Distributed Synchronization in a π-calculus with Bidirectional Communication [*]

Vladimir Alexiev
University of Alberta, Edmonton, Canada
`vladimir@cs.ualberta.ca`

Abstract
The (input) prefix operation of the π-calculus expresses *global* synchronization (blocking) of the prefixed process. We show how to implement synchronization in a completely distributed manner, by using *bidirectional* atomic communication and the principle of *provision* (data-dependency-based synchronization).

Keywords π-calculus, input prefix, distributed synchronization, bidirectional communication.

1 Introduction

The π-calculus (Milner *et al.*, 1992) is one of the most eminent tools for the theoretical study of concurrent processes. Its popularity is due to the parsimony of its basic constructs, which nevertheless provide a rich enough setting that can capture many computational phenomena.

There is some recent research concerned with the question if all π-calculus constructs are really necessary, or can some of them be translated to other constructs. Milner (1993b) showed how to model *polyadic* (n-ary) communication using only monadic communication. Honda and Tokoro (1991) expressed output prefix using only output atoms (the *zipper construction*), which grew into a study of an asynchronous variant of the π-calculus, called the ν-calculus. Nestmann and Pierce (1996) researched two implementations of input-guarded choice in a choice-free fragment, and Nestmann (1997) studied also output-guarded and mixed-guarded choice. Parrow (1997) showed how to encode the π-calculus with replication (but no choice) into a fragment where only sequences of at most three blocking prefixes $\pi_1.\pi_2.\pi_3$ are allowed. Honda and Yoshida (1994a,b) exhibited a combinatory (name-free) representation of the π-calculus.

These studies are important for several reasons:

- They explore the expressive power of the various fragments of the π-calculus. Palamidessi (1997) proved one of the few *negative* results in this area, namely that ν without choice cannot capture π with mixed-guarded choice.
- They further our understanding of the π-calculus, by exposing finer computational structure that the "aggregate" operators may have left implicit, by suggesting which of its constructs should be primitive and which may be taken as derived, and by exploring tradeoffs in the choice of primitive constructs.
- They make eventual implementation easier, by minimizing the required kernel language (*e.g.* the encoding of choice in Nestmann and Pierce (1996) is directly inspired by an implementation in the Pict language).

This work continues the trend by showing how to implement the synchronization embodied by (blocking) prefix in a completely distributed manner. It takes a modified version of the polyadic π-calculus that allows bidirectional communication (π_{bc}), and embeds it faithfully into its fragment that only has input/output *atoms*, but not prefixes.

[*]Supported by a University of Alberta Dissertation Scholarship.

2 The π-calculus with Bidirectional Communication

We consider a version of the polyadic π-calculus (Milner, 1993b) that we call π_{bc} (π with bidirectional communication) and which has the following modifications:

- We study only the finite fragment (no choice nor replication). We do not need them for our construction, and we believe that our results also hold for the full version. The omission of choice and replication simplifies the exposition.
- We allow non-blocking prefixes (which can also be considered as input/output atoms). The π-calculus already can express output atoms, but we require a special treatment of input atoms, in order to disentangle the issues of *scope* and *synchronization* (see §2.1).
- We allow atomic *bidirectional* communication, *i.e.* mixed input-output prefixes.

Lowercase letters and their subscripted variants denote any of countably many of *channel names.* Uppercase letters denote *processes* that are defined inductively as follows:

Zero 0 is the empty (do-nothing) process.

Parallel composition P, Q behaves as both P and Q, with possible interaction between the two.

Blocking Prefix $a\dagger_1 x_1 \ldots \dagger_n x_n.P$ (where $\dagger_i$ is ! for output and ? for input)[1] outputs some names through a, simultaneously and atomically inputs some names from a, binds these names into P, then proceeds like P where the placeholders are replaced with the received values.

Non-blocking Prefix Similarly, $(a\dagger_1 x_1 \ldots \dagger_n x_n, P)$ inputs/outputs names and binds the inputs in P, but it does not block P except for those parts of P that actually use names bound by inputs (see Provision below).

Hiding (Restriction) $(a)P$ is just like P, but the channel a is hidden: no values can be sent/received over that channel by other processes. The scope of the restriction extends to the next comma, or we may show it explicitly like so: $(a)(P, Q)$.

Notation We denote prefixes $a\dagger_1 x_1 \ldots \dagger_n x_n$ as $a\dagger\vec{x}$, or simply as π, π', *etc.* We say that a is in *subject* position and x_i are in *object* position. πP stands for either $\pi.P$ or π, P. Both $\pi.0$ and $\pi, 0$ are abbreviated as π (communication atom). $a!(v)$ denotes $(v)a!v$. $a!\vec{v}$ denotes $a!v_1 \ldots !v_n$ and $a?\vec{x}$ denotes $a?x_1 \ldots ?x_n$.

Names The *names* occurring in a process are classified as free or bound: $\mathrm{n}(P) = \mathrm{fn}(P) + \mathrm{bn}(P)$ (here $+$ denotes disjoint union). For bound names, we assume that α-renaming is used freely so as to satisfy the disjointness provisions embodied in the use of $+$ below

$$\begin{gathered}
\mathrm{fn}(0) = \mathrm{bn}(0) = \emptyset, \\
\mathrm{fn}(P, Q) = \mathrm{fn}(P) \cup \mathrm{fn}(Q), \mathrm{bn}(P, Q) = \mathrm{bn}(P) + \mathrm{bn}(Q) \\
\text{where } \mathrm{fn}(P) \cap \mathrm{bn}(Q) = \mathrm{bn}(P) \cap \mathrm{fn}(Q) = \emptyset, \\
\mathrm{fn}(c!\vec{v}?\vec{x}P) = \{c, \vec{v}\} \cup \mathrm{fn}(P), \mathrm{bn}(c!\vec{v}?\vec{x}P) = \{\vec{x}\} + \mathrm{bn}(P), \\
\mathrm{fn}((c)P) = \mathrm{fn}(P) \backslash \{c\}, \mathrm{bn}((c)P) = \{c\} + \mathrm{bn}(P).
\end{gathered}$$

2.1 Confounded Scope and Over-Synchronization in π

The input prefix $a?x.P$ of the π-calculus is overloaded with two roles:

- To pass the received value to P. For this it is important to delineate the scope of the name x bound by the prefix.

