Axiomatization without Prefix Combinator

Yuxi Fu*

Department of Computer Science
Shanghai Jiaotong University, Shanghai 200030, China
E-mail: fu-yx@cs.sjtu.edu.cn

Abstract. The chi calculus proposed several years ago enjoys some properties unknown from the experience with pi calculus, one of which is the ability to model concurrent computation without the use of prefix combinator. The atomic chi calculus studied in this paper is obtained from polyadic chi calculus by leaving out the prefix operator. This omission is impossible in the pi framework because it would render the input actions of pi useless. This paper focuses on complete systems of strong equivalence relations on finite atomic chi processes. The two equivalence relations investigated in this paper are strong bisimilarity and strong asynchronous bisimilarity. These bisimilarities are required to be closed under substitution on each bisimulation step. By exploring some properties enjoyed by the atomic chi calculus, it is shown that they coincide respectively with their ground counterparts. In the definitions of strong ground bisimilarity and strong asynchronous ground bisimilarity closure under substitution is not explicitly required. Based upon this fact complete systems are given for both relations. The axiomatic systems are novel in that they use none of the prefix, choice and match combinators.

Key Words: Process Algebra, Bisimulation, Axiomatization

1 Introduction

Operationally computations are achieved by substitutions. This is clear from λ-calculus, the canonical model for functional computation, and π-calculus, a model for concurrent computation. An abstraction term in λ-calculus is of the form $\lambda x.t$. Semantically it is a function that yields $t[s/x]$, the result of substituting s for x throughout t, when given s. This operational behaviour is formalized in the following β-reduction rule:

$$(\lambda x.t)s \rightarrow t[s/x]$$

The variable x in $\lambda x.t$ is bound. A bound name in a term can be replaced by a fresh name without affecting the meaning of the term. The input prefix operation of π-calculus ([19]) takes similar form as $a(x).P$. This is the process that can

* *Proceedings of the International Symposium on Domains and Processes '99*, K. Keimel et al eds., Kluwer Academic Publishers, 245-273, 2001.

receive a channel name y through x and then proceeds as $P[y/x]$. The channel name x in $a(x).P$ is bound. The semantics of a π-process in output prefix form $\overline{a}x.Q$ is different in that it is ready to emit a channel name x through a and then evolve as Q. In $a(x).P$ and $\overline{a}x.Q$ we say that P and Q are the continuations of the processes. As opposed to the x in $a(x).P$, the channel name x in $\overline{a}x.Q$ is free. A communication between input prefix process and output prefix process can happen when they share common channel. Communications are formalized by labeled transitions as in the following example:

$$a(x).P|\overline{a}y.Q \xrightarrow{\tau} P[y/x]|Q$$

The input and output processes differ in another aspect. In $\overline{a}x.Q$ the prefix operator act as a sequential combinator. It has no other function. On the other hand the prefix operator of $a(x).P$ plays a double role. It is both a sequential operator and a binding combinator. Another binding combinator in π-calculus is the localization operator. In $(x)P$ the component (x) localizes x to process P, meaning that the channel x can only be used within P. The localization operator adds a great deal of power to π-calculus.

The π-calculus is also called monadic π-calculus. The phrase 'monadic' indicates the fact that in a communication a process can emit or input only one channel at a time. In practice there is a need for processes to be able to send and receive a number of channels in one communication. The polyadic π-calculus extends the monadic π-calculus with this capacity ([18]). The input and output processes of the latter become respectively $a(\tilde{x}).P$ and $\overline{a}\tilde{y}.Q$ in the former, where $\tilde{x}$ and $\tilde{y}$ stand for finite sequences of channel names. A communication in the polyadic version looks as follows:

$$a(\tilde{x}).P|\overline{a}\tilde{y}.Q \xrightarrow{\tau} P[\tilde{y}/\tilde{x}]|Q$$

in which $[\tilde{y}/\tilde{x}]$ is a simultaneous substitution of $\tilde{y}$ for $\tilde{x}$.

One of the applications of the polyadic π-calculus is in the computational interpretation of classical proof theory. Girard was the first to point out possible connections between classical linear logic and parallelism ([13]). Abramsky made an important step in relating cut eliminations in linear logic to communications in π-calculus ([1]). It was discovered in his work that the prefix operator plays no role in modeling the dynamics of the proof theory. This raises the question of if prefix operators are really necessary in a model for concurrent computation. The asynchronous π-calculus can be seen as a partial answer to the question ([15, 3, 2]), in which output actions do not have continuations. It has been shown that the language has enough expressive power to do what π-calculus can do. The algebraic theory of the asynchronous π-calculus is slightly different from that of π-calculus. For one thing the standard definition of barbed bisimilarity need be modified to take into account the asynchronous nature of the language. In some aspects the algebraic theory is also a little harder. For example axiomatization of weak equivalence on asynchronous π-processes is unknown. In the asynchronous π-calculus the input prefix operator remains the same as in the π-calculus. There is absolutely no way to remove the continuations away from the processes in input

prefix form for that would have rendered the resultant language totally useless. The failure is due to the double role of the prefix operator mentioned above. The binding power of the operator would be reduced to none if there is nothing to bind over.

One solution to the problem is to disassociate the binding ability of the prefix operator from the sequentialization ability of the combinator. The χ-calculus proposed by present author ([4–12]) is a process calculus that achieves just that. The processes in input and the output forms are unified as $\alpha[x].P$, where α could be either a or $\overline{a}$ and x is global. The binding power of the language is provided solely by the localization operator. So for instance the channel name x in $(x)\alpha[x].P$ is local. A communication in χ-calculus amounts to instantiating a local channel name by a global channel name or identifying two local channel names. The following two examples of reduction should provide some intuition:

$$a[y].Q|(x)\overline{a}[x].P \xrightarrow{\tau} Q|P[y/x]$$
$$(y)a[y].Q|(x)\overline{a}[x].P \xrightarrow{\tau} (z)(Q[z/y]|P[z/x])$$

Now in χ-calculus one can give up on continuations altogether. The language one obtains has the following abstract grammar:

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

in which $!P$ is a replication process that provides potentially infinite copies of P. Unfortunately this language is too weak, the reason being that it lacks the ability to control the order of computations. A communication of the language transports only one token, which can be used either as a value or as control information but not both. This immediately suggests a solution: To abandon the prefix operator, one should work with a polyadic calculus. The polyadic χ-calculus has been studied by Parrow and Victor ([21–24]). They call it Fusion Calculus.

The atomic χ-calculus studied in this paper is obtained from the polyadic χ-calculus by leaving out the prefix operator. The abstract syntax of the language is as follows:

$$P := \mathbf{0} \mid \alpha[\tilde{x}] \mid P|P \mid (x)P \mid !P$$

The atomic χ-calculus has a great deal of control power. The following example suffices to demonstrate this point:

$$\begin{aligned}(x)(y)(z)((a[x,y]|\overline{a}[b,b])|(x[z]|\overline{y}[c])) &\xrightarrow{\tau} (z)((\mathbf{0}|\mathbf{0})|(b[z]|\overline{b}[c])) \\ &\xrightarrow{\tau} (\mathbf{0}|\mathbf{0})|(\mathbf{0}|\mathbf{0})\end{aligned}$$

It is apparent that the order of the two communications can not be swapped. Using ideas embodied in the above example, one can show that the lazy λ-calculus can be interpreted in the atomic χ-calculus. This calculus was proposed by Laneve and Victor ([16]). They call it Solos. However we will refer to the calculus as the atomic χ-calculus in the rest of the paper.

The theory of process calculus is mainly about algebraic properties of processes. Algebraic studies are based on equivalence relations, of which the most important ones are observational equivalences. The most well known observational equalities are the bisimulation equalities. Two processes are bisimilar if they can simulate each other's actions and evolve to two bisimilar processes. Bisimilarity equalities differ from one another in the extent actions can be observed. Another aspect of algebraic theory of process calculus is about axiomatic systems for congruence relations on processes. Each system consists of a set of conditional equations. A system should be both sound and complete in the sense that it derives equivalent and only equivalent processes with respect to the intended equality.

The focus of this paper is on algebraic theory of the atomic χ-calculus. Two bisimulation equivalence relations are investigated. They are strong bisimilarity and strong asynchronous bisimilarity. Like the situation in the asynchronous π-calculus the asynchronous bisimilarity takes care of some very special operational properties of the atomic χ-calculus. Let's see a typical example. The two processes $(x)(a[x]|\overline{a}[x])$ and $(a)(x)(a[x]|\overline{a}[x])$ are not bisimilar in the synchronous sense but are equivalent in the asynchronous view. The role of $(x)(a[x]|\overline{a}[x])$ is to absorb an atomic process of the form $a[b]$ or $\overline{a}[b]$ as in the following communication

$$\overline{a}[x]|(x)(a[x]|\overline{a}[x]) \xrightarrow{\tau} \mathbf{0}|(\mathbf{0}|\overline{a}[b])$$

The process $(a)(x)(a[x]|\overline{a}[x])|\overline{a}[b]$ can emulate the above reduction by performing an internal communication. This example should convince the reader that $(x)(a[x]|\overline{a}[x])$ and $(a)(x)(a[x]|\overline{a}[x])$ are observationally equivalent.

Both bisimilarities are closed under substitution of channel names. Closure under substitution of channel names is a reasonable requirement in order to guarantee congruence property because channel names in a mobile process do get changed as results of process interactions. If one does not insist on the closure property, one gets what we call ground bisimilarity. Usually the ground bisimilarity is a much weaker relation. We will prove however that in atomic χ-calculus strong bisimilarity and strong asynchronous bisimilarity coincide respectively with strong ground bisimilarity and strong asynchronous ground bisimilarity. This is anticipated by the work of Amadio, Castellani and Sangiorgi on asynchronous π-calculus. This fact is then explored to study axiomatization problem for the two bisimilarities. Usually complete systems for congruence relations on mobile processes make heavy use of choice and match combinators, not to mention prefix operator. We give in this paper complete systems for the two strong bisimilarities. These systems are novel in that none of the prefix combinator, the choice combinator and the match combinator is used.

In [16] Laneve and Victor have shown how to encode some well-known process combinators in the atomic χ-calculus. These encodings are faithful with respect to some particular algebraic equivalences. Their results demonstrate to some extent the expressive power of the calculus. While we should emphasize the importance of the work of Laneve and Victor, we would also like to point out that the study in this paper is neither about the language per se nor about its

expressiveness. It is about axiomatization of algebraic congruences with neither the choice combinator nor the prefix operator. Since the atomic χ-calculus, or Solos, is the only process calculus without the prefix combinator at the time of writing, it is the only framework one can work with at the moment.

The structure of the paper is as follows: Section 2 lays down some preliminaries. Section 3 introduces polyadic χ-calculus and some technical lemmas. The reason to go through the polyadic χ-calculus first is that most account of polyadic χ-calculus and the atomic χ-calculus are the same. The time spent on the former is worth the effort. Section 4 defines the semantics of the atomic χ-calculus and proves some crucial lemmas. Section 5 simplifies the definition of open bisimilarity by providing an alternative formulation in terms of ground bisimulation. Section 6 makes use of this result and gives a complete system for strong ground bisimilarity. Section 7 concludes with some final remarks.

