Electronic Notes in Theoretical Computer Science 16 No. 2 (1998)
URL: http://www.elsevier.nl/locate/entcs/volume16.html 8 pages

On the Expressive Power of Joint Input ⋆

Uwe Nestmann

BRICS: Basic Research in Computer Science
Centre of the Danish National Research Foundation
Aalborg University, Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø

Abstract

The join-calculus was introduced as an 'extended subset' of the asynchronous π-calculus by amalgamating the three operators for input, restriction, and replication into a single operator, called *definition*, but with the additional capability to describe the atomic *joint* reception of values from two different channels. In this paper, we just extend the asynchronous π-calculus with *joint input*. By studying its expressive power, using slight variations of previously investigated choice encodings, we also conclude on the expressiveness of the join-calculus.

1 Join-calculus

Fournet and Gonthier [FG96] introduced the join-calculus as an 'extended subset' of the asynchronous π-calculus, making the latter—offering an attractive basis for concurrent programming [PT98]—more amenable to distributed implementations. Syntactically, the amalgamation of the three operators input, restriction, and replication from the π-calculus into the definition operator

$$\mathsf{def}\ u\langle v\rangle = P\ \mathsf{in}\ Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} u)\,(\,!\,u(v).P \mid Q\,) \tag{1}$$

and its extension to admit joint inputs leads to the pleasantly simple grammar:

$$\begin{array}{rcl} P,Q & ::= & u\langle v\rangle \;\mid\; P|Q \;\mid\; \mathsf{def}\ D\ \mathsf{in}\ Q \\ D & ::= & u\langle v\rangle|w\langle x\rangle = P \end{array}$$

where D represents the *joint definition* with u, w called the defined names and v, x called the received names, which are all accessible from within P. The construct $\mathsf{def}\ D\ \mathsf{in}\ Q$ represents the amalgamated operator that limits the further scope of the defined names of D to Q (see also Equation 1).

⋆ Started under an ERCIM post-doc fellowship at INRIA Rocquencourt, supported by the ESPRIT CONFER-2 Working Group 21836, and continued under a BRICS post-doc grant.

The *full* join-calculus (fJC) differs from the *core* join-calculus (cJC) in allowing n-ary joins, conjoint definitions, and polyadic channels [FG96,Fou98]. Encodings for these features into the core have been presented in the above references showing that the full calculus is not more expressive than the core. By syntactic construction—and in constrast to asynchronous π-calculus—the join-calculus guarantees *locality* **(L)** and *uniformity* **(U)** of defined names:

locality (1) Defined names are activated at fixed locations within a process context: since the definition operator behaves like restriction for the defined names, it does not tolerate other definitions for the same name in parallel, so defined names in cJC are even unique. Amadio studied *unique receptors* explicitly within the (also asynchronous) π_1-calculus by means of a simple static type system [Ama98]. (2) Reception on a name can only happen in definitions, so the receiving channel (the defined name) is always freshly generated. Thus, it is not possible in JC to receive on received names, which would amount to re-define them. Consequently, this disciplined use of names (called *inversion of polarity* by Boreale [Bor98]) keeps stable the location of defined names during computation; it has been studied explicitly in the *Local* π-calculus (Lπ) [MS98] by Merro and Sangiorgi.

uniformity The behavior of a defined name, i.e. the continuation process P of D in the above grammar, is always the same, when triggered, because defined names are unique and replicated. In constrast to Amadio's unique receptors [Ama98], uniform receptors always behave like functions [Fou98]. *Uniform receptiveness* (unlike Palamidessi's 'uniform encodings' [Pal97]) was studied explicitly in the context of the π-calculus by Sangiorgi [San97].