[1]Instead of the traditional notation, we use this Occam-like notation, mainly because it is more lucid in email and in the typescript.

- To synchronize (block) the process P, so that no reduction can happen (neither inside P, nor between P and another process) before the prefix is reduced.

The scope of the value is confounded with the scope of synchronization. This makes it seemingly impossible to express the following simple idea in π without choice:[2] $a?x, b?y, x!y$. Here two names x and y are received from a and b *independently* and then one of them is sent on the other. Naturally, $x!y$ should wait until both of $a?x$ and $b?y$ reduce, but in the π-calculus we are also forced to impose an artificial ordering on the latter two, either $a?x.b?y.x!y$, or $b?y.a?x.x!y$. The reason is that π scopes must either be properly nested, or disjoint. Thus we cannot have a scope where both x and y are defined, unless their binders are nested. This leads to unnecessary synchronizations and decreases the inherent parallelism of a computation.

We therefore untie the value scope of a prefix from its synchronization scope,[3] by allowing an input atom to commute freely with another atom. However, we retain some measure of synchronization, embodied in the principle of Provision.

Definition 2.1.1 (Provision) *(Parrow, 1995) A prefix π containing a name $y \in fn(\pi)$ (in subject or output object position) is* provisioned *by a prefix $\pi' = a\dagger\vec{x}?y\dagger\vec{z}$ having y in input object position. π cannot reduce until π' is reduced and y is instantiated.*

Provision is implicit in the π-calculus because an input prefix blocks the entirety of its scope. We argue that provision is a natural principle, because it captures the *natural causal* dependencies between prefixes, and it corresponds to data-dependency and demand-driven architectures. It provides blocking of finer granularity than the explicit, aggregate blocking of π. This work shows that we can express aggregate blocking exactly by only using provision-based blocking.

The lack of finer synchronization mechanisms[4] has led to somewhat gratuitous use of blocking. For example, in the construction of Honda and Tokoro (1991)[5]

$$\begin{aligned} z![x_1x_2].P &::= (w)z!w, w?v_1.(v_1!x_1, w?v_2.(v_2!x_2, P) \\ z?(y_1y_2).Q &::= (v_1'v_2')z?w'.(w'!v_1', v_1'?y_1.(w'!v_2', v_2'?y_2.Q)) \end{aligned}$$

one can count at least 5 unnecessary blocks that we denote as a dependency relation below:

$$w?v_1 \prec v_1!x_1,\ w?v_2 \prec v_2!x_2,\ z?w' \prec w'!v_1',\ z?w' \prec v_1'?y_1,\ v_1'?y_1 \prec v_2'?y_2.$$

These blocks are already enacted by provision.

In a distributed implementation of π, every synchronization comes with a cost, therefore it is desirable to express various constructions by using only the necessary blocks, or "natural" (provision-based) synchronization.

2.2 Structural Congruence

A standard way to simplify the presentation of a transition system (see below) for a process calculus is to introduce structural rules describing equivalence classes of terms (processes) that are essentially the same, and only differ in their syntactical presentation. $\equiv$ is the smallest congruence (with respect to process constructors) that satisfies the following rules:

1. $P \equiv P[x/y]$ if $x, y \notin \text{fn}(P)$ (α-renaming). We already assumed above that this rule is applied freely so that all bound names are different (renamed apart) from each other and from the free names.

2. $0, P \equiv P; P, Q \equiv Q, P; (P, Q), R \equiv P, (Q, R)$: the parallel constructor is symmetric and associative, and 0 is its neutral element.

[2] And from a "true concurrency" viewpoint, $a.b + b.a$ is not the same and "inferior" to a, b anyway.
[3] Milner (1993a) has considered such distributed scopes in the actions of a calculus that he calls the *synchronous* π-calculus.
[4] And, perhaps, the notational simplicity of the dot.
[5] That implements polyadic prefix using monadic prefix.

3. $(x)0 \equiv 0$, $(x)(y)P \equiv (y)(x)P$.

4. $(x)P, Q \equiv (x)(P, Q)$ if $x \notin \mathrm{n}(Q)$: the scope of a restriction can be extended if there's nothing to hide. This and the previous rule imply that $Q \equiv (x)Q$ under the same proviso. Thus we can push all hidings to the outside and basically forget about them. Hidings are important when building a process, but they don't have much to do with the dynamics of processes.[6]

5. $\pi_1, (\pi_2, P) \equiv \pi_2, (\pi_1, P)$ unless π_2 is provisioned by π_1. Also $\pi, P \equiv P, \pi$ unless $\mathrm{bn}(\pi) \cap \mathrm{fn}(P) \neq \emptyset$.

The last rule reflects the discussion in the previous subsection.

2.3 Transition System

We present the operational semantics of π_{bc} as a labeled transition system. The set of labels (actions) μ is

$$\mu ::= a\dagger\vec{x} \mid \tau$$

where $\dagger\vec{x}$ may contain a mixture of inputs $?x$, outputs $!x$, and bound outputs $!(x)$; and τ is the internal (silent) action.

