The χ-Calculus*

Yuxi Fu†‡§

Abstract

The paper proposes a new process algebra, called χ-calculus. The language differs from π-calculus in several aspects. First it takes a more uniform view on input and output. Second, the closed names of the language is homogeneous in the sense that there is only one kind of bound names. Thirdly, the effects of communications in χ-calculus are delimited by localization operators, not by sequentiality combinator. Finally, the language cherishes more freedom of parallelism than π-calculus. The algebraic properties of χ-processes are studied in terms of local bisimulation. It is shown that local bisimilarity is a congruence equivalence on χ-processes.

Key Words: Bisimulation, π-Calculus

1. Questions on π-calculus

Communication is exchange of information. Two parties involved in a communication are in reciprocal positions. Symmetry of communications is a conspicuous feature of the basic CCS. When moving to value-passing calculi, symmetry is lost. The π-calculus ([4]) is but one such a language. Consider the process

$$(a)(a(x).x(y)|(v)\overline{a}v.\overline{v}w).$$

Operationally, $(a)(a(x).x(y)|(v)\overline{a}v.\overline{v}w)$ is no difference to $(a)((x)\overline{a}x.x(y)|a(v).\overline{v}w)$. Moreover, both processes are equivalent to $(a)(a(x).x(y)|(v)(\overline{a}v|\overline{v}w))$ and respectively $(a)((x)(\overline{a}x|x(y))|a(v).\overline{v}w)$. What do all these equivalences suggest? Well, they seem to indicate that there is little need to draw a clear-cut line between input actions and output actions when the output names are restricted. We might as well change the syntax of $a(x).P$ to $(x)ax.P$, where x is free in $ax.P$. With this change, we would have

$$(x)(ax|(y)xy)|(v)(\overline{a}v|\overline{v}w) \xrightarrow{\tau} (u)((y)uy|\overline{u}w).$$

This is an example of communication of local names.

> How do we extend this communication mechanism to communications with free outputs?

It might well be that there are more than one answers to the question.

What then is information? The simplest answer is that information is the content of communication. It is only through communication that a piece of information asserts itself. In value-passing CCS, the contents of communications are extralogical entities. Although a translation into the basic CCS is possible, such an enterprise relies on heavy mathematical assumptions. The greatest achievement of the π-calculus is that it presents itself, in a concise way, as a closed model for concurrent computations. But is there any room for improvement on π-calculus as far as the closedness is concerned? This brings us to a somewhat unpleasant feature of the π-calculus. In π there are two kinds of closed names. There are local names like y in $(y)P$ and there are abstract names like x in $m(x).P$. For a basic model of concurrent computations, two kinds of closed names are probably too more.

> Can we eliminate the abstract names from π-calculus in favour of the local names?

A conceivable solution is to regard the abstract names as special local names.

Often, information is used to control computations whereas communications, or the lack of them, make the control happen. When designing a process calculus, there are two complementary issues. One is to introduce a parallel mechanism so that events can happen in parallel to full extent. The other is to have a control mechanism to curb unwanted parallelism and to place order on life. The π-calculus, as well as most other process calculi, has a parallel mechanism that is beyond

Proceedings of the International Conference on Advances in Parallel and Distributed Computing, March 19-21, 1997, Shanghai, China, IEEE Computer Society Press, 74–81.

†Supported by the National Nature Science Foundation of China, grant number 69503006.

‡Department of Computer Science, Shanghai Jiao Tong University, 1954 Hua Shan Road, Shanghai 200030, China.

§E-mail: fu-yx@cs.sjtu.edu.cn.

question. The parallel combinator '|' intends to permit full-scale parallelism and to allow controls to have their ways. But are there any situations in π-calculus in which the orders of computations are always preordained? Suppose two processes P and Q share an abstract name x. The situation can be described as in $m(x).(P|Q)$. What if we want P to evolve independently from the instantiation of x through m? In π there is no way to do that. For a language that deals with concurrency, the requirement of this example is not over-demanding. A good control mechanism exerts controls, not unintended restrictions.

> How do we eliminate the restriction imposed by the control mechanism of the π-calculus?

Searching for an answer to the question is one of the motivations for the study carried out in this paper.

The restriction mentioned above is to do with the sequentiality operator[1] of the π-calculus. Let's take a close look at the combinator. The process $\overline{m}n.P$ emits n through medium m and then proceeds as P. The sequentiality appeared here is nice and simple. On the other hand, the behaviour of $m(x).P$ is different. After receiving a name n through m, the process continues to evolve as $P[n/x]$. From a purely set theoretical point of view, $m(x).P$ is nothing but a function. The functionality of $m(x).P$ has serious implication: the effect of any communication $m(x).P$ to participate is restricted to the body of the function, that is to P. In other words, the sequentiality operator also acts as an effect delimiter. But if the operator is to prescribe orders of actions, then why should it have anything to do with the scope of the effect of a communication?

> What if we, and how do we, free the sequentiality combinator from the duty of delimiting the effects of communications?

It would be a bad idea to introduce a completely new combinator to fulfill the role of effect delimiter. We already have in π-calculus a combinator that defines the scope of a name. It appears more sensible to let the localization operator play the role because that is what it is meant to be—a scope delimiter.

There is however a more dramatic question about the sequentiality combinator:

> Is sequentiality operator necessary for a concurrent computation model?

The answer depends to a large extent on how we see the relationship between sequential and concurrent computations. One view is that the notion of concurrency is based upon that of sequentiality. Without sequential computations, there is no way to talk about concurrent computations. This is the background against which the sequentiality operator is introduced. An alternative view is that sequential computations are special concurrent computations. For those of us who take this second attitude, we need a process calculus to back up our viewpoint.

In this paper, we propose a new process algebra, χ-calculus, that gives our answers to the first four questions. By omitting the sequentiality operator of the calculus, question 5 can also be investigated.

