $Q \overset{\delta}{\Rightarrow} (\vec{x'})(R'|Q_1^\circ)$ and $P_1 \approx^o Q_1$.

Proof: The proof of (ii) is simpler than that of (i). So we concentrate on (i). There are three cases:

- $(\vec{x})(R|P^\circ) \rightarrow P'$ is induced by a communication within R. Then $P' = (\vec{x'})(R'|P^\circ\sigma)$. So $(\vec{x})(R|Q^\circ) \rightarrow (\vec{x'})(R'|Q^\circ\sigma)$ and $P\sigma \approx^o Q\sigma$.
- $(\vec{x})(R|P^\circ) \rightarrow P'$ is induced by a communication in P°. Then by Lemma 33, $P' = (\vec{x})(R|P_1^\circ)$ and $P \rightarrow P_1$. So Q_1 exists such that $Q \rightarrow^* Q_1$ and $P_1 \approx^o Q_1$. By Theorem 32, $(\vec{x})(R|Q^\circ) \rightarrow^* (\vec{x})(R|Q_1^\circ)$.
- $(\vec{x})(R|P^\circ) \rightarrow P'$ is induced by a communication between R and P°. Without losing any generality, assuming $\vec{x}$ is just x. There are several subcases:
  · $R = (\vec{a})(S|\overline{m}[y].T)$, $P^\circ = (\vec{b})(A^\circ|(z)m[z].B^\circ)$ and
  $$(x)(R|P^\circ) \rightarrow (x)(\vec{a})(S|T|(\vec{b})(A^\circ|B^\circ[y/z]))$$
  is induced by the communication through m. Then $P^\circ \xrightarrow{my} (\vec{b})(A^\circ|B^\circ[y/z])$. By Theorem 32, $P \xrightarrow{my} (\vec{b})(A|B[y/z])$. As $P \approx^o Q$, Q_1 exists such that $Q \overset{my}{\Rightarrow} Q_1$ and $(\vec{b})(A|B[y/z]) \approx^o Q_1$. So $Q^\circ \overset{my}{\Rightarrow} Q_1^\circ$. Hence $(x)(R|Q^\circ) \rightarrow^* (x)(\vec{a})(S|T|Q_1^\circ)$ by Lemma 5.
  · $R = (\vec{a})(y)(S|m[y].T)$, $P^\circ = (\vec{b})(A^\circ|\overline{m}[z].B^\circ)$ and
  $$(x)(R|P^\circ) \rightarrow (x)(\vec{b})((\vec{a})(S[z/y]|T[z/y])|A^\circ|B^\circ)$$
  is induced by the communication through m. The argument is similar to the one in the above case.
  · $R = (\vec{a})(S|m[y].T)$, $P^\circ = (\vec{b})(A^\circ|\overline{m}[x].B^\circ)$ and $(x)(R|P^\circ) \rightarrow P'$ is
  $$(x)(R|P^\circ) \rightarrow (\vec{a})(S[y/x]|T[y/x]|(\vec{b})(A^\circ[y/x]|B^\circ[y/x])).$$
  Then $P \xrightarrow{\overline{m}[x]} (\vec{b})(A|B)$. It follows that Q_1 exists such that $Q \overset{\overline{m}[x]}{\Rightarrow} Q_1 \approx^o (\vec{b})(A|B)$. So $Q_1[y/x] \approx^o (\vec{b})(A[y/x]|B[y/x])$. By Lemma 5 and Lemma 3, one has $(x)(R|Q^\circ) \rightarrow^* (\vec{a})(S[y/x]|T[y/x]|Q_1[y/x]^\circ)$.
  · $R = (\vec{a})(S|m[x].T)$, $P^\circ = (\vec{b})(A^\circ|\overline{m}[y].B^\circ)$ and $(x)(R|P^\circ) \rightarrow P'$ is
  $$(x)(R|P^\circ) \rightarrow (\vec{b})((\vec{a})(S[y/x]|T[y/x])|A^\circ[y/x]|B^\circ[y/x]).$$
  The proof is similar to that in the previous case.

This completes the proof. □

Theorem 35 *For $P, Q \in \mathcal{P}$, $P \approx^o Q$ if and only if $P^\circ \approx Q^\circ$.*

PROOF: $\Rightarrow$: Let $\mathcal{R}$ be $\{((\vec{x})(P^\circ|R), (\vec{x})(Q^\circ|R)) \mid P \approx^o Q, P, Q \in \mathcal{P}, R \in \mathcal{C}$ and $\vec{x}$ names$\}$. Suppose $(\vec{x})(P^\circ|R)\mathcal{R}(\vec{x})(Q^\circ|R)$ and $(\vec{x})(P^\circ|R) \to P'$. By Lemma 34, x', R', P_1 and Q_1 exist such that $P' = (\vec{x'})(P_1^\circ|R')$, $(\vec{x})(Q^\circ|R) \to^* (\vec{x'})(Q_1^\circ|R')$ and $P_1 \approx^o Q_1$. So $P'\mathcal{R}(\vec{x'})(Q_1^\circ|R')$. The case when $(\vec{x})(P^\circ|R) \xrightarrow{\delta} P'$ is similar. Conclude that $\mathcal{R}$ is a local bisimulation.
$\Leftarrow$: Let $\mathcal{S}$ be $\{(P\sigma, Q\sigma) \mid P^\circ \approx Q^\circ, P, Q \in \mathcal{P}, \sigma$ substitution$\}$. Suppose $P\sigma\mathcal{S}Q\sigma$ and $P\sigma \to P_1$. Then $P^\circ\sigma \to P_1^\circ$ by Theorem 32. By Lemma 9 and Lemma 31, Q_1 exists such that $(Q\sigma)^\circ \to^* Q_1^\circ$ and $P_1^\circ \approx Q_1^\circ$. So $Q\sigma \to^* Q_1$ by Theorem 32. Other cases are similar. It follows that $\mathcal{S}$ is an open bisimulation. $\Box$

4.2 Pragmatics

In the formulation of χ-calculus, we use the same set of names for both global names and local names. Theoretically this is justified by the fact that if $P \to Q$ then $(x)P \to (x)Q$ and more importantly by the fact that if $P \approx Q$ then $(x)P \approx (x)Q$. The same can be said about π-calculus. But conceptually the identification is not always helpful. The standard bisimilarity ([41]) for the π-processes is not closed under input prefixing operation. This is because the variable names and the free names are regarded as semantically different in the approach. Sangiorgi's open bisimilarity is congruent. But still the local names are treated differently from the free names. In the χ-calculus, both local and global names are variable names, which is what local bisimilarity assumes. The situation is similar to that in λ-calculus, where both free and closed variables are, well, variables that can be instantiated by any λ-terms.

But variable names alone do not suffice. Pragmatically one definitely needs constant names! This is clear from the mobile process interpretation of object oriented languages ([65,66]). The usual practice is to identify some names as constant. This is the same as to say that $\mathcal{N}$ consists of two parts: a set $\mathcal{N}_v$ of variable names and a set $\mathcal{N}_c$ of constant names. We can now define χ-processes to be those in which all variable names are localized. Now there are two kinds of local names: local variable names and local constant names. A communication either identifies two local variable names or replaces a local variable name by a local or global constant name. A communication between two local constant names is prohibited. Bisimulation equivalence for χ-processes can be defined in a way similar to the standard bisimulation equivalence for π-processes. Assume that $\mathcal{N}_c$ is ranged over by a, b, c and $\mathcal{N}_c \cup \overline{\mathcal{N}_c}$ by α.

Definition 36 *Let $\mathcal{R}$ be a binary relation on the set of χ-processes. $\mathcal{R}$ is a simulation if $P\mathcal{R}Q$ implies*
(i) if $P \to P'$ then there exists some Q' such that $Q \to^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $P \xrightarrow{\alpha a} P'$ then there exists some Q' such that $Q \overset{\alpha a}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
(iii) if $P \xrightarrow{\alpha[a]} P'$ then there exists some Q' such that $Q \overset{\alpha[a]}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;

(iv) if $P \overset{\alpha(a)}{\rightarrow} P'$ then there exists some Q' such that $Q \overset{\alpha(a)}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$.
The relation $\mathcal{R}$ is a bisimulation if both $\mathcal{R}$ and its inverse are simulations. The bisimilarity $\approx^\chi$ is the largest bisimulation.

To know if two processes are *locally* bisimilar, one needs to examine their behaviour in all local contexts. To know if they are bisimilar, all one has to do is to see if they can simulate each other's observable actions; no contexts are necessary to make the judgement. For this reason, $\approx^\chi$ is much more tractable than $\approx$.

The π-calculus can be reexamined in this new setting. The input prefix operation restricts variable names whereas the localization operation always restricts constant names. The latter is due to the fact that in π-calculus a local name is never changed. π-processes are now defined to be those processes in which all variable names are restricted by input prefixes. The standard bisimilarity can be defined for these π-processes as follows:

Definition 37 *Let $\mathcal{R}$ be a binary relation on the set of π-processes. $\mathcal{R}$ is a simulation if $P\mathcal{R}Q$ implies*
(i) if $P \rightarrow P'$ then there exists some Q' such that $Q \rightarrow^* Q'$ and $P'\mathcal{R}Q'$;
(ii) if $P \overset{ca}{\rightarrow} P'$ then there exists some Q' such that $Q \overset{ca}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
(iii) if $P \overset{\overline{c}[a]}{\rightarrow} P'$ then there exists some Q' such that $Q \overset{\overline{c}[a]}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$;
(iv) if $P \overset{\overline{c}(a)}{\rightarrow} P'$ then there exists some Q' such that $Q \overset{\overline{c}(a)}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$.
The relation $\mathcal{R}$ is a bisimulation if both $\mathcal{R}$ and its inverse are simulations. The bisimilarity $\approx^\pi$ is the largest bisimulation.

Both $\approx^\chi$ and $\approx^\pi$ can be extended to process expressions with unrestricted variable names. For instance, if P and Q are two open χ-processes, then $P \approx^\chi Q$ if and only if $P\sigma \approx^\chi Q\sigma$ for all substitution σ that replaces all the variable names in $P|Q$ by constant names.

The translation given in Section 4.1 works in this new framework. It establishes an operational correspondence in the sense of Theorem 32. In addition we have the following full abstraction result with respect to the bisimilarity equivalence.