2 Preliminaries

A process calculus needs to deal with substitutions. In polyadic calculus more care is called for since one has to use simultaneous substitutions. Also in polyadic calculus a channel must be assigned a sort to prevent confusion. This section introduces some notations and preliminary definitions for substitution and sorting.

2.1 Normal Substitution

All process calculi proposed so far are based upon the notion of channels. Processes communicate through channels. Usually a channel is used only as a token. Its sole identity is that it is distinct from any other channel. For this reason one often talks about channel names or simply names. Let $\mathcal{N}$ be a set of names, ranged over by lower case letters. Let $\overline{\mathcal{N}}$ be the set of conames $\{\overline{x} \mid x \in \mathcal{N}\}$. Names and conames can be understood as the two ends of channels. Two processes can communicate if they are connected to the two ends of a same channel. The union $\mathcal{N} \cup \overline{\mathcal{N}}$ will be ranged over by α.

A sequence of names is often abbreviated by $\tilde{x}$. Accordingly $(x_1)\cdots(x_n)P$ is abbreviated by $(\tilde{x})P$. The length of $\tilde{x}$ is denoted by $|\tilde{x}|$. A name might occur more than once in a name sequence. For example x, y, x is a name sequence. But when used as in $(\tilde{x})P$, we always assume that all names occurring in $\tilde{x}$ are pairwise distinct. Occasionally we think of a sequence $\tilde{x} = x_1, \ldots, x_n$ as the multi-set $\{x_1, \ldots, x_n\}$. This is the case when we apply set theoretical operations on sequences of names. We will write $\{\tilde{x}\}$ for the set that contains precisely the elements appeared in $\tilde{x}$. For instance if $\tilde{x} = yzy$ then $\{\tilde{x}\} = \{y, z\}$. When the elements of $\tilde{x}$ are pairwise distinct, the set $\{\tilde{x}\}$ is also abbreviated to $\tilde{x}$.

A substitution σ is a function from $\mathcal{N}$ to $\mathcal{N}$ such that the set $\{x \mid x \in \mathcal{N},\ \sigma(x) \neq x\}$ is finite. The domain of a substitution σ, denoted by $dom(\sigma)$, is the set $\{x \mid x \in \mathcal{N},\ \sigma(x) \neq x\}$. The range of σ, $rng(\sigma)$, is defined as the image of $dom(\sigma)$. A substitution σ is often written as $[y_1/x_1, \ldots, y_n/x_n]$ when

$dom(\sigma) = \{x_1, \ldots, x_n\}$ and $rng(\sigma) = \{y_1, \ldots, y_n\}$. This is the function defined as follows:

$$\sigma(x) = \begin{cases} y_1, \text{ if } x = x_1 \\ \vdots \\ y_n, \text{ if } x = x_n \\ x, \text{ if } x \notin \{x_1, \ldots, x_n\} \end{cases}$$

The identity function is the vacuous substitution. It will be denoted by []. The composition of two substitutions σ_1 and σ_2, notation $\sigma_1\sigma_2$, is defined in terms of function composition: $\sigma_1\sigma_2$ is function σ_1 followed by σ_2. So $P\sigma_1\sigma_2$ is $(P\sigma_1)\sigma_2$. For a set of names S, write $\sigma^{-1}(S)$ for the set $\{x|\sigma(x) \in S\}$.

Definition 1. *A substitution σ is normal if $dom(\sigma) \cap rng(\sigma) = \emptyset$.*

Suppose σ' is defined as follows:

$$\sigma'(a) \stackrel{\text{def}}{=} \begin{cases} y, \text{ if } a = x \\ x, \text{ if } a = y \\ a, \text{ if otherwise} \end{cases}$$

Then σ' is typically not normal as $dom(\sigma') \cap rng(\sigma') = \{x, y\} \neq \emptyset$.

Suppose $\sigma = [y_1/x_1, \ldots, y_n/x_n]$ is a normal substitution. Define $\sigma{\uparrow}z$ to be the following function:

$$\sigma(x) = \begin{cases} y_i, \text{ if } x = x_i \wedge x \neq z \wedge (1 \leq i \leq n) \\ x, \text{ if otherwise} \end{cases}$$

For instance $[u/x, v/y, w/z]{\uparrow}y = [u/x, w/z]$. It is clear that $\sigma{\uparrow}z$ is a normal substitution. We will write $\sigma{\uparrow}\{z_1, \ldots, z_n\}$ for $(\ldots(\sigma{\uparrow}z_1)\ldots){\uparrow}z_n$.

Composition of two normal substitutions is not necessarily normal. A counter example is as follows: Both $\sigma_1 : [x \to a, y \to b]$ and $\sigma_2 : [a \to y, b \to x]$ are normal. But neither $\sigma_1\sigma_2$ nor $\sigma_2\sigma_1$ is normal. The next lemma gives a sufficient condition under which normal substitutions compose.

Lemma 2. *If σ_1 and σ_2 are normal substitutions and $dom(\sigma_1) \cap rng(\sigma_2) = \emptyset$ then $\sigma_1\sigma_2$ is normal.*

Proof. $dom(\sigma_1\sigma_2)$ is obviously finite.

$$\begin{aligned} & dom(\sigma_1\sigma_2) \cap rng(\sigma_1\sigma_2) \\ \subseteq\ & (dom(\sigma_1) \cup dom(\sigma_2)) \cap ((rng(\sigma_1) \cup rng(\sigma_2)) \setminus (rng(\sigma_1) \cap dom(\sigma_2))) \\ =\ & dom(\sigma_1) \cap rng(\sigma_1) \cup dom(\sigma_1) \cap rng(\sigma_2) \cup dom(\sigma_2) \cap rng(\sigma_1) \cup \\ & dom(\sigma_2) \cap rng(\sigma_2) \setminus ((dom(\sigma_1) \cup dom(\sigma_2)) \cap (rng(\sigma_1) \cap dom(\sigma_2))) \\ =\ & (rng(\sigma_1) \cap dom(\sigma_2)) \setminus (rng(\sigma_1) \cap dom(\sigma_2)) \\ =\ & \emptyset \end{aligned}$$

So $dom(\sigma_1\sigma_2) \cap rng(\sigma_1\sigma_2) = \emptyset$. □

On the other hand, the effect of a general substitution $[y_1/x_1, \ldots, y_n/x_n]$ on a term is the same as that of the composition of a finite number of normal substitutions as $[z_1/x_1] \ldots [z_n/x_n][y_1/z_1] \ldots [y_n/z_n]$ where $z_1, \ldots, z_n$ are fresh names that do not occur in the term the substitution is applied to. Substitutions of the form $[y/x]$ are called atomic substitutions.

Suppose $\tilde{x} = x_1, \ldots, x_n$ and $\tilde{y} = y_1, \ldots, y_n$ are two name sequences of length n. Then $\tilde{x}{=}\tilde{y}$, which denotes $x_1 = y_1, \ldots, x_n = y_n$, induces an equivalence relation on $\mathcal{N}$. Let $\sigma_{\tilde{x}=\tilde{y}}$ denote any chosen substitution that maps all elements of an equivalence class of the equivalence relation to a specific element of that class. Obviously $\sigma_{\tilde{x}=\tilde{y}}$ is a normal substitution.

Suppose $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are name sequences of length n. We say that $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are consistent if $\forall i, j \in \{1, \ldots, n\}.x_i{=}x_j \Leftrightarrow y_i{=}y_j$ and that $x_1, \ldots, x_n$ are replaceable by $y_1, \ldots, y_n$ if $\forall i, j \in \{1, \ldots, n\}.x_i{=}x_j \Rightarrow y_i{=}y_j$. When $\{x_1, \ldots, x_n\} \cap \{y_1, \ldots, y_n\} = \emptyset$ and $x_1, \ldots, x_n$ are replaceable by $y_1, \ldots, y_n$, $[y_1/x_1, \ldots, y_n/x_n]$ is a normal substitution.

2.2 Sorting

To avoid communication confusion, a name must be of some sort. The sort of a name indicates both the number of names it carries when communicating and the sorts of these names. The set $\mathcal{N}$ is partitioned into an infinite number of collections of names, each of which contains an infinite number of names. Each collection is called a subject sort. We write $a : S$ to mean that a is of sort S. A nonempty tuple of sorts $\langle S_1, \ldots, S_n \rangle$ is called an object sort. A sorting is a function $Sort$ from the set of subject sorts to the set of object sorts. For each subject sort S the object sort $Sort(S)$ declares the sorts of names associated to sort S. In the rest of this paper we fix a sorting function $Sort$.

3 Polyadic χ-Calculus

Parrow and Victor formulate the operational semantics in a late style. In this paper we use an early semantics.

The abstract syntax of polyadic χ-processes is given by the following BNF:

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

As usual $\mathbf{0}$ is the nil process that can do nothing. We will omit a trailing $\mathbf{0}$. $P|Q$ is the process of composition form, where P and Q can evolve independently and communicate if they want to. In $(x)P$ the name x is local, meaning that it can not been seen from outside. We will adopt the α-convention saying that a local name in a term can be replaced by a fresh name without changing its syntax. Let $gn(P)$ denote the set of global names, or nonlocal names, in P. The processes $a[\tilde{x}].P$ and $\overline{a}[\tilde{x}].P$ are in prefix form. Roughly they have to perform the prefix actions and then act as P. Both $a[x_1, \ldots, x_n].P$ and $\overline{a}[x_1, \ldots, x_n].P$ must be well-sorted in the sense that $x_1 : S_1, \ldots, x_n : S_n$ whenever $Sort(a) = \langle S_1, \ldots, S_n \rangle$. The set of polyadic χ-processes will be denoted by $\mathcal{P}$.

In the labeled transition semantics two kinds of labels are used: the set of output actions is the set

$$\{(\tilde{y})\alpha[\tilde{x}] \mid \tilde{y} \subseteq \tilde{x} \text{ where } \tilde{y} \text{ are pairwise distinct}\}$$

and the set of updates

$$\{\sigma_{\tilde{x}=\tilde{y}} \mid |\tilde{x}| = |\tilde{y}|\}$$

The latter set contains the empty substitution [] which will be denoted by τ.

We now define formally the operational semantics. In the following formulation, symmetric rules have been omitted.