2. Communication via information exchange

This section introduces a new process calculus. Syntactically, the language differs from the π-calculus by unifying input and output prefixes. Semantically, it represents a more drastic departure. The value-passing communication mechanism of the π-calculus is replaced by an information exchange communication mechanism. The language will be called χ-calculus, where χ stands for ex*ch*ange of *i*nformation.

Let $\mathcal{N}$ be a set of names ranged over by lower case letters, $\mathcal{P}_\chi$ the set of χ-processes defined as follows:

$$P := \mathbf{0} \mid m_i[x].P \mid P|P \mid (x)P \mid [x{=}y]P \mid !P,$$

where $i \in \{-1, 1\}$. Here m_{-1} and m_1 are the two ends of channel m. As usual $\mathbf{0}$ is the inactive process. A trailing inactive process will be omitted. $m_i[x].P$ is a process that must first perform a communication through channel m and then enacts $P[y/x]$, where y is the name received in the communication. $P|Q$ is a process of parasition form. Here P and Q can evolve independently and may communicate during the course of their evolution. $(x)P$ is a process where x is local to P, meaning that $(x)P$ is not allowed to communicate with another process through channel x. The conditional process $[x{=}y]P$ behaves the same as P if $x{=}y$, or it is the same as the inactive process $\mathbf{0}$. The replication process $!P$ provides potentially infinite copies of P. The set of local names appeared in P is denoted by $ln(P)$. The set of global names, or nonlocal names, in P is designated by $gn(P)$. The set $n(P)$ is the union of $ln(P)$ and $gn(P)$. We adopt the α-convention saying that a local name in a process can be replaced by a fresh name without changing the syntax of the process.

The following rules define the operational semantics of χ-calculus. Notice that we have left out all the sym-

[1] We call the dot in $\alpha.P$ a sequentiality combinator.

metric rules.

Sequention

$$\frac{}{m_i[x].P \xrightarrow{m_i\langle y/x\rangle} P[y/x]}\mathsf{S}_0$$

Parasition

$$\frac{Q \xrightarrow{\tau} Q'}{P|Q \xrightarrow{\tau} P|Q'}\mathsf{P}_0 \qquad \frac{Q \xrightarrow{m_i\langle y/x\rangle} Q'}{P|Q \xrightarrow{m_i\langle y/x\rangle} P[y/x]|Q'}\mathsf{P}_1$$

$$\frac{Q \xrightarrow{m_i\langle x\rangle} Q'}{P|Q \xrightarrow{m_i\langle x\rangle} P|Q'}\mathsf{P}_2 \qquad \frac{Q \xrightarrow{\langle y/x\rangle} Q'}{P|Q \xrightarrow{\langle y/x\rangle} P[y/x]|Q'}\mathsf{P}_3$$

Communication

$$\frac{P \xrightarrow{m_i\langle x/x\rangle} P' \quad Q \xrightarrow{m_{-i}\langle x\rangle} Q'}{P|Q \xrightarrow{\tau} P'|Q'}\mathsf{C}_0$$

$$\frac{P \xrightarrow{m_i\langle y/y\rangle} P' \quad Q \xrightarrow{m_{-i}\langle y/x\rangle} Q'}{P|Q \xrightarrow{\langle y/x\rangle} P'[y/x]|Q'}\mathsf{C}_1$$

$$\frac{P \xrightarrow{m_i\langle x\rangle} P' \quad Q \xrightarrow{m_{-i}\langle x\rangle} Q' \quad x \notin gn(P,Q)}{P|Q \xrightarrow{\tau} (x)(P'|Q')}\mathsf{C}_2$$

Localization

$$\frac{P \xrightarrow{\alpha} P' \quad x \notin n(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}\mathsf{L}_0 \qquad \frac{P \xrightarrow{\langle y/x\rangle} P' \quad x \neq y}{(x)P \xrightarrow{\tau} P'}\mathsf{L}_1$$

$$\frac{P \xrightarrow{m_i\langle y/x\rangle} P' \quad x \neq m}{(x)P \xrightarrow{m_i\langle y\rangle} P'}\mathsf{L}_2 \qquad \frac{P \xrightarrow{\langle x/x\rangle} P'}{(x)P \xrightarrow{\tau} (x)P'}\mathsf{L}_3$$

Replication *Condition*

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'|!P}\mathsf{R} \qquad \frac{P \xrightarrow{\alpha} P'}{[x{=}x]P \xrightarrow{\alpha} P'}\mathsf{C}$$

In the above rules, α ranges over the set $\{\tau\} \cup \{m_i\langle y/x\rangle, m_i\langle x\rangle, \langle y/x\rangle \mid m, x, y \in \mathcal{N}\}$. The notation $[y/x]$ stands 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 applying a substitution to a process is defined as follows:

$$\begin{aligned} P[] &\stackrel{\text{def}}{=} P \\ P[y_1/x_1]\ldots[y_n/x_n] &\stackrel{\text{def}}{=} (\ldots P[y_1/x_1]\ldots)[y_n/x_n], \end{aligned}$$

where [] is the empty substitution. Substitutions will be ranged over by σ. The reflexive and transitive closure of $\xrightarrow{\tau}$ is denoted by $\Longrightarrow$. For $\alpha \neq \tau$, the notation $\stackrel{\alpha}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$.