In fJC, locality and uniformity are slightly relaxed since in conjoint definitions, as in $\mathsf{def}\ D_1 \wedge \ldots \wedge D_n\ \mathsf{in}\ Q$, the defined names may be shared among the 'parallel' D_i. Although uniqueness of defined names is (syntactically) lost that way, locality is still assured since no process outside the definition construct, within the defined processes P_i, or within process Q, can ever supply further receptors for these defined names.[1] Similarly, uniformity of defined names holds in the sense that the set of possible behaviors of a name is fixed from the beginning and never changed during computation.

As the main advantage, locality and uniformity enable simpler distributed implementations of the join-calculus compared to the π-calculus: the question of where to install the receptors, aka: functions, is answered syntactically.

Since, up to now, we did not mention the aspect of joint inputs at all, why do we need the join in the join-calculus? If we only had the single-input definition suggested in Equation 1, then no synchronization between parallel

[1] In implementations, as mimicked by the encoding of full definitions into core definitions, a local automaton can be supplied that correctly manages the possible sharing of defined names among conjoint definitions, even in a divergence-free manner [Fou98]. So, the uniqueness of defined names can be recovered.

components would ever be possible. For example, with D being just $u\langle v\rangle = P$, the processes Q_1 and Q_2 in $\mathsf{def}\ D\ \mathsf{in}\ Q_1|Q_2$ can never synchronize with each other in any sensible way, because they are only separately able to send on their shared defined name u. Here, the following behavioral equality holds:

$$\mathsf{def}\ D\ \mathsf{in}\ Q_1|Q_2 \quad \approx \quad \mathsf{def}\ D\ \mathsf{in}\ Q_1 \mid \mathsf{def}\ D\ \mathsf{in}\ Q_2 \tag{2}$$

By contrast, with a joint definition D given as $u_1\langle v\rangle|u_2\langle x\rangle = P$, the processes Q_1 and Q_2 can synchronize by sending to u_1 and u_2, respectively, where the joint availability of these signals can be detected and checked by both processes via further signals eventually available from P. Apparently, joint input contributes considerably to the expressive power of JC (see §3).

The operational semantics of JC was originally presented using the framework of reflexive chemical abstract machines [FG96], but one can also resort to a simple reduction semantics based on a straightforwardly defined structural congruence relation $\equiv$, according to Fournet [Fou98]. With J abbreviating a join pattern $u_1\langle v\rangle|u_2\langle x\rangle$, the main (simplified) reduction rule of JC is:

$$\mathsf{def}\ J = P\ \mathsf{in}\ J\sigma|Q \quad \to \quad \mathsf{def}\ J = P\ \mathsf{in}\ P\sigma|Q$$

Whenever an instantiation $J\sigma$ of a definition's join-pattern J can be found in top-level of the scope of the definition, then reduction may replace this instance with the corresponding instantiation $P\sigma$ of the defined continuation P. For the full semantics, we refer to Fournet [Fou98].

2 Asynchronous π-calculus with joint input

We introduce a family of polyadic π-calculi that only differ in their sequential components, which determine the respective synchronization capabilities.

In the following, let $\mathbf{N}$ be a countable set of *names*, typically u, v, w, x, y, or z, and let $\tilde{x}$ denote a finite tuple $x_1, \cdots, x_n$ of names. The languages π_Σ with $\Sigma \in \{\mathrm{a}, \mathrm{j}, \mathrm{mix}\}$ are then defined as shown in Figure 1, where I ranges over finite sets of indices i. Apart from the (mis)match operator, π_a is the usual asynchronous π-calculus [ACS98], whereas π_mix is the standard π-calculus, but with (mixed-) guarded choice [Mil93]. The language π_j is new: to asynchronous output, it adds a *joint input* prefix, where synchronization takes place atomically on two different channels, so in the grammar we require that $u \neq w$ and the $\tilde{v}, \tilde{x}$ be pairwise distinct. We abbreviate $\overline{x}\langle\rangle$ to $\overline{x}$, and $x()$ to x, respectively. Inaction $\mathbf{0}$ is derived as $(\boldsymbol{\nu} a)\,\overline{a}$ or $\sum_{i\in\emptyset}$. Single input is derived using dummy signals, as in $y(\tilde{x}).P := (\boldsymbol{\nu} u)(\overline{u}|\{u|y(\tilde{x})\}.P)$ for the non-replicated, and in $!\,y(\tilde{x}).P := (\boldsymbol{\nu} u)(\overline{u}|!\,\{u|y(\tilde{x})\}.(\overline{u}|P))$ for the replicated case.