Sequentialization

$$\frac{}{\alpha[\tilde{x}].P \xrightarrow{\alpha[\tilde{x}]} P} Sqn$$

Composition

$$\frac{P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P' \quad \tilde{y} \cap gn(Q) = \emptyset}{P|Q \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P'|Q} Cmp_0 \qquad \frac{P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} P'}{P|Q \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} P'|Q\sigma_{\tilde{x}=\tilde{y}}} Cmp_1$$

Communication

$$\frac{P \xrightarrow{(\tilde{u})\alpha[\tilde{x}]} P' \quad Q \xrightarrow{(\tilde{v})\overline{\alpha}[\tilde{y}]} Q' \quad \begin{cases} \tilde{u} \cap \tilde{v} = \emptyset \\ \tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v} \\ (\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v} \end{cases}}{P|Q \xrightarrow{\sigma_{\tilde{x}=\tilde{y}} \restriction \tilde{u}\tilde{v}} (\tilde{w})(P'\sigma_{\tilde{x}=\tilde{y}}|Q'\sigma_{\tilde{x}=\tilde{y}})} Cmm$$

Localization

$$\frac{P \xrightarrow{\delta} P' \quad z \notin n(\delta)}{(z)P \xrightarrow{\delta} (z)P'} Loc_0 \qquad \frac{P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} P' \quad z \in dom(\sigma_{\tilde{x}=\tilde{y}})}{(z)P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}} \restriction z} P'} Loc_1$$

$$\frac{P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P' \quad z \in \{\tilde{x}\} \setminus \tilde{y} \quad \alpha \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z)(\tilde{y})\alpha[\tilde{x}]} P'} Loc_2$$

Replication

$$\frac{P|!P \xrightarrow{\delta} P'}{!P \xrightarrow{\delta} P'} Rpl$$

The side condition on Cmm is a bit involved. Here is an example of applying the rule:

$$\frac{P \xrightarrow{(c)(x)a[c,x,x,u]} P' \qquad Q \xrightarrow{(d)(v)\overline{a}[d,y,w,v]} Q'}{P|Q \xrightarrow{[y/w]} (c)(P[c/d, y/x, y/w, u/v]|Q[c/d, y/x, y/w, u/v])}$$

where $\tilde{u} = cx$, $\tilde{x} = cxxu$, $\tilde{v} = dv$, $\tilde{y} = dywv$, $\tilde{w} = c$ and $\sigma_{\tilde{x}=\tilde{y}}$ is chosen to be $[c/d, y/x, y/w, u/v]$.

In what follows, $\sigma_{\tilde{x}=\tilde{y}}$ will often be abbreviated to σ and $\xrightarrow{[]}$ will always be simplified to $\xrightarrow{\tau}$. In other words τ is identified to [] when used as a label in the transitional semantics.

Two processes are strongly bisimilar if they can simulate each other's actions and evolve into processes that are still strongly bisimilar. In this approach all actions, whether they are internal or external, are treated with equal importance. The relation introduced in the following definition would be called strong open bisimilarity by some researchers. We will however simply call it strong bisimilarity.

Definition 3. *Suppose $\mathcal{R}$ is a symmetric binary relation on $\mathcal{A}$. The relation $\mathcal{R}$ is a strong bisimulation if whenever $P\mathcal{R}Q$ then the following properties hold:*
(i) If $P\sigma \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} P'$ (including the case when $\sigma_{\tilde{x}=\tilde{y}} = \tau$) for a substitution σ then some Q' exists such that $Q\sigma \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} Q'$ and $P'\mathcal{R}Q'$.
(ii) If $P\sigma \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P'$ for a substitution σ then some Q' and $\tilde{y'}$ exist such that $Q\sigma \xrightarrow{(\tilde{y'})\alpha[\tilde{x}]} Q'$, $\{\tilde{y}\} = \{\tilde{y'}\}$ and $P'\mathcal{R}Q'$.
The strong bisimilarity $\sim$ is the largest strong bisimulation.

The closure under substitution ensures that $\sim$ is a congruence relation. The proof of this fact is routine. The two processes $(x)(y)a[x,y].P$ and $(y)(x)a[x,y].P$ are bisimilar. The transition $(x)(y)a[x,y].P \xrightarrow{(x)(y)a[x,y]} P$ is matched up by $(y)(x)a[x,y].P \xrightarrow{(y)(x)a[x,y]} P$. In the two actions the order of the appearances of (x) and (y) are not the same. This explains the side condition in (ii) of the above definition.

Theorem 4. *The strong bisimilarity is congruent.*

In the rest of the section, we introduce some lemmas necessary to the development of the remaining paper.

Lemma 5. *Suppose σ is a normal substitution. The following properties hold:*
(i) If $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P'$ and $\tilde{y} \cap (dom(\sigma) \cup rng(\sigma)) = \emptyset$ then $P\sigma \xrightarrow{(\tilde{y})\alpha[\tilde{x}\sigma]} P'\sigma$.
(ii) If $P \xrightarrow{\sigma'_{\tilde{x}=\tilde{y}}} P'$ then $P\sigma \xrightarrow{\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}} P'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}$ for some chosen $\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}$.

Proof. Both (i) and (ii) are proved by induction on the height of derivation.
(i) Suppose the last rule applied is Loc_2. By induction hypothesis

$$P\sigma \xrightarrow{(\tilde{y})\alpha\sigma[\tilde{x}\sigma]} P'\sigma$$

Using the condition $\{z\tilde{y}\} \cap (dom(\sigma) \cup rng(\sigma)) = \emptyset$ one obtains that $z \in \{\tilde{x}\sigma\} \setminus \{\tilde{y}\}$ and $\alpha\sigma \notin \{z, \overline{z}\}$. Using Loc_2, one has that

$$(z)P\sigma \xrightarrow{(z)(\tilde{y})\alpha\sigma[\tilde{x}\sigma]} P'\sigma$$

(ii) Suppose the last rule applied is Cmm. By α-convention and (i)

$$P\sigma \xrightarrow{(\tilde{v})\alpha\sigma[\tilde{x}\sigma]} P'\sigma$$
$$Q\sigma \xrightarrow{(\tilde{u})\overline{\alpha}\sigma[\tilde{y}\sigma]} Q'\sigma$$

By Cmm, one has

$$(P|Q)\sigma \equiv P\sigma|Q\sigma \xrightarrow{\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}} (\tilde{w})(P'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}|Q'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}) \equiv (\tilde{w})(P'|Q')\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}$$

Suppose the last rule applied is Loc_1. Then $P\sigma \xrightarrow{\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}} P'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}$ by induction hypothesis. Applying Loc_1 one obtains that $(z)P\sigma \xrightarrow{\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}\uparrow z} P'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}$. We are done by observing that $\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma}\uparrow z = \sigma'_{\tilde{x'}\sigma=\tilde{y'}\sigma}$ for some $\tilde{x'}$ and $\tilde{y'}$ and some chosen $\sigma'_{\tilde{x'}\sigma=\tilde{y'}\sigma}$. As $z \notin gn(P'\sigma)$, $P'\sigma\sigma'_{\tilde{x}\sigma=\tilde{y}\sigma} = P'\sigma\sigma'_{\tilde{x'}\sigma=\tilde{y'}\sigma}$. □

The reverse of the above lemma also holds.

Lemma 6. *Suppose σ is a normal substitution. The following properties hold:*
(i) If $P\sigma \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P'$ and $\tilde{y} \cap (dom(\sigma) \cup rng(\sigma)) = \emptyset$ then there exist some α', $\tilde{x'}$ and P_1 such that $P \xrightarrow{(\tilde{y})\alpha'[\tilde{x'}]} P_1$ and $\alpha = \alpha'\sigma$, $\tilde{x} = \tilde{x'}\sigma$ and $P' \equiv P_1\sigma$.
(ii) If $P\sigma \xrightarrow{\sigma_1} P'$ is caused by a communication through a local name then $P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}} P_1$ for some $\tilde{x}$, $\tilde{y}$, $\sigma_{\tilde{x}=\tilde{y}}$ and P_1 such that $\sigma_1 = \sigma_{\tilde{x}\sigma=\tilde{y}\sigma}$ and $P' \equiv P_1\sigma\sigma_{\tilde{x}\sigma=\tilde{y}\sigma}$ for some chosen $\sigma_{\tilde{x}\sigma=\tilde{y}\sigma}$.

Proof. This is a simple proof by induction on the height of derivation. □

Lemma 7. *Suppose $a \notin gn(P)$. If $(\tilde{x})(P|a[\tilde{x}]) \xrightarrow{\tau} P'|a[\tilde{y}]$ then $P \xrightarrow{[\tilde{y}/\tilde{x}]} P'$.*

Proof. It is obvious that $(\tilde{x})(P|a[\tilde{x}]) \xrightarrow{\tau} P'|a[\tilde{y}]$ is obtained by applying Loc_1 several times. So $P|a[\tilde{x}] \xrightarrow{[\tilde{y}/\tilde{x}]} P'|a[\tilde{y}]$, which must follows from $P \xrightarrow{[\tilde{y}/\tilde{x}]} P'$. □

4 A Process Calculus without Precedence

Almost all process calculi has prefix operator. But the role of the combinator has not been clarified. The main objective of having prefix operator is to impose causal dependency which is important to all computation models. Apart from using prefix operator, causality can be achieved by other means such as mobility. In the rest of the paper we investigate a process calculus without prefix operator. This model is obtained from the polyadic χ-calculus by simply removing the prefix operator. Its abstract syntax is as follows:

$$P := \alpha[\tilde{x}] \mid P|P' \mid (x)P \mid !P$$

The operational semantics is that of the polyadic χ-calculus minus those concerned with the prefix combinator. In addition one needs th following rule:

$$\frac{}{\alpha[\tilde{x}] \xrightarrow{\alpha[\tilde{x}]} \mathbf{0}} Act$$

where $\mathbf{0}$ is defined as $(a)a[a]$ for some a of suitable sort.

We will call this language atomic χ-calculus. The set of atomic χ-processes is denoted by $\mathcal{A}$.

The atomic χ-calculus enjoys some properties not satisfied by the (polyadic) χ-calculus. For instance the two actions of

$$(w)(a[x,y,z]|(b)(b[x,z]|\overline{b}[y,w])) \xrightarrow{a[x,y,z]} \xrightarrow{[y/x]} \mathbf{0}|(b)(\mathbf{0}|\mathbf{0})$$

can be swapped as

$$(w)(a[x,y,z]|(b)(b[x,z]|\overline{b}[y,w])) \xrightarrow{[y/x]} \xrightarrow{a[y,y,z]} \mathbf{0}|(b)(\mathbf{0}|\mathbf{0})$$

After reordering the two actions are not necessarily the same as the original ones. Some modifications are necessary. But the actions are incurred by the same components, so to speak, as the original ones. For another example one notices that

$$(x)(y)(a[x,y,z]|(b)(b[x,z]|\overline{b}[y,y])) \xrightarrow{(x)(y)a[x,y,z]} \xrightarrow{[y/x,y/z]} \mathbf{0}|(b)(\mathbf{0}|\mathbf{0})$$

can be reordered as

$$(x)(y)(a[x,y,z]|(b)(b[x,z]|\overline{b}[y,y])) \xrightarrow{\tau} \xrightarrow{a[z,z,z]} \mathbf{0}|(b)(\mathbf{0}|\mathbf{0})$$

The drop of prefix operator allows one to rearrange the order of actions in many occasions. The next lemma deals with two situations.