The operational rules are simpler than they appear to be. To help understand the communication rules, we now give some examples. In the process $(x)(R|(m_{-1}[n].P|m_1[x].Q))$, a communication via m is possible. The effect of the communication is to replace the local name x by the global name n throughout the term over which the localization operator (x) applies. The global name n in P however remains unchanged. In other words, global information overwrites. The reduction

$$(x)(R|(m_{-1}[n].P|m_1[x].Q)) \xrightarrow{\tau} (R|(P|Q))[n/x]$$

is not derivable by a single rule. The only communication rule applicable here is C_1. But C_1 does only part of the job. The result is a labeled reduction representing an incomplete action:

$$m_{-1}[n].P|m_1[x].Q \xrightarrow{\langle n/x\rangle} P[n/x]|Q[n/x].$$

To continue, we use the rule P_3 to get

$$R|(m_{-1}[n].P|m_1[x].Q) \xrightarrow{\langle n/x\rangle} R[n/x]|(P|Q)[n/x].$$

An application of L_1 then finishes the job. Another situation arises if we modify the example to $m_{-1}[n].P|(x)(R|m_1[x].Q)$. The difference is that here the term $m_{-1}[n].P$ lies outside the scope of the localization operator (x). This time we need to use C_0. But before that an application of rule L_2 is necessary. Next consider the following reduction:

$$(x)m_{-1}[x].P|(y)m_1[y].Q \xrightarrow{\tau} (z)(P[z/x]|Q[z/y]).$$

It is an example of communications between local names. The communication rule applied is C_2. A rule that also calls for explanation is L_3. It is necessary when two processes exchange a same local name. For example the reduction

$$(x)(m_{-1}[x].P|m_1[x].Q) \xrightarrow{\tau} (x)(P|Q)$$

is obtained from $m_{-1}[x].P|m_1[x].Q \xrightarrow{\langle x/x\rangle} P|Q$ by applying L_3.

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.

Let $\mathcal{SC}$ be the set of contexts of the form

$$(\vec{x_n})(\ldots(\vec{x_1})(\_|R_1)\ldots|R_n),$$

where $n \geq 1$ and $R_1, \ldots, R_n \in \mathcal{P}_\chi$. Notice that the length of $\vec{x_i}$, for $i \leq n$, could be zero. An element of $\mathcal{SC}$ is called a static context. One useful property of a static context $C[]$ is that any derivative of $C[P]$ is of the form $C'[P']$, where $C'[]$ is static and P' is what P has turned into.

3. Bisimulation congruence

A difficulty with the bisimulation equivalence for χ-processes is to do with the incomplete actions. Suppose P and Q are bisimilar, should we require $P \stackrel{\langle y/x\rangle}{\longrightarrow}$ be matched by $Q \stackrel{\langle y/x\rangle}{\longrightarrow}$? If incomplete actions are completely ignored, the bisimilarity would not be closed under localization. But if we treat them the same as the other actions, the resulting equivalence would be difficult to analyze. We propose to use local bisimulation as a tool to study the algebraic properties of χ-processes. Throughout this paper, we concentrate on weak equivalences.

Definition 3.1 *Suppose $\mathcal{R} \subset \mathcal{P}_\chi \times \mathcal{P}_\chi$. The relation $\mathcal{R}$ is a local simulation if whenever $P\mathcal{R}Q$, then for any χ-process R and any sequence $\vec{x}$ of names it holds that*

if $(\vec{x})(P|R) \stackrel{m_i\langle x\rangle}{\Longrightarrow} P'$, then Q' exists such that $(\vec{x})(Q|R) \stackrel{m_i\langle x\rangle}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$.

$\mathcal{R}$ is a local bisimulation if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are local simulations. P is locally bisimilar to Q, notation $P \approx Q$, if $P\mathcal{R}Q$ for some local bisimulation $\mathcal{R}$.

Theorem 3.2 *$\approx$ is an equivalence relation.*

Definition 3.3 *Suppose $\mathcal{R} \subset \mathcal{P}_\chi \times \mathcal{P}_\chi$. The relation $\mathcal{R}$ is a local simulation up to $\approx$ if $P\mathcal{R}Q$ implies that for any process R and sequence $\vec{x}$ of names it holds that*

if $(\vec{x})(P|R) \stackrel{m_i\langle x\rangle}{\Longrightarrow} (\vec{y})P'$, then Q' exists such that $(\vec{x})(Q|R) \stackrel{m_i\langle x\rangle}{\Longrightarrow} (\vec{y})Q'$ and $P' \approx \mathcal{R} \approx Q'$.

$\mathcal{R}$ is a local bisimulation up to $\approx$ if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are local simulations up to $\approx$.

Theorem 3.4 *If $\mathcal{R}$ is a local bisimulation up to $\approx$, then $\mathcal{R} \subset \approx$.*

The proof of this theorem uses the next result.

Proposition 3.5 *If $P \approx Q$, then*
(i) for any χ-process R and any sequence $\vec{x}$ of names, if $(\vec{x})(P|R) \Longrightarrow P'$ then Q' exists such that $(\vec{x})(Q|R) \Longrightarrow Q'$ and $P' \approx Q'$;
(ii) if $P \Longrightarrow P'$ then $Q \Longrightarrow Q'$ for some Q' such that $P' \approx Q'$;
(iii) if $P \stackrel{m_i\langle x\rangle}{\Longrightarrow} P'$ then $Q \stackrel{m_i\langle x\rangle}{\Longrightarrow} Q'$ for some Q' such that $P' \approx Q'$.

PROOF: (i) Suppose $(\vec{x})(P|R) \Longrightarrow P''$. Then P'' must be of the form $(\vec{x'})(P'|R')$. If $a \notin \{\vec{x}\} \cup n(P,Q)$, then $(\vec{x})(P|(R|(y)a_1[y])) \stackrel{a_1\langle z\rangle}{\Longrightarrow} (\vec{x'})(P'|(R'|\mathbf{0}))$. In order to match this action, we must have

$$(\vec{x})(Q|(R|(y)a_1[y])) \stackrel{a_1\langle z\rangle}{\Longrightarrow} (\vec{x''})(Q'|(R''|\mathbf{0}))$$

such that $(\vec{x'})(P'|(R'|\mathbf{0})) \approx (\vec{x''})(Q'|(R''|\mathbf{0}))$. It follows easily that $(\vec{x'})(P'|R') \approx (\vec{x''})(Q'|R'')$. But then $(\vec{x})(Q|R) \Longrightarrow (\vec{x''})(Q'|R'')$. Done. (ii) is a special case of (i) and (iii) can be proved similarly. □