The transition relation $P \xrightarrow{\mu} Q$ is the smallest relation generated by the axiom and rules below.

Prefix $a\dagger\vec{x}P \xrightarrow{a\dagger\vec{x}} P$. A prefix (either blocking or non-blocking) produces a corresponding action.

Struc If $P \xrightarrow{\mu} Q$ and $P \equiv P'$, $Q \equiv Q'$ then $P' \xrightarrow{\mu} Q'$. This incorporates the structural congruence into the transition relation.

Alpha If $P \xrightarrow{\mu} Q$ then $P[x/y] \xrightarrow{\mu[x/y]} Q[x/y]$ where $x \notin \mathrm{fn}(\mu)$ and y is a fresh name. This α-renaming for transitions makes it possible to always satisfy the proviso of the Par rule below.

Par If $P \xrightarrow{\mu} Q$ then $P, R \xrightarrow{\mu} Q, R$, provided that $\mathrm{bn}(\mu) \cap \mathrm{fn}(R) = \emptyset$. Adding a parallel component does not decrease the possibilities for reduction. The proviso ensures that a free name of R is not captured inadvertently by the action.

Res If $P \xrightarrow{\mu} Q$ then $(x)P \xrightarrow{\mu} (x)Q$, provided that $x \notin \mathrm{n}(\mu)$. Restriction does not decrease the possibilities for reduction of the inside process.

Scope If $P \xrightarrow{a!v\dagger\vec{v}} Q$ then $(v)P \xrightarrow{a!(v)\dagger\vec{v}} (v)Q$, provided $a \neq v$ (here $!v$ does not have to be the first object of the prefix, and all occurrences of v are hidden in the action of the conclusion). This allows to deduce reduction of bound outputs.

Comm If $P \xrightarrow{a\dagger\vec{x}} P'$ and $Q \xrightarrow{a\bar{\dagger}\vec{x}} Q'$ then $P, Q \xrightarrow{\tau} (\vec{b})(P', Q')$, where

1. $\bar{?} = !$ and $\bar{!} = ?$.
2. If one action (say the one of $P \to P'$) has $?x$ then the other may have either $!x$ or $!(x)$ in the corresponding position. In the latter case (bound output), there is a proviso that $x \notin \mathrm{fn}(P)$, and x is hidden in the result: $x \in \vec{b}$.

The Prefix and Comm rules are the essence of π_{bc}. The other rules extend the transition relation in various contexts. It is important to note that transition is not a congruence with respect to blocking prefix (does not happen under blocking prefix). Also, communication over a hidden channel $(a)a\dagger\vec{x}$ is not a valid action, but one can deduce two non-hidden communications $P \xrightarrow{a\dagger\vec{x}} P'$ and $Q \xrightarrow{a\bar{\dagger}\vec{x}} Q'$, "summarize" them using Comm to $P, Q \xrightarrow{\tau} P', Q'$, and deduce that the same is possible on a hidden channel using Res: $(a)(P, Q) \xrightarrow{\tau} (a)(P', Q')$.

The weak versions of the transition relation are defined as

$$\Rightarrow ::= \overset{\tau}{\Rightarrow} ::= \xrightarrow{\tau}^{*}, \quad \overset{\mu}{\Rightarrow} ::= \Rightarrow\xrightarrow{\mu}\Rightarrow.$$

[6]This is not the case if we consider replication, in which case hiding creates new names "at runtime".

3 Distributed Synchronization

In this section we define a translation of a π_{bc} process P to a process $[\![P]\!]$ in a fragment of π_{bc} that we call π_{ds} (π with Distributed Synchronization). π_{ds} is defined as π_{bc} without the Blocking Prefix construction.

3.1 The Translation

We define $[\![P]\!]$ by induction on the structure of P. Simultaneously with it we define an auxiliary set $\mathcal{B}[\![P]\!]$, the set of *blocking points* of $[\![P]\!]$. $\mathcal{B}[\![P]\!]$ are private names that will be provisioned if the process P is prefixed. They are used only in the Blocking Prefix rule below.

We add two auxiliary names to *every* prefix π, one that is used to block the prefix (b), and another that is used to unblock any processes that are provisioned by the prefix (u).

Zero $[\![0]\!] ::= 0$; $\mathcal{B}[\![0]\!] ::= \emptyset$.

Parallel $[\![P, Q]\!] ::= [\![P]\!], [\![Q]\!]$; $\mathcal{B}[\![P, Q]\!] ::= \mathcal{B}[\![P]\!] + \mathcal{B}[\![Q]\!]$.

Hiding $[\![(a)P]\!] ::= (a)[\![P]\!]$; $\mathcal{B}[\![(a)P]\!] ::= \mathcal{B}[\![P]\!]$.

Blocking Prefix We assign b and u for a prefix π according to the type of the first object of π.[7]

- If $\pi = a!x_1\dagger_2 x_2 \dots \dagger_n x_n$ then $[\![\pi]\!] ::= a!(b)?u\dagger\vec{x}$.
- If $\pi = a?x_1\dagger_2 x_2 \dots \dagger_n x_n$ then $[\![\pi]\!] ::= a?u!(b)\dagger\vec{x}$.

$$[\![\pi.P]\!] ::= [\![\pi]\!], \textstyle\prod_{c\in\mathcal{B}[\![P]\!]}(u?c, u!(c')), [\![P]\!]'; \quad \mathcal{B}[\![\pi.P]\!] ::= \mathcal{B}[\![\pi]\!] ::= \{b\}.$$

Here $\prod$ denotes indexed parallel composition and c' are fresh private names. $[\![P]\!]$ is structurally equivalent to $(\vec{c})P'$ for some P', and $[\![P]\!]'$ is obtained from it by unhiding the names $\vec{c}$ ($[\![P]\!]' = P'$), so that they can be made input objects of u.