For the sake of readability, we also use primitive booleans t and f, equipped with conjunction $\wedge$, negation $\neg$, and a conditional $\mathsf{if}\ x\ \mathsf{then}\ P\ \mathsf{else}\ Q$ with the obvious meaning [NP96]. The behaviour can be incorporated into a structural congruence relation ($\equiv$) by $\mathsf{if}\ \mathsf{t}\ \mathsf{then}\ P\ \mathsf{else}\ Q \equiv P$ and $\mathsf{if}\ \mathsf{f}\ \mathsf{then}\ P\ \mathsf{else}\ Q \equiv Q$.

$$
\begin{array}{lcl}
P, Q & ::= & P|Q \;\mid\; (\boldsymbol{\nu} y)\, P \;\mid\; [x = y]P, Q \;\mid\; N_\Sigma \\
N_a & ::= & \overline{y}\langle \tilde{z} \rangle \;\mid\; y(\tilde{x}).P \;\mid\; !\, y(\tilde{x})\,.\,P \\
N_{\mathrm{j}} & ::= & \overline{y}\langle \tilde{z} \rangle \;\mid\; \{u(\tilde{v})\,|\,w(\tilde{x})\}\,.\,P \;\mid\; !\,\{u(\tilde{v})\,|\,w(\tilde{x})\}\,.\,P \\
N_{\mathrm{mix}} & ::= & \sum_{i \in I} \alpha_i.P_i \;\mid\; !\, y(\tilde{x})\,.\,P \\
\alpha & ::= & \overline{y}\langle \tilde{z} \rangle \;\mid\; y(\tilde{x})
\end{array}
$$

Fig. 1. π-calculus syntax

As usual, we may give a reduction semantics for the π_Σ, where the main reduction rule for π_{j} describes the atomic consumption of two messages:

$$\overline{u}\langle \tilde{y} \rangle \mid \overline{w}\langle \tilde{z} \rangle \mid \{u(\tilde{v})\,|\,w(\tilde{x})\}\,.\,P \quad \rightarrow \quad P\{^{\tilde{y}}/_{\tilde{v}}\}\{^{\tilde{z}}/_{\tilde{x}}\}$$

In the corresponding rule for replicated input, the joint input prefix persists. The behavior of mismatch can be incorporated into the usual structural congruence rules like $[x = y]P, Q \equiv P$ for $x = y$, and $[x = y]P, Q \equiv Q$ for $x \neq y$. For the further semantics of the π_Σ, we refer to [Nes97,ACS98].

3 Expressiveness

Fournet and Gonthier [FG96,Fou98] proved that cJC and π_{a} are equally expressive by providing a pair of fully-abstract mutual encodings. However, only the direction cJC $\rightarrow \pi_{\mathrm{a}}$ is obviously compositional (see: cJC $\rightarrow$ Lπ [MS98]).

Palamidessi has shown that π_{mix} is strictly more expressive than π_{a}: there is no compositional encoding from π_{mix} into π_{a} that preserves a 'reasonable' semantics [Pal97]. Here, we show that π_{j}, the extension of π_{a} with joint input is expressive enough as a target for such a compositional encoding of π_{mix}. The main problems of encoding mixed-guarded choice have been outlined by Nestmann [Nes97] as caused by the unwanted possibility of deadlock due to

(i) symmetric cyclic dependencies, as in $\overline{y_0}.P_0 + y_1.P_1 \mid y_0.Q_0 + \overline{y_1}.Q_1$, and
(ii) incestuous requests, as in $\overline{y}.P + y.Q$ (with $+$ denoting binary choice).