Theorem 3.6 *The following properties hold:*
(i) $(m)m_i[x].P \approx \mathbf{0}$;
(ii) $(x)m_i[y].P \approx m_i[y].(x)P$, where $x \notin \{m, y\}$;
(iii) $(x)\mathbf{0} \approx \mathbf{0}$;
(iv) $(x)(y)P \approx (y)(x)P$;
(v) $(x)(P|Q) \approx P|(x)Q$, where $x \notin gn(P)$;
(vi) $P|\mathbf{0} \approx P$;
(vii) $P_1|P_2 \approx P_2|P_1$;
(viii) $P_1|(P_2|P_3) \approx (P_1|P_2)|P_3$;
(ix) $[x{=}x]P \approx P$;
(x) $(x)[y{=}z]P \approx [y{=}z](x)P$, where $x \notin \{y, z\}$;
(xi) $(x)[y{=}z]P \approx \mathbf{0}$, where $x \in \{y, z\}$ and $y \neq z$;
(xii) $[x{=}y][v{=}w]P \approx [v{=}w][x{=}y]P$.

The proof of the theorem is standard. Notice that in general $(x)(R|[x{=}a]P)$ is not locally bisimilar to $(x)R$. This is one of the differences between χ-calculus and π-calculus. The next lemma is crucial to showing that $\approx$ is a congruence relation.

Lemma 3.7 *If $P \approx Q$, then $P[y/x] \approx Q[y/x]$.*

PROOF: Let $\mathcal{R}$ be the union of $\approx$ and the following

$$\left\{ (C[P\sigma], C[Q\sigma]) \;\middle|\; \begin{array}{l} P \approx Q,\ C[] \in \mathcal{SC}, \\ \sigma \equiv [y_1/x_1]\ldots[y_n/x_n], \\ \vec{x} \text{ pairwise distinct} \end{array} \right\}.$$

Suppose $P \approx Q$ and $C[P\sigma] \stackrel{m_i\langle a\rangle}{\Longrightarrow} S$. Two cases:

- $P\sigma$ does not participate in the sequence of actions $C[P\sigma] \stackrel{m_i\langle a\rangle}{\Longrightarrow} S$. Then S must be of the form $C'[P\sigma\sigma']$, where σ' is some substitution $[y'_1/x'_1]\ldots[y'_{n'}/x'_{n'}]$. Notice that $x'_1, \ldots, x'_{n'}$ were local names. So by α-convention we can assume that $x_1, \ldots, x_n, x'_1, \ldots, x'_{n'}$ are pairwise distinct. But then we have $C[Q\sigma] \stackrel{m_i\langle a\rangle}{\Longrightarrow} C'[Q\sigma\sigma']$ with $S\mathcal{R}C'[Q\sigma\sigma']$.
- $P\sigma$ does participate in the sequence of actions $C[P\sigma] \stackrel{m_i\langle a\rangle}{\Longrightarrow} S$. As the process $C[P\sigma|(\mathbf{0}|\mathbf{0})]$ is locally bisimilar to $C[P\sigma]$, some S' exists such that $C[P\sigma|(\mathbf{0}|\mathbf{0})] \stackrel{m_i\langle a\rangle}{\Longrightarrow} S'$ and $S \approx S'$. Suppose σ is the substitution $[y_1/x_1]\ldots[y_n/x_n]$ and o is not in

$n(C[P], C[Q]) \cup \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$. Let $D[]$ be the static context $C[(\vec{x})(\_|A)]$ where A is $o_{-1}[x_1].\cdots.o_{-1}[x_n]|o_1[y_1].\cdots.o_1[y_n]$. Now

$$(1) \qquad D[P] \stackrel{\tau}{\Longrightarrow} C[P\sigma|(\mathbf{0}|\mathbf{0})] \stackrel{m_i\langle a\rangle}{\Longrightarrow} S'.$$

By assumption, $P \approx Q$. So T' exists such that

$$(2) \qquad D[Q] \stackrel{m_i\langle a\rangle}{\Longrightarrow} T'$$

and $S' \approx T'$. For (2) to match (1), all actions of $o_{-1}[x_1].\cdots.o_{-1}[x_n]$ must have reacted upon the corresponding actions of $o_1[y_1].\cdots.o_1[y_n]$ when $D[Q]$ has evolved to T', or $S' \approx T'$ would not hold. Next we show that (2) can be factorized as

$$(3) \qquad D[Q] \stackrel{\tau}{\Longrightarrow} C[Q\sigma|(\mathbf{0}|\mathbf{0})] \stackrel{m_i\langle a\rangle}{\Longrightarrow} T'.$$

For simplicity, we assume that (2) is

$$(4) \qquad \begin{array}{ll} & C[(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))] \\ \stackrel{m_i\langle a\rangle}{\longrightarrow} & C'[(x_1)(Q'|(o_{-1}[x_1]|o_1[y_1']))] \\ \stackrel{\tau}{\longrightarrow} & T', \end{array}$$

and we need to prove that

$$(5) \qquad \begin{array}{ll} & C[(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))] \\ \stackrel{\tau}{\longrightarrow} & C[Q[y_1/x_1]|(\mathbf{0}|\mathbf{0})] \\ \stackrel{m_i\langle a\rangle}{\longrightarrow} & T'. \end{array}$$

The general case can be proved by induction. (4) can be rewritten as

$$\begin{array}{ll} & C[(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))] \\ \stackrel{m_i\langle a\rangle}{\longrightarrow} & C'[(x_1)(Q\sigma|(o_{-1}[x_1]|o_1[y_1\sigma]))] \\ \stackrel{\tau}{\longrightarrow} & C'[Q\sigma[y_1\sigma/x_1]|(\mathbf{0}|\mathbf{0})] \\ \equiv & C'[Q[y_1/x_1]\sigma|(\mathbf{0}|\mathbf{0})]. \end{array}$$