Non-blocking Prefix $[\![\pi, P]\!] ::= [\![\pi]\!], [\![P]\!]$; $\mathcal{B}[\![\pi, P]\!] ::= \mathcal{B}[\![\pi]\!], \mathcal{B}[\![P]\!]$. As an optimization, $\pi.0$ is considered a non-blocking prefix (atom).

The translation acts homomorphically on all constructors but blocking prefix. The opening that $[\![P]\!]'$ embodies is the only "non-compositional" aspect of our translation.

For example,

$$[\![a!x.d!y]\!] = a!(b_a)?u_a!x,\, u_a?b_d,\, u_a!(b'_d),\, d!b_d?u_d!y.$$

Here the output on a can reduce as soon as there is a matching input. However, the output on d has to wait until b_d is instantiated. This can only happen by the communication on u_a, which can only happen after u_a is instantiated (which happens when $a!(b_a)?u!x$ reduces),

In general, two complementary prefixes π and π' cannot communicate until both their blocking points b and b' are provisioned. Once this happens, they communicate and transmit the "data" channels $\dagger\vec{x}$ as expected. In addition, they provision each other's unblocking channels u and u'. Now the communications $u?c, u!(c')$ (and similarly for u') become possible, and they unblock the subordinated process(es).

As a more complex example, here is $[\![z![x_1 x_2].P]\!]$, the output part of the Honda-Tokoro construction (see the end of §2.1):

$$\begin{gathered}(w)z!(b_z)?u_z!w,\, w?u_{w1}!(b_{w1})?v_1,\, u_{w1}?b_{v_1},\, u_{w1}!(b'_{v_1}),\, v_1!b_{v_1}?u_{v_1}!x_1,\, u_{w1}?b_{w2},\, u_{w1}!(b'_{w2}),\\ w?u_{w2}!b_{w2}?v_2,\, u_{w2}?b_{v_2},\, u_{w2}!(b'_{v_2}),\, v_2!b_{v_2}?u_{v_2}!x_2,\, \textstyle\prod_{b\in\mathcal{B}[\![P]\!]}(u_{w2}?b,\, u_{w2}!(b')),\, [\![P]\!].\end{gathered}$$

Compare this to the variant with two unnecessary blockings removed ($v_i!x_i$ are already data-dependent upon $w?v_i$):

$$\begin{gathered}(w)z!(b_z)?u_z!w,\, w?u_{w1}!(b_{w1})?v_1,\, v_1!(b_{v_1})?u_{v_1}!x_1,\, u_{w1}?b_{w2},\, u_{w1}!(b'_{w2}),\\ w?u_{w2}!b_{w2}?v_2,\, v_2!(b_{v_2})?u_{v_2}!x_2,\, \textstyle\prod_{b\in\mathcal{B}[\![P]\!]}(u_{w2}?b,\, u_{w2}!(b')),\, [\![P]\!].\end{gathered}$$

[7] Therefore we do not allow the input/output of 0 names, but we can easily recover this ability, by assigning dummy types to such prefixes, *e.g.* $a![]$ and $a?()$.

3.2 Complexity of the Translation

We now prove that the size of our translation is linear with respect to the size of the source process. Let define the size $\mathcal{S}(P)$ of a process as the number of name occurrences, excluding restrictions. For example, $\mathcal{S}((x)a!x, a?y) = 4$.

Lemma 3.2.1 *The number of blocking points of* $[\![P]\!]$ *is* $|\mathcal{B}[\![P]\!]| = \mathcal{T}(P)$, *where* $\mathcal{T}(P)$ *is the number of top-level parallel components of* P.

Proof Easy after noticing that $|\mathcal{B}[\![\pi.P]\!]| = 1$. $\diamond$

Theorem 3.2.2 $\mathcal{S}[\![P]\!] = \mathcal{S}(P) + 6\mathcal{P}(P) - 4\mathcal{T}(P)$ *where* $\mathcal{P}(P)$ *is the number of prefixes/atoms in* P.

Proof Induction on the definition of $[\![P]\!]$. The homomorphic cases are obvious, because the above measures are "additive". For atom, $\mathcal{S}[\![\pi]\!] = \mathcal{S}(\pi) + 2 = \mathcal{S}(\pi) + 6\mathcal{P}(\pi) - 4\mathcal{T}(\pi)$ (since $\mathcal{P}(\pi) = \mathcal{T}(\pi) = 1$). For prefix, $\mathcal{S}[\![\pi.P]\!] = \mathcal{S}(\pi)+2+4\,|\mathcal{B}[\![P]\!]|+\mathcal{S}[\![P]\!]$. By induction hypothesis and the lemma, the RHS equals $\mathcal{S}(\pi)+2+4\mathcal{T}(P)+\mathcal{S}(P)+6\mathcal{P}(P)-4\mathcal{T}(P) = \mathcal{S}(\pi.P)+2+6\mathcal{P}(\pi.P)-6 = \mathcal{S}(\pi.P)+6\mathcal{P}(\pi.P)-4\mathcal{T}(\pi.P)$ (since $\mathcal{T}(\pi.P) = 1$). $\diamond$

Corollary 3.2.3 $\mathcal{S}[\![P]\!] < 7\mathcal{S}(P)$. $\diamond$

For the case of the finite π-calculus (no replication) it is trivial to see that a similar claim holds about the number of reduction steps of $[\![P]\!]$, because this number is bound by $\mathcal{P}([\![P]\!])$ and thus $\mathcal{S}([\![P]\!])$. Furthermore, as can be seen from our completeness proof (A.0.1), a refinement of the claim is also true for the infinitary π_{bc}, because π_{ds} reduction takes a syntactically (therefore uniformly) bounded number of steps to simulate one step of π_{bc} reduction.

