Is It a "Good" Encoding of Mixed Choice?*

Kirstin Peters and Uwe Nestmann

Technische Universität Berlin, Germany

Abstract. Mixed choice is a widely-used primitive in process calculi. It is interesting, as it allows to break symmetries in distributed process networks. We present an encoding of mixed choice in the context of the π-calculus and investigate to what extent it can be considered "good". As a crucial novelty, we introduce a suitable criterion to measure whether the degree of distribution in process networks is preserved.

1 Introduction

It is well-known [Pal03, Gor10, PN10] that there is no good encoding from the full π-calculus—the synchronous π-calculus including mixed choice—into its asynchronous variant if the encoding translates the parallel operator rigidly (a criterion included in *uniformity* in [Pal03]). Palamidessi was the first to point out that mixed choice strictly raises the absolute expressive power of the synchronous π-calculus compared to its asynchronous variant. Analysing this result [PN10], we observed that it boils down to the fact that the full π-calculus can break merely syntactic symmetries, where its asynchronous variant can not. However, the condition of rigid translation of the parallel operator is rather strict. Therefore, Gorla proposed the weaker criterion of compositional translation of the source language operators (see Definition 4 at page 214). We show that this weakening of the structural condition on the encoding of the parallel operator turns the separation result into an encodability result, by presenting a good encoding of mixed choice[1]. So, merely considering the (abstract) behaviour of terms, the full π-calculus and its asynchronous variant have the same expressive power.

The situation changes again if we additionally take into account the *degree of distribution*. In the area of distributed communicating systems it is natural to consider distributed algorithms, that perform at least some of their tasks concurrently. Thus, an answer to the question whether an encoding preserves the degree of distribution of the original algorithm, becomes important. In order to measure the preservation of the degree of distribution we introduce a novel but intuitive (semantic) criterion, which is strictly weaker than the (syntactic) requirement of rigid translation of the parallel operator. Using this criterion in addition to the criteria presented in [Gor10], we again obtain, as expected, a separation result by showing that there is no good encoding of mixed choice that preserves the degree of distribution of source terms.

* Supported by the DFG (German Research Foundation), grant NE-1505/2-1.

[1] Note that this encoding is neither prompt nor is the assumed equivalence $\asymp$ strict, so the similar separation results of [Gor08] and [Gor10] do not apply here.

L. Birkedal (Ed.): FOSSACS 2012, LNCS 7213, pp. 210–224, 2012.

Overview of the Paper. In Section 2, we introduce the considered variants of the π-calculus, some abbreviations to simplify the presentation of encodings, and the criteria of [Gor10] to measure the correctness of encodings. In Section 3, we revisit the encoding given in [Nes00]; based on it, we present a novel encoding, denoted by $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, of mixed choice. Section 4 discusses in how far an encoding like $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ can preserve the degree of distribution. We conclude in Section 5.

2 Technical Preliminaries

2.1 The π-Calculus

Our source language is the monadic π-calculus as described for instance in [SW01]. As already demonstrated in [Pal03] the most interesting operator for a comparison of the expressive power between the full π-calculus and its asynchronous variant is mixed choice, i.e., choice between input and output capabilities. Thus we denote the full π-calculus also by π_{m}. Let $\mathcal{N}$ denote a countably infinite set of names with $\tau \notin \mathcal{N}$ and $\overline{\mathcal{N}}$ the set of co-names, i.e., $\overline{\mathcal{N}} = \{\overline{n} \mid n \in \mathcal{N}\}$. We use lower case letters $a, a', a_1, \ldots, x, y, \ldots$ to range over names.

Definition 1 (π_{m}). *The set of process terms of the* synchronous π-calculus (with mixed choice)*, denoted by $\mathcal{P}_{\mathrm{m}}$, is given by*

$$P ::= (\nu\, n)\, P \quad | \quad P_1 \mid P_2 \quad | \quad [\, a = b \,]\, P \quad | \quad y^*\,(x)\,.P \quad | \quad \sum_{i \in I} \pi_i.P_i$$

$$\pi ::= y\,(x) \;\mid\; \overline{y}\,\langle z \rangle \;\mid\; \tau$$

where $n, a, b, x, y, z \in \mathcal{N}$ range over names and I ranges over finite index sets.

The interpretation of process terms is as usual. We consider two subcalculi of π_{m}. The process terms $\mathcal{P}_{\mathrm{s}}$ of π_{s} —the *π-calculus with separate choice*—are obtained by restricting the choice primitive such that in each choice either no input guarded or no output guarded alternatives appear. The process terms $\mathcal{P}_{\mathrm{a}}$ of the *asynchronous π-calculus* π_{a} [Bou92, HT91] are obtained by limiting each sum to at most one branch and requiring that outputs can only guard the empty sum.

Note that we augment all three variants of the π-Calculus with matching, because we need it at least in π_{a} to encode mixed choice. Of course, the presence of match influences the expressive power of π_{a}. However, we do not know, whether the use of match in the encoding of mixed choice can be circumvented, although there are reasons indicating that this is indeed not possible. We leave the consideration of this problem to further research.