If the first action doesn't have any effect on $(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))$, then σ is $[]$. If it does affect the process $(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))$, then σ must be $[a/z]$ such that x_1, a and z are pairwise distinct. In either case, the syntactical equality holds. On the other hand, (5) is

$$\begin{array}{ll} & C[(x_1)(Q|(o_{-1}[x_1]|o_1[y_1]))] \\ \stackrel{\tau}{\longrightarrow} & C[Q[y_1/x_1]|(\mathbf{0}|\mathbf{0})] \\ \stackrel{m_i\langle a\rangle}{\longrightarrow} & C'[Q[y_1/x_1]\sigma|(\mathbf{0}|\mathbf{0})]. \end{array}$$

So we have proved that (2) can be rearranged as (3). Now $C[Q\sigma|(\mathbf{0}|\mathbf{0})] \stackrel{m_i\langle a\rangle}{\Longrightarrow} T'$ implies that some T exists such that $C[Q\sigma] \stackrel{m_i\langle a\rangle}{\Longrightarrow} T$ with $T' \approx T$. So $S \approx T$. This closes up the bisimulation game.

So $\mathcal{R}$ is a local bisimulation. □

It is worth remarking that the lemma does not hold for π-calculus. Therefore local bisimilarity for π-processes is not closed under input prefixing operation.

Theorem 3.8 *$\approx$ is a congruence equivalence:*
(i) $m_i[x].P \approx m_i[x].Q$ whenever $P \approx Q$;
(ii) $P|O \approx Q|O$ whenever $P \approx Q$;
(iii) $(x)P \approx (x)Q$ whenever $P \approx Q$;
(iv) $[x{=}y]P \approx [x{=}y]Q$ whenever $P \approx Q$;
(v) $!P \approx !Q$ whenever $P \approx Q$.

PROOF: (i) The proof is easy.
(ii) It suffices to show that $\{(P|O, Q|O) \mid P \approx Q \wedge O \in \mathcal{P}_\chi\} \cup \approx$ is a local bisimulation up to $\approx$. Suppose $R \in \mathcal{P}_\chi$ and $\vec{x}$ is a sequence of names. It is rather easy to show that $(\vec{x})((P|O)|R) \approx (\vec{x})(P|(O|R))$ and $(\vec{x})((Q|O)|R) \approx (\vec{x})(Q|(O|R))$. If $(\vec{x})((P|O)|R) \stackrel{m_i\langle y\rangle}{\Longrightarrow} P'$, then by (ii) of proposition 3.5 P'' exists such that $(\vec{x})(P|(O|R)) \stackrel{m_i\langle y\rangle}{\Longrightarrow} P''$ and $P' \approx P''$. So by definition, Q'' exists with $(\vec{x})(Q|(O|R)) \stackrel{m_i\langle y\rangle}{\Longrightarrow} Q''$ and $P'' \approx Q''$. Using proposition 3.5 again, one has Q' such that $(\vec{x})((Q|O)|R) \stackrel{m_i\langle y\rangle}{\Longrightarrow} Q'$ and $Q'' \approx Q'$.
(iii) The proof is similar to that of (ii).
(iv) $\{(C[[x{=}y]P], C[[x{=}y]Q]) \mid P \approx Q \wedge C[] \in \mathcal{SC}\}$ is a local bisimulation. If $x{=}y$, then the result follows from (ix) of theorem 3.6, (ii) and (iii). So assume $x \neq y$ and $C[[x{=}y]P] \stackrel{m_i\langle z\rangle}{\Longrightarrow} P'$. Consider only one case. Suppose $C[[x{=}y]P] \stackrel{m_i\langle z\rangle}{\Longrightarrow} C'[([x{=}y]P)\sigma]$ for some static context $C'[]$ and some substitution σ such that $x\sigma = y\sigma$. Correspondingly we have $C[[x{=}y]Q] \stackrel{m_i\langle z\rangle}{\Longrightarrow} C'[([x{=}y]Q)\sigma]$. By lemma 3.7, $P\sigma \approx Q\sigma$. The rest of the argument is similar to that when $x = y$.
(v) $\mathcal{R} \stackrel{\text{def}}{=} \{(C[!P], C[!Q]) \mid P \approx Q \wedge C[] \in \mathcal{SC}\}$ is a local bisimulation up to $\approx$. Consider only one case. If $C[!P] \stackrel{m_i\langle x\rangle}{\Longrightarrow} C[P'|!P]$ is induced by $P \stackrel{m_i\langle x\rangle}{\Longrightarrow} P'$, then $Q \stackrel{m_i\langle x\rangle}{\Longrightarrow} Q'$ with $P' \approx Q'$ by proposition 3.5. So $C[!Q] \stackrel{m_i\langle x\rangle}{\Longrightarrow} C[Q'|!Q]$. Now by (vii) of theorem 3.6, (ii) and (iii), $C[P'|!P] \approx C[Q'|!P] \approx C[!P|Q']\mathcal{R}C[!Q|Q'] \approx C[Q'|!Q]$. □

In [5] barbed bisimulation is proposed as a universal tool applicable to a variety of process calculi. Let $\approx_b$ be the barbed bisimilarity on χ-processes.

Theorem 3.9 *$\approx_b$ is the same as $\approx$.*

The proof of the theorem consists of two parts. In one direction, one shows that $\approx_b$ is a local bisimulation. In the other direction, one proves that $\approx$ is a barbed bisimulation. Both are straightforward.