Theorem 38 *For π-processes P and Q, $P \approx^\pi Q$ if and only if $P^\circ \approx^\chi Q^\circ$.*

Proof: The proof is similar to that of Theorem 35. □

So practically speaking, π is a subcalculus of χ. Anything one can do using π-calculus can be done with χ-calculus. The converse problem has not been investigated. We believe that the two languages are equally expressive.

5 Lambda Calculus via Chi Calculus

A concurrent computation model has to answer the question of whether it captures sequential computation successfully. The issue is often addressed by relating variants of λ-calculus to the model. Milner's encodings ([34,53,54]) of the lazy λ-calculus ([1,43]) and the weak call-by-value λ-calculus carry over to the present calculus. There is no point in repeating the programme. Our focus in this section will be on the call-by-name λ-calculus ([50]), whose semantics is defined by the following rules:

$$\frac{}{(\lambda x.M)N \to M[N/x]} \qquad \frac{M \to M'}{MN \to M'N} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

Let Λ denote the set of λ-terms. The set of free variables in a term M is denoted by $fv(M)$.

The following translation, which is Milner's encoding of the lazy λ-calculus with slight modification, serves as an encoding of the call-by-name λ-calculus in χ-calculus:

$$\begin{aligned}
[\![x]\!]u &\overset{\text{def}}{=} \overline{x}[u] \\
[\![\lambda x.M]\!]u &\overset{\text{def}}{=} (v)(x)(u[x].u[v] | [\![M]\!]v) \\
[\![MN]\!]u &\overset{\text{def}}{=} (v)(x)([\![M]\!]v | \overline{v}[x].\overline{v}[u].x(w)*[\![N]\!]w)
\end{aligned}$$

The parallel composition of $\overline{u}[x].\overline{u}[v]$ and $[\![M]\!]v$ allows $[\![M]\!]v$ to evolve independently, thus modeling reduction under λ-abstraction. Let us see an example:

$$\begin{aligned}
& [\![\lambda x.(\lambda y.y)N]\!]o \\
&= (u)(x)(o[x].o[u] | (v)(z)((w)(y)(v[y].v[w] | [\![y]\!]w) | \overline{v}[z].\overline{v}[u].z(w)*[\![N]\!]w)) \\
&\to (u)(x)(o[x].o[u] | (v)(y)((w)(v[w] | \overline{y}[w]) | \overline{v}[u].y(w)*[\![N]\!]w)) \\
&\to (u)(x)(o[x].o[u] | (y)(\overline{y}[u] | y(w)*[\![N]\!]w)) \\
&\to (u)(x)(o[x].o[u] | (y)([\![N]\!]u | y(w)*[\![N]\!]w)) \\
&\approx (u)(x)(o[x].o[u] | [\![N]\!]u) \\
&= [\![\lambda x.N]\!]o.
\end{aligned}$$

The computation is of a call-by-need nature. The following definition is taken from [34]. Here we have to deal with open λ-terms.

Definition 39 *Let the relation $\lhd \subset \Lambda \times \mathcal{C}$ contain all the pairs (L, P) such that for some $k \geq 0$, some $M, N_1, \ldots, N_k \in \Lambda$ and distinct variables $x_1, \ldots, x_k$:*
(i) $fv(N_i) \cap \{x_1, \ldots, x_i\} = \emptyset$ for all $1 \leq i \leq k$;

(ii) $L \equiv M[N_1/x_1]\dots[N_k/x_k]$;
(iii) $P = (\vec{x})([\![M]\!]u|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w)$.

Intuitively $M \lhd P$ means that the process P codes up the operational behaviour of λ-term M. In particular $M \lhd [\![M]\!]u$. As a matter of fact $P_1 \approx P_2$ whenever $M \lhd P_1$ and $M \lhd P_2$. To prove that, we need the following lemma. Its proof idea can be found in [35].

Lemma 40 *Suppose every occurrence of a in P, Q and R is in negative subject position. Then*
(i) $(a)(\alpha[x].P|a(w)*R) \approx \alpha[x].(a)(P|a(w)*R)$ *if* $a \notin \{x, \alpha, \overline{\alpha}\}$;
(ii) $(a)(P|Q|a(w)*R) \approx (a)(P|a(w)*R)|(a)(Q|a(w)*R)$;
(iii) $(a)(m(x)*P|a(w)*Q) \approx m(x)*(a)(P|a(w)*Q)$.

The following proposition can be proved by induction using Lemma 40.

Proposition 41 *If $M \lhd P$ then $[\![M]\!]u \approx P$.*

Proof: Suppose $P = (\vec{x})([\![M_0]\!]u|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w)$ and $M \equiv M_0[N_1/x_1]\dots[N_k/x_k]$. The following argument is carried out by induction on the size of M.

- $M_0 \equiv x_i$ for $i \in \{1,\dots,k\}$. Then

$$\begin{aligned} P &= (\vec{x})([\![x_i]\!]u|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w) \\ &\approx (\vec{x})([\![N_i]\!]u|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w) \\ &\approx [\![M]\!]u \end{aligned}$$

  by induction hypothesis.
- $M_0 \equiv \lambda x.A$. Then

$$\begin{aligned} P &= (\vec{x})((v)(x)(u[x].u[v]|[\![A]\!]v)|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w) \\ &= (v)(x)(u[x].u[v]|(\vec{x})([\![A]\!]v|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w)) \\ &\approx (v)(x)(u[x].u[v]|[\![A[N_1/x_1]\dots[N_k/x_k]]\!]v) \\ &= [\![M]\!]u \end{aligned}$$

  by induction hypothesis.
- $M_0 \equiv AB$. Then

$$\begin{aligned} P &= (\vec{x})((v)(x)([\![A]\!]v|\overline{v}[x].\overline{v}[u].x(w)*[\![B]\!]w) \\ &\qquad\qquad |x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w) \\ &\approx (v)(x)((\vec{x})([\![A]\!]v|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w) \\ &\qquad\qquad |(\vec{x})(\overline{v}[x].\overline{v}[u].x(w)*[\![B]\!]w|x_1(w)*[\![N_1]\!]w|\dots|x_k(w)*[\![N_k]\!]w)) \\ &\approx (v)(x)([\![A[N_1/x_1]\dots[N_k/x_k]]\!]v \\ &\qquad\qquad |\overline{v}[x].\overline{v}[u].x(w)*[\![B[N_1/x_1]\dots[N_k/x_k]]\!]w) \end{aligned}$$

$= [\![M]\!]u$

by Lemma 40 and induction hypothesis.

We are done. $\Box$

The next theorem relates the operational behaviour of P to that of L whenever $L \triangleleft P$.

Theorem 42 *Suppose $L \triangleleft P$. Then*
(i) if $L \longrightarrow L'$ then P' exists such that $P \longrightarrow^+ P'$ and $L' \triangleleft P'$;
(ii) if $P \longrightarrow P'$ then either $L \triangleleft P'$ or $P' \longrightarrow^+ P''$ for some P'' and $L \longrightarrow L'$ for some L' such that $L' \triangleleft P''$.

Proof: Suppose $L \triangleleft P$. The general form of L is $\lambda z_1 . \ldots . \lambda z_n . A_1 A_2 \ldots A_m$. The proof of (i) is routine. The proof of (ii) goes by examining two possible cases for A_1: (a) A_1 is an abstraction term; (b) A_1 is one of the variables $x_1, \ldots, x_k$. Suppose P is $(\vec{x})([\![\lambda z_1 . \ldots . \lambda z_n . A'_1 A'_2 \ldots A'_m]\!]u | x_1(w) * [\![N_1]\!]w | \ldots | x_k(w) * [\![N_k]\!]w)$ and $P \longrightarrow P'$.

- A'_1 is an abstraction term. Then $P' \longrightarrow^+ P''$ for some P'' and $L \longrightarrow L'$ for some L' such that $L' \triangleleft P''$.
- A'_1 is one of the variables $x_1, \ldots, x_k$. Then $L \triangleleft P'$.

The details of proof is very much the same as the proof of the corresponding result in [34]. $\Box$

So the operational semantics of the call-by-name λ-calculus can be simulated within χ.

6 Towards an Integration of Chi and Lambda

Can χ-calculus simulate the operational semantics of the full λ-calculus? The same question has been asked about the π-calculus. There are two problems one encounters when trying to answer the question. The first is how to model reduction under λ-abstraction. The second is how to model reduction $MN \longrightarrow MN'$ induced by $N \longrightarrow N'$. The two problems are of different nature. The former is to do with parallel computation. There is no reason why it should pose any problem for concurrent computation. This view is supported by the result in Section 5. The latter is to do with recursion because the λ-term N may be duplicated in future reduction. In any structural interpretation, this

N must be translated into the body of a replicator or guarded replicator. So if the N induces an infinite reduction, the interpretation of MN would have no terminating reduction sequence. This would imply that the behaviour of the λ-term $\mathbf{KI\Omega}$ is not faithfully captured by the interpretation. It is our view that the second problem is orthogonal to concurrent computation. It is caused essentially by the operational incompatibility of the two recursion mechanisms.

In this section we take a look at a higher order calculus combining the communication mechanism of the χ-calculus with the recursion mechanism of the λ-calculus. The purpose of this investigation is to see if the two mechanisms fit coherently and if local bisimilarity suffices as a tool for studying the algebraic properties of the resultant language.

6.1 Chi with Call by Name Lambda

Let the set $\mathcal{H}$ of higher order χ-processes be defined by the following grammar:

$$E := \mathbf{0} \mid X \mid \alpha[x].E \mid E|E' \mid (x)E \mid \alpha(X)E \mid \alpha[E],$$

where X is a process variable. E, F, G and H will denote higher order χ-processes. The operational semantics of the higher order χ-calculus is defined by the relevant rules of the first order χ-calculus together with the rules incorporating a call-by-name recursion mechanism:

$$(x)(G|\alpha[x].E|\overline{\alpha}[y].F) \rightarrow (x)(G[y/x]|E[y/x]|F[y/x])$$

$$\frac{E \rightarrow E'}{E|F \rightarrow E'|F} \qquad \frac{E \rightarrow E'}{(x)E \rightarrow (x)E'}$$

$$\frac{}{\alpha(X)E|\overline{\alpha}[F] \rightarrow E[F/X]} \qquad \frac{E \rightarrow F}{\alpha(X)E \rightarrow \alpha(X)F}$$

Free and closed variables are defined in the standard way. Local and global names are defined as in the first order case with additional postulation that $gn(X) = ln(X) = \emptyset$. We will assume that local names are renamed to avoid being captured in higher order communications and higher order substitutions.