We use capital letters $P, P', P_1, \ldots, Q, R, \ldots$ to range over processes. Let $\mathsf{fn}(P)$, $\mathsf{bn}(P)$, and $\mathsf{n}(P)$ denotes the sets of free names, bound names and all names occurring in P, respectively. Their definitions are completely standard. Given an input prefix $y\,(x)$ or an output prefix $\overline{y}\,\langle x \rangle$ we call y the subject and x

$$\textsc{Tau}_{\mathrm{m}} \quad \ldots + \tau.P + \ldots \longmapsto P$$

$$\textsc{Com}_{\mathrm{m}} \quad (\ldots + y\,(x)\,.P + \ldots) \mid (\ldots + \overline{y}\,\langle z\rangle\,.Q + \ldots) \longmapsto \{^{z}/_{x}\}\,P \mid Q$$

$$\textsc{Par} \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \textsc{Res} \quad \frac{P \longmapsto P'}{(\nu\, x)\,P \longmapsto (\nu\, x)\,P'}$$

$$\textsc{Cong} \quad \frac{P \equiv P' \quad P' \longmapsto Q' \quad Q' \equiv Q}{P \longmapsto Q}$$

Fig. 1. Reduction Semantics of π_{m}

the object of the action. We denote the subject of an action also as link or channel name, while we denote the object as value or parameter. Note that in case the object does not matter we omit it, i.e., we abbreviate an input guarded term $y\,(x)\,.P$ or an output guarded term $\overline{y}\,\langle x\rangle\,.P$ such that $x \notin \mathsf{fn}(P)$ by $y.P$ or $\overline{y}.P$, respectively. We denote the empty sum with $\mathbf{0}$ and often omit it in continuations. As usual, we sometimes write a sum $\sum_{i\in\{i_1,\ldots,i_n\}} \pi_i.P_i$ as $\pi_{i_1}.P_{i_1} + \ldots + \pi_{i_n}.P_{i_n}$.

We use $\sigma, \sigma', \sigma_1, \ldots$ to range over substitutions. A substitution is a mapping $\{^{x_1}/_{y_1}, \ldots, ^{x_n}/_{y_n}\}$ from names to names. The application of a substitution on a term $\{^{x_1}/_{y_1}, \ldots, ^{x_n}/_{y_n}\}\,(P)$ is defined as the result of simultaneously replacing all free occurrences of y_i by x_i for $i \in \{1, \ldots, n\}$, possibly applying alpha-conversion to avoid capture or name clashes. For all names in $\mathcal{N} \setminus \{y_1, \ldots, y_n\}$, the substitution behaves as the identity mapping. Let id denote identity, i.e. id is the empty substitution. We naturally extend substitutions to co-names, i.e. $\forall \overline{n} \in \overline{\mathcal{N}}\,.\,\sigma\,(\overline{n}) = \overline{\sigma\,(n)}$ for all substitutions σ.

The *reduction semantics* of π_{m} is given by the transition rules in Figure 1, where *structural congruence*, denoted by $\equiv$, is defined as usual[2]. The reduction semantics of π_{s} is the same, and it is even simpler for π_{a} because of the restrictions on its syntax. As usual, we use $\equiv_\alpha$ if we refer to alpha-conversion only.

Let $P \longmapsto (P \not\longmapsto)$ denote existence (non-existence) of a step from P, i.e. there is (no) $P' \in \mathcal{P}$ such that $P \longmapsto P'$. Moreover, let $\Longmapsto$ be the reflexive and transitive closure of $\longmapsto$ and let $\longmapsto^{\infty}$ define an infinite sequence of steps.

In Section 2.3, we present several criteria to measure the quality of an encoding. The first of these criteria relies on the notion of a context. A *context* $\mathcal{C}\,([\cdot]_1, \ldots, [\cdot]_n)$ is a π-term, i.e., a π_{a}-term in case of Definition 4, with n so-called *holes*. Plugging the π_{a}-terms $P_1, \ldots, P_n$ into the respective holes $[\cdot]_1, \ldots, [\cdot]_n$ of the context, yields a term denoted $\mathcal{C}\,(P_1, \ldots, P_n)$. A context $\mathcal{C}\,([\cdot]_1, \ldots, [\cdot]_n)$ can be seen as a function of type $\mathcal{P}_{\mathrm{a}} \times \ldots \times \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ of *arity* n, applicable to *parameters* $P_1, \ldots, P_n$. Note that a context may bind some free names of $P_1, \ldots, P_n$.

[2] Note that, since we do not use "!" but replicated input, the common rule $!P \equiv P \mid !P$ becomes $y^*\,(x)\,.P \equiv y\,(x)\,.P \mid y^*\,(x)\,.P$.

2.2 Abbreviations

To shorten the presentation and ease the readability of the rather lengthy encoding function in Section 3, we use some abbreviations on π_{a}-terms. First note that we defined only monadic versions of the calculi π_{m}, π_{s}, and π_{a}, where across any link exactly one value is transmitted. However, within the presented encoding function in Section 3, we treat the target language π_{a} as if it allows for polyadic communication. More precisely, we allow asynchronous links to carry any number of values from zero to five, of course under the requirement that within each π_{a}-term no link name is used twice with different multiplicities. Let $\tilde{x}$ denote a sequence of names. Note that these polyadic actions can be simply translated into monadic actions by a standard encoding as given in [SW01]. Thus, we silently use the polyadic version of π_{a} in the following. Second, as already done in [Nes00], we use the following abbreviations to define boolean values and a conditional construct.

Definition 2 (Tests on Booleans). *Let* $\mathbb{B} \triangleq \{\top, \bot\}$ *be the set of* boolean values*, where* $\top$ *denotes* true *and* $\bot$ *denotes* false*.*

Let $l, t, f \in \mathcal{N}$ *and* $P, Q \in \mathcal{P}_{\mathrm{a}}$*. Then a* boolean instantiation *of* l*, i.e., the allocation of a boolean value to a link* l*, and a* test-statement *on a boolean instantiation are defined by*

$$\begin{aligned} \overline{l}\,\langle \top \rangle &\triangleq l\,(t,f)\,.\,\overline{t} \\ \overline{l}\,\langle \bot \rangle &\triangleq l\,(t,f)\,.\,\overline{f} \\ \mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q &\triangleq (\nu\, t,f)\left(\overline{l}\,\langle t,f\rangle \mid t.P \mid f.Q\right) \end{aligned}$$

for some $t, f \notin \mathsf{fn}(P) \cup \mathsf{fn}(Q)$*.*

Finally, we define forwarders, i.e., a simple process to forward each received message along some specified set of links.