4. Higher order χ-calculus

There are two typical kinds of recursion mechanisms employed in process algebras. In CCS the recursion mechanism consists of the fix-expressions and the rule

$$\frac{E[\mathsf{fix}(E)/X] \xrightarrow{\alpha} E'}{\mathsf{fix}(E) \xrightarrow{\alpha} E'}\mathsf{rec.}$$

What one should notice about the rule is that it adds nothing to the communication mechanism. What the rule really says is that $E[\mathsf{fix}(E)/X]$ and $\mathsf{fix}(E)$ can be regarded as syntactically the same. The same can be said about the recursion mechanism via the duplicator in π-calculus. So the first order approach essentially admits infinite long expressions that can be coded up by finite number of symbols. In a first order calculus, the recursion power is achieved syntactically. The other approach is the higher order one which allows a process to be the content of a communication. In a higher order process calculus, unlike in the first order case, the recursion and the communication mechanisms are intertwined; the former is part of the latter. That is to say that the recursion power is achieved semantically. The two methods are not unrelated. The higher order one can code up the first order one, whereas the first order one, in case of the higher order polyadic π-calculus ([6]) and the (Plain) CHOCS ([7, 8]), is able to imitate the higher order one.

We now take a look at χ^ω, the higher order χ-calculus. Part of the agenda in this section is to demonstrate that local bisimulation is the right technical tool for proving equivalence of higher order processes. Let $\mathcal{V}$ be the set of process variables, ranged over by $X, Y, Z, \ldots$. The set $\mathcal{E}$ of process expressions is the least set consisting of the expressions defined by the following abstract syntax:

$$\begin{aligned} E \quad := \quad & \mathbf{0} \mid X \mid m_i[n].E \mid E|E \mid (x)E \\ & \mid m_i(X).E \mid m_i[E].E, \end{aligned}$$

where $i \in \{-1, 1\}$. For a subset V of $\mathcal{V}$, let $\mathcal{E}[V]$ denote the set of process expressions whose free variables are all in V. When V is a finite set $\{X_1, \ldots, X_n\}$, we write $\mathcal{E}[X_1, \ldots, X_n]$ for $\mathcal{E}[\{X_1, \ldots, X_n\}]$. A higher order χ-process is a process expression that does not contain any variables. The set of all higher order χ-processes is denoted by $\mathcal{P}_{\chi^\omega}$, which will be ranged over by $P, Q, R, \ldots$. The operational semantics for the higher order χ-calculus is defined by the operational rules of the first order χ-calculus together with the following rules, noticing that we have again left out all the symmetric rules:

Sequention

$$\frac{}{m_i[A].P \xrightarrow{m_i[A]} P}\mathsf{S}_1 \qquad \frac{gn(A) \cap ln(P) = \emptyset}{m_i(X).P \xrightarrow{m_i\langle A\rangle} P[A/X]}\mathsf{S}_2$$

Parasition

$$\frac{Q \xrightarrow{m_i\langle A\rangle} Q'}{P|Q \xrightarrow{m_i\langle A\rangle} P|Q'}\mathsf{P}_4$$

$$\frac{Q \xrightarrow{(\vec{x})m_i[A]} Q' \quad \{\vec{x}\} \cap gn(P) = \emptyset}{P|Q \xrightarrow{(\vec{x})m_i[A]} P|Q'}\mathsf{P}_5$$

Communication

$$\frac{P \xrightarrow{(\vec{x})m_i[A]} P' \quad Q \xrightarrow{m_{-i}\langle A\rangle} Q' \quad \{\vec{x}\} \cap gn(Q) = \emptyset}{P|Q \xrightarrow{\tau} (\vec{x})(P'|Q')}\mathsf{C}_3$$

Localization

$$\frac{P \xrightarrow{m_i\langle A\rangle} P' \quad x \notin gn(A)}{(x)P \xrightarrow{m_i\langle A\rangle} (x)P'}\mathsf{L}_4$$

$$\frac{P \xrightarrow{(\vec{x})m_i[A]} P' \quad y \in gn(A) \setminus \{\vec{x}\}}{(y)P \xrightarrow{(y)(\vec{x})m_i[A]} P'}\mathsf{L}_5$$

$$\frac{P \xrightarrow{(\vec{x})m_i[A]} P' \quad y \notin gn(A) \setminus \{\vec{x}\}}{(y)P \xrightarrow{(\vec{x})m_i[A]} (y)P'}\mathsf{L}_6$$

In the rules, α ranges over all actions. Notice that in this language, the replicator is unnecessary, a fact to be justified below. Algebraic equivalences for higher order processes are more interesting than those in the first order case. Again the difficulty is to do with the treatment of output actions. When comparing for example $a_i[A].P$ and $a_i[B].Q$, it is much too strong to require A and B be syntactically the same in order for $a_i[A].P$ and $a_i[B].Q$ to be equivalent. A refined treatment allows us to regard the processes as equivalent whenever A and B, P and Q are equivalent. The higher order bisimilarities used in [7, 8] are precisely equivalences of this kind. But still, these equivalences are too strong. The reason is that when comparing $a_i[A].P$ and $a_i[B].Q$ in this approach, the processes A and P, respectively B and Q, are considered separately. This overlooks any possible localization operators that might surround $a_i[A].P$. To take this into account, context bisimilarity is proposed in [6]. It is shown in the paper that context bisimilarity coincides with the barbed bisimilarity on higher order π-processes.

Our approach to the equivalence of higher order χ-processes is similar to that based on context bisimulations. The definition of local bisimulation for higher order χ-processes is completely the same as definition 3.1;

and the notion of local bisimulation up to $\approx$ can be similarly defined as in 3.3. It follows that the results in theorem 3.2, theorem 3.4, proposition 3.5, theorem 3.6, and lemma 3.7 all hold for higher order χ-processes. The proofs of them are just as simple.