These problems cannot be coped with in π_{a}, except by either invalidating compositionality or by admitting severe possibilities of divergence. However, as we will see in the following paragraphs, the availability of joint input gives us further expressive power to attack the problem.

First attempt
In Figures 2 and 3, we present a direct adaptation of Nestmann's encodings from $\pi_{\mathrm{mix}} \rightarrow \pi_{\mathrm{a}}$ to the setting $\pi_{\mathrm{mix}} \rightarrow \pi_{\mathrm{j}}$, where we now take advantage of joint input. For the deeper understanding of the algorithm, we refer to [Nes97].

$$
\begin{array}{lcl}
[\![\, P_1 \mid P_2 \,]\!] & \stackrel{\text{def}}{=} & [\![\, P_1 \,]\!] \mid [\![\, P_2 \,]\!] \\
[\![\, (\boldsymbol{\nu} x)\, P \,]\!] & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x)\, [\![\, P \,]\!] \\
[\![\, [x = y] P, Q \,]\!] & \stackrel{\text{def}}{=} & [x = y] [\![\, P \,]\!], [\![\, Q \,]\!]
\end{array}
$$

Fig. 2. Encoding $\pi_{\text{mix}} \to \pi_{\text{j}}$ (base)

To prevent deadlocks of type (i) in π_{a}, the so-called *locks* l and r either had to be tested separately according to some globally defined order, or it was needed to enable undoing of tests, which opened up for inherently uncontrollable divergent behavior. Joint input allows us to express the check of both locks atomically, so it avoids deadlocks directly.

To deal with deadlocks of type (ii), we use the mismatch operator, which is checking the locks' identity and in the positive case just resends the requests and restarts the receiver. This still results in possible divergence, but it is less harmful since it is easily ruled out in implementations: for unique receptors, incestuous requests are thrown away immediately; for channels with several receptors, unique channel managers processes store incoming requests in queues and only consider non-incestuous requests for entering the protocol.

Unfortunately, the encoding of Figure 3 invalidates both properties—locality and uniformity—that are valid for joint input as in JC, when using locks. Locality **(L)**: as soon as the implementation of an input-branch receives a request, it dynamically installs a new receptor on the received sender-lock r. Uniformity **(U)**: as soon as there are at least two input-branches in a choice, there will be potentially different behaviors for the receiver-lock's (l) continuation processes; they are just inherited from the source term. Obviously, the handling of locks in this encoding is not optimal with respect to the properties as required in JC. Could we have done better?