Lemma 8. *The following properties hold:*
(i) If $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ *and* $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$ *then* $P \xrightarrow{\sigma\uparrow\tilde{y}} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma]} P'$*, where* $\tilde{y'} = \tilde{y} \setminus dom(\sigma)$.
(ii) If $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{(\tilde{y'})\alpha'[\tilde{x'}]} P'$ *then* $P \xrightarrow{(\tilde{z'})\alpha'[\tilde{x'}]} \xrightarrow{(\tilde{z})\alpha[\tilde{x}]} P'$ *such that* $\tilde{z'} = \tilde{y'} \cup (\{\tilde{x'}\} \cap \tilde{y})$ *and* $\tilde{z} = \tilde{y} \setminus \{\tilde{x'}\}$.

Proof. The proof is carried out by induction on the height of derivation. There are several cases:

- P is $\alpha[\tilde{z}]$. Impossible.
- P is of form $P_1|P_2$. If the two actions are caused solely by either P_1 or P_2 individually then apply the induction hypothesis. Otherwise $\tilde{y} \cap (dom(\sigma) \cup rng(\sigma)) = \emptyset$. Suppose $P_1 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P_1'$, $P_2 \xrightarrow{\sigma} P_2'$ and $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is

  $$P \equiv P_1|P_2 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P_1'|P_2 \xrightarrow{\sigma} P_1'\sigma|P_2' \equiv P'$$

  Then

  $$P \equiv P_1|P_2 \xrightarrow{\sigma} P_1\sigma|P_2' \xrightarrow{(\tilde{y})\alpha[\tilde{x}\sigma]} P_1'\sigma|P_2' \equiv P'$$

  by Lemma 5.

– P is of the form $(v)P_1$. There are several subcases:

1. $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is caused by $P_1 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P_1'$ such that $v \notin \{\tilde{x}\}$ and $v \notin dom(\sigma) \cup rng(\sigma)$. By induction hypothesis, $P_1 \xrightarrow{\sigma\uparrow\tilde{y}} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma]} P_1'$, where $\tilde{y'} = \tilde{y} \setminus dom(\sigma)$. So $P \xrightarrow{\sigma\uparrow\tilde{y}} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma]} P'$.
2. $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is caused by $P_1 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma'} P_1'$ such that $v \notin \{\tilde{x}\}$, $v \in dom(\sigma')$ and $\sigma = \sigma'\uparrow v$. By assumption $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$. So $(\sigma')^{-1}(\tilde{y} \cap rng(\sigma')) \subseteq \tilde{y}v$. There are two further subcases:
   (a) $(\sigma')(v) \notin \tilde{y}$. Then $(\sigma')^{-1}(\tilde{y} \cap rng(\sigma')) \subseteq \tilde{y}$. By induction hypothesis, $P_1 \xrightarrow{\sigma'\uparrow\tilde{y}} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma']} P_1'$ such that $\tilde{y'} = \tilde{y} \setminus dom(\sigma')$. Hence $P \xrightarrow{\sigma'\uparrow\tilde{y}\uparrow v} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma']} P'$. Now $\tilde{y'} = \tilde{y} \setminus dom(\sigma') = \tilde{y} \setminus dom(\sigma)$, $\sigma'\uparrow\tilde{y}\uparrow v = \sigma'\uparrow v\uparrow\tilde{y} = \sigma\uparrow\tilde{y}$ and $\tilde{x}\sigma' = \tilde{x}\sigma$ as $v \notin \{\tilde{x}\}$. So $P \xrightarrow{\sigma\uparrow\tilde{y}} \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma]} P'$, where $\tilde{y'} = \tilde{y} \setminus dom(\sigma)$.
   (b) $(\sigma')(v) \in \tilde{y}$. Suppose $(\sigma')(v)$ is z. Let σ_1 be defined as follows:

$$\sigma_1(x) \stackrel{\text{def}}{=} \begin{cases} \sigma'(x) & x \in dom(\sigma') \setminus (\sigma')^{-1}(z) \\ v & x \in (\sigma')^{-1}(z) \setminus \{v\} \\ v & x = z \\ x & \text{otherwise} \end{cases}$$

   By Lemma 5, $P_1 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma_1} P_1'[v/z]$. Clearly $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$ implies that $\sigma_1^{-1}(\tilde{y} \cap rng(\sigma_1)) \subseteq \tilde{y}$. It follows by induction that $P_1 \xrightarrow{\sigma_1\uparrow\tilde{y}} P_1'' \xrightarrow{(\tilde{y'})\alpha[\tilde{x}\sigma_1]} P_1'[v/z]$ for some P_1'' such that $\tilde{y'} = \tilde{y} \setminus dom(\sigma_1)$. By Loc_0,

$$(v)P_1 \xrightarrow{\sigma_1\uparrow\tilde{y}} (v)P_1'' \xrightarrow{(v)(\tilde{y'})\alpha[\tilde{x}\sigma_1]} P_1'[v/z]$$

   By α-conversion and Lemma 5, one obtains

$$(v)P_1 \xrightarrow{\sigma_1\uparrow\tilde{y}} (z)P_1''[z/v] \xrightarrow{(z)(\tilde{y'})\alpha[\tilde{x}\sigma']} P_1'$$

   Now $\sigma\uparrow\tilde{y} = (\sigma'\uparrow v)\uparrow\tilde{y} = \sigma_1\uparrow\tilde{y}$ and $\tilde{x}\sigma' = \tilde{x}\sigma$. Therefore

$$(v)P_1 \xrightarrow{\sigma\uparrow\tilde{y}} \xrightarrow{(z)(\tilde{y'})\alpha[\tilde{x}\sigma]} P_1'$$

   It also follows from $\tilde{y'} = \tilde{y} \setminus dom(\sigma_1)$ that $z\tilde{y'} = \tilde{y} \setminus dom(\sigma)$.
3. $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is caused by $P_1 \xrightarrow{(\tilde{y'})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ such that $\tilde{y} = \tilde{y'}v$ and $v \notin dom(\sigma) \cup rng(\sigma)$. The assumption $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$ and the condition $v \notin dom(\sigma) \cup rng(\sigma)$ imply that $\sigma^{-1}(\tilde{y'} \cap rng(\sigma)) \subseteq \tilde{y'}$. By induction hypothesis $P_1 \xrightarrow{\sigma\uparrow\tilde{y'}} \xrightarrow{(\tilde{y''})\alpha[\tilde{x}\sigma]} P'$ such that $\tilde{y''} = \tilde{y'} \setminus dom(\sigma)$. So $(v)P_1 \xrightarrow{\sigma\uparrow\tilde{y'}} \xrightarrow{(v)(\tilde{y''})\alpha[\tilde{x}\sigma]} P'$. Clearly $v\tilde{y''} = v\tilde{y'} \setminus dom(\sigma) = \tilde{y} \setminus dom(\sigma)$.
4. $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is caused by $P_1 \xrightarrow{(\tilde{y'})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ such that $\tilde{y} = \tilde{y'}v$ and $v \in dom(\sigma)$. By assumption $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$. So $\sigma^{-1}(\tilde{y'} \cap rng(\sigma)) \subseteq \tilde{y}$. The rest of the proof is similar to the subcase 2.

5. $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ is caused by $P_1 \xrightarrow{(\tilde{y'})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ such that $\tilde{y} = \tilde{y'}v$ and $v \in rng(\sigma)$. The assumption $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$ implies $\sigma^{-1}(\tilde{y'} \cap rng(\sigma)) \subseteq \tilde{y'}$. By induction hypothesis $P_1 \xrightarrow{\sigma\uparrow\tilde{y'}(\tilde{y''})\alpha[\tilde{x}\sigma]} P'$ such that $\tilde{y''} = y' \setminus dom(\sigma)$. It follows from the assumption $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$ that $v \notin dom(\sigma\uparrow\tilde{y'}) \cup rng(\sigma\uparrow\tilde{y'})$. So $(v)P_1 \xrightarrow{\sigma\uparrow\tilde{y'}(v)(\tilde{y''})\alpha[\tilde{x}\sigma]} P'$. That is $(v)P_1 \xrightarrow{\sigma\uparrow\tilde{y'}v(v)(\tilde{y''})\alpha[\tilde{x}\sigma]} P'$. It is clear that $v\tilde{y''} = \tilde{y'}v \setminus dom(\sigma)$.

– P is of the form $!P_1$. Then $P_1|!P_1 \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\sigma} P'$ and $\sigma^{-1}(\tilde{y} \cap rng(\sigma)) \subseteq \tilde{y}$. By induction hypothesis $P_1|!P_1 \xrightarrow{\sigma\uparrow\tilde{y}(\tilde{y'})\alpha[\tilde{x}\sigma]} P'$ such that $\tilde{y'} = \tilde{y} \setminus dom(\sigma)$. Therefore $P \xrightarrow{\sigma\uparrow\tilde{y}(\tilde{y'})\alpha[\tilde{x}\sigma]} P'$.

This completes the proof of (i).

(ii) The proof is similar. □

Corollary 9. *If* $P \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} \xrightarrow{\tau} P'$ *then* $P \xrightarrow{\tau} \xrightarrow{(\tilde{y})\alpha[\tilde{x}]} P'$.

Proof. This is a special case of Lemma 8. □

In polyadic χ-calculus $P \xrightarrow{(x)a[x]} \xrightarrow{\overline{a}[y]} P'$ does not imply $P \xrightarrow{\tau} P'[y/x]$. Take for example P to be $(x)a[x].\overline{a}[y]$. In the atomic χ-calculus however one can conclude $P \xrightarrow{\tau} P'[y/x]$ from $P \xrightarrow{(x)a[x]} \xrightarrow{\overline{a}[y]} P'$. This is one of the virtues of working without prefix operator. On the other hand if the communication $P \xrightarrow{\tau} P'$ takes place through a global name a then it can be decomposed as it were into two output actions with both subject names being a. The next lemma generalizes this observation to updates.

Lemma 10. *The following properties hold:*
(i) If $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$ *then there exist some* $\tilde{w}$, $\sigma_{\tilde{x}=\tilde{y}}$ *and* P' *such that* $P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}} P'$, $(\tilde{w})(P_1'\sigma_{\tilde{x}=\tilde{y}}) \sim P'$, $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ *and* $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$.
(ii) If $P \xrightarrow{\sigma} P'$ *is caused by a communication through a global name* a *then there exist some* $\tilde{x}$, $\tilde{y}$, $\tilde{u}$, $\tilde{v}$, $\tilde{w}$, $\sigma_{\tilde{x}=\tilde{y}}$, P_1 *and* P_1' *such that* $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$, $\sigma = \sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}$, $(\tilde{w})(P_1'\sigma_{\tilde{x}=\tilde{y}}) \sim P'$, $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ *and* $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$.

Proof. Suppose $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$. We prove by induction on the height of derivation.

– P is of the form $\alpha[\tilde{z}]$. This case is impossible.

