Math. Struct. in Comp. Science (2016), *vol.* 26, *pp.* 1054–1106.
doi:10.1017/S0960129514000346 First published online 12 November 2014

Breaking symmetries[†]

KIRSTIN PETERS and UWE NESTMANN

Technische Universität Berlin, Germany
Email: kirstin.peters@tu-berlin.de, uwe.nestmann@tu-berlin.de

Received 7 April 2011; revised 28 March 2012

A well-known result by Palamidessi tells us that π_{mix} (the π-calculus with mixed choice) is more expressive than π_{sep} (its subset with only separate choice). The proof of this result analyses their different expressive power concerning leader election in symmetric networks. Later on, Gorla offered an arguably simpler proof that, instead of leader election in symmetric networks, employed the reducibility of 'incestual' processes (mixed choices that include both enabled senders and receivers for the same channel) when running two copies in parallel. In both proofs, the role of *breaking (initial) symmetries* is more or less apparent. In this paper, we shed more light on this role by re-proving the above result – based on a proper formalization of what it means to break symmetries – without referring to another problem domain like leader election.
Both Palamidessi and Gorla rephrased their results by stating that there is no uniform and reasonable encoding from π_{mix} into π_{sep}. We indicate how their proofs can be adapted and exhibit the consequences of varying notions of uniformity and reasonableness. In each case, the ability to break initial symmetries turns out to be essential. Moreover, by abandoning the uniformity criterion, we show that there indeed is a reasonable encoding. We emphasize its underlying principle, which highlights the difference between breaking symmetries locally instead of globally.

1. Introduction

The context of this paper is the formal analysis of both the expressive power and the (distributed) implementability of specification and programming languages for concurrent systems. Concentrating on the computational essence of such languages, we focus on so-called process calculi, as exemplified by the family of π-calculi, which contain as few syntactic primitives as possible. Here, the primitives cover parallel composition, name generation, name passing via handshake interactions between parallel components and, depending on the respective variant, also some notion of choice that allows to express a competitive selection among alternative communication capabilities. As it turned out earlier (see next paragraph), and as we hope to shed further light upon in this paper, the kind of such competitions – mixing inputs and outputs, or not – allowed within choice operators has a great impact on the expressive power of the language. The reason relies on their different power to break symmetries.

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

The well-known result of Palamidessi (2003) tells us that π_{mix} (the π-calculus with mixed choice) is more expressive than π_{sep} (its subset with only separate choice). More technically, the result states that there exists no 'good' – i.e. uniform (structure-preserving) and reasonable (semantics-preserving) – encoding from π_{mix} into π_{sep}. Nestmann (2000) proved that there is a 'good' encoding from π_{sep} to π_{a} (the choice-free asynchronous subset of the π-calculus). He also exhibited various encodings from π_{mix} to π_{sep}, which were not considered 'good' by Palamidessi, as they were not uniform or reasonable enough.

The proof of Palamidessi (2003) analyses the different expressive power of the involved calculi concerning leader election in symmetric networks. More precisely, Palamidessi proves that there is no symmetric network in π_{sep} that solves leader election, whereas there are such networks in π_{mix}. The proof implicitly uses the fact that it is not possible in π_{sep} to break initial symmetries, while this is possible in π_{mix}. To this end, a rather strong notion of symmetry consisting of a syntactic and a semantic component is used to ensure that solving leader election requires breaking initial symmetries. With this result, inspired by the work of Bougé (1988) in the context of *CSP*, Palamidessi proves that there is no uniform and reasonable encoding from π_{mix} into π_{sep}.

Later on, Gorla (2008b) offered an arguably simpler proof for the non-existence of a 'good' encoding from π_{mix} into π_{sep}. Instead of leader election in symmetric networks, it employed the reducibility of 'incestual'processes (mixed choices that include both enabled senders and receivers for the same channel) when running two copies in parallel. Gorla's proof does not explicitly use a notion of symmetry.

Palamidessi's proof that there are no symmetric networks in π_{sep} that solve leader election addresses the *absolute* expressive power of π_{sep}, whereas the proofs of the non-existence of a uniform encoding by Palamidessi and Gorla address the often-called *relative* expressive power of the languages (Parrow 2008). In the following, we discuss these two approaches in more detail, as this allows us to clarify the role of symmetry-breaking in the respective proofs.