3.3 Correspondence

Fact 3.3.1 (Structural Correspondence) $P \equiv Q$ *iff* $[\![P]\!] \equiv [\![Q]\!]$.

Proof Induction on the definition of structural equivalence in §2.2. All cases are trivial, due to the homomorphic action of our translation on 0, parallel composition and hiding. $\diamond$

In order to prove the faithfulness of our encoding, we define observational equivalences $\approx$ and $\approx_2$ that are appropriate for our purposes (implementation of one process by another).

Definition 3.3.2 (Bisimulation) *A binary relation* R *is a weak* simulation *if* $(P, Q) \in R$ *and* $P \overset{\mu}{\Rightarrow} P'$ *implies that there is a* Q' *such that* $Q \overset{\mu}{\Rightarrow} Q'$ *and* $(P', Q') \in R$. R *is a weak* bisimulation *if both* R *and* R^{-1} *are weak simulations. Two processes* P *and* Q *are weakly* bisimilar *(denoted* $P \approx Q$ *if there is a weak bisimulation* R *such that* $(P, Q) \in R$.

The study of bisimulations for π_{bc} and π_{ds} is outside the scope of the present paper, and we only state the following

Fact 3.3.3 $\approx$ *is a congruence on processes.*

Since our translations introduce two private auxiliary names in the beginning of every prefix, a modified version of bisimulation is appropriate. We define a function μ_2 on actions that throws away the first two names:

$$\begin{aligned}
(a!(b)?u!x_1\dagger_2 x_2 \ldots \dagger_n x_n)_2 &::= a!x_1\dagger_2 x_2 \ldots \dagger_n x_n \\
(a?u!(b)?x_1\dagger_2 x_2 \ldots \dagger_n x_n)_2 &::= a?x_1\dagger_2 x_2 \ldots \dagger_n x_n \\
(\tau)_2 &::= \tau
\end{aligned}$$

and μ_2 is undefined if μ is not of one of these forms.

Definition 3.3.4 (Bisimulation-up-to-shortening) *A binary (asymmetric) relation R is a weak* bisimulation-up-to-shortening *if whenever* $(P,Q) \in R$

1. *If* $P \overset{\mu}{\Rightarrow} P'$ *then there exist* Q' *and* μ' *such that* $\mu'_2 = \mu$ *and* $Q \overset{\mu'}{\Rightarrow} Q'$ *and* $(P',Q') \in R$.
2. *If* $Q \overset{\mu}{\Rightarrow} Q'$ *then there exists a* P' *such that* $P \overset{\mu_2}{\Rightarrow} P'$ *and* $(P',Q') \in R$.

Two processes P and Q are weakly bisimilar-up-to-shortening *(denoted $P \approx_2 Q$) if there is a weak bisimulation-up-to-shortening R such that $(P,Q) \in R$.*

Note that although shortening throws away information, we can recover it unambiguously up to α-renaming.

Lemma 3.3.5 *If* $\mu_2 = \mu'_2$ *then* $\mu =_\alpha \mu'$.

Proof If one of μ and μ' is τ then the other must also be τ. Else the types of the third objects of μ and μ' must be the same, because they are the same as the types of the first objects of μ_2 and μ'_2. Then also the types of the first two objects of μ and μ' must match, because $(\cdot)_2$ is undefined on other labels. Furthermore, these two objects will be private names, again due to the definition of $(\cdot)_2$. Thus we can make them the same in μ and μ' by α-renaming. The rest of μ and μ' match because they are the same as μ_2 and μ'_2 respectively. $\diamond$

Lemma 3.3.6 (Compositionality of $\approx_2$ etc) *Let* ${}_2{\approx} ::= (\approx_2)^{-1}$ *and* $\cdot$ *denote composition of relations. Then*

(1) $$\approx_2 \cdot \approx \;=\; \approx_2$$
(2) $$_2{\approx} \cdot \approx_2 \;=\; \approx$$
(3) $$_2{\approx} \cdot \approx \;=\; {}_2{\approx}$$
(4) $$\approx_2 \cdot {}_2{\approx} \;=\; \approx$$

Proof The proofs are similar, so we only do 1 and 2.

1. Assume that $P \approx_2 Q \approx R$. Let $P \overset{\mu}{\Rightarrow} P'$. Then there exist Q' and μ' such that $\mu'_2 = \mu$ and $Q \overset{\mu'}{\Rightarrow} Q'$ and $P' \approx_2 Q'$. Furthermore, there exists an R' such that $R \overset{\mu'}{\Rightarrow} R'$ and $Q' \approx R'$. We can continue this construction (from (P,Q,R) to (P',Q',R'), *etc*) indefinitely long.

   Conversely, let $R \overset{\mu}{\Rightarrow} R'$. Then there exists a Q' such that $Q \overset{\mu}{\Rightarrow} Q'$ and $Q' \approx R'$. Furthermore, there exist P' such that $P \overset{\mu_2}{\Rightarrow} P'$ and $P' \approx_2 Q'$. Again, we can continue the construction indefinitely. From this the result follows by coinduction.