Usually a bisimulation equivalence for a higher order process calculus is defined for closed processes. This tractable approach is used by the authors of [60–62,51,52] in studying bisimulation equivalences for CHOCS and higher order

π-calculus. But the method breaks down in the presence of the reduction rule

$$\frac{E \to F}{\alpha(X)E \to \alpha(X)F}$$

A bisimulation equivalence for higher order χ-processes has to be defined on open processes. For that purpose, let's say that a binary relation $\mathcal{R}$ on $\mathcal{H}$ is substitution closed if $E\mathcal{R}F$ implies $E[E_1/X_1, \ldots, E_i/X_i]\mathcal{R}F[E_1/X_1, \ldots, E_i/X_i]$ for $E_1, \ldots, E_i \in \mathcal{H}$ and process variables $X_1, \ldots, X_i$.

As in the first order case, a labeled transition system is defined. In the following rules, δ ranges over $\{\overset{\alpha x}{\to}, \overset{\alpha[x]}{\to}, \overset{\alpha(x)}{\to} | \alpha \in \mathcal{N} \cup \overline{\mathcal{N}}, x \in \mathcal{N}\}$:

$$\frac{}{(y)(R|\alpha[y].P) \overset{\alpha x}{\to} R[x/y]|P[x/y]} \qquad \frac{}{\alpha[x].P \overset{\alpha[x]}{\to} P}$$

$$\frac{P \overset{\alpha[x]}{\to} P'}{(x)P \overset{\alpha(x)}{\to} P'} \qquad \frac{P \overset{\delta}{\to} P' \quad ln(\delta) \cap gn(Q) = \emptyset}{P|Q \overset{\delta}{\to} P'|Q} \qquad \frac{P \overset{\delta}{\to} P' \quad x \notin n(\delta)}{(x)P \overset{\delta}{\to} (x)P'}$$

Let $\overset{\delta}{\Rightarrow}$ denote the relation $\to^* \overset{\delta}{\to} \to^*$.

Definition 43 *A substitution closed binary relation $\mathcal{R}$ on $\mathcal{H}$ is a local bisimulation if $E\mathcal{R}F$ implies that for any higher order process G and any sequence $\vec{x}$ of names it holds that*
(i) if $(\vec{x})(E|G) \overset{\delta}{\Rightarrow} E'$ then F' exists such that $(\vec{x})(F|G) \overset{\delta}{\Rightarrow} F'$ and $E'\mathcal{R}F'$;
(ii) if $(\vec{x})(F|G) \overset{\delta}{\Rightarrow} F'$ then E' exists such that $(\vec{x})(E|G) \overset{\delta}{\Rightarrow} E'$ and $E'\mathcal{R}F'$.
The local bisimilarity $\approx^\omega$ is the largest local bisimulation on higher order processes.

It should be remarked that $\approx^\omega$ is by definition substitution closed.

We can define bisimulation up to $\approx^\omega$ similar to Definition 10. Lemma 8, Lemma 9 and Lemma 11 through Lemma 14 all hold for $\approx^\omega$.

Theorem 44 *$\approx^\omega$ is a congruence equivalence on higher order processes: if $E \approx^\omega F$ and $G \in \mathcal{H}$ then*
(i) $\alpha[x].E \approx^\omega \alpha[x].F$;
(ii) $(x)E \approx^\omega (x)F$;
(iii) $E|G \approx^\omega F|G$;
(iv) $\alpha(X)E \approx^\omega \alpha(X)F$;
(v) $\alpha[E] \approx^\omega \alpha[F]$.

PROOF: (ii) and (iii) follow directly from definition. The proof of (i) is the same as the one in the first order χ.

(iv) Suppose $(\vec{x})(R|\alpha(X)E) \rightarrow (\vec{x'})(R'|E[A/X])$ is induced by a higher order communication involving $\alpha(X)E$. Then $(\vec{x})(R|\alpha(X)F) \rightarrow (\vec{x'})(R'|F[A/X])$. As $\approx^\omega$ is substitution closed, $E[A/X] \approx^\omega F[A/X]$. It follows from (ii) and (iii) that $(\vec{x'})(R'|E[A/X]) \approx^\omega (\vec{x'})(R'|F[A/X])$. Other cases are simple. Hence $\alpha(X)E \approx^\omega \alpha(X)F$.

(v) For the sake of this proof, let's define $\mathcal{H}_o[X]$ to be the set of all higher order processes E such that each occurrence of X is within $\alpha[K]$ for some $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}}$ and some $K \in \mathcal{H}$. We first point out an auxiliary fact: Assuming $E \in \mathcal{H}_o[X]$ and $A \approx^\omega B$,

- if $(\vec{x})E[A/X] \rightarrow G$ then $(\vec{x})E[B/X] \rightarrow H$ such that $G = (\vec{x'})F[A/X]$ and $H \approx^\omega (\vec{x'})F[B/X]$ for some F in $\mathcal{H}_o[X]$;
- if $(\vec{x})E[A/X] \xrightarrow{\delta} G$ then $(\vec{x})E[B/X] \xrightarrow{\delta} H$ such that $G = (\vec{x'})F[A/X]$ and $H \approx^\omega (\vec{x'})F[B/X]$ for some F in $\mathcal{H}_o[X]$.

The proof of the fact goes as follows:

- If $(\vec{x})E[A/X] \xrightarrow{\delta} G$ is a first order action, then $G = (\vec{x'})F[A/X]$ for some $F \in \mathcal{H}_o[X]$ and $(\vec{x})E[B/X] \xrightarrow{\delta} (\vec{x'})F[B/X]$.
- If $(\vec{x})E[A/X] \rightarrow G$ is a first order communication, then $G = (\vec{x'})F[A/X]$ for some $F \in \mathcal{H}_o[X]$ and $(\vec{x})E[B/X] \rightarrow (\vec{x'})F[B/X]$.
- If $(\vec{x})E[A/X] \rightarrow G$ is a higher order communication, then $G = (\vec{x})F[A/X]$ for some $F \in \mathcal{H}_o[X]$. Now $(\vec{x})E[B/X] \rightarrow H$ by carrying out the 'same' communication. If we replace by A all the occurrences of B in H that do not appear in a subprocess of the form $\alpha[K]$, we get $(\vec{x})F[B/X]$. By (i) through (iv), $(\vec{x})F[B/X] \approx^\omega H$.

This concludes the proof of the auxiliary fact. Let $\mathcal{R}$ be the locally closed and substitution closed relation

$$\{((\vec{x})E[A/X], (\vec{x})E[B/X]) \mid A \approx^\omega B,\ E \in \mathcal{H}_o[X],\ \vec{x} \text{ names}\}.$$

Now suppose $(\vec{x})E[A/X]\mathcal{R}(\vec{x})E[B/X]$ and $(\vec{x})E[A/X] \rightarrow G$ or $(\vec{x})E[A/X] \xrightarrow{\delta} G$. Then $G = (\vec{x'})F[A/X]$ for some $F \in \mathcal{H}_o[X]$. By the above fact, we can find some $H \in \mathcal{H}$ such that $(\vec{x})E[B/X] \rightarrow H$ or $(\vec{x})E[A/X] \xrightarrow{\delta} H$ with $(\vec{x'})F[B/X] \approx^\omega H$. It follows from Lemma 11 that $\mathcal{R}$ is a local bisimulation up to $\approx^\omega$. Thus $\alpha[E] \approx^\omega \alpha[F]$ since $\alpha[X] \in \mathcal{H}_o[X]$ and $E \approx^\omega F$. $\square$

In the remaining part of Section 6, we justify our claim that the higher order χ-calculus is a combination of χ and λ.

In [61], the general recursion $recX.E$ is defined as follows:

$$recX.E \overset{\text{def}}{=} (a)(a?X.(E|a!X)|a!(a?X.(E|a!X))).$$

One has

$$recX.E \to (a)(E[a?X.(E|a!X)/X]|a!(a?X.(E|a!X))).$$

This reduction looks like an operational unfolding of the recursion $recX.E$. Unfortunately the equation is not verified by the higher order bisimilarity defined in [61]. This is because the higher order bisimilarity has too strong a distinguishing power.

As a test for local bisimilarity, we examine Thomsen's recursion in this section. Suppose that E contains free variable X and a does not occur in E. The following abbreviations will be used:

$$\begin{aligned}&W_X^a(E) \overset{\text{def}}{=} a[a]|a(X)(\overline{a}[a].E|\overline{a}[X]),\\ &\mathsf{rec}X.E \overset{\text{def}}{=} (a)(W_X^a(E)|\overline{a}[W_X^a(E)]).\end{aligned}$$

We remark that $\mathsf{rec}X.E$ defined here is slightly different. The idea is to make sure that $W_X^a(E)$ is inert before being activated. To prove the main property concerning $\mathsf{rec}X.E$, we first establish a useful result.