The *absolute expressive power* of a language describes what kind of behaviour or operations on behaviour are expressible in it (see Gorla (2008a,b); Parrow (2008) and even Lipton *et al.* (1974)). Analysing the absolute expressive power of a language usually consists of analysing which 'problems' can be solved in it and which cannot. It is often difficult to identify a suitable problem instance or problem domain to properly measure the expressive power of a language. For instance, one might consider Turing-completeness to measure the computational power of a language. In fact, Turing-completeness has been used in the context of process algebras, e.g. for Linda (Busi *et al.* 2000). Instead, Palamidessi, inspired by Bougé (Bougé 1988), uses the distributed coordination problem of leader election. More precisely, the problem refers to initially symmetric networks, where all potential leaders have equal chances and all processes run the same – read: symmetric – code. There, to solve the leader election problem, it is required that in all possible executions a leader is elected. Usually, it is argued that it is necessary – again in all possible executions – to break the initial symmetry in order to do so. On the other hand, if there is just a single execution in which the symmetry is somehow perpetually maintained or at least restored, then also leader election may fail, and thus the leader election problem is not solved. One may conclude that, at a closer look, Palamidessi's proof furthermore

addresses another problem: the problem of breaking initial symmetries. Therefore, we suggest to promote 'breaking symmetries' from a mere auxiliary proof technique to a proper problem of its own. It turns out that, by doing so, we can significantly weaken the definition of symmetry and at the same time provide a stronger proof applicable to problem instances different from leader election.

Now, to *compare* the absolute expressive power of *two* languages, we may simply choose a problem that can be solved in one language, but not in the other language. Actually, as soon as we compare two languages, it makes sense to use the term *relative expressive power*, as we can now relate the two languages. Unfortunately, the terminology was introduced differently. It has been attributed (see Parrow (2008)) to the comparison of the expressive power of two languages by means of the existence or non-existence of encodings from one language into the other language, subject to various conditions on the encoding. In our opinion, the term 'relative expressive power' is misleading. First, as mentioned above, also the absolute expressive power can directly be used to *relate* two languages. Second, results on the encodability of a language have to be understood relative to the specific conditions on the encoding – it is not always clear to what aspect the 'relative' refers. Thus, in this paper, we prefer the notion of *translational expressive power* to refer to comparisons of the expressiveness of two languages by analysing the existence or non-existence of an encoding, subject to various conditions. Both Palamidessi and Gorla state results of this kind; they prove that there is no uniform and reasonable encoding from π_{mix} into π_{sep}, for varying interpretations of the conditions uniform and reasonable.

In this paper, we show that the problem of breaking initial symmetries, compared to the problem of leader election, appears to be a more suitable problem instance to separate π_{mix} from π_{sep}. There are two great benefits in proving an absolute separation result instead of a translational one. First, in opposite to translational separation results which are always equipped with the conditions on the encoding, we can formulate a separation result without any pre- or side conditions. Second, as we show in Section 3.3, we can prove several translational separation results due to different definitions of reasonableness as simple consequences of our absolute separation result. For our work, we had to develop answers to two related questions of definition:

(1) How exactly should one define *symmetric* networks?
(2) What exactly does it mean to *break symmetries*?

By comparing the proofs of the translational separation results of Section 3.3, we observe their dependency on the absolute separation result as well as their dependency on the homomorphic translation of the parallel operator. Gorla (2010) points out that the homomorphic translation of the parallel operator is a rather strict condition. Instead, he proposes (weak) compositionality in combination with four other criteria to define the notion of a 'good' encoding. We claim that this weakening of the structural condition of homomorphic translation of the parallel operator by compositional translation of the parallel operator suffices to turn the translational separation result (negative) into an encodability result (positive). To underpin that claim, we present an encoding from π_{mix} into π_{a} in Section 4, based on a known encoding from π_{sep} into π_{a} (Nestmann 2000), and discuss some of its properties. Note that in Peters and Nestmann (2012a,b) we present

another encoding from π_{mix} into π_a based on similar ideas as well as an exhaustive argumentation for the proof of its correctness with respect to the criteria of Gorla. Since, by our absolute separation result, it is not possible to break initial symmetries in π_{sep}, thus neither in π_a, it is up to the proposed encoding function to break initial source term symmetries. This idea was also the basis for some of the encodings of Nestmann (2000), where the symmetry was broken globally by means of some centralized artefacts. However, as we show in Section 4, it is also possible by means of weak compositionality to exploit the parallel structure of source terms to break symmetries locally.