2. Assume that $P \;{}_2{\approx}\; Q \approx_2 R$. Let $P \overset{\mu}{\Rightarrow} P'$. Then there exists a Q' such that $Q \overset{\mu_2}{\Rightarrow} Q'$ and $P' \;{}_2{\approx}\; Q'$. Furthermore, there exist R' and μ' such that $\mu'_2 = \mu_2$ and $R \overset{\mu'}{\Rightarrow} R'$ and $Q' \approx_2 R'$. By Lemma 3.3.5 $\mu' =_\alpha \mu$. We can continue this construction (from (P,Q,R) to (P',Q',R'), *etc*) indefinitely long.

   Conversely, let $R \overset{\mu}{\Rightarrow} R'$. Then there exists a Q' such that $Q \overset{\mu_2}{\Rightarrow} Q'$ and $Q' \approx_2 R'$. Furthermore, there exist P' and μ' such that $\mu'_2 = \mu_2$ and $P \overset{\mu'}{\Rightarrow} P'$ and $P' \;{}_2{\approx}\; Q'$. Again, $\mu' =_\alpha \mu$ and we can continue the construction indefinitely. From this the result follows by coinduction. $\diamond$

Theorem 3.3.7 (Operational Correspondence) $P \approx_2 [\![P]\!]$.

Proof We use the soundness and completeness lemmas in Appendix A. Assume that $P \approx_2 [\![P]\!]$. Let $P \overset{\mu}{\Rightarrow} P'$. By completeness $[\![P]\!] \overset{\mu'}{\Rightarrow} [\![P']\!]$ where $\mu'_2 = \mu$.

Now let $[\![P]\!] \overset{\mu}{\Rightarrow} P''$. By soundness there exists a P' such that $P'' \approx [\![P']\!]$ and $P \overset{\mu_2}{\Rightarrow} P'$. Now we use lemma 3.3.6.1. From this the result follows by coinduction. $\diamond$

One may wonder what is the relevance of the above result, since $\approx_2$ is an ad-hoc bisimulation. We argue that the result is significant, because for example it allows us to prove a result in terms of the standard bisimulation $\approx$.

Theorem 3.3.8 (Full Abstraction) *$P \approx Q$ iff $[\![P]\!] \approx [\![Q]\!]$.*

Proof

If Let $[\![P]\!] \approx [\![Q]\!]$. Then $P \approx_2 [\![P]\!] \approx [\![Q]\!] {}_2{\approx} Q$, and the result follows by 3.3.6.

Only if Let $P \approx Q$. Then $[\![P]\!] {}_2{\approx} P \approx Q \approx_2 [\![Q]\!]$, and the result follows by 3.3.6. $\diamond$

4 Discussion

I got the idea of implementing blocking prefix in a fragment without blocking from working on a translation of the π-calculus into a non-deterministic variant of Interaction Nets.[8] The idea of having chains of blocks (and therefore only one block per prefix and a linear complexity of the translation) transferred well from IN into the π-calculus setting, however it was not so easy to implement the blocks themselves in a block-free fragment.

One problem was the separation of control and data. I got the idea of using polyadic communication for this purpose from Parrow (1995). This work describes Interaction Diagrams, a graphical notation for the π-calculus. It works out traditional π-calculus examples, such as the implementation of the λ-calculus, without using blocking prefix,. It also puts forward the idea that provision and an additional "control" channel may be used to implement blocking.

Unfortunately, the sketch given in Parrow (1995, sec. 10) appears to be incorrect (or correct for only a limited setting). The author proposes to implement $a?\vec{x}.P$ by adding a control channel b to every output prefix $d!\vec{y}$ appearing in P, and making $a?\vec{x}$ instantiate these channels when it reduces. Input prefixes $d?\vec{y}$ also get an auxiliary channel, but it is not "hooked up" to $a?\vec{x}$, its purpose only being to match the control channel of $d!\vec{y}$. This indeed stops internal communications of $d?\vec{y}$, but it does not stop communications with the outside of P. Therefore the construction only works if all subjects in P are private channels. Nor will it be correct to block *all* interactions on d, because in $a?x.(d!y, d?y), d!z, d?z$, the prefixes $d\dagger y$ must wait, but $d\dagger z$ may proceed.

4.1 Why π_{bc}?

I tried to find an implementation in the polyadic π-calculus (without bidirectional communication), but failed. The problem is that while one can block an output $a!x$ by using an additional non-provisioned object $(a!bx)$, *an input $a?x$ is always ready to receive.*[9] Therefore $[\![a?x]\!]$ needs to send back any received messages until it is unblocked, as in the divergent encoding of choice in Nestmann and Pierce (1996).[10] After $[\![a?x]\!]$ is unblocked, it should consume the received message and proceed with unblocking its subordinated process. I was not able to express the sequencing needed for this scheme in a block- and choice-free fragment. The lock tests in Nestmann and Pierce (1996) use blocking prefix.

4.2 Limited Blocking

Parrow (1997) describes a translation of the π-calculus with replication (but no choice) into a fragment that allows only processes of a certain simple form (*concerts*): $(\vec{x})\prod(\pi_1.\pi_2.\pi_3)$. Sequences of three prefixes are called *trios*; trios may optionally be replicated in a concert. The main advantages of this approach compared to ours is its adherence to the π-calculus (no bidirectional communication is needed) and the simplicity of the forms (as opposed to the complexity of the translation). We see the following disadvantages:

[8] To be reported in my dissertation.

[9] Unless we make the subject a non-provisioned. However, a should be ready to serve as channel for other communications that are not in the scope of the blocking.

[10] We need replication here, because an indefinite number of outputs may attempt to send to $[\![a?x]\!]$ while it is blocked.