Lemma 45 *Suppose E and F have free variable X and F' is obtained from $F[W_X^a(E)/X]$ by replacing some of the occurrences of $W_X^a(E)$ by $W_X^b(E)$. Then $(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \approx^\omega (a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])$. Here a and b are fresh.*

Proof: If $(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \to P$ is induced by a higher order communication between $F[W_X^a(E)/X]$ and $\overline{a}[W_X^a(E)]$, then clearly

$$P \approx^\omega (\vec{x})(a)(H[W_X^a(E)/X]|\overline{a}[W_X^a(E)])$$

for some H with free variable X but without a. By carrying out the 'same' higher order communication either between F' and $\overline{a}[W_X^a(E)]$ or between F' and $\overline{b}[W_X^b(E)]$, we get

$$\begin{aligned}&(\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])\\ \rightarrow\approx^{\omega}\ &(\vec{x})(a)(b)(H'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])\end{aligned}$$

where H' is obtained from $H[W_X^a(E)/X]$ by replacing some occurrences of $W_X^a(E)$ by $W_X^b(E)$. Conversely, if

$$\begin{aligned}&(\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])\\ \rightarrow\approx^{\omega}\ &(\vec{x})(a)(b)(H'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])\end{aligned}$$

then

$$(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \rightarrow\approx^{\omega} (\vec{x})(a)(H[W_X^a(E)/X]|\overline{a}[W_X^a(E)])$$

such that H' is obtained from $H[W_X^a(E)/X]$ by replacing some occurrences of $W_X^a(E)$ by $W_X^b(E)$. If $(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \rightarrow P$ by a communication within $F[W_X^a(E)/X]$ or $(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \xrightarrow{\delta} P$, then

$$P = (\vec{x'})(a)(H[W_X^a(E)/X]|\overline{a}[W_X^a(E)])$$

for some H with free variable X but without a. And by performing the same action, we have

$$(\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)]) \rightarrow (\vec{x'})(a)(b)(H'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])$$

or

$$(\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)]) \xrightarrow{\delta} (\vec{x'})(a)(b)(H'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])$$

where H' is obtained from $H[W_X^a(E)/X]$ by replacing some occurrences of $W_X^a(E)$ by $W_X^b(E)$. Similar argument can establish the following fact:

> Q exists such that $(\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)]) \overset{\delta}{\Rightarrow} Q$ and $P \approx^{\omega} \mathcal{R} \approx^{\omega} Q$ whenever $(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \overset{\delta}{\Rightarrow} P$,

where $\mathcal{R}$ is the following locally closed and substitution closed relation

$$\left\{ \begin{array}{l|l} & E,F \text{ have free } X, \\ & \vec{x} \in \mathcal{N},\ a,b \notin n(E,F), \\ ((\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]), & F' \text{ is obtained from} \\ (\vec{x})(a)(b)(F'|\overline{a}[W_X^a(E)]|\overline{b}[W_X^b(E)])) & F[W_X^a(E)] \text{ by replacing} \\ & \text{some occurrences of} \\ & W_X^a(E) \text{ by } W_X^b(E) \end{array} \right\}$$

It follows that $\mathcal{R}$ is a local bisimulation up to $\approx^\omega$. □

Theorem 46 *If E contains free variable X then* $\mathsf{rec}X.E \approx^\omega E[\mathsf{rec}X.E/X]$.

Proof: We show that the locally closed and substitution closed relation

$$\left\{ \begin{array}{l|l} ((\vec{x})F[\mathsf{rec}X.E/X], & E \text{ and } F \text{ contain free variable } X, \\ & a \notin n(E,F),\ gn(E) \cap ln(F) = \emptyset, \\ (\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)])) & \vec{x} \text{ is a sequence of names} \end{array} \right\}$$

is a local bisimulation up to $\approx^\omega$. We sketch the general idea. Suppose

$$(\vec{x})(a)(F[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \to P$$

is induced by a communication between $F[W_X^a(E)/X]$ and $\overline{a}[W_X^a(E)]$. Without loss of generality, let F be $X|F_1$. Then by Lemma 45

$$\begin{aligned} P &= (\vec{x})(a)(a[a]|\overline{a}[a].E[W_X^a(E)/X]|\overline{a}[W_X^a(E)]|F_1[W_X^a(E)/X]) \\ &\approx^\omega (\vec{x})(a)(E[W_X^a(E)/X]|\overline{a}[W_X^a(E)]|F_1[W_X^a(E)/X]) \\ &\approx^\omega (\vec{x})(a)(b)(E[W_X^b(E)/X]|\overline{b}[W_X^b(E)]|F_1[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \\ &= (\vec{x})(a)((b)(E[W_X^b(E)/X]|\overline{b}[W_X^b(E)])|F_1[W_X^a(E)/X]|\overline{a}[W_X^a(E)]) \\ &= (\vec{x})(a)(H[W_X^a(E)/X]|\overline{a}[W_X^a(E)]), \end{aligned}$$

where $H = (b)(E[W_X^b(E)]|\overline{b}[W_X^b(E)])|F_1$. Correspondingly, we have

$$\begin{aligned} (\vec{x})F[\mathsf{rec}X.E/X] &= (\vec{x})(\mathsf{rec}X.E|F_1[\mathsf{rec}X.E/X]) \\ &\to^* (\vec{x})((a)(E[W_X^a(E)/X]|\overline{a}[W_X^a(E)])|F_1[\mathsf{rec}X.E/X]) \\ &= (\vec{x})H[\mathsf{rec}X.E/X] \end{aligned}$$

to match the previous reduction. The details of the proof are omitted. Hence

$$\mathsf{rec}X.E \approx^{\omega} (a)(E[W^a_X(E)/X]|\overline{a}[W^a_X(E)]) \approx^{\omega} E[\mathsf{rec}X.E/X].$$

This completes the proof. □

6.3 Conservativity

In this section we show that the higher order χ can be seen as an extension of the first order χ. A fallout of the result is a justification of the claim that the first order recursion is completely unnecessary in the higher order χ-calculus. Let χ^+ be the higher order χ-calculus enriched with the guarded replication. The language χ^+ can be investigated along the same line as the higher order χ has been. $\mathcal{H}^+$ and $\approx^+$ are defined accordingly. It can also be shown that $\approx^+$ is a congruence relation. The translation $\widehat{\ }$ from χ^+-processes to χ^{ω}-processes is defined as follows:

$$\begin{aligned}
\widehat{X} &\stackrel{\text{def}}{=} X \\
\widehat{E|F} &\stackrel{\text{def}}{=} \widehat{E}|\widehat{F} \\
\widehat{(x)E} &\stackrel{\text{def}}{=} (x)\widehat{E} \\
\widehat{\alpha[x].E} &\stackrel{\text{def}}{=} \alpha[x].\widehat{E} \\
\widehat{\alpha(X)E} &\stackrel{\text{def}}{=} \alpha(X)\widehat{E} \\
\widehat{\alpha[F]} &\stackrel{\text{def}}{=} \alpha[\widehat{F}] \\
\widehat{\alpha(x)*E} &\stackrel{\text{def}}{=} (a)((x)\alpha[x].(\widehat{E}|a(X)(X|\overline{a}[X]))|\overline{a}[(x)\alpha[x].(\widehat{E}|a(X)(X|\overline{a}[X]))]) \\
&\quad\ \text{where } a \text{ is fresh.}
\end{aligned}$$

The translation $\widehat{\ }$ projects the guarded replication out, as it were. A χ^+-process P and its translation can perform same actions in the sense of the following proposition:

Proposition 47 *Let $P \in \mathcal{H}^+$. Then*
(i) if $P \xrightarrow{\delta} P'$ ($P \rightarrow P'$) then $\widehat{P} \stackrel{\delta}{\Rightarrow} \widehat{P'}$ ($\widehat{P} \rightarrow^+ \widehat{P'}$);
(ii) if $\widehat{P} \xrightarrow{\delta} P''$ ($\widehat{P} \rightarrow P''$) then $P \xrightarrow{\delta} P'$ ($P \rightarrow P'$) for some P' such that $P'' \approx^+ \widehat{P'}$.

Remark: (i) The proof of this proposition is similar to that of Lemma 45. (ii) As in Lemma 45, if $\widehat{P} \stackrel{\delta}{\Rightarrow} P''$ ($\widehat{P} \rightarrow^+ P''$) then $P \stackrel{\delta}{\Rightarrow} P'$ ($P \rightarrow^+ P'$) for some P' such that $P'' \approx^+ \widehat{P'}$.

Theorem 48 *For $P \in \mathcal{H}^+$, $P \approx^+ \widehat{P}$.*

PROOF: Using Proposition 47, one shows that $\{(P, \widehat{P}) | P \in \mathcal{H}^+\}$ is a local bisimulation up to $\approx^+$. $\square$

Theorem 49 *The following properties hold:*
(i) Suppose P and Q are in $\mathcal{H}$. Then $P \approx^+ Q$ if and only if $P \approx^\omega Q$.
(ii) Suppose P and Q are in $\mathcal{H}^+$. Then $P \approx^+ Q$ if and only if $\widehat{P} \approx^\omega \widehat{Q}$.
(iii) If $\widehat{P} \xrightarrow{\delta} P''$ ($\widehat{P} \rightarrow P''$) then $P \xrightarrow{\delta} P'$ ($P \rightarrow P'$) for some P' such that $P'' \approx^\omega \widehat{P'}$.

PROOF: (i) Suppose P, Q are in $\mathcal{H}$. $P \approx^+ Q$ clearly implies $P \approx^\omega Q$. Suppose $P \approx^\omega Q$. Then $(\vec{x})\widehat{(P|R)} = (\vec{x})(P|\widehat{R})$ and $(\vec{x})\widehat{(Q|R)} = (\vec{x})(Q|\widehat{R})$, where $R \in \mathcal{H}^+$. By Theorem 48, $(\vec{x})(P|R) \approx^+ (\vec{x})(P|\widehat{R})$ and $(\vec{x})(Q|R) \approx^+ (\vec{x})(Q|\widehat{R})$. It is now easy to see that $\approx^\omega$ is a local bisimulation up to $\approx^+$.
(ii) By Theorem 48, $P \approx^+ Q$ if and only if $\widehat{P} \approx^+ \widehat{Q}$. By (i) $\widehat{P} \approx^+ \widehat{Q}$ if and only if $\widehat{P} \approx^\omega \widehat{Q}$.
(iii) This is obtained from Proposition 47 by replacing $P'' \approx^+ \widehat{P'}$ with $P'' \approx^\omega \widehat{P'}$. $\square$

We now complete the picture by relating the first order χ to χ^+. In terms of operational semantics, the former is clearly a sublanguage of the latter. As for the algebraic semantics, we have the following result.

Theorem 50 $\approx$ *and* $\approx^+$ *coincide on first order χ-processes.*

PROOF: For $P, Q \in \mathcal{C}$, the implication from $P \approx^+ Q$ to $P \approx Q$ is trivial. To establish the other half of the theorem, it suffices to show that $\approx$ is a local bisimulation in χ^+-calculus. Suppose $P \approx Q$ and $G \in \mathcal{H}^+$. Let $\vec{x}$ be a sequence of names. If $(\vec{x})(P|G) \xrightarrow{\delta} (\vec{x'})(P'|G')$ or $(\vec{x})(P|G) \rightarrow (\vec{x'})(P'|G')$ by a first order communication, then using the technique employed in the proof of Lemma 21, one shows that some Q' exists such that $P' \approx Q'$ and $(\vec{x})(Q|G) \stackrel{\delta}{\Rightarrow} (\vec{x'})(Q'|G')$ or $(\vec{x})(Q|G) \rightarrow^* (\vec{x'})(Q'|G')$. If $(\vec{x})(P|G) \xrightarrow{\delta} (\vec{x'})(P'|G')$ is induced by a higher order communication, then $(\vec{x'})(P'|G')$ must be of the form $(\vec{x})(P|G')$ and $G \rightarrow G'$. But then $(\vec{x})(Q|G) \rightarrow (\vec{x})(Q|G')$. $\square$

From Theorem 49 and Theorem 50, we conclude that $P \approx Q$ if and only if $\widehat{P} \approx^\omega \widehat{Q}$ for first order χ-processes P and Q. In other words, the higher order χ-calculus can be seen as a conservative extension of the first order χ-calculus.