Definition 3 (Forwarder). *Let* I *be a finite index set and for all* $i \in I$ *let* y *and* y_i *be channel names with multiplicity* $n \in \mathbb{N}$*, then a* forwarder *is given by:*

$$y \twoheadrightarrow \{y_i \mid i \in I\} \triangleq y^*\,(x_1, \ldots, x_n)\,.\left(\prod_{i \in I} \overline{y_i}\,\langle x_1, \ldots, x_n\rangle\right)$$

In case of a singleton set we omit the brackets, i.e., $y \twoheadrightarrow y' \triangleq y \twoheadrightarrow \{y'\}$*.*

2.3 Quality Criteria for Encodings

Within this paper we consider two encodings, (1) an encoding from π_{s} into π_{a} presented in [Nes00], denoted by $[\![\,\cdot\,]\!]^{\mathrm{s}}_{\mathrm{a}}$, and (2) a new encoding from π_{m} into π_{a}, denoted by $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$. To measure the quality of such an encoding, Gorla [Gor10] suggested five criteria well suited for language comparison. Accordingly, we consider an encoding to be "good", if it satisfies Gorla's five criteria.

As in [Gor10], an encoding is a mapping from a source into a target language; in our case, π_{m} and π_{s} are source languages and π_{a} is the target language. To distinguish terms on these languages or definitions for the respective encodings, we use m, s, and a as super- and subscripts. Thereby, the superscript usually refers to the source and the subscript to the target language. Moreover, we use $S, S', S_1, \ldots$ to range over terms of the source languages and $T, T', T_1, \ldots$ to range over terms of the target language.

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *name invariance* and (2) *compositionality*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection* and (5) *success sensitiveness*. We do not repeat them formally, here, except for compositionality. Note that for name invariance and operational correspondence a behavioural equivalence $\asymp$ on the target language is assumed. Its purpose is to describe the "abstract" behaviour of a target process, where abstract basically means "with respect to the behaviour of the source term".

(1) The first structural criterion *name invariance* states that the encoding should not depend on specific names used in the source term. This is important, as sometimes it is necessary to translate a source term name into a sequence of names or reserve some names for the encoding function. To ensure that there are no conflicts between these reserved names and the source term names, the encoding is equipped with a *renaming policy*, more precisely, a substitution ϕ from names into sequences of names. Note that in the case of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ the renaming policies are injective substitutions from names into single names. Based on such a renaming policy, an encoding is independent of specific names if it preserves all substitutions σ on source terms by a substitution σ' on target terms such that σ' respects the changes made by the renaming policy.

(2) The second structural criterion *compositionality* aims at relaxing rigidity. We call the translation of a operator $\mathbf{op}$ *rigid*, if $[\![\mathbf{op}]\!]$ is mapped onto $\mathbf{op}^{\phi}$, i.e., essentially the same operator, but possibly adapted to comply with the renaming policy ϕ. For example, $[\![(\nu\, x)\, P]\!] = (\nu\, \phi(x))\, [\![P]\!]$ translates the restriction on a single name into the restriction on the sequence of associated names. Now, we call the translation of an operator $\mathbf{op}$ "merely" *compositional* if $[\![\mathbf{op}]\!]$ is defined quite more liberally as a context $\mathcal{C}^{\phi}_{\mathbf{op}}$ (of the same arity as $\mathbf{op}$) that mediates between the translations of $\mathbf{op}$'s parameters, while those parameters are still translated independently of $\mathbf{op}$. In order to realize this mediation, the context $\mathcal{C}_{\mathbf{op}}$ must at least be allowed to know some of the parameters' free names.

Definition 4 (Compositionality). *The encoding* $[\![\cdot]\!]$ *is* compositional *if, for every k-ary operator* $\mathbf{op}$ *of the source language and for every subset of names* N*, there exists a k-ary context* $\mathcal{C}^{N}_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_k)$ *such that, for all* $S_1, \ldots, S_k$ *with* $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_k) = N$*, it holds that*

$$[\![\mathbf{op}\,(S_1, \ldots, S_k)]\!] = \mathcal{C}^{N}_{\mathbf{op}}\,([\![S_1]\!], \ldots, [\![S_k]\!])\,.$$

Note that Gorla requires the parallel composition operator "|" to be binary and unique in both the source and the target language. Thus, compositionality prevents us from introducing a global coordinator or to use global knowledge, i.e.,

$$
\begin{aligned}
\left[\!\!\left[\sum_{i\in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu\, l)\left(\overline{l}\,\langle\top\rangle \mid \prod_{i\in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\\
[\![\, \tau.P \,]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \left(\overline{l}\,\langle\bot\rangle \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\ \mathsf{else}\ \overline{l}\,\langle\bot\rangle\\
[\![\, \overline{y}\,\langle z\rangle .P \,]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu\, s)\left(\overline{y}\,\langle l, s, z\rangle \mid s.[\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\\
[\![\, y\,(x).P \,]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu\, r)\left(\overline{r} \mid r^*.y\left(l', s, x\right).\right.\\
&\qquad \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\\
&\qquad\qquad \mathsf{else}\ \overline{l}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{r}\\
&\qquad \left.\mathsf{else}\ \overline{l}\,\langle\bot\rangle \mid \overline{y}\langle l', s, x\rangle\right)\\
[\![\, y^*\,(x).P \,]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq y^*\,(l, s, x).\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\,\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\,\langle\bot\rangle
\end{aligned}
$$

Fig. 2. Encode π_{s} into π_{a} [Nes00]

knowledge about surrounding source terms or the structure of the parameters. We discuss this point in Section 4.