- The translation still uses blocking prefix (though only to a depth of 3).
- The translation is not homomorphic with respect to neither of parallel composition, 0 and restriction. For example, Palamidessi (1997) argues against non-uniform translations (ones that introduce additional machinery in parallel compositions).
- The translation does not preserve structural congruence.
- A central name server process is used, which is not a realistic assumption for a distributed implementation. This also necessitates the use of replication.

If we adopt the idea of limited blocking which is central to the previously mentioned work, we can implement distributed synchronization in a monadic π-calculus with provision, and using only *duos* $\pi_1.\pi_2$:

$$\begin{aligned} [\![a!x.P]\!] &::= b!(z'), b!(u'), b?z.a!(a'), a'!x.b?u, [\![P]\!], \quad [u/b_i]_{b_i \in \mathcal{B}[\![P]\!]}, \\ [\![a?x.Q]\!] &::= b!(z'), b!(u'), b?z.a?(a'), a'?x.b?u, [\![Q]\!], \quad [u/b_i]_{b_i \in \mathcal{B}[\![Q]\!]}. \end{aligned}$$

Here $[u/b_i]_{b_i \in \mathcal{B}[\![P]\!]}$ means to equate all blocking points of $[\![P]\!]$ (being the blocking points of its top-level prefixes) to the unblocking point of the prefix. (Some optimization is possible for prefixes that are not blocked and/or blocking.)

The construction works as follows: until b is instantiated, nothing can reduce, because $b\dagger$ are not provisioned, no communication is yet possible on the private channel a', and the block of $[\![P]\!]$ and $[\![Q]\!]$ (being u) is not provisioned. After b is provisioned and $b?z$ reduces, the private channel a' and then the datum x can be communicated. After that, u can be instantiated, which provisions the subordinated processes.

Note that this construction does not invalidate the claim in Parrow (1997) that duos are not sufficient to implement synchronization, because we still use non-blocking input prefix and the principle of provision.

4.3 Combinatory Representation

Our construction can easily be transformed into a combinatory representation for the finitary π-calculus, because it "implements out" the synchronization aspect of the prefix. As to the value aspect of the prefix, we find that there is no need to introduce duplication machinery to copy a received name to all occurrences of the bound variable. Instead, we find it more natural to keep all such occurrences merged, and to merge them to the received name, as in Interaction Diagrams, π-nets, and our non-deterministic Interaction Nets. This will be reported in more detail in my dissertation.

Honda and Yoshida (1994a,b) is another combinatory representation of π, but we find that it is too complicated and unnatural. Their combinators correspond to prefixes and more complex constructs, instead of simply names. They use forwarders instead of merging channel names, which introduces unnecessary bureaucracy. They introduce blocking machinery for every combinator in a process, instead of blocking the process in "layers", which renders their translation exponential. Finally, it is not at all clear how to read back a process from a net of combinators.

4.4 Future Work

Certain trade-offs relating to synchronization have emerged. Please note that bidirectional communication can be introduced as a derived construct in the π-calculus similarly to polyadic communication, by following the technique of Honda and Tokoro (1991).[11] However, this requires sequentialization (blocking).

One can implement distributed synchronization using one of:

- Polyadic π, replication and trios (Parrow, 1997).
- Monadic π, provision and duos (sketched above).

[11] Which roughly consists of transmitting the names sequentially instead of in parallel.

- Bidirectional π and provision (this paper).
- Combinator/graph systems (Honda and Yoshida, 1994a).

A challenging problem is to find a framework that unifies these approaches and explains the tradeoffs systematically.

It is also interesting to integrate some of these approaches to synchronization with approaches that implement away other π constructs, for example the choice encodings of Nestmann (1997). We conjecture that π_{bc} does not suffer from the inability to break symmetries that make the ν calculus unable to capture mixed-prefix choice (Palamidessi, 1997).

A Proof of Soundness and Completeness

Lemma A.0.1 (Completeness) *Let $P \xrightarrow{\mu} P'$. Then there exists a μ' such that $\mu'_2 = \mu$ and $[\![P]\!] \overset{\mu'}{\Rightarrow} [\![P']\!]$.*

Proof Induction on the definition of $P \xrightarrow{\mu} P'$.

Prefix The case $\pi, P \xrightarrow{\pi} P$ is easy: $[\![\pi]\!], [\![P]\!] \xrightarrow{[\![\pi]\!]} [\![P]\!]$ by Prefix (or Prefix and Par if we regard the LHS as parallel composition). $[\![\pi]\!]_2 = \pi$ by observation.

The case $\pi.P \xrightarrow{\pi} P$ involves silent actions: $[\![\pi]\!], \prod_{c\in\mathcal{B}[\![P]\!]}(u?c, u!(c')), [\![P]\!]' \xrightarrow{[\![\pi]\!]} \prod_{c\in\mathcal{B}[\![P]\!]}(u?c, u!(c')), [\![P]\!]'$ $\xrightarrow{\tau}^{n} [\![P]\!][\vec{b'}/\vec{b}] \equiv_\alpha [\![P]\!]$.

Struc Let $P' \xrightarrow{\mu} Q'$ due to $P \xrightarrow{\mu} Q$ and $P \equiv Q$, $P' \equiv Q'$. Let also assume that $P' \approx_2 [\![P']\!]$. We have to prove that $Q' \approx_2 [\![Q']\!]$, but this follows immediately from 3.3.1.

Alpha Again follows from 3.3.1 and the observation that $\mu[x/y] = \mu'[x/y]_2$ implies that $\mu = \mu'_2$.

Par Follows immediately from the fact that the translation is homomorphic.

Res Same.