The higher order calculus investigated so far is the combination of the χ-calculus and the call-by-name λ-calculus. An integration of χ with the full λ-calculus is the higher order calculus extended with the following rule

$$\frac{E \to F}{\alpha[E] \to \alpha[F]}$$

Definition 43 now gives rise to an equivalence relation on the set of all processes of the fully integrated calculus. The results in Section 6.2 and Section 6.3 hold for this language. The (i) through (iv) of Theorem 44 also hold. But so far we haven't been able to prove the (v) of Theorem 44 for the fully integrated calculus. Although we believe that $\approx^\omega$ is a congruence equivalence in the fully integrated language, we also think its proof will turn out to be hard.

The operational semantics of the full λ-calculus can be simulated in the fully integrated calculus. The following is one possible encoding:

$$\begin{aligned}
[\![x]\!]_u &\stackrel{\text{def}}{=} \overline{x}[u]|X_x \\
[\![\lambda x.M]\!]_u &\stackrel{\text{def}}{=} (x)(v)(u[v].u[x]|x(X_x)[\![M]\!]_v) \\
[\![MN]\!]_u &\stackrel{\text{def}}{=} (x)(v)([\![M]\!]_v|\overline{v}[u].\overline{v}[x]|\overline{x}[(w)(x[w]|[\![N]\!]_w)])
\end{aligned}$$

where the subscript of X_x indicates the correlation between the lambda variable x and the process variable.

Theorem 51 *Suppose M is a λ-term. If $M \to N$ then $[\![M]\!]_u \to^+ [\![N]\!]_u$.*

Proof: Consider the reduction $(\lambda x.A)B \to A[B/x]$. In the encoding it is simulated as follows:

$$\begin{aligned}
& [\![(\lambda x.A)B]\!]_u \\
= \;& (x)(v)((x)(w)(v[w].v[x]|x(X_x)[\![A]\!]_w)|\overline{v}[u].\overline{v}[x]|\overline{x}[(w)(x[w]|[\![B]\!]_w)]) \\
\to \;& (x)(v)((x)(v[x]|x(X_x)[\![A]\!]_u)|\overline{v}[x]|\overline{x}[(w)(x[w]|[\![B]\!]_w)]) \\
\to \;& (x)(x(X_x)[\![A]\!]_u|\overline{x}[(w)(x[w]|[\![B]\!]_w)]) \\
\to \;& (x)(\ldots\overline{x}[a_1]|(w)(x[w]|[\![B]\!]_w)\ldots\overline{x}[a_i]|(w)(x[w]|[\![B]\!]_w)\ldots) \\
\to^* \;& (x)(\ldots[\![B]\!]_{a_1}\ldots[\![B]\!]_{a_i}\ldots) \\
= \;& [\![A[B/x]]\!]_u.
\end{aligned}$$

This is enough to prove the theorem. □

The above encoding is operationally sound. But it does not preserve algebraic equality: $M =_\beta N$ does not imply $[\![M]\!]_u \approx^\omega [\![N]\!]_u$. For λ-terms M and N, $[\![M]\!]_u \approx^\omega [\![N]\!]_u$ implies $[\![\lambda x.M]\!]_u \approx^\omega [\![\lambda x.N]\!]_u$. Conversely, suppose $[\![\lambda x.M]\!]_u \approx^\omega [\![\lambda x.N]\!]_u$. Then it can be easily seen that some E and F exists such that $[\![M]\!]_u \rightarrow^* E \approx^\omega [\![N]\!]_u$ and $[\![N]\!]_u \rightarrow^* F \approx^\omega [\![M]\!]_u$. It follows from Lemma 12 that $[\![M]\!]_u \approx^\omega [\![N]\!]_u$. So we are forced to deal with open λ-terms. Now

$$\begin{aligned}
& [\![(\lambda x.x)y]\!]_u \\
= \;& (x)(v)([\![\lambda x.x]\!]_v|\overline{v}[u].\overline{v}[x]|\overline{x}[(w)(x[w]|\overline{y}[w]|Y_y)]) \\
= \;& (x)(v)((x)(w)(v[w].v[x]|x(X_x)[\![x]\!]_w)|\overline{v}[u].\overline{v}[x]|\overline{x}[(w)(x[w]|\overline{y}[w]|Y_y)]) \\
\rightarrow^* \;& (x)(x(X_x)[\![x]\!]_u|\overline{x}[(w)(x[w]|\overline{y}[w]|Y_y)]) \\
\rightarrow \;& (x)(\overline{x}[u]|(w)(x[w]|\overline{y}[w]|Y_y)) \\
\rightarrow \;& \overline{y}[u]|Y_y \\
= \;& [\![y]\!]_u.
\end{aligned}$$

Obviously $(x)(\overline{x}[u]|(w)(x[w]|\overline{y}[w]|y[y])) \rightarrow (x)(\overline{x}[u]|x[y])$ is not matched by any derivative of $\overline{y}[u]|y[y]$. It follows that $(x)(\overline{x}[u]|(w)(x[w]|\overline{y}[w]|Y_y))$ is not bisimilar to $[\![y]\!]_u$. But $[\![(\lambda x.x)y]\!]_u$ is bisimilar to $(x)(\overline{x}[u]|(w)(x[w]|\overline{y}[w]|Y_y))$. Consequently $[\![(\lambda x.x)y]\!]_u \approx^\omega [\![y]\!]_u$ is false.

We need to search for an alternative that verifies β-equality. Here is a modification of the above interpretation:

$$\begin{aligned}
[\![x]\!]_u &\stackrel{\text{def}}{=} (a)(\overline{x}[a].\overline{a}[u].\overline{a}[\mathbf{0}])|X_x \\
[\![\lambda x.M]\!]_u &\stackrel{\text{def}}{=} (x)(v)(u[v].u[x]|x(X_x)[\![M]\!]_v) \\
[\![MN]\!]_u &\stackrel{\text{def}}{=} (x)(v)([\![M]\!]_v|\overline{v}[u].\overline{v}[x]|\overline{x}[(a)(w)(x[a].a[w]|a(Y)[\![N]\!]_w)]) \\
&\quad\;\; \text{where } a \text{ is fresh and } Y \text{ does not appear in } [\![N]\!]_w.
\end{aligned}$$

Theorem 51 holds for this new encoding. In addition the interpretation maps β-equal terms to bisimilar processes. To justify this claim we need to establish an auxiliary result.

A process Q is akin to a process P if (i) $P \rightarrow Q$ (ii) $\forall a \in \mathcal{N}.\neg(P{\downarrow}a)$ and (iii) if $P \rightarrow P'$ then either $P' = Q$ or $Q \rightarrow^* Q'$ for some Q' such that Q' is akin to P'.

Lemma 52 *Suppose E is a process with at least process variables $X_1, \ldots, X_k$. Q_i is akin to P_i for each $i \in \{1, \ldots, k\}$. Then $E[P_1/X_1, \ldots, P_k/X_k] \approx^\omega E[Q_1/X_1, \ldots, Q_k/X_k]$.*

Proof: First observe the following facts:

- If $E[Q_1/X_1, \ldots, Q_k/X_k] \rightarrow F$ then $E[P_1/X_1, \ldots, P_k/X_k] \rightarrow^* F$.
- If $E[P_1/X_1, \ldots, P_k/X_k] \rightarrow F[P_1/X_1, \ldots, P_k/X_k]$ is induced by a communication within E then $E[Q_1/X_1, \ldots, Q_k/X_k] \rightarrow F[Q_1/X_1, \ldots, Q_k/X_k]$.
- If $E[P_1/X_1, \ldots, P_k/X_k] \rightarrow E[P_1'/X_1, \ldots, P_k/X_k]$ is induced by $P_1 \rightarrow P_1'$ with $\neg(P_1' = Q_1)$, then $E[Q_1/X_1, \ldots, Q_k/X_k] \rightarrow^* E[Q_1'/X_1, \ldots, Q_k/X_k]$ for some Q_1' such that $Q_1 \rightarrow^* Q_1'$ and P_1' is akin to Q_1'.

The rest of the proof is a matter of formality. □

Theorem 53 *Suppose M and N are two λ-terms. Then $M =_\beta N$ implies $[\![M]\!]_u \approx^\omega [\![N]\!]_u$.*

Proof: By Church-Rosser property, one only has to prove that $[\![M]\!]_u \approx^\omega [\![N]\!]_u$ whenever $M \rightarrow N$. Suppose $M \rightarrow N$ is induced by $(\lambda x.A)B \rightarrow A[B/x]$. Then

$$\begin{aligned}
[\![M]\!]_u &= P_0 \stackrel{\text{def}}{=} \ldots (x)(v)((x)(w)(v[w].v[x]|x(X_x)[\![A]\!]_w)| \\
&\qquad\qquad \overline{v}[o].\overline{v}[x]|\overline{x}[(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w)]) \ldots \\
&\rightarrow P_1 \stackrel{\text{def}}{=} \ldots (x)(v)((x)(v[x]|x(X_x)[\![A]\!]_o)| \\
&\qquad\qquad \overline{v}[x]|\overline{x}[(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w)]) \ldots \\
&\rightarrow P_2 \stackrel{\text{def}}{=} \ldots (x)(x(X_x)[\![A]\!]_o|\overline{x}[(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w)]) \ldots \\
&\rightarrow P_3 \stackrel{\text{def}}{=} \ldots (x)(\ldots (a)(\overline{x}[a].\overline{a}[a_1].\overline{a}[\mathbf{0}])|(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w) \\
&\qquad \ldots (a)(\overline{x}[a].\overline{a}[a_i].\overline{a}[\mathbf{0}])|(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w) \ldots) \ldots \\
&\rightarrow^* P_4 \stackrel{\text{def}}{=} \ldots [\![B]\!]_{a_1} \ldots [\![B]\!]_{a_i} \ldots \\
&= [\![N]\!]_u.
\end{aligned}$$

By constructing appropriate bisimulations, one can prove that $P_i \approx^\omega P_{i+1}$ for $i \in \{0, 1, 2\}$ using Lemma 52. To show $P_3 \approx^\omega P_4$, notice that P_3 is bisimilar to P_3' which is

$$\begin{aligned}
&\ldots (x)((a)(\overline{x}[a].\overline{a}[a_1].\overline{a}[\mathbf{0}])|(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w)) \\
&\ldots (x)((a)(\overline{x}[a].\overline{a}[a_i].\overline{a}[\mathbf{0}])|(a)(w)(x[a].a[w]|a(Y)[\![B]\!]_w)) \ldots .
\end{aligned}$$

This is established in a similar way as the (ii) of Lemma 40. Then prove $P_3' \approx^\omega P_4$ using Lemma 52. □

Notice that a congruence result would make the proof of Theorem 53 a little bit easier.