(3) The first semantic criterion and usually the most elaborate one to prove is *operational correspondence*, which consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be emulated by its translation, i.e., the translation does not shrink the set of computations of the source term. Note that encodings often translate single source term steps into sequences of target term steps. We call such sequences *emulations* of the corresponding source term step. *Soundness* requires that every computation of a target term corresponds to some computation of the corresponding source term, i.e., the translation does not introduce new computations.

(4) With *divergence reflection* we require, that any infinite execution of a target term corresponds to an infinite execution of the respective source.

(5) The last criterion *success sensitiveness* links the behaviour of source terms to the behaviour of their encodings. With Gorla [Gor10], we assume a *success* operator $\checkmark$ as part of the syntax of both the source and the target language, i.e., of π_{m}, π_{s}, and π_{a}. Since $\checkmark$ can not be further reduced, the operational semantics is left unchanged in all three cases. Moreover, note that $\mathsf{n}(\checkmark) = \mathsf{fn}(\checkmark) = \mathsf{bn}(\checkmark) = \emptyset$, so also interplay of $\checkmark$ with the rules of structural congruence is smooth and does not require explicit treatment. The test for reachability of success is standard. Finally, an encoding preserves the abstract behaviour of the source term if it and its encoding answer the tests for success in exactly the same way.

For a more exhaustive and formal description we refer to [Gor10].

3 Encoding Mixed Choice

Nestmann [Nes00] presents an encoding from π_{s} into π_{a}, in the following denoted by $[\![\, \cdot \,]\!]^{\mathrm{s}}_{\mathrm{a}}$, that encodes the parallel operator rigidly: $[\![\, P \mid Q \,]\!]^{\mathrm{s}}_{\mathrm{a}} \triangleq [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\, Q \,]\!]^{\mathrm{s}}_{\mathrm{a}}$. In the following, for simplicity, we omit the indication of the renaming policy.

The full details are given in [PN12]. The encodings of sum, guarded terms, and replicated input are given in Figure 2 where, in the last four clauses, we assume that the name l that is used on the right-hand sides is an implicit parameter of the encoding function, as supplied in the first of the above clauses. The remaining operators are translated rigidly (compare to their translations in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$). The main idea of this encoding is to introduce a so-called sum lock l carrying a boolean value for each sum. In order to emulate a step on a source term summand the respective sum lock is checked. In case its boolean value is $\top$ the respective source term step is emulated, else the emulation is aborted and the terms corresponding to this emulation attempt remain as junk (compare to [Nes00] for a more detailed discussion of this encoding). [Nes00] argues for the correctness of this encoding by proving its deadlock- and divergence-freedom; he also discussed the possibilities to state full abstraction results. In [PN12], we present a proof of its correctness with respect to the criteria presented in Section 2.3.

As already proved in [Pal03] and later on rephrased in [Gor10, PN10], it is not possible to encode π_{m} into π_{a}, while translating the parallel operator rigidly. However, by weakening this requirement, the separation result no longer holds—instead, an encodability result is possible. To prove this, we give an encoding from π_{m} into π_{a}, denoted as $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, that is correct with respect to the criteria established by [Gor08, Gor10].

As stated in [Nes00], the encoding presented above introduces deadlock when applied in the case of mixed choice, due to the nested test-statements in the encoding of an input-guarded source term. However, [Nes00] also states that all potential deadlocks can be avoided by using a total ordering on the sum locks. Of course, compositionality forbids to simply augment the encoding with an arbitrary previously created ordering, because this would require some form of global knowledge on the source terms. So, the main idea behind the design of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is to augment $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ with an algorithm to dynamically compute an order on the sum locks—at runtime. Unfortunately, this algorithm fairly blows up the translation of the parallel operator and replicated input.

For sums, the translation via $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ follows exactly the scheme of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$.

$$\left[\!\!\left[\sum_{i \in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{m}}_{\mathrm{a}} \triangleq (\nu\, l) \left(\overline{l}\,\langle \top \rangle \mid \prod_{i \in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \right)$$

This translation splits up the encoded summands in parallel and introduces the sum locks, which are initialised by $\top$. To *order* these sum locks, we first have to transport them to a surrounding parallel operator encoding: for example, in $P \mid Q$, with P and Q being sequential processes, the sums occurring in either P or Q will have their locks ordered by means of the translation $[\![\, P \mid Q \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Therefore, in the translation, we let input- and output-guarded source terms not communicate directly, but instead require that they first register their send/receive abilities to a surrounding parallel operator encoding, by sending an output request $\overline{p_o}\,\langle y, l, s, z \rangle$ or an input request $\overline{p_i}\,\langle y, l, r \rangle$. A request carries all necessary information to resolve a nested test-statement, i.e., the translated link name, the corresponding sum lock, a sender lock or a receiver lock, and, in case of

an output request, the translation of the transmitted value. Note that a sender lock, i.e., the s in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, is used to guard the encoded continuation of the sender, while over the receiver lock, i.e., the r in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, the ordered sum locks are transmitted back to the receiver. For convenience, the mapping $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is implicitly parametrised by the names p_o and p_i; in the clause for parallel composition, some of their occurrences will be bound, while others will be free.

$$
\begin{aligned}
[\![\, \tau.P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\,\langle\bot\rangle \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\,\langle\bot\rangle \\
[\![\, \overline{y}\,\langle z\rangle.P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu\, s)\,(\overline{p_o}\,\langle y, l, s, z\rangle \mid s.\,[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}) \\
[\![\, y\,(x).P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu\, r)\,\big(\overline{p_i}\,\langle y, l, r\rangle \mid r^*\,(l_1, l_2, -, s, x)\,. \\
&\qquad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\,\langle\bot\rangle \mid \overline{l_2}\,\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_1}\,\langle\top\rangle \mid \overline{l_2}\,\langle\bot\rangle \\
&\qquad \mathsf{else}\ \overline{l_1}\,\langle\bot\rangle\,\big)
\end{aligned}
$$