$$
\begin{array}{lcl}
\left[\!\!\left[\sum_{i \in I} \pi_i.P_i \right]\!\!\right] & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} l) \left(\; \overline{l}\langle \mathsf{t} \rangle \;\mid\; \prod_{i \in I} [\![\, \pi_i.P_i \,]\!]_l \; \right) \\
[\![\, \overline{y}\langle \tilde{z} \rangle.P \,]\!]_r & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} a) \left(\, \overline{y}\langle r, a, \tilde{z} \rangle \mid a.[\![\, P \,]\!] \, \right) \\
[\![\, y(\tilde{x}).P \,]\!]_l & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} b) \Big(\; \overline{b} \mid \; ! \;\; b \,.\, y(r, a, \tilde{x}) \,.\, [l = r] \; (\, \overline{b} \mid \overline{y}\langle r, a, \tilde{x} \rangle \,) \;, \\
& & \qquad \{ l(b_l) | r(b_r) \} \,. \\
& & \qquad \mathsf{if} \;\; b_l \wedge \;\; b_r \mathsf{then}\; \overline{l}\langle \mathsf{f} \rangle \mid \overline{r}\langle \mathsf{f} \rangle \mid \overline{a} \mid [\![\, P \,]\!] \;\; \mathsf{else} \\
& & \qquad \mathsf{if} \;\; b_l \wedge \neg b_r \mathsf{then}\; \overline{l}\langle \mathsf{t} \rangle \mid \overline{r}\langle \mathsf{f} \rangle \mid \overline{b} \;\; \mathsf{else} \\
& & \qquad \mathsf{if} \neg b_l \wedge \;\; b_r \mathsf{then}\; \overline{l}\langle \mathsf{f} \rangle \mid \overline{r}\langle \mathsf{t} \rangle \mid \overline{y}\langle r, a, \tilde{x} \rangle \;\; \mathsf{else} \\
& & \qquad \mathsf{if} \neg b_l \wedge \neg b_r \mathsf{then}\; \overline{l}\langle \mathsf{f} \rangle \mid \overline{r}\langle \mathsf{f} \rangle \;\; \mathsf{else}\; \mathbf{0} \; \Big) \\
[\![\, ! \, y(\tilde{x}).P \,]\!] & \stackrel{\text{def}}{=} & ! \, y(r, a, \tilde{x}) \,.\, r(b) \,.\, \mathsf{if}\; b \;\mathsf{then}\; \overline{r}\langle \mathsf{f} \rangle \mid \overline{a} \mid [\![\, P \,]\!] \;\mathsf{else}\; \overline{r}\langle \mathsf{f} \rangle
\end{array}
$$

Fig. 3. Encoding $\pi_{\text{mix}} \to \pi_{\text{j}}$

Second attempt
It turns out that we can do better, if we invert the direction of communications for interrogating lock values: since locks behave like mutex channels, there is always at most one message available for a given lock. By applying a standard encoding trick for booleans, the inversion of directionality can be easily done and, moreover, allows us to use unique (local) lock receptors.

In Figure 4, we give a respective adaptation of the previous encoding. The behavior of the former lock l is now centralized such that it better matches the locality properties: l is split into three names $\tilde{l}{=}l_{\mathrm{c}}, l_{\mathrm{t}}, l_{\mathrm{f}}$ for checking the lock (by sending two fresh names to l_{c}) and setting it to true ($\overline{l_{\mathrm{t}}}$), e.g. as its initial 'value', or false ($\overline{l_{\mathrm{f}}}$), respectively. Note that the single joint input of the encoding in Figure 3 now splits up into four parallel (conjoint) joint inputs—one for each combination of received boolean values. Furthermore, the invariant of the former encoding that for a given lock l "at any time there is at most one message available on l" is now rephrased to: "at any time there is at most one receptor for l_{c}"—and these receptors are even local.

In Figure 4, all channels that are introduced by the encoding are local **(L)**, referring to the relaxed notion of locality for fJC: the conjoint composition of joint inputs behaves just like conjoint definitions in fJC. However, it is crucial for this encoding that some names are allowed to behave non-uniformly, thus invalidating property **(U)**: the lock's check channels l_{c} must initially (and at most once) reply on the received name t, and only after their first reply they then behave uniformly by always replying on the received name f.

Theorem 3.1 *Both encodings $\pi_{\mathrm{mix}} \to \pi_{\mathrm{j}}$ are compositional and deadlock-free.*

Proof. Compositionality is trivial with Figure 2. Deadlock-freedom follows from a rather restricted full-abstraction result (cf. [Nes97]), where we optimize and only consider source terms, where communication *on selectable channels* is always restricted and communication *of selectable channels* is forbidden. □