7 Category of Chi Processes

Abramsky's proof-as-process approach is one of the motivations for present work. In this section we come back to the issue to see how Abramsky's interpretation looks like using χ-calculus. We do this by casting the interpretation in a categorical framework.

A χ-process is linear if it is of the form

$$(\vec{x})(a_1[y_1]|\ldots|a_m[y_m]|P|\overline{b_1}[z_1]|\ldots|\overline{b_n}[z_n])$$

for some natural numbers m and n satisfying the following conditions:

(1) $a_1,\ldots,a_m,b_1,\ldots,b_n$ are pairwise distinct;
(2) $\{a_1,\ldots,a_m,b_1,\ldots,b_n\}\cap\{y_1,\ldots,y_m,z_1,\ldots,z_n\}=\emptyset$;
(3) $\{y_1,\ldots,y_m,z_1,\ldots,z_n\}=\{\vec{x}\}$;
(4) $gn(P)\subseteq\{y_1,\ldots,y_m,z_1,\ldots,z_n\}$.

In what follows, we abbreviate, say, the above linear process to $(\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{b}}[\vec{z}])$.

For two linear χ-processes $(\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{b}}[\vec{z}])$ and $(\vec{x'})(\vec{c}[\vec{y'}]|P'|\vec{\overline{d}}[\vec{z'}])$, define

$$(\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{b}}[\vec{z}])\asymp(\vec{x'})(\vec{c}[\vec{y'}]|P'|\vec{\overline{d}}[\vec{z'}])$$

if $\vec{y}=\vec{y'}$, $\vec{z}=\vec{z'}$ and $P\approx P'$. $\asymp$ is obviously an equivalence relation. Let $I(a,b)$ denote the linear χ-process $(x)(a[x]|\overline{b}[x])$.

Lemma 54 *Suppose $(\vec{x})(a[y]|P|\overline{b}[z])$ is a linear χ-process. Then for a fresh name c, the following holds: $(a)(I(c,a)|(\vec{x})(a[y]|P|\overline{b}[z]))\asymp(\vec{x})(c[y]|P|\overline{b}[z])$ and $(b)((\vec{x})(a[y]|P|\overline{b}[z])|I(b,c))\asymp(\vec{x})(a[y]|P|\overline{c}[z])$.*

The category $\mathbf{P}$ has natural numbers as objects. Objects of $\mathbf{P}$ are denoted by $\underline{0}$, $\underline{1}$, $\underline{2}$ and so on. A morphism from $\underline{m}$ to $\underline{n}$ is a $\asymp$-equivalence class of linear χ-processes of the form

$$(\vec{x})(a_1[y_1]|\ldots|a_m[y_m]|P|\overline{b_1}[z_1]|\ldots|\overline{b_n}[z_n]).$$

We confuse notationally a $\asymp$-equivalence class with one of its members. The composition of morphisms $(\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{b}}[\vec{z}]):\underline{l}\to\underline{m}$ and $(\vec{x'})(\vec{c}[\vec{y'}]|P'|\vec{\overline{d}}[\vec{z'}]):\underline{m}\to\underline{n}$ is

$$(\vec{e})((\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{e}}[\vec{z}])|(\vec{x'})(\vec{e}[\vec{y'}]|P'|\vec{\overline{d}}[\vec{z'}])):\underline{l}\to\underline{n}$$

for fresh $\vec{e}$. It is clearly well defined. Associativity of composition obviously holds. By Lemma 54 the identity morphism on $\underline{m}$ can be defined as

$$I(a_1, b_1)| \dots |I(a_m, b_m)$$

where $a_1, \dots, a_m, b_1, \dots, b_m$ are pairwise distinct. In particular the identity morphism on $\underline{0}$ is the inactive process $\mathbf{0}$.

Given morphisms $\phi = (\vec{x})(\vec{a}[\vec{y}]|P|\vec{\overline{b}}[\vec{z}]) : \underline{m} \to \underline{n}$ and $\psi = (\vec{x'})(\vec{c}[\vec{y'}]|P'|\vec{\overline{d}}[\vec{z'}]) : \underline{m'} \to \underline{n'}$ such that $\vec{a}, \vec{b}, \vec{c}, \vec{d}$ are all different, define $\phi \otimes \psi$ to be $\phi|\psi : \underline{m+m'} \to \underline{n+n'}$. This gives rise to a symmetric monoidal category, the unit of the tensor product being $\underline{0}$.

For a χ-processes P, $P^{\perp}$ denotes the process obtained by replacing every positive (negative) occurrence of a name in P by a negative (positive) occurrence of the same name. For instance, $((x)(m[a]|\overline{b}[x]|x[m]))^{\perp}$ is $(x)(\overline{m}[a]|b[x]|\overline{x}[m])$. Clearly $(P^{\perp})^{\perp} = P$. $(\_)^{\perp}$ extends to a functor from $\mathbf{P}$ to $\mathbf{P}^{op}$ if we define $\underline{m}^{\perp}$ to be $\underline{m}$ for $m \in \omega$. As $(\underline{m} \xrightarrow{\psi} \underline{n})^{\perp}$ is $\underline{n} \xrightarrow{\psi^{\perp}} \underline{m}$, $(\_)^{\perp}$ is an involution. Let $\underline{m}\wp\underline{n}$ be $(\underline{m}^{\perp} \otimes \underline{n}^{\perp})^{\perp} = \underline{m} \otimes \underline{n}$ and let $\underline{m} \multimap \underline{n}$ be $\underline{m}^{\perp}\wp\underline{n} = \underline{m} \otimes \underline{n}$. With these constructions, $\mathbf{P}$ becomes a $*$-autonomous category ([9]). A bijective correspondence between $\mathbf{P}(\underline{l} \otimes \underline{m}, \underline{n})$ and $\mathbf{P}(\underline{l}, \underline{m} \multimap \underline{n})$ sends $(\vec{x})(\vec{a}[\vec{l}]|\vec{b}[\vec{m}]|P|\vec{\overline{c}}[\vec{n}])$ to $(\vec{x})(\vec{a}[\vec{l}]|P|\vec{\overline{b}}[\vec{m}]|\vec{\overline{c}}[\vec{n}])$. As a matter of fact $\mathbf{P}$ is a compact closed category as the functor $(\_)^{\perp}$ is self-dual ([10,6]).

It is well-known that a $*$-autonomous category is a model of the multiplicative linear logic ([58,10]).

8 Final Remark

Proof theory turns out to be helpful in constructing models for concurrent computation. The χ-calculus designed with proofs in mind subsumes the π-calculus. It is worth remarking that the extension is not achieved by adding extra operators but by unifying two combinators in π-calculus. If we can simplify a formalism without sacrificing its expressive power, we go ahead with the simplification.

As mentioned previously, there are two kinds of restricted name in π-calculus. For a basic model of concurrent computation, two is probably too many. The χ-calculus is what one gets when one tries to unify the two classes of restricted names. The payoff is that one has to adjust his familiar perception of communication as value-passing operation. In χ, the distinction between global names and local names are made minimal. It is not too exaggerating to say that in χ there is no difference between these two kinds of names. If one adopts a closed-world viewpoint, then global names are local names seen from within local declarations.

Usually a process calculus consists of two parts, a communication mechanism defining the operational behaviour of the system and a recursion mechanism giving the language the Turing computability. The χ-calculus differs from the π-calculus mainly in the effect of communication. π adopts a value-passing mechanism inherited from a functional framework. A communication in π-calculus can be seen as a concurrent functional application. The fact that the language as defined in [34] is capable of encapsulating the lazy λ-calculus ([34,53,54]), which is a model for functional programming, but not the full λ-calculus can be seen as an indication that it is a model of concurrent functional computation. The χ-calculus deviates from the π-calculus by removing some functionality away. To communicate is to share information. In $(x)(R|m[x].P)$ and $(y)(S|\overline{m}[y].Q)$, the local x and y are two secrets known only to the respective communities they belong to. The two communities share the secrets as a result of the following communication:

$$(x)(R|m[x].P)|(y)(S|\overline{m}[y].Q) \rightarrow (z)(R[z/x]|P[z/x]|S[z/y]|Q[z/y]) \quad (1)$$

In a basic model of concurrent computation, there ought not to be operations like ‘putting two pieces of information together’. Our solution has been to use a fresh name to denote the resulting information. In the following communication

$$(x)(R|m[x].P)|(S|\overline{m}[y].Q) \rightarrow R[y/x]|P[y/x]|S|Q \quad (2)$$

the process $(x)(R|m[x].P)$ has to reveal the secret to public. Once a secret is known to everybody, it is no more a secret. This is the intuition behind the above reduction. A stronger justification of (2) is that it is consistent with (1). (1) is a global view of an exchange while (2) is a local description of the communication.

A pleasant feature of χ-calculus is its symmetry. Other symmetric systems have been proposed in the literature. Sangiorgi's πI is one example ([55,13]). It has been observed that πI has almost all the expressive power of π. From the π-calculus point of view, πI is obtained from π by making the output prefixes the same as the input prefixes, whereas χ is what one gets by forcing the input prefixes to be the same as the output prefixes. The syntax of πI introduces a distinction between local names and variable names, although the latter appear pseudo. In χ, however, the distinction completely vanishes. Another variant of π-calculus is the so-called asynchronous π-calculus ([25,14,26,7]). Like πI, the asynchronous π-calculus is a sublanguage of π. The former can simulate the latter at least from an operational point of view. It is remarkable that the proposed sublanguages of π do not sacrifice the expressive power of π in any essential way. This seems to suggest that the idea of name manipulation is a robust one. In this paper we have shown that χ has at least the expressive power of π. Whether it is strictly more powerful is left for further study. If π and χ turn out to be equally expressive, then there ought to be a translation from χ to πI or a symmetric version of the asynchronous π-calculus that preserves expressive power. This is an important question that certainly deserves investigation. To some degree, one hopes that the answer is positive as that would add force to our belief that calculi of mobile processes are the right tool to describe concurrent computation.