The main contributions of this paper are then as follows. (1) We present a separation result between π_{mix} and π_{sep} that does not require any additional preconditions. In particular, it is completely independent of what it means for an encoding to be 'good' or 'reasonable'. (2) This absolute separation result, i.e. the inability of π_{sep} in opposite to π_{mix} to break initial symmetries, implies that any encoding from π_{mix} into π_{sep}– if there is any – has to be able to break initial source term symmetries. (3) Since we use a weaker notion of symmetry, and because we do not focus on the leader election problem, our separation result is more general than the one in Palamidessi (2003), i.e. it widens the gap between π_{mix} and π_{sep}. It also allows us to derive a number of translational separation results using counterexamples different from leader election. (4) We prove a stronger translational separation result in comparison to Palamidessi (2003), Vigliotti *et al.* (2007) and (the first setting of) Gorla (2008b) by weakening the conditions on the encodings used. This strengthening of the translational separation results reveals their strong dependency on the homomorphic translation of the parallel operator as one of its fundamental preconditions. By abandoning this precondition, and as a novelty going well beyond our previous article (Peters and Nestmann 2010), (5) we manage to present, as we conjecture, a 'good' encoding from π_{mix} into π_a.

1.1. *Overview of the paper*

In Section 2, we introduce the two process calculi that we intend to compare. Moreover we discuss notions of 'good' encodings and revisit the five criteria Gorla presented in Gorla (2010) to measure the quality of an encoding. In Section 3, we revisit the notion of symmetry used by Palamidessi to propose her separation result and define symmetry as we use it. Then we prove an absolute separation result, i.e. we prove that π_{mix} is strictly more expressive than π_{sep}, by proving the inability of π_{sep} to break initial symmetries. Based on this result, we prove that there is no uniform and reasonable encoding from π_{mix} to π_{sep} examining different notions of reasonableness. In Section 4, we present an attempt to encode π_{mix} into π_a, show how initial source term symmetries are broken by the proposed encoding function, and discuss some of the properties of this encoding attempt. We conclude with Section 5.

2. Technical preliminaries

2.1. *The π-calculus*

Our source language is the monadic π-calculus as described for instance in Sangiorgi and Walker (2001). Since the main reason for the absolute difference in the expressiveness of

the full π-calculus compared to the asynchronous π-calculus is the power of mixed choice we denote the full π-calculus also by π_{mix}.

Let $\mathcal{N}$ denote a countably infinite set of names 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. In π_{mix} (and its subcalculi) names are used for two different purposes. First, they serve as names of communication *links* and so are used by processes to interact. Second, they are used as *values* within interactions, i.e. the objects that can be exchanged over links, and so they allow to constitute dynamic communication networks. Sometimes, we denote the value sent over a link as *parameter* of this link.

Definition 2.1 (π_{mix})**.** The set of process terms of the *π-calculus (with mixed choice)*, denoted by $\mathcal{P}_{\text{mix}}$, are given by

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

where

$$\pi \triangleq y(x) \quad | \quad \overline{y}\langle z \rangle \quad | \quad \tau$$

for some names $n, a, b, x, y, z \in \mathcal{N}$ and a finite index set I.

The interpretation of the defined process terms is as usual. *Restriction* $(\nu\, n)\, P$ restricts the scope of the name n to the definition of P. The *parallel composition* $P_1 \mid P_2$ defines the process in which P_1 and P_2 may proceed independently, possibly interacting using shared links. The operator $[\, a = b \,]$ is called *matching*. It works as a conditional guard, which can be removed if and only if a and b are equal. The process term $\sum_{i \in I} \pi_i.P_i$ represents *finite guarded choice*; as usual, the term $\pi_1.P_1 + \pi_2.P_2$ denotes binary choice, and we use $\mathbf{0}$ as abbreviation for the empty sum, i.e. in case of $I = \varnothing$. $y^*(x).P$ denotes *input-guarded replication.*

We observe that recursion is defined for input-guarded processes only and that processes within sums are also always guarded. A *guard* is either an input prefix $y(x)$, an output prefix $\overline{y}\langle z \rangle$, or the prefix τ. We sometimes refer to input and output prefixes as action prefixes. The input prefix $y(x)$ is used to describe the ability of receiving the value x over link y and, analogously, the output prefix $\overline{y}\langle z \rangle$ describes the ability to send a value z over link y. The prefix τ describes the ability to perform an internal, not observable action. A term whose outermost operator is either choice or replication is called *guarded*, else it is *unguarded*. Moreover an *input/output-guarded* term is a term guarded by an input/output guard, respectively.