$$
\begin{array}{lcl}
\left[\!\left[\sum_{i\in I} \pi_i.P_i \right]\!\right] & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} l_{\mathrm{c}}, l_{\mathrm{t}}, l_{\mathrm{f}}) \left(\; !\, l_{\mathrm{t}}.l_{\mathrm{c}}(t,f).\overline{t} \mid !\, l_{\mathrm{f}}.l_{\mathrm{c}}(t,f).\overline{f} \mid \overline{l_{\mathrm{t}}} \mid \prod_{i\in I} [\![\, \pi_i.P_i \,]\!]_{\tilde{l}} \right) \\
[\![\, \overline{y}\langle\tilde{z}\rangle.P \,]\!]_{\tilde{r}} & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} a)\, (\, \overline{y}\langle\tilde{r}, a, \tilde{z}\rangle \mid a.[\![\, P \,]\!]\,) \\
[\![\, y(\tilde{x}).P \,]\!]_{\tilde{l}} & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} b) \Big(\; \overline{b} \mid !\;\; b\,.\, y(\tilde{r}, a, \tilde{x})\,.\,[l_{\mathrm{c}} = r_{\mathrm{c}}]\; (\, \overline{b} \,|\, \overline{y}\langle\tilde{r}, a, \tilde{x}\rangle\,)\;, \\
 & & \qquad (\boldsymbol{\nu} t_l, f_l)\, (\boldsymbol{\nu} t_r, f_r)\, (\overline{l_{\mathrm{c}}}\langle t_l, f_l\rangle \,|\, \overline{r_{\mathrm{c}}}\langle t_r, f_r\rangle \\
 & & \qquad |\{t_l|t_r\}\,.\,(\overline{l_{\mathrm{f}}}|\overline{r_{\mathrm{f}}}|\overline{a}|[\![\, P \,]\!]) \\
 & & \qquad |\{t_l|f_r\}\,.\,(\overline{l_{\mathrm{t}}}|\overline{r_{\mathrm{f}}}|\overline{b}) \\
 & & \qquad |\{f_l|t_r\}\,.\,(\overline{l_{\mathrm{f}}}|\overline{r_{\mathrm{t}}}|\overline{y}\langle r, a, \tilde{x}\rangle) \\
 & & \qquad |\{f_l|f_r\}.\,(\overline{l_{\mathrm{f}}}|\overline{r_{\mathrm{f}}})\;) \Big) \\
[\![\, !\, y(\tilde{x}).P \,]\!] & \stackrel{\text{def}}{=} & !\, y(\tilde{r}, a, \tilde{x})\;.\; (\boldsymbol{\nu} t, f)\, (\, \overline{r_{\mathrm{c}}}\langle t, f\rangle \mid t.(\overline{r_{\mathrm{f}}}|\overline{a}|[\![\, P \,]\!]) \mid f.\overline{r_{\mathrm{f}}}\,)
\end{array}
$$

Fig. 4. Encoding (L)$\pi_{\mathrm{mix}} \to$ (L)π_{j}

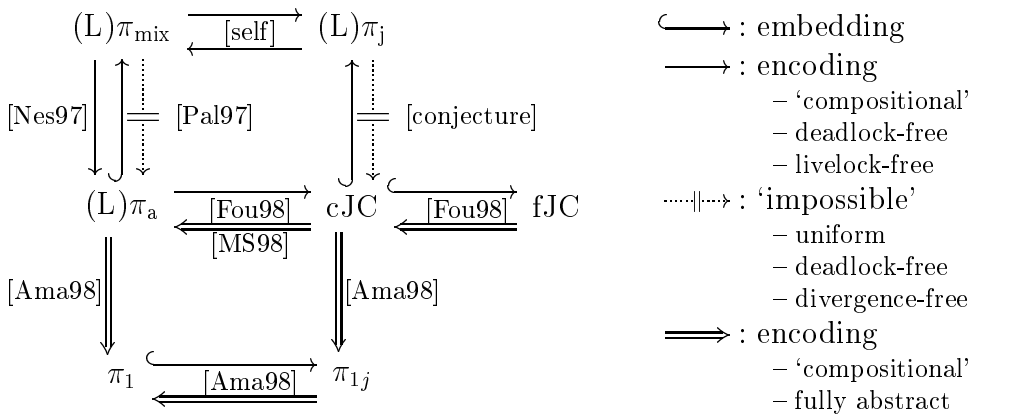

Fig. 5. Some join- and π-calculi

Comparing the expressive power
In Figure 5, we have intentionally simplified the relations among the calculi, e.g., we are not very strict about polyadicity or compositionality, and the indicated full-abstraction results do sometimes only hold for restricted source terms. The various (L)π (cf. [Bor98,MS98]) indicate encodings that do not depend on the target language admitting non-local names—in fact, it seems that one can always reprogram non-local names with local ones, like in Figure 4.