Theorem 4.1 *$\approx$ is preserved by all combinators:*
(i) $m_i[x].P \approx m_i[x].Q$ whenever $P \approx Q$;
(ii) $P|O \approx Q|O$ whenever $P \approx Q$;
(iii) $(x)P \approx (x)Q$ whenever $P \approx Q$;
(iv) $[x{=}y]P \approx [x{=}y]Q$ whenever $P \approx Q$;
(v) $!P \approx !Q$ whenever $P \approx Q$;
(vi) $m_i[A].P \approx m_i[A].Q$ whenever $P \approx Q$;
(vii) $m_i[A].P \approx m_i[B].P$ whenever $A \approx B$.

PROOF: We only prove (vii). For the sake of this proof, let's define $\mathcal{E}_o[X]$ to be the set of all process expressions E in $\mathcal{E}[X]$ such that each occurrence of X is within $a_i[F]$ for some $a \in \mathcal{N}$ and some process expression F. We first prove an auxiliary fact: Suppose $E \in \mathcal{E}_0[X]$. Then

- if $E[A/X] \stackrel{m_i\langle x\rangle}{\longrightarrow} P$, then $E[B/X] \stackrel{m_i\langle x\rangle}{\longrightarrow} Q$ such that $P \equiv F[A/X]$ and $Q \equiv F[B/X]$ for some F in $\mathcal{E}_o[X]$;
- if $E[A/X] \stackrel{\tau}{\longrightarrow} P$, then $E[B/X] \stackrel{\tau}{\longrightarrow} Q$ such that $P \equiv F[A/X]$ and $Q \approx F[B/X]$ for some F in $\mathcal{E}_o[X]$.

The proof of the fact goes as follows:

- If $E[A/X] \stackrel{m_i\langle x\rangle}{\longrightarrow} P$, then clearly $P \equiv F[A/X]$ for some $F \in \mathcal{E}_o[X]$ and $E[B/X] \stackrel{m_i\langle x\rangle}{\longrightarrow} F[B/X]$.
- If $E[A/X] \stackrel{\tau}{\longrightarrow} P$ is a first order communication, then $P \equiv F[A/X]$ for some $F \in \mathcal{E}_o[X]$ and $E[B/X] \stackrel{\tau}{\longrightarrow} F[B/X]$.
- If $E[A/X] \stackrel{\tau}{\longrightarrow} P$ is a higher order communication, then $P \equiv F[A/X]$ for some $F \in \mathcal{E}_o[X]$. Now $E[B/X] \stackrel{\tau}{\longrightarrow} Q$ by carrying out the 'same' communication. If we replace by A all the occurrences of B in Q that do not appear in an output expression, we get $F[B/X]$. By (i) through (vi), $F[B/X] \approx Q$.

This concludes the proof of the auxiliary fact. Let $\mathcal{R}$ be $\{(E[A/X], E[B/X]) \mid A \approx B \wedge E \in \mathcal{E}_o[X]\}$. Now suppose $E[A/X]\mathcal{R}E[B/X]$ and $E[A/X] \stackrel{m_i\langle x\rangle}{\Longrightarrow} P$. Then $P \equiv F[A/X]$ for some $F \in \mathcal{E}_o[X]$. By the above fact, we can find, by repeatedly using (ii) of proposition 3.5, some Q such that $E[B/X] \stackrel{m_i\langle x\rangle}{\Longrightarrow} Q$ and $F[B/X] \approx Q$. This should be enough of an evidence that $\mathcal{R}$ is a local bisimulation up to $\approx$. The result follows by observing that $(m_i[A].P)\mathcal{R}(m_i[B].P)$. □

Let $E, F \in \mathcal{E}[X_1, \ldots, X_n]$. Define $E \approx F$ if for any processes $P_1, \ldots, P_n \in \mathcal{P}_{\chi^\omega}$, it holds that $E[\vec{P}/\vec{X}] \approx F[\vec{P}/\vec{X}]$.

Corollary 4.2 *Suppose $E \in \mathcal{E}[X]$. If $A \approx B$, then $E[A/X] \approx E[B/X]$.*

5. General recursion in χ^ω

In [7], the general recursion $recX.E$ is defined as follows:

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

One has

$$recX.E \stackrel{\tau}{\longrightarrow} (a)(E[a?X.(E|a!X)]|a!(a?X.(E|a!X))).$$

This reduction looks like an operational simulation of the recursion equation $recX.E = E[recX.E/X]$. Unfortunately the equation is not verified by the higher order bisimilarity. This is because the higher order bisimilarity has too strong a distinguishing power. However, the encoding *per se* is good, a fact we are going to prove.

Suppose $E \in \mathcal{E}[X]$ and a does not occur in E. The following abbreviations will be used:

$$\begin{aligned} W_a(E) &\stackrel{\text{def}}{=} a_1(X).(E|a_{-1}[X]), \\ \mathsf{rec}X.E &\stackrel{\text{def}}{=} (a)(W_a(E)|a_{-1}[W_a(E)]). \end{aligned}$$

Before proving the property concerning $\mathsf{rec}X.E$, we first establish a useful result. Let $F[\Omega_E]$ denote the expression $(a)(F[W_a(E)/X]|a_{-1}[W_a(E)])$ where a is fresh.