– P is of the form $A_1|A_2$. If $A_1 \xrightarrow{(\tilde{u})a[\tilde{x}](\tilde{v})\overline{a}[\tilde{y}]} A_1'$ then $A_1 \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}} A'$ for some chosen $\sigma_{\tilde{x}=\tilde{y}}$ and A' and $(\tilde{w})(A_1'\sigma_{\tilde{x}=\tilde{y}}) \sim A'$ for some $\tilde{w}$ such that $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$. Then $A_1|A_2 \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}} A'|A_2\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}$. Therefore

$$A'|A_2\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v} \sim (\tilde{w})(A_1'\sigma_{\tilde{x}=\tilde{y}})|A_2\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v} \sim (\tilde{w})(A_1'\sigma_{\tilde{x}=\tilde{y}}|A_2\sigma_{\tilde{x}=\tilde{y}})$$

because $\tilde{u}\tilde{v} \cap gn(A_2) = \emptyset$.

– P is of the form $A_1|A_2$ and the two consecutive actions $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$ are performed by A_1 and A_2 respectively. We are done by applying Cmm.
– P is of the form $(z)A$ and $z \notin \tilde{x}\tilde{y}$. This case is simple.
– P is of the form $(z)A$ and $z \in \tilde{x}$. Let $\tilde{u'}$ be $\tilde{u} \setminus z$. Then $A \xrightarrow{(\tilde{u'})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$. By induction hypothesis, $A \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u'}\tilde{v}} P'$ and $(\tilde{w})(P_1'\sigma_{\tilde{x}=\tilde{y}}) \sim P'$ for some chosen $\sigma_{\tilde{x}=\tilde{y}}$ and for some $\tilde{w}$ such that $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$. There are two subcases:
  • $z \in dom(\sigma_{\tilde{x}=\tilde{y}})$. Then $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$ and $P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}} P'$. There is nothing more to be proved.
  • $z \in rng(\sigma_{\tilde{x}=\tilde{y}})$. Then $P \xrightarrow{(\tilde{u})a[\tilde{x}]} P_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} P_1'$ and $P \xrightarrow{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}} (z)P'$. Now clearly $(\tilde{w}z)(P_1'\sigma_{\tilde{x}=\tilde{y}}) \sim (z)P'$. So this case is fine as well.
– P is of the form $(z)A$ and $z \in \tilde{y}$. The situation is similar to the above case.
– P is of the form $!P_1$. Use induction hypothesis.

Suppose $P \xrightarrow{\sigma} P'$ is caused by a communication through a global name a. The the following proof goes by structural induction.

– P is of the form $\alpha[\tilde{z}]$. This case is impossible.
– P is of the form $A_1|A_2$. If the update action is caused solely by either A_1 or A_2, then apply induction hypothesis. If the update action is caused by $A_1 \xrightarrow{(\tilde{u})a[\tilde{x}]} A_1'$ and $A_2 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} A_2'$, then $\tilde{u} \cap \tilde{v} = \emptyset$ by α-convention. We are done by applying Cmm-rule.
– P is of the form $(z)A_1$. There are two subcases:
  • If $P \xrightarrow{\sigma} P'$ is obtained from $A_1 \xrightarrow{\sigma} A_1'$ such that $P \equiv (z)A_1$, $P' \equiv (z)A_1'$ and $z \notin dom(\sigma) \cup rng(\sigma)$. By induction hypothesis, there exist some a, $\tilde{x}$, $\tilde{y}$, $\tilde{u}$, $\tilde{v}$, $\tilde{w}$, $\sigma_{\tilde{x}=\tilde{y}}$, B_1 and B_1' such that $A_1 \xrightarrow{(\tilde{u})a[\tilde{x}]} B_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} B_1'$, $\sigma = \sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}$, $(\tilde{w})(B_1'\sigma_{\tilde{x}=\tilde{y}}) \sim A_1'$, $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$. It follows that $P \xrightarrow{(\tilde{u})a[\tilde{x}]} (z)B_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} (z)B_1'$ and
$$(\tilde{w})((z)B_1'\sigma_{\tilde{x}=\tilde{y}}) \sim (z)A_1' \equiv P'$$
  • If $P \xrightarrow{\sigma} P'$ is obtained from $A_1 \xrightarrow{\sigma'} P'$ such that $P \equiv (z)A_1$, $z \in dom(\sigma')$ and $\sigma = \sigma'\uparrow z$. By induction hypothesis, there exist some a, $\tilde{x}$, $\tilde{y}$, $\tilde{u}$, $\tilde{v}$, $\tilde{w}$, $\sigma_{\tilde{x}=\tilde{y}}$, B_1 and B_1' such that $A_1 \xrightarrow{(\tilde{u})a[\tilde{x}]} B_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} B_1'$, $\sigma' = \sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}$, $(\tilde{w})(B_1'\sigma_{\tilde{x}=\tilde{y}}) \sim P'$, $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$. Then either $P \xrightarrow{(z)(\tilde{u})a[\tilde{x}]} B_1 \xrightarrow{(\tilde{v})\overline{a}[\tilde{y}]} B_1'$ or $P \xrightarrow{(\tilde{u})a[\tilde{x}]} (z)B_1 \xrightarrow{(z)(\tilde{v})\overline{a}[\tilde{y}]} B_1'$. In either case $\sigma = \sigma'\uparrow z = \sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}z$. It is also obvious that $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}z$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}z$.
– P is of the form $!P_1$. Use induction hypothesis.

This completes the proof. □

The next lemma spells out another property enjoyed by atomic χ-calculus. It is not satisfied by the polyadic χ-calculus.

Lemma 11. *If* $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ *then* $P \sim (\tilde{y})(\alpha[\tilde{x}]|P')$.

Proof. The proof is carried out by structural induction.

- P is of the form $\alpha[\tilde{x}]$. Then $P \sim \alpha[\tilde{x}]|\mathbf{0} \equiv \alpha[\tilde{x}]|P'$.
- P is of the form $P_1|P_2$ and $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ is caused by $P_1 \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P_1'$. By induction hypothesis $P_1 \sim (\tilde{y})(\alpha[\tilde{x}]|P_1')$. Hence

$$P \sim (\tilde{y})(\alpha[\tilde{x}]|P_1')|P_2 \sim (\tilde{y})(\alpha[\tilde{x}]|(P_1'|P_2))$$

- P is of the form $(z)P_1$. If $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ is caused by $P_1 \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P_1'$ such that $z \notin \tilde{x}$. Then $P' \equiv (z)P_1'$ and $P_1 \sim (\tilde{y})(\alpha[\tilde{x}]|P_1')$. Therefore $P \sim (z)(\tilde{y})(\alpha[\tilde{x}]|P_1') \sim (\tilde{y})(\alpha[\tilde{x}]|(z)P_1') \equiv (\tilde{y})(\alpha[\tilde{x}]|(z)P')$. If $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ is caused by $P_1 \stackrel{(\tilde{y'})\alpha[\tilde{x}]}{\longrightarrow} P'$ such that $z \in \tilde{x}$ and $\tilde{y} = z\tilde{y'}$. Then $P_1 \sim (\tilde{y'})(\alpha[\tilde{x}]|P')$. Consequently $P \sim (\tilde{y})(\alpha[\tilde{x}]|P')$.
- P is of the form $!P_1$. Then $P_1|!P_1 \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$. So $P \sim P_1|!P_1 \sim (\tilde{y'})(\alpha[\tilde{x}]|P')$.

This completes the proof. □

5 Ground Bisimilarity for Atomic χ-Calculus

In the study of asynchronous π-calculus it was observed that ground bisimilarity is closed under substitution and therefore is a congruence relation. This is significant in view of the fact that the ground bisimilarity for the π-calculus is not closed under substitution. This closure property is essentially due to three facts for the asynchronous language. The first is that a communication through a global name can be decomposed into an output action followed immediately by an input action. The second is that the ability to communicate through a local name is not affected by substitution. And the third is the property that $P \stackrel{\delta}{\longrightarrow} P'$ implies $P\sigma \stackrel{\delta\sigma}{\longrightarrow} P'\sigma$ for all substitution σ. All the three properties hold of the atomic χ-calculus. Therefore it should not come as a surprise that strong bisimilarity is equivalent to its ground counterpart. Compared with the situation in the asynchronous π-calculus, additional attention should be paid to update actions in atomic χ-calculus.

Definition 12. *Suppose* $\mathcal{R}$ *is a symmetric binary relation on* $\mathcal{A}$. *The relation* $\mathcal{R}$ *is a strong ground bisimulation if whenever* $P\mathcal{R}Q$ *then the following properties hold:*
(i) If $P \stackrel{\sigma_{\tilde{x}=\tilde{y}}}{\longrightarrow} P'$ *(including the case when* $\sigma_{\tilde{x}=\tilde{y}} = \tau$*) then some* Q' *exists such that* $Q \stackrel{\sigma_{\tilde{x}=\tilde{y}}}{\longrightarrow} Q'\mathcal{R}P'$.
(ii) If $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ *then some* Q' *exists such that* $Q \stackrel{(\tilde{y'})\alpha[\tilde{x}]}{\longrightarrow} Q'\mathcal{R}P'$ *and* $\{\tilde{y}\} = \{\tilde{y'}\}$.
The strong ground bisimilarity, notation $\sim_g$, *is the largest strong ground bisimulation.*

The next result is crucial to axiomatization of the strong bisimilarity.

Theorem 13. *$\sim_g$ is the same as $\sim$.*

Proof. It is obvious that $\sim \subseteq \sim_g$. For the reverse inclusion we prove that

$$\mathcal{R} \stackrel{\text{def}}{=} \{(P\sigma, Q\sigma) \mid P \sim_g Q \text{ and } \sigma \text{ is a substitution}\}$$

is a strong bisimulation. Since a substitution is the composition of a finite number of normal substitutions, we may consider normal substitutions only. Suppose $P \sim_g Q$ and σ is a normal substitution. There are three cases:

- $P\sigma \stackrel{(\tilde{y})a[\tilde{x}]}{\longrightarrow} P'$. By Lemma 10, there must exist some b, $\tilde{x'}$ and P_1 such that $P \stackrel{(\tilde{y})b[\tilde{x'}]}{\longrightarrow} P_1$, $b\sigma = a$, $\tilde{x'}\sigma = \tilde{x}$ and $P_1\sigma = P'$. By definition some Q_1 and $\tilde{y'}$ exist such that $Q \stackrel{(\tilde{y'})b[\tilde{x'}]}{\longrightarrow} Q_1 \approx_g P_1$ and $\{\tilde{y}\} = \{\tilde{y'}\}$. By Lemma 5, $Q\sigma \stackrel{(\tilde{y'})a[\tilde{x}]}{\longrightarrow} Q_1\sigma$.
- $P\sigma \stackrel{\sigma_1}{\longrightarrow} P'$ is caused by an interaction through global name a. By Lemma 10, there exist some $\tilde{x}$, $\tilde{y}$, $\tilde{u}$, $\tilde{v}$, $\tilde{w}$, $\sigma_{\tilde{x}=\tilde{y}}$, P_1 and P_1' such that $P\sigma \stackrel{(\tilde{u})a[\tilde{x}](\tilde{v})\overline{a}[\tilde{y}]}{\longrightarrow} P_1'$, $\sigma_1 = \sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}$, $(\tilde{w})(P_1'\sigma_{\tilde{x}=\tilde{y}}) \sim P'$, $\tilde{w} = rng(\sigma_{\tilde{x}=\tilde{y}}) \cap \tilde{u}\tilde{v}$ and $(\sigma_{\tilde{x}=\tilde{y}})^{-1}(\tilde{w}) \subseteq \tilde{u}\tilde{v}$. It follows from Lemma 6 that $P \stackrel{(\tilde{u})a'[\tilde{x'}](\tilde{v})\overline{a''}[\tilde{y'}]}{\longrightarrow} A_1'$ for some a', a'', $\tilde{x'}$ and $\tilde{y'}$ such that $\tilde{x'}\sigma = \tilde{x}$, $\tilde{y'}\sigma = \tilde{y}$, $a = a'\sigma = a''\sigma$ and $P_1' \equiv A_1'\sigma$. By definition some $\tilde{u'}$, $\tilde{v'}$, B_1 and B_1' exist such that $Q \stackrel{(\tilde{u})a'[\tilde{x'}](\tilde{v})\overline{a''}[\tilde{y'}]}{\longrightarrow} B_1' \sim_g A_1'$, $\{\tilde{u}\} = \{\tilde{u'}\}$ and $\{\tilde{v}\} = \{\tilde{v'}\}$. Using Lemma 5 one gets that $Q\sigma \stackrel{(\tilde{u})a[\tilde{x}](\tilde{v})\overline{a}[\tilde{y}]}{\longrightarrow} B_1'\sigma$. By Lemma 9, $Q\sigma \stackrel{\sigma_{\tilde{x}=\tilde{y}}\uparrow\tilde{u}\tilde{v}}{\longrightarrow} Q'$ for some Q' such that $Q' \sim (\tilde{w})(B_1'\sigma\sigma_{\tilde{x}=\tilde{y}})$. Notice that $P' \sim (\tilde{w})(A_1'\sigma\sigma_{\tilde{x}=\tilde{y}}) \equiv (\tilde{w})(P_1'\sigma_{\tilde{x}=\tilde{y}})$.
- $P\sigma \stackrel{\sigma_1}{\longrightarrow} P'$ is caused by an interaction through a local name. By Lemma 6, $P \stackrel{\sigma_{\tilde{x}=\tilde{y}}}{\longrightarrow} P_1$ for some $\sigma_{\tilde{x}=\tilde{y}}$ and P_1 such that $\sigma_{\tilde{x}\sigma=\tilde{y}\sigma} = \sigma_1$ and $P_1\sigma\sigma_{\tilde{x}\sigma=\tilde{y}\sigma} \equiv P'$. By definition some Q_1 exists such that $Q \stackrel{\sigma_{\tilde{x}=\tilde{y}}}{\longrightarrow} Q_1 \sim_g P_1$. By (iii) of Lemma 5, $Q\sigma \stackrel{\sigma_{\tilde{x}\sigma=\tilde{y}\sigma}}{\longrightarrow} Q_1\sigma\sigma_{\tilde{x}\sigma=\tilde{y}\sigma}$.

Conclude that $\mathcal{R}$ is a strong bisimulation up to $\sim$ and localization. It is then easy to show that $\mathcal{R}$ is included in the strong bisimilarity. □

Similarly one can introduce strong asynchronous ground bisimilarity. This bisimilarity formalizes the intuition that $(\tilde{x})(a[\tilde{x}]|\overline{a}[\tilde{x}])$ should be equivalent to $(a)(\tilde{x})(a[\tilde{x}]|\overline{a}[\tilde{x}])$.

Definition 14. *Suppose $\mathcal{R}$ is a symmetric binary relation on $\mathcal{A}$. The relation $\mathcal{R}$ is a strong asynchronous ground bisimulation if it satisfies the following property:*
(i) If $P \stackrel{\sigma}{\longrightarrow} P'$ then Q' exists such that $Q \stackrel{\sigma}{\longrightarrow} Q'\mathcal{R}P'$.
(ii) If $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ and $|\tilde{y}| < |\tilde{x}|$ then Q' and $\tilde{y'}$ exist such that $Q \stackrel{(\tilde{y'})\alpha[\tilde{x}]}{\longrightarrow} Q'\mathcal{R}P'$ and $\{\tilde{y}\} = \{\tilde{y'}\}$.
(iii) If $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ and $|\tilde{y}| = |\tilde{x}|$ then Q' exists such that one of the following properties holds:

– $\tilde{y'}$ *exist such that* $Q \xrightarrow{(\tilde{y'})\alpha[\tilde{x}]} Q'\mathcal{R}P'$ *and* $|\tilde{y'}| = |\tilde{x}|$.
– $Q \xrightarrow{\tau} Q'$ *and some* P'' *exists such that* $P' \xrightarrow{\overline{\alpha}[\tilde{x}]} P''\mathcal{R}Q'$.

The strong asynchronous ground bisimilarity, notation $\sim_g^a$*, is the largest strong asynchronous ground bisimulation.*

Theorem 15. $\sim_g^a$ *is closed under substitution.*

Proof. The proof is similar to that of Theorem 13. □

It follows that $\sim_g^a$ is a congruence relation. If in Definition 14 we have required the relation to be closed under substitution we would get strong asynchronous bisimilarity $\sim_a$. Theorem 15 says that the strong asynchronous bisimilarity is the same as the strong asynchronous ground bisimilarity.

6 Axiomatization in the Absence of Prefix, Summation and Match Combinators

The problems of axiomatization for equivalences on mobile processes are much more interesting than those in CCS. Extra effort is necessary to deal with mobility in an interleaving scenario. For that purpose match combinator is introduced in π-calculus. Together with the prefix and the summation combinators, the match operator retains the mobility when converting a concurrent process to a nondeterministic process. All the three operators, prefix, summation and match combinators, play crucial role in axiomatic treatment of π-calculus.

In [2] the authors have given a complete system for strong ground bisimilarity on finite asynchronous π-processes. The axiomatization uses restricted forms of prefix and summation operators. The match operator is not necessary.

In this section we make an initial step towards an axiomatic theory of atomic χ-calculus. We will be focusing on the strong bisimilarity and strong asynchronous bisimilarity. Our treatment is novel in that it uses none of the prefix combinator, the summation combinator and the match combinator.

Let $P_1, \ldots, P_n$ be n atomic χ-processes. We will write

$$\prod_{i=1}^{n} P_i \quad \text{or} \quad \prod_{i\in\{1,\ldots,n\}} P_i$$

for $(\ldots((P_1|P_2)|P_3)\ldots)|P_n$. When $n = 0$, $\prod_{i=1}^{n} P_i$ is just $\mathbf{0}$. We say that $\tilde{x}$ are the global name occurrences in P if the sequence of all the occurrences of global names appeared in P from left to right is $\tilde{x}$. We say that $\tilde{x}$ are the local name occurrences in P if the sequence of all the occurrences of local names appeared in P from left to right is $\tilde{x}$. Local name occurrences ignore the names appeared in localization operators. For instance the global name occurrences of $(x)(a[x]|(y)(\overline{a}[y]|c[c,x]))$ are *aacc* and the local name occurrences

of $(x)(a[x]|(y)(\overline{a}[y]|c[c,x]))$ are xyx. Suppose $\tilde{x_1},\ldots,\tilde{x_n}$ are the global name occurrences in $P_1,\ldots,P_n$ respectively and $\tilde{b_1},\ldots,\tilde{b_n}$ are the local name occurrences in $P_1,\ldots,P_n$ respectively. Suppose further that $\tilde{z_1},\ldots,\tilde{z_n}$ and $\tilde{d_1},\ldots,\tilde{d_n}$ are pairwise distinct fresh names such that $|\tilde{z_1}| = |\tilde{x_1}|,\ldots,|\tilde{z_n}| = |\tilde{x_n}|$ and $|\tilde{d_1}| = |\tilde{b_1}|,\ldots,|\tilde{d_n}| = |\tilde{b_n}|$. Let $\tilde{c_i}$, for each $i \in \{1,\ldots,n\}$, be obtained from $\tilde{b_i}$ by removing repetitive occurrences. It follows that $\{\tilde{c_i}\} = \{\tilde{b_i}\}$ and $\tilde{c_i}$ are pairwise distinct. Abbreviate $\tilde{x_1}\ldots\tilde{x_n}$, $\tilde{b_1}\ldots\tilde{b_n}$, $\tilde{z_1}\ldots\tilde{z_n}$, $\tilde{d_1}\ldots\tilde{d_n}$ and $\tilde{c_1}\ldots\tilde{c_n}$ by $\tilde{x}$, $\tilde{b}$, $\tilde{z}$, $\tilde{d}$ and $\tilde{c}$ respectively. Let $\tilde{v_1},\ldots,\tilde{v_n}$ and $\tilde{w_1},\ldots,\tilde{w_n}$ be sequences of names such that $|\tilde{v_1}| = |\tilde{w_1}|,\ldots,|\tilde{v_n}| = |\tilde{w_n}|$ and let $\tilde{v_1}\ldots\tilde{v_n}$, $\tilde{w_1}\ldots\tilde{w_n}$ be abbreviated respectively by $\tilde{v}$ and $\tilde{w}$. We will write $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ for the following atomic χ-process:

$$(e)(a)(\prod_{i=1}^{n}(\tilde{z})(\tilde{d})(a[e\tilde{z}\tilde{d}\tilde{w_1}\ldots\tilde{w_{i-1}}\tilde{v_i}\tilde{w_{i+1}}\ldots\tilde{w_n}]|P_i[\tilde{z_i}\tilde{d_i}])|(\tilde{c})\overline{a}[e\tilde{x}\tilde{b}\tilde{w}])$$

where e is a fresh name, a is fresh with suitable sort and $P_i[\tilde{z_i}\tilde{d_i}]$, for $i \in \{1,\ldots,n\}$, is P_i obtained by removing all the localization operators in P_i and replacing the global name occurrences $\tilde{x_i}$ by $\tilde{z_i}$ and local name occurrences $\tilde{b_i}$ by $\tilde{d_i}$. Let's see one example. Suppose P_1 is $(a)(z)(a[x]|(\overline{a}[z]|b[b]))$ and P_2 is $\overline{c}[y]|d[d]$. Suppose further that $\tilde{v_1} = b$, $\tilde{w_1} = c$, $\tilde{v_2} = cd$ and $\tilde{w_2} = ef$. Then $\sum_{i=1}^{2}[\tilde{v_i}{=}\tilde{w_i}].P_i$ denotes the following process

$$(t)(s)(((\tilde{z})(\tilde{b})(s[t\tilde{z}bef]|b_1[z_1]|(\overline{b_2}[b_3]|z_2[z_3]))|(\tilde{z})(s[t\tilde{z}ccd]|(\overline{z_4}[z_5]|z_6[z_7])))|A)$$

where A stands for $(a)(z)\overline{s}[txbbcyddaazcef]$, $\tilde{z}$ stands for $z_1z_2z_3z_4z_5z_6z_7$ and $\tilde{b}$ stands for $b_1b_2b_3$.