Scope Same, with the additional observation that if $(a!(b)?u!v\dagger\vec{v})_2 = a!v\dagger\vec{v}$ then $(a!(b)?u!(v)\dagger\vec{v})_2 = a!(v)\dagger\vec{v}$.

Comm Let $P, Q \xrightarrow{\tau} (\vec{b})(P', Q')$ due to $P \xrightarrow{a\dagger\vec{x}} P'$ and $Q \xrightarrow{\bar{a}\bar{\dagger}\vec{x}} Q'$. We have to prove $[\![P, Q]\!] \overset{\tau}{\Rightarrow} [\![(\vec{b})(P', Q')]\!]$. From the induction hypothesis on the premises, $[\![P]\!] \overset{a!(b)?u\dagger\vec{x}}{\Rightarrow} [\![P']\!]$ and $[\![Q]\!] \overset{a?u!(b)\bar{\dagger}\vec{x}}{\Rightarrow} [\![Q']\!]$.[12] By an application of Comm, $[\![P]\!], [\![Q]\!] \overset{\tau}{\Rightarrow} (b, u, \vec{b})([\![P']\!], [\![Q']\!])$, from which the claim follows by $[\![(\vec{b})(P', Q')]\!] \equiv [\![(b, u, \vec{b})(P', Q')]\!]$, Struc, and the homomorphism of the translation. $\diamond$

As usual for implementation encodings, soundness is harder to prove than completeness, because we cannot limit consideration to single transition steps of the implementing process. But for our translation, the proof is relatively easy, because the auxiliary steps are confluent.

Lemma A.0.2 (Soundness) *Let $[\![P]\!] \overset{\mu}{\Rightarrow} P''$. Then there exists a P' such that $P'' \approx [\![P']\!]$ and $P \xrightarrow{\mu_2} P'$.*

Proof [Sketch] Induction on the structure of $[\![P]\!]$. The homomorphic cases are trivial, so we are left with blocking prefix (refer to §3.1). The only transition of $[\![\pi.P]\!]$ is $[\![\pi.P]\!] \xrightarrow{[\![\pi]\!]} P''$ where $P'' = \prod_{c\in\mathcal{B}[\![P]\!]}(u?c, u!(c')), [\![P]\!]'$, because u in $(u?c, u!(c')$ is not yet instantiated, and all parallel components of $[\![P]\!]'$ are dependent on c (from which by induction follows that nothing in $[\![P]\!]'$ can reduce). There is a corresponding transition $\pi_2.P \xrightarrow{\pi_2} P$.

We now have to prove that $P'' \approx [\![P]\!]$. Obviously $P'' \Rightarrow [\![P]\!]$, by reduction of the $(u?c, u!(c'))$ components. Furthermore, $\xrightarrow{u?c}$ and $\xrightarrow{u!(c')}$ are all the single-step transitions of P'' ($[\![P]\!]'$ cannot reduce), and every one can be completed to a τ transition by using its complementary action. $\diamond$

[12] Or with $?u$ and $!(b)$ swapped, depending on the type of the first object of the action.

References

Kohei Honda and Mario Tokoro, 1991. An object calculus for asynchronous communication. In Pierre America, ed., *European Conference on Object-Oriented Programming (ECOOP'91)*, vol. 512 of *LNCS*. Geneva.

Kohei Honda and Nobuko Yoshida, 1994a. Combinatory representation of mobile processes. In *Principles of Programming Languages (POPL'94)*, pp. 348–360. Portland, Oregon. ISBN 0-89791-636-0.

Kohei Honda and Nobuko Yoshida, 1994b. Replication in concurrent combinators. In M. Hagiya and J. C. Mitchell, eds., *Theoretical Aspects of Computing Science (TACS'94)*, no. 789 in LNCS, pp. 786–805. Sendai, Japan.

Robin Milner, 1993a. An action structure for the synchronous π-calculus. In Z. Esik, ed., *Fundamentals of Computation Theory (FCT'93)*, no. 710 in LNCS, pp. 87–105. Szeged, Hungary.

Robin Milner, 1993b. The polyadic π-calculus: a tutorial. In F. L. Bauer (or Hamer?), W. Brauer and H. Schwichtenberg, eds., *Logic and Algebra of Specification*, p. 50. Also University of Edinburgh TR ECS-LFCS-91-180.

Robin Milner, Joachim Parrow *et al.*, 1992. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77.

Uwe Nestmann, 1997. What is a 'good' encoding of guarded choice? In Catuscia Palamidessi and Joachim Parrow, eds., *4th Workshop on Expressiveness in Concurrency (EXPRESS'97)*, vol. 7 of *ENTCS*. Santa Margherita Ligure, Italy.

Uwe Nestmann and Benjamin C. Pierce, 1996. Decoding choice encodings (extended abstract). In *Intl. Conf. on Concurrency Theory (CONCUR'96)*, no. 1119 in LNCS, pp. 179–194. Full version is ERCIM Research Report 10/97-R051, August 1997, 51pp.

Catuscia Palamidessi, 1997. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *Principles of Programming Languages (POPL'97)*, pp. 256–265.

Joachim Parrow, 1995. Interaction diagrams. *Nordic Journal of Computing*, (2):407–443. Earlier version appeared in *A Decade of Concurrency: Reflections and Perspectives. REX School and Symposium*, J.W. de Bakker, W.-P. de Roever and G. Rozenberg (ed), June 1993, LNCS 803; and as SICS Research report R93:06.

Joachim Parrow, 1997. Trios in concert. In G. Plotkin, C. Stirling and M. Tofte, eds., *Proof, Language and Interaction: a Festschrift in Honour of Robin Milner*. MIT Press. To appear.