Apart from requests, the encoding of guarded terms is very similar to $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. The requests push the task of finding matching source term communication partners to the surrounding parallel operator encodings. There, a strict policy controls the redirection of requests. First, it restricts the request channels p_o and p_i for both of its parameters to be able to distinguish requests from the left from those from the right side.

$$
\begin{aligned}
[\![\, P \mid Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} \triangleq\ & (\nu\, m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,\big(\\
& (\nu\, p_o, p_i)\,\big([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big) \\
& \mid (\nu\, p_o, p_i)\,\big([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big) \\
& \mid \mathsf{pushReq}\big)
\end{aligned}
$$

Note that, since "|" is binary, a source term is structured as a binary tree—its *parallel structure*—with a sum or a replicated input in its leafs and a parallel operator in each remaining node. At each such node, a matching pair of communication partners can meet. More precisely, for every pair of matching communication partners, there is exactly one node such that one partner is at its left and the other at its right side. Therefore, each parallel operator encoding pushes all received (left or right) requests further upwards to a surrounding parallel operator encoding by means of the forwarders in $\mathsf{pushReq} \triangleq p_{o,up} \twoheadrightarrow p_o \mid p_{i,up} \twoheadrightarrow p_i$.

Requests from the left are forwarded to the links $p_{o,up}$ or $p_{i,up}$, to be pushed further upwards with $\mathsf{pushReq}$. Moreover, in order to combine requests from the left with requests from the right side, all left requests are forwarded to the right side over m_o and m_i. Thus left requests are processed by two simple forwarders, $\mathsf{procLeftOutReq} \triangleq p_o \twoheadrightarrow \{m_o, p_{o,up}\}$ and $\mathsf{procLeftInReq} \triangleq p_i \twoheadrightarrow \{m_i, p_{i,up}\}$.

The processing of requests from the right is more difficult. Intuitively, the encoding ensures that any request of the left side is combined exactly once with each opposite request of the right side. Then, the respective first parameters of each pair of requests are matched, to reveal a pair that results from the translation of matching communication partners. If such a pair is found, then the

information necessary to resolve the respective test-statement are retransmitted over the receiver lock back to the receiver, where the test-statement completes the emulation in case of positive instantiated sum locks. To avoid deadlock, the sum lock of the left request is always checked first. Since the encoding relies on the parallel structure of the source term, which is a binary tree, to prefer always the left lock indeed results in a total ordering of the sum locks.

$$
\begin{aligned}
\mathsf{procRightOutReq} \triangleq\ & \overline{c_o}\,\langle m_i\rangle \mid {c_o}^*\,(m_i)\,.p_o\,(y, l_s, s, z)\,.\big(\\
& (\nu\, m_{i,up})\,\big(\, {m_i}^*\,(y', l_r, r)\,.\,([\,y' = y\,]\,\overline{r}\,\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\,\langle y', l_r, r\rangle) \\
& \qquad \mid (\nu\, m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\,\langle m_i\rangle)\,\big) \\
& \mid \overline{p_{o,up}}\,\langle y, l_s, s, z\rangle\,\big)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{procRightInReq} \triangleq\ & \overline{c_i}\,\langle m_o\rangle \mid {c_i}^*\,(m_o)\,.p_i\,(y, l_r, r)\,.\big(\\
& (\nu\, m_{o,up})\,\big(\, {m_o}^*\,(y', l_s, s, z)\,.\,([\,y' = y\,]\,\overline{r}\,\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\,\langle y', l_s, s, z\rangle) \\
& \qquad \mid (\nu\, m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\,\langle m_o\rangle)\,\big) \\
& \mid \overline{p_{i,up}}\,\langle y, l_r, r\rangle\,\big)
\end{aligned}
$$

In order to emulate arbitrary source term steps, all pairs of left and right requests have to be checked at least once. On the other side, a careless checking of the same pairs infinitely often introduces divergence. Thus, only a single copy of each left request is transmitted to the right side and, there, each pair of left and right requests is combined exactly once. To do so, the right requests are linked together within two chains; one for right output requests and one for right input requests. The first member of the chain receives all left requests via m_o or m_i, combines them with its own information, and sends a copy of each left request to the next member over $m_{o,up}$ or $m_{i,up}$, respectively. Subsequent members of a chain are linked by m_o or m_i, i.e., each member creates a new version of the corresponding name and sends this new version over c_o or c_i to enable the addition of a new member. Moreover, it transmits all received left requests along this new version. A new member is then added to the chain by the consumption of its request, also triggering to transmit a copy to $\mathsf{pushReq}$ via $p_{o,up}$ or $p_{i,up}$.

Finally restriction, match, and success are translated rigidly:

$$
\begin{aligned}
[\![\, (\nu\, x)\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu\, \varphi^{\mathrm{m}}_{\mathrm{a}}\,(x))\,[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
[\![\, [\, a = b\,]\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq [\, \varphi^{\mathrm{m}}_{\mathrm{a}}\,(a) = \varphi^{\mathrm{m}}_{\mathrm{a}}\,(b)\,]\,[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
[\![\, \checkmark \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \checkmark
\end{aligned}
$$

In the discussion so far, we omitted the encoding of replicated input, because it is slightly tricky. The crux is that each replicated input implicitly represents an unbounded number of copies of the respective input in parallel. Each such copy changes the parallel structure of the source term, on which our encoding function relies. Obviously, a compositional encoding can not first compute the number of required copies. By the reduction semantics, the copies of a replicated input are generated as soon they are needed. Likewise, the encoding of replicated

input adds a branch to the constructed parallel structure, for each emulated communication with a replicated input. To do so, it adapts the parallel operator translation for each unguarded continuation in encodedContinuations.