As we said before, the basic reduction rule of χ-calculus

$$(x)(R|\alpha[x].P|\overline{\alpha}[y].Q) \rightarrow (x)(R[y/x]|P[y/x]|Q[y/x])$$

can be split into two:

$$(x)(R|\alpha[x].P|\overline{\alpha}[y].Q) \rightarrow R[y/x]|P[y/x]|Q[y/x], \quad \text{where } x \neq y \tag{3}$$

$$(x)(R|\alpha[x].P|\overline{\alpha}[x].Q) \rightarrow (x)(R|P|Q) \tag{4}$$

Rule (3) is the essence of communication in χ-calculus. On the other hand, (4) is open for modification. Two possibilities arise. One is to remove rule (4) completely. The other is to replace it with the following

$$\alpha[x].P|\overline{\alpha}[x].Q \rightarrow P|Q \tag{5}$$

Both modifications result in calculi that have simpler labeled transition systems than the χ of this paper. Conceptually we prefer (4) to (5) as the former keeps the uniformity of communication.

The Church-Turing thesis is supported by the fact that all proposed models for computable operations are equally expressive in the sense that they can simulate each other's operational behaviour. In the study of concurrent computation models, expressive power ought to be an important issue. But unlike the situation of sequential computation, there does not seem to be any consensus on criteria for one concurrent computation model being more expressive than another. How do we mean, for example, that one calculus is a subcalculus of another? Do we mean that the syntax of one is part of the syntax of the other with operational semantics inherited? Or do we mean that there is a translation from one to the other that preserves operational semantics? Should the translation be fully abstract with respect to some observable equivalence? If full abstraction is required, then which observational equivalence are we talking about? Can it be any observational equivalence that happens to enjoy a full abstraction property? The answers may well depend on what one does with these models.

There could be some questions concerning some of the design decisions of the χ-calculus. One is that the guarded replication introduces extra restricted names. This is not in line with our starting point that in a basic computation model there should be only one kind of restricted names. The second is that the second order communication is not symmetric, which seems to destroy the integrity of the first order communication. The first question can be easily avoided by using unguarded replication. Guarded replication $m(x){*}P$, which can be seen as an abbreviation of $!(x)(m[x].P)$, is better behaved and is easier to implement. The higher order communication mechanism used in this paper could be made symmetric, but there is nothing to be gained from that. In this paper we see higher order feature as providing a recursion mechanism rather than as an extension of communication mechanism. The essence of a concurrent computation model is defined by its communication mechanism, upon which one can introduce whatever recursion mechanism to fulfill one's purpose.

The χ-calculus has its motivation from reaction graphs ([18]), which is related to Lafont's interaction nets ([28,29]), Parrow's interaction diagrams ([45]) and Milner's π-nets ([37]). It would be interesting to investigate χ-calculus in the general framework of action calculus ([39,36,40,38]).

Does the translation $[\![\_]\!]$ defined in Section 5 preserve the algebraic semantics in the sense that $M =_\beta N$ implies $[\![M]\!]u \approx [\![N]\!]u$? The answer is negative. Now

$$\begin{aligned}
[\![(\lambda x.y)x]\!]u &= (v)(x)((o)(x)(v[x].v[o]|\overline{y}[o])|\overline{v}[x].\overline{v}[u].x(w){*}\overline{x}[w]) \\
&\rightarrow (v)(x)((o)(v[o]|\overline{y}[o])|\overline{v}[u].x(w){*}\overline{x}[w]) \\
&\rightarrow (x)(\overline{y}[u]|x(w){*}\overline{x}[w]) \\
&\approx [\![y]\!]u.
\end{aligned}$$

It is easily seen that $[\![(\lambda x.y)x]\!]u \approx (v)(x)((o)(v[o]|\overline{y}[o])|\overline{v}[u].x(w){*}\overline{x}[w])$. But

$$(v)(x)((o)(v[o]|\overline{y}[o])|\overline{v}[u].x(w){*}\overline{x}[w])|y[a] \rightarrow (v)(x)(v[a]|\overline{v}[u].x(w){*}\overline{x}[w])$$

is not matched by anything from $(x)(\overline{y}[u]|x(w){*}\overline{x}[w])|y[a]$. Hence $[\![(\lambda x.y)x]\!]u \not\approx [\![y]\!]u$. Moreover $[\![M]\!]u \approx [\![N]\!]u$ if and only if $[\![\lambda y.M]\!]u \approx [\![\lambda y.N]\!]u$. It follows that $[\![\lambda y.(\lambda x.y)x]\!]u \not\approx [\![\lambda y.y]\!]u$.

The category defined in Section 7 is an example of compact closed category. Weak product, and therefore coproduct, exists for objects $\underline{m}$ and $\underline{n}$ with

$m{+}n \neq 0$. Does the category have enough structure to model the modalities of linear logic? Or does it have the essential structures to be an interaction category ([2,5,21,6])? These are interesting questions.

The syntax of χ-calculus suggests another variant of π-calculus. It is a symmetric presentation of π-calculus, in which communications are exchange of information. Its abstract syntax is the same as that of χ-calculus. Let's denote this language by Π. One nice thing about the symmetric presentation is that it renders unnecessary abstract names as x in $m(x).P$ in the asymmetric version. In other words, the names in symmetric π-calculus is more homogeneous in the sense that if we replace the guarded recursion $m(x){*}R$ by replicator $!P$ then there is only one kind of restricted names. The communication mechanism of Π can be defined by a single rule

$$a[x].P|\overline{a}[y].Q \rightarrow P[y/x]|Q[x/y]$$

together with the structural rules

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \frac{P \rightarrow P'}{(x)P \rightarrow (x)P'}$$

Here are some examples of reduction in Π-calculus:

$$\begin{aligned}
R|m[x].P|\overline{m}[y].Q &\rightarrow R|P[y/x]|Q[x/y],\\
(x)(R|m[x].P)|\overline{m}[y].Q &\rightarrow (x)(R|P[y/x]|Q[x/y]),\\
(x)(R|m[x].P)|(y)(S|\overline{m}[y].Q) &\rightarrow (x)(y)(R|P[y/x]|S|Q[x/y]).
\end{aligned}$$

There is a structural translation from π to Π defined as follows:

$$\begin{aligned}
(\mathbf{0})^{\diamond} &\stackrel{\text{def}}{=} \mathbf{0},\\
(m(x).P)^{\diamond} &\stackrel{\text{def}}{=} (x)m[x].P^{\diamond},\\
(\overline{m}x.P)^{\diamond} &\stackrel{\text{def}}{=} (s)(\overline{m}[x].\overline{s}[s]|s[s].P^{\diamond}),\ \text{ where } s \text{ is fresh},\\
(P|Q)^{\diamond} &\stackrel{\text{def}}{=} P^{\diamond}|Q^{\diamond},\\
((x)P)^{\diamond} &\stackrel{\text{def}}{=} (x)P^{\diamond},\\
(\alpha(x){*}P)^{\diamond} &\stackrel{\text{def}}{=} \alpha(x){*}P^{\diamond}.
\end{aligned}$$

In [19] it is shown that the translation preserves both operational and algebraic semantics.

After the publication of [17], Parrow and Victor proposed a language they call Update Calculus ([47]), which is just an asymmetric version of χ-calculus.

More recently they have related Fusion Calculus, which is a polyadic version of χ-calculus, to concurrent constraint programming ([48,64]).

Axiomatizations of π-processes have been intensively studied. A couple of complete systems have been obtained. These include those for early and late congruence relations ([41,46,24,30]), for open bisimulation congruence ([56]), for finite control π-processes ([31,32]), and for asynchronous π-calculus ([7]). In χ-calculus, early and late bisimilarities coincide. In π-calculus, axiomatizations are complicated by the fact that there are two kinds of restricted names. In χ-calculus this complication is absent. Parrow and Victor have studied complete systems for weak hyperequivalence on Fusion processes ([49]). Fu has in [20] worked out complete axiom systems for both local congruence and barbed congruence. We believe that Fu's system for barbed congruence on χ-processes can be generalized to a complete system for barbed congruence on finite Fusion processes. More recently Fu has worked out complete systems for all the eighteen L-congruences on asymmetric χ-calculus and discovered a mistake made in both [49] and [20].

We have considered in this paper essentially two distinct bisimulation congruences on χ-processes. Are there any other reasonable bisimulation congruences on χ-processes? In [20] a possible classification of bisimilarities on χ-processes is given. A uniform definition of dozens of bisimilarities, called L-bisimilarities, is introduced. It is pointed out that these L-bisimilarities collapse into a number of distinct equivalence relations. Ordered by set inclusion, they form a lattice called bisimulation lattice. The local bisimilarity and the barbed bisimilarity studied in this paper are respectively the bottom and the top of the lattice. In [20] bisimulation lattices for asymmetric χ-processes, asynchronous χ-processes, asymmetric and asynchronous χ-processes are also discovered.

Acknowledgements

I have had the pleasure to discuss the material covered in this paper with G. Boudol, W. Chen, Y. Chen, L. Du and E. Shen. F. Song has proof-read the paper. B. Victor pointed out to me that the $\lhd$ of Definition 39 ought to be a relation from the set of *open* λ-terms to the set of χ-processes. Two anonymous referees' comments have helped improve the quality of the paper. The editors of the special issue have put a lot of care into my final presentation of the material covered in the paper. My thanks to all of them.

The work reported here has been supported by the National Nature Science Foundation of China, grant number 69873032, and by the National 863 High-Tech Project, grant number 863-306-ZT06-02-2.

References

[1] S. Abramsky, The Lazy Lambda Calculus, in: D. Turner, eds., *Declarative Programming* (Addison-Wesley, 1988) 65–116.

[2] S. Abramsky, Interaction Categories, in: G. Burn, S. Gay and M. Ryan, eds., *Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods* (Springer-Verlag, 1993) 57–70.