We call $[\tilde{v_i}{=}\tilde{w_i}].P_i$ a summand of $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ and $[\tilde{v_i}{=}\tilde{w_i}]$ an update prefix and P_i the corresponding continuation. We call $[\tilde{v_i}{=}\tilde{w_i}]$ a tau prefix, written τ, if both $\tilde{v_i}$ and $\tilde{w_i}$ are empty. When $n = 1$ we abbreviate $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ to $[\tilde{v_1}{=}\tilde{w_1}].P_1$, which will be further abbreviated to $[\tilde{v_1}{=}\tilde{w_1}]$ if P_1 is $\mathbf{0}$.

The reason to introduce a lot of fresh names in $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ is to make sure that

$$\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i \xrightarrow{\sigma_{\tilde{v_i}=\tilde{w_i}}} {\sim_g}\ P_i\sigma_{\tilde{v_i}=\tilde{w_i}}$$

and that $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ can not perform any other actions. To conform to the standard notation we will often write $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$ alternatively as

$$[\tilde{v_1}{=}\tilde{w_1}].P_1 + \ldots + [\tilde{v_n}{=}\tilde{w_n}].P_n$$

We write $\ldots + P_i + \ldots$ for instance when we want to emphasize particular summand(s) of $\sum_{i=1}^{n}[\tilde{v_i}{=}\tilde{w_i}].P_i$.

For update prefix $[\tilde{v}{=}\tilde{w}]$ let $[(\tilde{v}{=}\tilde{w}) \setminus x]$ denote any update prefix $[\tilde{y}{=}\tilde{z}]$ such that $\forall a,b.(a \neq x) \wedge (b \neq x) \Rightarrow ((\tilde{v}{=}\tilde{w} \Rightarrow a = b) \Leftrightarrow (\tilde{y}{=}\tilde{z} \Rightarrow a = b))$. For example $[(xy{=}yw) \setminus y]$ can be $[x{=}w]$.

E1	P	$= P$
E2	P	$= Q$ if $Q = P$
E3	P	$= R$ if $P = Q$ and $Q = R$
E4	$P\|R$	$= Q\|R$ if $P = Q$
E5	$(x)P$	$= (x)Q$ if $P = Q$
C1	$P\|\mathbf{0}$	$= P$
C2	$P\|Q$	$= Q\|P$
C3	$P\|(Q\|R)$	$= (P\|Q)\|R$
L1	$(x)\mathbf{0}$	$= \mathbf{0}$
L2	$(x)(y)P$	$= (y)(x)P$
L3	$(x)(P\|Q)$	$= P\|(x)Q$ if $x \notin gn(P)$
S1	$\ldots+[\tilde{v}=\tilde{w}].P+[\tilde{v'}=\tilde{w'}].P'+\ldots$	$= \ldots+[\tilde{v'}=\tilde{w'}].P'+[\tilde{v}=\tilde{w}].P+\ldots$
S2	$\ldots+[\tilde{v}=\tilde{w}].P+[\tilde{v}=\tilde{w}].P+\ldots$	$= \ldots+[\tilde{v}=\tilde{w}].P+\ldots$
S3	$\ldots+[\tilde{v}=\tilde{w}].P+\ldots$	$= \ldots+[\tilde{v}=\tilde{w}].P[w_i/v_i]+\ldots$ if v_i is the i-th component of $\tilde{v}$ and w_i is the i-th component of $\tilde{w}$
S4	$\ldots+[\tilde{v}=\tilde{w}].P+\ldots$	$= \ldots+[\tilde{v'}=\tilde{w'}].P+\ldots$ if $\tilde{v}=\tilde{w} \Leftrightarrow \tilde{v'}=\tilde{w'}$
S5	$(x)\sum_{i=1}^{n}[\tilde{v_i}=\tilde{w_i}].P_i$	$= \sum_{i=1}^{n}[(\tilde{v_i}=\tilde{w_i}) \setminus x].(x)P_i[x_i/x]$ where, for each $i \in \{1,\ldots,n\}$, $\tilde{v_i}=\tilde{w_i} \Rightarrow x = x_i$ and $x_i = x$ only if $\forall y.y \neq x \Rightarrow (\tilde{v_i}=\tilde{w_i} \not\Rightarrow x = y)$
A	$(\tilde{x})(a[\tilde{x}]\|\overline{a}[\tilde{x}])$	$= \tau$

Fig. 1. Axioms for $\sim_g$ and $\sim_g^a$

Let ASA be the set of axioms given in Figure 1 together with the following expansion law

$$(\tilde{x})(\prod_{i\in\{1,\ldots,n\}}\alpha_i[\tilde{v_i}]) = (\tilde{x})\sum_{\alpha_i=\overline{\alpha_j}}^{1\le i<j\le n}[\tilde{v_i}=\tilde{v_j}].\prod_{k\in\{1,\ldots,n\}\setminus\{i,j\}}\alpha_k[\tilde{v_i}]$$

where for each $i \in \{1,\ldots,n\}$ either $\alpha_i \in \tilde{x}$ or $\overline{\alpha_i} \in \tilde{x}$. Let AS be $ASA \setminus \{A\}$. We will write $AS \vdash P = Q$ to mean that the equality $P = Q$ is derivable from the rules and axioms of AS. When no confusion arises, we simply write $P = Q$. We will also write $P \stackrel{R}{=} Q$ to indicate that R is the major axiom applied to derive $P = Q$. The meaning of $ASA \vdash P = Q$ is similar.

Lemma 16. *If P is finite and $P \stackrel{(\tilde{y})\alpha[\tilde{x}]}{\longrightarrow} P'$ then $AS \vdash (\tilde{y})(\alpha[\tilde{x}]|P') = P$.*

Proof. This is a simple proof by structural induction. □

A process P is in normal form if it is of the form

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j)$$

such that the following conditions hold:

1. Each P_j, for $1 \leq j \leq m$, is in normal form.
2. $\tilde{x} \subseteq gn(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}])$.
3. There is an enumeration $i_1, \ldots, i_n$ of $\{1, \ldots, n\}$ such that

$$(\tilde{x})(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j) \xrightarrow{(\tilde{y_1})\alpha_{i_1}[\tilde{b_{i_1}}]} \ldots \xrightarrow{(\tilde{y_n})\alpha_{i_n}[\tilde{b_{i_n}}]} \sim_g \sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j$$

for some $\tilde{y_1}, \ldots, \tilde{y_n}$ with $\tilde{x} = \tilde{y_1} \ldots \tilde{y_n}$. When $|\tilde{x}| = 0$ and $m = n = 0$ the normal form is $\mathbf{0}$.

Lemma 17. *Every finite atomic χ-process is provably equal in AS to a normal form process.*

Proof. Suppose P is a finite atomic χ-process. Then $P \xrightarrow{(\tilde{y_1})\alpha_1[b_1]} \ldots \xrightarrow{(\tilde{y_n})\alpha_n[b_n]} P'$, for $n \geq 0$, such that all subject names in P' are local. By (ii) of Lemma 8 the process P' is not affected if $(\tilde{y_1})\alpha_1[b_1], \ldots, (\tilde{y_n})\alpha_n[b_n]$ are performed in a different order as long as it is legal. By C-axioms, L-axioms and Lemma 16, one derives that

$$AS \vdash P = (\tilde{y_1} \ldots \tilde{y_n})(\alpha_1[\tilde{b_1}]| \ldots |\alpha_n[\tilde{b_n}]|P')$$

When $y \in \{\tilde{y_1} \ldots \tilde{y_n}\} \setminus gn(\alpha_1[\tilde{b_1}]| \ldots |\alpha_n[\tilde{b_n}])$, we can use the L-laws and S5 to push (y) inwards. So we may assume that $\{\tilde{y_1} \ldots \tilde{y_n}\} \subseteq gn(\alpha_1[\tilde{b_1}]| \ldots |\alpha_n[\tilde{b_n}])$. Apply the expansion law and S5 to P', one gets

$$AS \vdash P = (\tilde{y_1} \ldots \tilde{y_n})(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j)$$

For each $j \in \{1, \ldots, m\}$, P_j is structurally simpler than P. So we can use induction hypothesis on P_j. □

We are now in a position to prove completeness theorems.

Theorem 18. *AS is sound and complete for $\sim_g$.*

Proof. The soundness can be easily verified. Suppose $P \sim_g Q$. By Lemma 17, we can assume that both P and Q are in normal form. Let P be

$$(\tilde{x})(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j)$$

and Q be

$$(\tilde{x'})(\prod_{i=1}^{n'} \alpha'_i[\tilde{b'_i}]|\sum_{j=1}^{m'}[\tilde{v'_j}=\tilde{w'_j}].Q_j)$$

By definition there is an enumeration $i_1, \ldots, i_n$ of $\{1, \ldots, n\}$ such that

$$(\tilde{x})(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j) \xrightarrow{(\tilde{y_1})\alpha_{i_1}[\tilde{b_{i_1}}]} \ldots \xrightarrow{(\tilde{y_n})\alpha_{i_n}[\tilde{b_{i_n}}]} = \sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j$$

for some $\tilde{y_1}, \ldots, \tilde{y_n}$ with $\tilde{x} = \tilde{y_1} \ldots \tilde{y_n}$. Now Q has to match up the above sequence of actions by

$$(\tilde{x'})(\prod_{i=1}^{n'} \alpha'_i[\tilde{b'_i}] | \sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j) \xrightarrow{(\tilde{y'_1})\alpha_{i_1}[\tilde{b_{i_1}}]} \ldots \xrightarrow{(\tilde{y'_n})\alpha_{i_n}[\tilde{b_{i_n}}]} = \sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j$$

such that $\sum_{j=1}^{m} [\tilde{v_j}{=}\tilde{w_j}].P_j \sim_g \sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j$ and $\{\tilde{y'_i}\} = \{\tilde{y_i}\}$ for each $i \in \{1, \ldots, n\}$. This says that by using C-laws, L-laws and α-conversion one can rewrite Q as

$$(\tilde{x})(\prod_{i=1}^{n} \alpha_i[\tilde{b_i}] | \sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j)$$

So we only have to show that

$$AS \vdash \sum_{j=1}^{m} [\tilde{v_j}{=}\tilde{w_j}].P_j = \sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j \tag{1}$$

Now for each $j \in \{1, \ldots, m\}$ one has that

$$\sum_{j=1}^{m} [\tilde{v_j}{=}\tilde{w_j}].P_j \xrightarrow{\sigma_{\tilde{v_j}=\tilde{w_j}}} = P_j \sigma_{\tilde{v_j}=\tilde{w_j}} \tag{2}$$

for any chosen $\sigma_{\tilde{v_j}=\tilde{w_j}}$. To match up there must exist some $j' \in \{1, \ldots, m'\}$ such that

$$\sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j \xrightarrow{\sigma_{\tilde{v'_j}=\tilde{w'_j}}} = Q_{j'} \sigma_{\tilde{v'_{j'}}=\tilde{w'_{j'}}} \sim_g P_j \sigma_{\tilde{v_j}=\tilde{w_j}}$$

and $\sigma_{\tilde{v'_{j'}}=\tilde{w'_{j'}}} = \sigma_{\tilde{v_j}=\tilde{w_j}}$. The latter implies $\tilde{v'_{j'}}{=}\tilde{w'_{j'}} \Leftrightarrow \tilde{v_j}{=}\tilde{w_j}$. It is clear that $P_j \sigma_{\tilde{v_j}=\tilde{w_j}}$ and $Q_{j'} \sigma_{\tilde{v'_{j'}}=\tilde{w'_{j'}}}$ are structurally simpler than P and Q respectively. By induction hypothesis

$$AS \vdash P_j \sigma_{\tilde{v_j}=\tilde{w_j}} = Q_{j'} \sigma_{\tilde{v'_{j'}}=\tilde{w'_{j'}}}$$