$$\begin{aligned}[\![\, y^*\,(x)\,.P\,]\!]^{\mathrm{m}}_{\mathrm{a}} \triangleq\ & (\nu\, l, r, c_{r1}, c_{r2}, r_o, r_i)\,\big(\overline{p_i}\,\langle y, l, r\rangle \\ & \mid r^*\,(-,-,l_s,s,x)\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\,\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\,\langle x\rangle\ \ \mathsf{else}\ \overline{l_s}\,\langle\bot\rangle \\ & \mid \overline{r_i}\,\langle y, l, r\rangle \mid \overline{l}\,\langle\top\rangle \mid \mathsf{encodedContinuations}\big)\end{aligned}$$

To direct the flow of requests among the additional branches, they are again ordered into a chain.

$$\begin{aligned}\mathsf{encodedContinuations} \triangleq\ & \overline{c_{r2}}\,\langle r_o, r_i\rangle \mid {c_{r1}}^*\,(x)\,.c_{r2}\,(r_o, r_i)\,. \\ & (\nu\, m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,\big(\mathsf{pushReqIn} \\ & \mid (\nu\, p_o, p_i)\,([\![\, P\,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\ & \mid (\nu\, r_o, r_i)\,(\overline{c_{r2}}\,\langle r_o, r_i\rangle \mid \mathsf{pushReqOut})\,\big)\end{aligned}$$

For each successful emulation on a replicated input, a new branch with the encoded continuation is unguarded by transmitting the received source term value over c_{r1}. As in the chains of right requests, each branch in encodedContinuations restricts its own versions of r_o and r_i to receive all requests from its successor. These links are transmitted over c_{r2} to the respective next member. The translation of the replicated input serves itself as first member of the chain by providing its own request over r_i. Note that the third line of encodedContinuations is exactly the same as the right side of a parallel operator encoding. There, all received requests are combined with the requests of the respective continuation to enable the emulation of a communication with the replicated input or another of its unguarded continuations. Moreover, to enable an emulation of a communication with the rest of the term, its requests are pushed upwards. The remaining terms pushReqIn and pushReqOut direct the flow of requests.

$$\begin{aligned}\mathsf{pushReqIn} &\triangleq r_o \twoheadrightarrow \{m_o, r_{o,up}\} \mid r_i \twoheadrightarrow \{m_i, r_{i,up}\} \\ \mathsf{pushReqOut} &\triangleq p_{o,up} \twoheadrightarrow \{p_o, r_o\} \mid r_{o,up} \twoheadrightarrow r_o \mid p_{i,up} \twoheadrightarrow \{p_i, r_i\} \mid r_{i,up} \twoheadrightarrow r_i\end{aligned}$$

pushReqIn receives all requests from a predecessor in the chain and forwards one copy to the encoded continuation over m_o and m_i and one copy to pushReqOut. There, all requests of the encoded continuation are pushed upwards to a surrounding parallel operator encoding over p_o or p_i, and for all such requests and all requests received from a previous member, a copy is forwarded to the successor over r_o or r_i.

For a more exhaustive description of the algorithm implemented by this encoding and how it emulates source term steps, we refer to the proof of its correctness in [PN12].

Theorem 1. *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *are good.*

Observations

The existence of a good encoding from π_{m} into π_{a} shows that π_{a} is as expressive as π_{m} with respect to the abstract behaviour of terms. This looks surprising. From [Pal03, Gor10, PN10], we know that it is not possible to implement mixed choice without introducing some additional amount of coordination. The existence of the good encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ proves that, to do so, no global coordination is necessary. Instead the little amount of local coordination, which is allowed in compositional encodings, suffices to completely implement the full power of mixed (guarded) choice within an asynchronous and thus choice-free setting.

However, the encoding presented above comes with some drawbacks. The most crucial of these drawbacks—at least with respect to efficiency measures—is the impact of the encoding function on the degree of distribution of source terms. We consider this problem in the next section. Another drawback is the necessity of the match operator in the target language. Examining the proof of the main result in [PSN11], we observe that it already indicates how to solve the problem of the deadlocks in the test-statements of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. Moreover, it reveals a second solution to that problem. Instead of implementing an algorithm to order the sum locks, we could always test the sum lock of the receiver first, if we restrict the number of emulations that can be performed simultaneously. To do so, we augment $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ with an aditional coordinator lock—an output $\overline{c}$ on a new channel c—for each encoding of a parallel operator and require that this lock must be available in order to send an output over the receiver lock r. Then, each completion of a test-statement in the encoding of input or replicate input—regardless of its outcome—restores the coordinator lock of the respective parallel operator encoding. Due to the restriction of simultaneous emulations, the impact of such an encoding on the degree of distribution of source terms is even worse than it is the case for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. However, for both solutions, it is necessary to send some kind of input and output requests and to combine requests of communication partners in order to emulate a communication step. Due to scope extrusion in the source and the necessity in the target to restrict the request channels, we cannot ensure that the requests of different source term steps can be distinguished by their channel names. Thus, to examine which pairs of requests refer to matching communication partners, we need the matching primitive.

A problem that already occurs in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ is the introduction of observable junk, i.e., of observable remainders left over by further emulations. In π_{m}, if we perform a step on a summand of a sum, immediately any other summand of that sum disappears. In the implementation, we have to split up the encoded summands in parallel, such that it is not possible to immediately withdraw the encoded summands as soon as one summand is used within an emulation. In $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ such observable junk is marked by a false instantiation of its sum locks. As a consequence, the encodings are not good w.r.t. a standard equivalence $\asymp$, as asynchronous barbed congruence. However, for both encodings, we can prove correctness with respect to a non trivial variant of barbed equivalence, by redefining the notion of barbs to the result of translating source term barbs. The result is a congruence w.r.t. contexts that respect the protocol of the encoding.