Lemma 5.1 *Suppose $E, F \in \mathcal{E}[X]$, b is fresh and F' is obtained from $F[W_a(E)/X]$ by replacing some occurrences of $W_a(E)$ by $W_b(E)$. Then $F[\Omega_E]$ is locally bisimilar to $(a)(b)(F'|(a_{-1}[W_a(E)]|b_{-1}[W_b(E)]))$.*

PROOF: Suppose $(a)(F[W_a(E)/X]|a_{-1}[W_a(E)]) \stackrel{\tau}{\longrightarrow} P$ is caused by a higher order communication between $F[W_a(E)/X]$ and $a_{-1}[W_a(E)]$. Then clearly

$$P \approx (a)(H[W_a(E)/X]|a_{-1}[W_a(E)])$$

for some $H \in \mathcal{E}[X]$. By carrying out essentially the same higher order communication between either F' and $a_{-1}[W_a(E)]$ or F' and $b_{-1}[W_b(E)]$, we get, for some $Q \in \mathcal{P}_{\chi^\omega}$,

$$\begin{aligned} & (a)(b)(F'|(a_{-1}[W_a(E)]|b_{-1}[W_b(E)])) \\ \stackrel{\tau}{\longrightarrow}\approx \quad & (a)(b)(H'|(a_{-1}[W_a(E)]|b_{-1}[W_b(E)])) \end{aligned}$$

where H' is obtained from $H[W_a(E)/X]$ by replacing some occurrences of $a_{-1}[W_a(E)]$ by $b_{-1}[W_b(E)]$. From this example, it is easy to see that

$$\left\{ \begin{array}{l} ((\vec{x})(a)(F[W_a(E)/X]|a_{-1}[W_a(E)]), \\ (\vec{x})(a)(F'|(a_{-1}[W_a(E)]|b_{-1}[W_b(E)]))) \end{array} \;\middle|\; \phi \right\}$$

is a local bisimulation up to $\approx$, where ϕ is

> $E, F \in \mathcal{E}[X]$, $a, b \notin n(E,F)$, $x_1, \ldots, x_n \in \mathcal{N}$, and F' is obtained from $F[W_a(E)/X]$ by replacing some occurrences of $a_{-1}[W_a(E)]$ by $b_{-1}[W_b(E)]$.

Details are omitted. □

Theorem 5.2 $\mathsf{rec}X.E \approx E[\mathsf{rec}X.E/X]$ *provided that* $E \in \mathcal{E}[X]$.

Proof: We show that the following relation

$$\left\{ (F[\Omega_E], F[\mathsf{rec}X.E]) \;\middle|\; \begin{array}{l} gn(E) \cap ln(F) = \emptyset, \\ E \in \mathcal{E}[X],\ F \in \mathcal{E}[X] \end{array} \right\}$$

is a local bisimulation up to $\approx$. Let $\mathcal{R}$ be a process and $\vec{x}$ a sequence of names. Suppose

$$(\vec{x})((a)(F[W_a(E)/X]|a_{-1}[W_a(E)])|R) \xrightarrow{\tau} P.$$

The most interesting case is when a higher order communication between $F[W_a(E)/X]$ and $a_{-1}[W_a(E)]$ happen. Without loss of generality, let F be $X|F_1$. Then

$$\begin{array}{rcl} P & \equiv & (\vec{x})((a)(((E[W_a(E)/X]|a_{-1}[W_a(E)]) \\ & & |F_1[W_a(E)/X])|\mathbf{0})|R) \\ & \approx & (\vec{x})((a)(b)(((E[W_b(E)/X]|b_{-1}[W_b(E)]) \\ & & |F_1[W_a(E)/X])|a_{-1}[W_a(E)])|R) \\ & \approx & (\vec{x})(a)((((b)(E[W_b(E)/X]|b_{-1}[W_b(E)]) \\ & & |R)|F_1[W_a(E)/X])|a_{-1}[W_a(E)]) \\ & \equiv & (\vec{x})H[\Omega_E], \end{array}$$

where $H \equiv ((b)(E[W_b(E)/X]|b_{-1}[W_b(E)])|R)|F_1 \in \mathcal{E}[X]$. Correspondingly, we have

$$\begin{array}{rcl} & & (\vec{x})(F[\mathsf{rec}X.E/X]|R) \\ & \equiv & (\vec{x})((\mathsf{rec}X.E|F_1[\mathsf{rec}X.E/X])|R) \\ & \xrightarrow{\tau} & (\vec{x})(((a)(E[W_a(E)/X]|a_{-1}[W_a(E)]) \\ & & |F_1[\mathsf{rec}X.E/X])|R) \\ & \approx & (\vec{x})(H[\mathsf{rec}X.E/X]) \end{array}$$

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

$$\begin{array}{rcl} \mathsf{rec}X.E & \approx & (a)(E[W_a(E)/X]|a_{-1}[W_a(E)]) \\ & \approx & E[\mathsf{rec}X.E/X], \end{array}$$

which completes the proof. □

6. Conclusion

There are several questions one usually asks about a mathematical model for concurrent computations. One is concerned with the primitivity of the model. Does it capture the essence of concurrent computations in a simple yet elegant way? The χ-calculus is grammatically simple and conceptually coherent. Another question is to do with concurrency degree. To what extent does the model permit parallelism? There is little doubt that the χ-calculus allows more freedom than the π-calculus in this respect. A manifestation of this fact is that the operational semantics of the call-by-name λ-calculus can be imitated in χ-calculus in a straightforward manner.

The paper has just begun the study of χ-like process algebras. Quite a lot of interesting questions await to be answered. It is hoped that further research will tell us more about what χ really differs from π and will eventually lead us to a canonical process calculus that encapsulates concurrent computations, object-oriented programming ([9]), full λ-calculus ([1, 3]) and proofs of linear logic ([2]).

References

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

[2] G. Bellin and P. Scott, 1992. On the π-Calculus and Linear Logic. Technical Report, LFCS, Department of Computer Science, University of Edinburgh.

[3] R. Milner, 1992. Functions as Processes. *Journal of Mathematical Structures in Computer Science*, **2**, 119–141, Cambridge University Press.

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

[5] R. Milner and D. Sangiorgi, 1992. Barbed Bisimulation, *19th ICALP*, *LNCS 623*, W. Knich ed. 685–695, Springer Verlag.

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

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

[8] B. Thomsen, 1995. A Theory of Higher Order Communicating Systems. *Information and Computation*, **116**, 38–57, Academic Press.

[9] D. Walker, 1995. Objects in the π-Calculus. *Information and Computation*, **116**, 253–271, Academic Press.