[3] S. Abramsky, Computational Interpretations of Linear Logic, *Theoretical Computer Science*, **111** (1993) 3–57.

[4] S. Abramsky, Proofs as Processes, *Theoretical Computer Science*, **135** (1994) 5–9.

[5] S. Abramsky, Interaction Categories and Communicating Sequential Processes, in: A. Roscoe, eds., *A Classical Mind: Essays in Honour of C.A.R. Hoare* (Prentice Hall, 1994) 1–15.

[6] S. Abramsky, S. Gay and R. Nagarajan, Interaction Categories and the Foundations of Typed Concurrent Programming, *Proceedings of the 1994 Marktoberdorf Summer School on Deductive Program Design*, NATO ASI, Series F (Springer-Verlag, 1996).

[7] R. Amadio, I. Castellani and D. Sangiorgi, On Bisimulations for the Asynchronous π-Calculus, *CONCUR '96*, Lecture Notes in Computer Science 1119 (Springer-Verlag, 1996) 146–162.

[8] H. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and Foundations of Mathematics (North-Holland, 1984).

[9] M. Barr, $*$-*Autonomous Categories*, Lecture Notes in Mathematics 752 (Springer-Verlag, 1979).

[10] M. Barr, $*$-Autonomous Categories and Linear Logic, *Mathematical Structures in Computer Science*, **1** (1991) 159–178.

[11] G. Berry and G. Boudol, The Chemical Abstract Machine, *Theoretical Computer Science*, **96** (1992) 217–248.

[12] M. Boreale and R. DeNicola, Testing Equivalence for Mobile Processes, *CONCUR '92*, Lecture Notes in Computer Science 630 (Springer-Verlag, 1992) 2–16.

[13] M. Boreale, On the Expressiveness of Internal Mobility in Name-Passing Calculi, *CONCUR '96*, Lecture Notes in Computer Science 1119 (Springer-Verlag, 1996) 161–178.

[14] G. Boudol, Asynchrony and π-Calculus, Research Report 1702 (INRIA, Sophia-Antipolis, 1991).

[15] G. Bellin and P. Scott, On the π-Calculus and Linear Logic, *Theoretical Computer Science*, **135** (1994) 11–65.

[16] Y. Fu, The χ-Calculus, *Proceedings of the International Conference on Advances in Parallel and Distributed Computing* (March 19th-21th, Shanghai, IEEE Computer Society Press, 1997) 74–81.

[17] Y. Fu, A Proof Theoretical Approach to Communications, *ICALP '97*, Lecture Notes in Computer Science 1256 (July 7th-11th, Bologna, Italy, Springer-Verlag, 1997) 325–335.

[18] Y. Fu, Reaction Graphs, *Journal of Computer Science and Technology*, **13** (1998) 510–530.

[19] Y. Fu, Symmetric π-Calculus, *Journal of Computer Science and Technology*, **13** (1998) 202–208.

[20] Y. Fu, Bisimulation Lattice of Chi Processes, *ASIAN '98*, Lecture Notes in Computer Science 1538 (December 8-10, Manila, The Philippines, Springer-Verlag, 1998) 245–262.

[21] S. Gay, Linear Types for Communicating Processes, PhD thesis (Department of Computing, Imperial College, 1995).

[22] J. Girard, Linear Logic, *Theoretical Computer Science*, **50** (1987) 1–102.

[23] J. Girard, Towards a Geometry of Interaction, in: J.W. Gray and A. Scedrov, eds., *Categories in Computer Science and Logic*, Contemporary Mathematics, **92** (AMS, 1989) 69–108.

[24] M. Hennessy and H. Lin, Symbolic Bisimulations, *Theoretical Computer Science*, **138** (1995) 353–389.

[25] K. Honda and M. Tokoro, An Object Calculus for Asynchronous Communication, *ECOOP '91*, Lecture Notes in Computer Science 512 (Springer-Verlag, 1991) 133–147.

[26] K. Honda and M. Tokoro, On Asynchronous Communication Semantics, *Object-Based Concurrent Computing*, Lecture Notes in Computer Science 612 (Springer-Verlag, 1991) 21–51.

[27] Honda and Yoshida, On Reduction Based Process Semantics, *Theoretical Computer Science*, **152** (1995) 437–486.

[28] Y. Lafont, Interaction Nets, *POPL '90* (ACM, 1990) 95–108.

[29] Y. Lafont, Interaction Combinators, *Information and Computation*, **137** (1997) 69–101.

[30] H. Lin, Complete Inference Systems for Weak Bisimulation Equivalences in the π-Calculus, *Proceedings of Sixth International Joint Conference on the Theory and Practice of Software Development*, Lecture Notes in Computer Science 915 (Springer-Verlag, 1995) 187–201.

[31] H. Lin, Unique Fixpoint Induction for Mobile Processes, *CONCUR '95*, Lecture Notes in Computer Science 962 (Springer-Verlag, 1995) 88–102.

[32] H. Lin, Complete Proof Systems for Observation Congruences in Finite-Control π-Calculus, *ICALP '98*, Lecture Notes in Computer Science 1443 (Springer-Verlag, 1998) 443–454.

[33] R. Milner, *Communication and Concurrency* (Prentice Hall, 1989).

[34] R. Milner, Functions as Processes, *Mathematical Structures in Computer Science*, **2** (1992) 119–146.

[35] R. Milner, The Polyadic π-Calculus: a Tutorial, *Proceedings of the 1991 Marktoberdorf Summer School on Logic and Algebra of Specification*, NATO ASI, Series F (Springer-Verlag, 1993).

[36] R. Milner, Action Calculi, or Syntactic Action Structure, *MFCS '93*, Lecture Notes in Computer Science 711 (Springer-Verlag, 1993) 105–121.

[37] R. Milner, π-Nets: a Graphical Form of π-Calculus, *ESOP '94*, Lecture Notes in Computer Science 788 (Springer-Verlag, 1994) 26–42.

[38] R. Milner, Calculi for Interaction, *Acta Informatica*, **33** (1995) 707–737.

[39] R. Milner, Action Structures and the π-Calculus, *Proceedings of the 1993 Marktoberdorf Summer School on Proof and Computation*, NATO ASI, Series F (Springer-Verlag, 1995).

[40] A. Mifsud, R. Milner and J. Power, Control Structure, *LICS '95* (IEEE Computer Society Press, 1995) 188–198.

[41] R. Milner, J. Parrow and D. Walker, A Calculus of Mobile Processes, *Information and Computation*, **100** (1992) 1–40 (Part I), 41–77 (Part II).

[42] R. Milner and D. Sangiorgi, Barbed Bisimulation, *ICALP '92*, Lecture Notes in Computer Science 623 (Springer-Verlag, 1992) 685–695.

[43] L. Ong, *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*, PhD thesis (Imperial College of Science and Technology, University of London, 1988).

[44] D. Park, Concurrency and Automata on Infinite Sequences, Lecture Notes in Computer Science 154 (Springer-Verlag, 1981) 561–572.

[45] J. Parrow, Interaction Diagrams, *A Decade of Concurrency*, Lecture Notes in Computer Science 803 (Springer-Verlag, 1993) 477–508.

[46] J. Parrow and D. Sangiorgi, Algebraic Theories for Name-Passing Calculi, *Information and Computation*, **120** (1995) 174–197.

[47] J. Parrow and B. Victor, The Update Calculus, *AMAST '97*, Lecture Notes in Computer Science 1119 (Sydney, December 13-17, Springer-Verlag, 1997) 389–405.

[48] J. Parrow and B. Victor, The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes, *LICS '98* (IEEE Computer Society, 1998) 176–185.

[49] J. Parrow and B. Victor, The Tau-Laws of Fusion, *CONCUR '98*, Lecture Notes in Computer Science 1466, (Springer-Verlag, 1998) 99–114.

[50] G. Plotkin, Call-by Name, Call-by-Value and the λ-Calculus, *Theoretical Computer Science*, **1** (1975) 125–159.

[51] D. Sangiorgi, *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, PhD thesis (Department of Computer Science, University of Edinburgh, 1993).

[52] D. Sangiorgi, From π-Calculus to Higher Order π-Calculus—and Back, *TAPSOFT '93*, Lecture Notes in Computer Science 668 (Springer-Verlag, 1993) 151–166.

[53] D. Sangiorgi, The Lazy Lambda Calculus in a Concurrency Scenario, *Information and Computation*, **111** (1994) 120–153.

[54] D. Sangiorgi, Lazy Functions and Mobile Processes, Technical Report RR-2515 (INRIA, 1995).

[55] D. Sangiorgi, π-Calculus, Internal Mobility and Agent-Passing Calculi, *Theoretical Computer Science*, **167** (1996) 235–274.

[56] D. Sangiorgi, A Theory of Bisimulation for π-Calculus, *Acta Informatica*, **3** (1996) 69–97.

[57] D. Sangiorgi and R. Milner, Techniques of "Weak Bisimulation Up To", *CONCUR '92*, Lecture Notes in Computer Science 630 (Springer-Verlag, 1992).

[58] R. Seely, Linear Logic, $*$-Autonomous Categories and Cofree Coalgebras, in: J.W. Gray and A. Scedrov, *Categories in Computer Science and Logic*, Contemporary Mathematics, **92** (AMS, 1989) 371–382.

[59] G. Takeuti, *Proof Theory*, Studies in Logic 81 (North-Holland, 1975).

[60] B. Thomsen, A Calculus of Higher Order Communicating Systems, *POPL '89* (ACM, 1989) 143–154.

[61] B. Thomsen, Plain CHOCS—A Second Generation Calculus for Higher Order Processes, *Acta Informatica*, **30** (1993) 1–59.

[62] B. Thomsen, A Theory of Higher Order Communicating Systems, *Information and Computation*, **116** (1995) 38–57.

[63] A. Troelstra, *Lectures on Linear Logic*, CSLI Lecture Notes (1992).

[64] B. Victor and J. Parrow, Concurrent Constraints in the Fusion Calculus, *ICALP '98*, Lecture Notes in Computer Science 1443 (Springer-Verlag, 1998) 455–469.

[65] D. Walker, π-Calculus Semantics for Object-Oriented Programming Languages, *TACS '91*, Lecture Notes in Computer Science 526 (Springer-Verlag, 1991) 532–547.

[66] D. Walker, Objects in the π-Calculus, *Information and Computation*, **116** (1995) 253–271.