As mentioned above, our encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ augments the parallel structure of source terms to order sum locks. Accordingly, another parallel structure of the source—e.g. as a result of applying the rule $P \mid Q \equiv Q \mid P$ to it—results in a different order of the respective sum locks. Hence, it is possible that, for some source terms S_1 and S_2, the target terms $[\![S_1 \mid S_2]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S_2 \mid S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ differ in the number of necessary pre- or postprocessing steps within an emulation, but also in the reachability of "intermediate", i.e.: "partially committed", states. Although these states exhibit different observables, their differences do not introduce deadlock or influence the possibility to emulate source term steps, i.e. $[\![S_1 \mid S_2]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S_2 \mid S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ still have the same abstract behaviour. Interestingly, the alternative solution on coordinator locks reveals similar problems with the rule $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. To overcome this problem, the equivalence $\asymp$ has either to abstract from the reachability of intermediate states or we have to avoid the rule Cong in our reduction semantics.

For a more formal and exhaustive discussion of these drawbacks and a definition of the used equivalences, we refer to [PN12]. We believe that none of the described drawbacks can be circumvented. In this sense, we think that the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ given above is the best encoding from π_{m} into π_{a} we can achieve.

4 Distributability

The first result comparing the expressive power of π_{m} and π_{a} is given by the separation result in [Pal03]. The main difference to our encodability result in the last section is due to the requirement on the rigid translation of "|". This requirement ensures that the encoding "preserves the degree of distribution" of the source term, which—thinking of distributed computing systems—is a crucial measure for the efficiency of such an encoding. A distributed system is a network of parallel processes. Accordingly, a distributed algorithm is an algorithm that may perform at least some of its tasks in parallel. Therefore, a main issue when considering an encoding between distributed algorithms is to ensure that it does not sequentialise all tasks by introducing a *global coordinator*. On the other side, the rigid translation of the parallel operator is a rather hard requirement. Therefore, Gorla instead requires the compositional translation of all source term operators. Note that also this requirement already prevents the use of global coordinators. In that view, compositionality can be seen as a minimal criterion to ensure the preservation of the degree of distribution.

However, sometimes—as in the current case—compositionality alone is too weak to consider the preservation of the degree of distribution, because it still allows for *local coordinators*: a compositional encoding may still sequentialise some of the parallel tasks of a distributed algorithm. If we are not only interested in the expressive power in terms of the abstract behaviour but additionally in how far problems can be solved exploiting at least the same degree of distribution, we must consider an additional criterion.

Up to now, there have been various approaches to explicitly consider the concurrent execution of independent steps directly within an operational semantics, often called *step semantics* (e.g., [Lan07] for the case of process calculi), and also

in the form of dedicated behavioral equivalences. In our case, we do not want to explicitly quantify the degree of distribution in the source and the target term, but only to measure whether it is preserved by an encodings. To this aim, we choose a simple and intuitive formulation—in the style of Gorla's criteria—of our additional requirement based on the notion of parallel component.

Note that it does *not* suffice to consider the *initial* degree of distribution, i.e., to require that each source term and its encoding are distributed in the same way. We also have to require, that whenever a part of a source term can solve a task independently of the rest—i.e., it can reduce on its own—then the respective part of its encoding must also be able to emulate this reduction independent of the rest of the encoded term. Accordingly, we require that not only the source term and its encoding are distributed in the same way, but also their derivatives. In the following, $\equiv_1$ is the usual structural congruence naturally of the source language and $\equiv_2$ is the usual structural congruence of the target language.

Definition 5. *An encoding* $[\![\cdot]\!]$ preserves *the* degree of distribution *if, for every* S *such that* $S \equiv_1 S_1 \mid \ldots \mid S_n$ *and* $S_i \Longmapsto S_i'$ *for all* i *with* $1 \leq i \leq n$*, there exists* $T_1, \ldots, T_n$ *and a context* $\mathcal{C}$ *with* n *holes such that* $[\![S]\!] \equiv_2 \mathcal{C}\,(T_1, \ldots, T_n)$ *and* $T_i \Longmapsto\asymp [\![S_i']\!]$ *for all* i *with* $1 \leq i \leq n$.

Here, the context $\mathcal{C}$ is introduced to allow, e.g., for some global restrictions or parts of the encoded term that may be necessary to emulate communications between S_i and S_j for $i \neq j$. This only makes sense because compositionality already rules out global coordinators. Since the parallel operator is considered to be binary, context $\mathcal{C}$ can be the result of assembling parts of the contexts introduced by several parallel operator encodings. In essence, the requirement in Definition 5 is a concurrency-enhanced adaptation of operational completeness: whenever a source term can perform n steps in parallel, then its encoding must be able to emulate all n steps in parallel; note that the T_i must be able to move independent of the context $\mathcal{C}$. So, Definition 5 describes a *semantic* criterion.

Definition 5 is not the only way to measure the preservation of the degree of distribution. However, when considering the degree of distribution, we find it natural and appealing to require that parallel source term steps can be emulated truly in parallel, i.e., that for each pair of independent source term steps there is at least the possibility to emulate them independently. Moreover, by the following consideration, we observe that this requirement indeed suffices to reveal a fundamental difference in the expressive power of π_{m} compared to π_{s} or π_{a} considering the degree of distribution. Since $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ translates the parallel operator rigidly, it naturally preserves the degree of distribution.

Lemma 1. *The encoding* $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ *preserves the degree of distribution.*

Notably, in the proof of the lemma above we do not use any features of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ except that it satisfies operational completeness, i.e., it is a good encoding, translates the parallel operator rigidly, and preserves structural congruence. So, any such encoding preserves the degree of distribution. Not surprisingly, the most crucial requirement here is the rigid translation of "|".