The caluli π_{mix} and π_{j} appear at the same (top) level of expressiveness, witnessed by a pair of mutual encodings. In fact, in this document we have just shown one direction, but the opposite is not difficult, either: implementing joint input with guarded choices can be done by successively reading messages or—here we need mixed guards—giving back previously read messages until enough messages have been read in order to trigger a continuation. Thus, such an encoding resembles the one from fJC into cJC, except that we cannot exploit JC's locality property, such that we always need to consider that there might be other external receptors competing with the current local ones. Therefore, the encoding $\pi_{\text{j}} \to \pi_{\text{mix}}$ (necessarily) introduces divergence.

Note that π_1 and π_{1j} (its extension with *filtered joined input* [Ama98]) are not subsets of Lπ_{a} and cJC, but only of π_{a} and π_{j}, respectively, because unique receptors do neither necessarily obey the L-discipline, nor are they necessarily uniform. On the other hand, in comparison to locality in Lπ_{j}, the uniqueness of names in π_{1j} seems to prevent us from expressing Lπ_{j} in π_{1j}.

4 Conclusion

Since joint input, as a language primitive, is thought of as being too expressive, also the join-calculus is sometimes considered too far-reaching, even if Fournet and Gonthier have shown that JC is not more expressive than π_{a}. The current document underlines the expressive power of unconstrained joint input by providing a succinct deadlock-free encoding of π_{mix} with mixed-guarded choice

into π_j, the asynchronous π-calculus enhanced with joint input. Yet, this study suggests that joint input only overshoots the mark (and maybe spoils the ease of implementability), when paired with non-uniformity to undermine the functional behavior of names; non-locality does not have the same impact. However, within JC, the expressive power of joint input is adequately tamed.

References

[ACS98] R. M. Amadio, I. Castellani and D. Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[Ama98] R. M. Amadio. On modelling mobility. A revised extended version of "An Asynchronous Model of Locality, Failure, and Process Mobility", in *Proceedings of COORDINATION '97*, volume 1282 of *LNCS*. June 1998.

[Bor98] M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science*, 195(2):205–226, 1998.

[FG96] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *Proceedings of POPL '96*, pages 372–385. 1996.

[Fou98] C. Fournet. *The Join-Calculus: A Calculus for Distributed Mobile Programming*. PhD thesis, INRIA Rocquencourt, 1998. To appear.

[Mil93] R. Milner. The Polyadic π-Calculus: A Tutorial. In F. L. Bauer, W. Brauer and H. Schwichtenberg, eds, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

[MS98] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In K. G. Larsen, S. Skyum and G. Winskel, eds, *Proceedings of ICALP '98*, volume 1443 of *LNCS*, pages 856–867. Springer, July 1998.

[Nes97] U. Nestmann. What Is a 'Good' Encoding of Guarded Choice? In C. Palamidessi and J. Parrow, eds, *Proceedings of EXPRESS '97*, volume 7 of *ENTCS*. Elsevier Science Publishers, 1997.

[NP96] U. Nestmann and B. C. Pierce. Decoding Choice Encodings. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 179–194. Springer, 1996.

[Pal97] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, Jan. 1997.

[PT98] B. C. Pierce and D. N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. To appear.

[San97] D. Sangiorgi. The Name Discipline of Uniform Receptiveness. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, eds, *Proceedings of ICALP '97*, volume 1256 of *LNCS*, pages 303–313. Springer, 1997.