Now

$$\begin{aligned}
\sum_{j=1}^{m'} [\tilde{v'_j}{=}\tilde{w'_j}].Q_j &\equiv \ldots + [\tilde{v'_{j'}}{=}\tilde{w'_{j'}}].Q_{j'} + \ldots \\
&\overset{S3}{=} \ldots + [\tilde{v'_{j'}}{=}\tilde{w'_{j'}}].Q_{j'} \sigma_{\tilde{v'_{j'}}=\tilde{w'_{j'}}} + \ldots \\
&= \ldots + [\tilde{v'_{j'}}{=}\tilde{w'_{j'}}].P_j \sigma_{\tilde{v_j}=\tilde{w_j}} + \ldots \\
&\overset{S4}{=} \ldots + [\tilde{v_j}{=}\tilde{w_j}].P_j \sigma_{\tilde{v_j}=\tilde{w_j}} + \ldots \\
&\overset{S3}{=} \ldots + [\tilde{v_j}{=}\tilde{w_j}].P_j + \ldots
\end{aligned}$$

For each $j \in \{1, \ldots, m\}$ we consider all the summands of $\sum_{j=1}^{m'}[\tilde{v'_j}{=}\tilde{w'_j}].Q_j$ that can match the action in (2) and carry out the transformation. In this way we can rewrite $\sum_{j=1}^{m'}[\tilde{v'_j}{=}\tilde{w'_j}].Q_j$ into a form each of its summands is syntactically identical to a summand of $\sum_{j=1}^{m}[\tilde{v_j}{=}\tilde{w_j}].P_j$. Then use S1 and S2 to get (1). □

Theorem 19. *ASA is sound and complete for* $\sim_g^a$.

Proof. Suppose $P \sim_g^a Q$. Let P be

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}{=}\tilde{w_j}].P_j)$$

and Q be

$$(\tilde{x'})(\prod_{i=1}^{n'}\alpha'_i[\tilde{b'_i}]|\sum_{j=1}^{m'}[\tilde{v'_j}{=}\tilde{w'_j}].Q_j)$$

If

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}{=}\tilde{w_j}].P_j) \xrightarrow{(\tilde{b'_i})\alpha[\tilde{b_i}]} P'$$

where $\{\tilde{b'_i}\} = \{\tilde{b_i}\}$ and $|\tilde{b'_i}| = |\tilde{b_i}|$, is matched up by

$$(\tilde{x'})(\prod_{i=1}^{n'}\alpha'_i[\tilde{b'_i}]|\sum_{j=1}^{m'}[\tilde{v'_j}{=}\tilde{w'_j}].Q_j) \xrightarrow{\tau} Q'$$

for some Q' such that

$$P' \xrightarrow{\overline{\alpha}[\tilde{b_i}]} P'' \sim_g^a Q'$$

Then $\tilde{b_i} \cap gn(Q') = \emptyset$ and $\tilde{b_i} \cap gn(P'') = \emptyset$ for otherwise P'' can perform a sequence of actions which Q' can not simulate. Using L-laws, one shows that

$$AS \vdash P = (\tilde{b_i})(\alpha[\tilde{b_i}]|\overline{\alpha}[\tilde{b_i}])|P_1$$

for some normal form process P_1. Consequently

$$ASA \vdash P = \tau|P_1$$

by induction hypothesis. If on the other hand

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}{=}\tilde{w_j}].P_j) \xrightarrow{(\tilde{b'_i})\alpha[\tilde{b_i}]} P' \xrightarrow{\overline{\alpha}[\tilde{b_i}]} P'' \tag{3}$$

such that $\tilde{b_i} \cap gn(P'') \neq \emptyset$, then by the above analysis

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}{=}\tilde{w_j}].P_j) \xrightarrow{(\tilde{b'_i})\alpha[\tilde{b_i}]} P'$$

must be simulated by

$$(\tilde{x'})(\prod_{i=1}^{n'}\alpha'_i[\tilde{b'_i}]|\sum_{j=1}^{m'}[\tilde{v'_j}=\tilde{w'_j}].Q_j) \stackrel{(\tilde{b''_i})\alpha[\tilde{b_i}]}{\longrightarrow} Q'$$

for some Q' and $\tilde{b''_i}$ such that $\{\tilde{b''_i}\} = \{\tilde{b_i}\}$ and $|\tilde{b''_i}| = |\tilde{b_i}|$.

To continue observe that, by expansion law,

$$ASA \vdash \tau|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j = \sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].(\tau|P_j) + \tau.\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j$$

Here the right hand side stands for

$$\sum_{j=1}^{m+1}[\tilde{v'_j}=\tilde{w'_j}].P'_j$$

where $\tilde{v'_j} = \tilde{v_j}$, $\tilde{w'_j} = \tilde{w_j}$, $P'_j \equiv P_j$, for $j \in \{1,\ldots,m\}$, $\tilde{v'_{m+1}}$ and $\tilde{w'_{m+1}}$ are empty sequence, and P_{m+1} is $\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j$. So we can now assume that P is of the normal form

$$(\tilde{x})(\prod_{i=1}^{n}\alpha_i[\tilde{b_i}]|\sum_{j=1}^{m}[\tilde{v_j}=\tilde{w_j}].P_j)$$

such that $\tilde{b_i} \cap gn(P'') \neq \emptyset$ whenever (3) holds. Similar assumption can be made for Q.

The rest of the proof is similar to that of Theorem 18. □

7 Final Remarks

The popular way of defining the operational semantics of the Fusion Calculus is to impose a structural congruence on processes. This approach makes it easy to define communications. In this paper we have adopted a pure labeled transition semantics. In our view this latter approach simplifies rather than complicates the algebraic investigation.

Complete systems for weak congruences are much more difficult than those for strong congruences. In CCS ([14, 17]) it is well known that complete system for weak equivalence can be obtained from that for strong equivalence by adding three famous tau laws of Milner's. For some time it had been conjectured that the situation in π-calculus is similar. It was pointed out recently by the present author ([10]) however that in general Milner's tau laws fail to lift a complete system for a strong congruence on finite mobile processes to a complete system for the corresponding weak congruence. A fourth tau law is necessary to deal with mobility. Axiomatization for asynchronous π-calculus is more difficult than for π-calculus. Weak congruence for finite asynchronous π-processes has not been axiomatized. As atomic χ-calculus shares properties with the asynchronous π-calculus one expects that its axiomatization problem for weak congruence is

equally difficult. This seems to be the most important question about atomic χ-calculus. Until we have answered this question, we can't really say that we understand the algebraic theory of the calculus.

Acknowledgement The author is supported by the NNSFC (69873032), the 863 Hi-Tech Project (863-306-ZT06-02-2), the Young Scientist Research Fund and the University Scholar Funding Scheme. He is also supported by BASICS, Center of Basic Studies in Computing Science, sponsored by Shanghai Education Commission. BASICS is affiliated to the Department of Computer Science of Shanghai Jiaotong University.

References

1. S. Abramsky, Proofs as Processes, *Theoretical Computer Science*, **135**, 5–9, 1994.
2. R. Amadio, I. Castellani, D. Sangiorgi. On Bisimulation for the Asynchronous π-Calculus. *CONCUR'96*, Lecture Notes in Computer Science 1119, Springer, 1996.
3. G. Boudol. Asynchrony and the π-calculus. Research Report 1702, INRIA, Sophia-Antipolis, 1992.
4. Y. Fu. The χ-Calculus. *Proceedings of the International Conference on Advances in Parallel and Distributed Computing*, 74-81, March 19th-21th, Shanghai, IEEE Computer Society Press, 1997.
5. Y. Fu. A Proof Theoretical Approach to Communications. *ICALP'97*, 325–335, July 7th-11th, Bologna, Italy, Lecture Notes in Computer Science 1256, Springer, 1997.
6. Y. Fu. Symmetric π-Calculus. *Journal of Computer Science and Technology*, **13**: 202–208, 1998.
7. Y. Fu. Reaction Graphs. *Journal of Computer Science and Technology*, **13**: 510–530, 1998.
8. Y. Fu. Bisimulation Lattice of Chi Processes. *ASIAN'98*, December 8-10, Manila, The Philippines, Lecture Notes in Computer Science 1538, Springer, 245–262, 1998.
9. Y. Fu. Variations on Mobile Processes. *Theoretical Computer Science*, Vol. 221, 327-368, 1999.
10. Y. Fu. Open Bisimulations of Chi Processes. *CONCUR'99*, Eindhoven, The Netherlands, August 24-27, Lecture Notes in Computer Science 1664, 304-319, Springer, 1999.
11. Y. Fu, Z. Yang. Chi Calculus with Mismatch. CONCUR 2000, 22-25 August, Pennsylvenia, USA, Lecture Notes in Computer Science 1877, 596-610, Springer, 2000.
12. Y. Fu, Z. Yang. The Ground Congruence for Chi Calculus. FST&TCS 2000, December, India, Lecture Notes in Computer Science, Springer, 2000.
13. J. Girard. Linear Logic, *Theoretical Computer Science*, **50**, 1–102, 1987.
14. M. Hennessy, R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of ACM*, **67**: 137–161, 1985.
15. K. Honda, M. Tokoro. An Object Calculus for Asynchronous Communication. *Proc. ECOOP '91*, Geneve, 1991.
16. C. Laneve, B. Victor. Solos in Concert. *ICALP'99*, July 11th-15th, Prague, Hungary, Lecture Notes in Computer Science, Springer, 1999.
17. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

18. 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, 1993.
19. R. Milner, J. Parrow, D. Walker. A Calculus of Mobile Processes. *Information and Computation*, **100**: 1–40 (Part I), 41–77 (Part II), Academic Press.
20. M. Merro, D. Sangiorgi. On Asynchrony in Name-Passing Calculi. *ICALP'98*, Lecture Notes in Computer Science 1443, Springer, 1998.
21. J. Parrow, B. Victor. The Update Calculus. *AMAST'97*, Sydney, December 13-17, 1997.
22. J. Parrow, B. Victor. The Fusion Calculus: Expressive and Symmetry in Mobile Processes. *LICS'98*, 1998.
23. J. Parrow, B. Victor. The Tau-Laws of Fusion. *CONCUR'98*, Lecture Notes in Computer Science, 1998.
24. B. Victor, J. Parrow. Concurrent Constraints in the Fusion Calculus. *ICALP '98*, Lecture Notes in Computer Science 1443, 455–469, Springer, 1998.