Lemma 2. *Any good encoding, that translates the parallel operator rigidly and preserves enough of the structural congruence on source terms to ensure that* $S \equiv_1 S_1 \mid \ldots \mid S_n$ *implies* $[\![S]\!] \equiv_2 [\![S_1 \mid \ldots \mid S_n]\!]$*, preserves the degree of distribution.*

Thus, the (semantic) criterion formalised in Definition 5 can be considered to be at most as hard as the (syntactic) criterion to rigidly translate the parallel operator. To see that it is not an equivalent requirement, but indeed strictly weaker, we may consider an encoding (spelled out in [PN12]) from π_{m} (without replicated input) into π_{a}^2, the asynchronous π-calculus augmented with a two-level polyadic synchronisation by Carbone and Maffeis [CM03]. It is a simplified version of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$, based on the same way to order sum locks but without the necessity to link right requests in chains. To prove that it is good, an argumentation similar as for $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ can be applied. Moreover, [CM03] prove that there is no good encoding from π_{m} into π_{a}^2 that translates "|" rigidly; this separation result does not rely on replication, i.e., it also implies that there is no such encoding from π_{m} without replicated input into π_{a}^2. On the other side, since all parts of the context introduced by the parallel operator encoding are replicated inputs, it preserves the degree of distribution.

The encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ does not preserve the degree of distribution, because we can not distribute the linking of right requests within a chain at the right side of a parallel operator encoding. Because of that, all steps on communication partners that meet at the same parallel operator in the source term, can never be emulated independently even if the source term steps are.

Lemma 3. *The encoding* $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ *does* not *preserve the degree of distribution.*

This lemma is not due to an awkward design of the encoding function $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$, but is a general restriction on the encodability of mixed choice, i.e., it is not possible to design a good encoding from π_{m} into π_{a} that preserves the degree of distribution. This fact is a direct consequence of the theorem proved in [PSN11].

Theorem 2. *There is no good encoding from* π_{m} *into* π_{a} *that preserves the degree of distribution.*

5 Conclusion

We present a novel and for some readers perhaps surprising encodability result, showing that the asynchronous π-calculs is "as expressive as" the synchronous π-calculus with mixed choice, if the non-rigid translation of parallel composition is allowed. Furthermore, we present a fundamental limitation of each good encoding between these two languages concerning a novel criterion that measures the preservation of the degree of distribution. In contrast to the three semantic criteria of operational correspondence, divergence reflection, and success sensitivity, our new criterion does not primarily consider the (abstract) behaviour of terms but an additional dimension: the potential for concurrent execution. We conclude that considering the behaviour of terms, the full π-calculus and its asynchronous variant have the same expressive power. Our result complements

Palamidessi's result [Pal03], as her rigidity criterion includes more than just abstract behavior. Likewise, as expected, then translating a π_{m}-algorithm into a π_{a}-algorithm by such an encoding, one must tolerate losses in the efficiency of the respective algorithm—which Palamidessi's rigidity requirement would forbid.

Note that the separation of criteria considering the behaviour from additional requirements as the degree of distribution offers additional advantages, because we can more easily analyse the reasons of separation results and in how far they limit the degree of distribution of the encoded algorithm. There is no way to overcome the theoretical border stated by our separation result. However, the proof that an encoding does not preserve the degree of distribution can point out ways to nevertheless optimise a translation of algorithms, because it exactly states which parts can not be distributed.

Of course, only a study of other process calculi and corresponding encoding functions can reveal whether the proposed criterion is suited to measure the degree of distribution in general.

Acknowledgements. We thank Daniele Gorla for his very constructive comments and some fruitful discussions on preliminary versions of this work.

References

[Bou92] Boudol, G.: Asynchrony and the π-calculus (note). Note, INRIA (May 1992)

[CM03] Carbone, M., Maffeis, S.: On the Expressive Power of Polyadic Synchronisation in π-Calculus. Nordic Journal of Computing 10, 1–29 (2003)

[Gor08] Gorla, D.: Towards a Unified Approach to Encodability and Separation Results for Process Calculi. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 492–507. Springer, Heidelberg (2008)

[Gor10] Gorla, D.: Towards a Unified Approach to Encodability and Separation Results for Process Calculi. Information and Computation 208(9), 1031–1053 (2010)

[HT91] Honda, K., Tokoro, M.: An Object Calculus for Asynchronous Communication. In: America, P. (ed.) ECOOP 1991. LNCS, vol. 512, pp. 133–147. Springer, Heidelberg (1991)

[Lan07] Lanese, I.: Concurrent and Located Synchronizations in π-Calculus. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 388–399. Springer, Heidelberg (2007)

[Nes00] Nestmann, U.: What is a "Good" Encoding of Guarded Choice? Information and Computation 156(1-2), 287–319 (2000)

[Pal03] Palamidessi, C.: Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculi. Mathematical Structures in Computer Science 13(5), 685–719 (2003)

[PN10] Peters, K., Nestmann, U.: Breaking Symmetries. In: Fröschle, S.B., Valencia, F.D. (eds.) EXPRESS. EPTCS, vol. 41, pp. 136–150 (2010)

[PN12] Peters, K., Nestmann, U.: Is it a "Good" Encoding of Mixed Choice? (Technical Report). Technical Report, TU Berlin, Germany (January 2012), `http://arxiv.org/corr/home`

[PSN11] Peters, K., Schicke-Uffmann, J.-W., Nestmann, U.: Synchrony vs Causality in the Asynchronous Pi-Calculus. In: Luttik, B., Valencia, F.D. (eds.) EXPRESS. EPTCS, vol. 64, pp. 89–103 (2011)

[SW01] Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, New York (2001)