As common nowadays, we restrict our attention to guarded choice. However, in the original definition, as presented in Milner *et al.* (1992), the π-calculus contained free choice, i.e. summands can appear unguarded. This restriction does not influence the results presented in Section 3, i.e. they remain valid even for the more general case of free choice in π_{mix} (as long as choice in π_{sep} is restricted to be guarded). But the restriction is necessary to obtain the encoding from π_{mix} into π_{a} in Section 4.

For simplicity, we often omit the continuation $\mathbf{0}$, so $y.\mathbf{0}$ becomes y. In addition, for simplicity in the presentation of examples, we sometimes omit an action's object when it

does not effectively contribute to the behaviour of a term. Typically, we do this when it would be enough to use a CCS-like example, but the monadic π-calculus would force us to carry *some* object along that would never be used on a receiver side, e.g. as in $y(x).\mathbf{0}$, which would be written as $y.\mathbf{0}$ or y.

As target languages, we consider two subcalculi of π_{mix}: π_{sep}, the π-calculus with separate choice, and π_{a}, the asynchronous π-calculus. In π_{sep}, both output and input can be used as guards, but within a single choice term either there are no input or no output guards, i.e. we have input- and output-guarded choice, but no mixed choice.

Definition 2.2 (π_{sep}). The set of process terms of the *π-calculus with separate choice*, denoted by $\mathcal{P}_{\text{sep}}$, are given by

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

where

$$\pi^O \triangleq \overline{y}\langle z \rangle \quad | \quad \tau \qquad \text{and} \qquad \pi^I \triangleq y(x) \quad | \quad \tau$$

for some names $n, a, b, x, y, z \in \mathcal{N}$ and a finite index set I.

As expected, the definitions of π_{sep} and π_{mix} differ in the definition of choice only. Moreover, since each separate choice construct also meets the requirements of a mixed choice but not vice versa, the set of π_{sep}-terms is a strictly smaller subset of the set of π_{mix}-terms, i.e. $\mathcal{P}_{\text{sep}} \subset \mathcal{P}_{\text{mix}}$. For strictness, consider $P = x + \overline{y}$. Obviously, $P \in \mathcal{P}_{\text{mix}}$ but $P \notin \mathcal{P}_{\text{sep}}$.

Asynchronous variants of the π-calculus were introduced independently by Honda and Tokoro (1991) and Boudol (1992). In asynchronous communication, a process has no chance to directly determine, i.e. without a hint by another process, whether a value sent by it was already received or not. To model that fact in π_{a}, output action is not allowed to guard a process different from $\mathbf{0}$. Accordingly, within an asynchronous setting, the interpretation of output guards within a choice construct is delicate. Here, we use the variant of π_{a}, where choice is not allowed at all. Let $\mathcal{T}(M)$ denotes the set of tuples over a set M.

Definition 2.3 (π_{a}). The set of process terms of the *asynchronous π-calculus*, denoted by $\mathcal{P}_{\text{a}}$, are given by

$$\begin{aligned} P \triangleq\ & (\nu\, n)\, P \quad | \quad P_1 \mid P_2 \quad | \quad [\, a = b \,]\, P \\ & | \quad \mathbf{0} \quad | \quad \overline{y}\langle \tilde{z} \rangle \quad | \quad y(\tilde{x}).P \quad | \quad y^*(\tilde{x}).P \end{aligned}$$

for some names $n, a, b, y \in \mathcal{N}$ and $\tilde{x}, \tilde{z} \in \mathcal{T}(\mathcal{N})$.

Since π_{a} has no choice, and thus no nullary choice, we include $\mathbf{0}$ as a primitive. To simplify the definition of the encoding in Section 4, we use a polyadic version of π_{a}. As usual, the tuple notation $\tilde{x} \in \mathcal{T}(\mathcal{N})$ denotes finite sequences $x_1, \ldots, x_n$ of names in $\mathcal{N}$. Moreover, we use $(\nu\, \tilde{x})$ for a sequence $\tilde{x} = x_1, \ldots, x_n$ to abbreviate $(\nu\, x_1) \ldots (\nu\, x_n)$ and $\tilde{x} \setminus M$ for a set of

$P \equiv Q$ if Q can be obtained from P by renaming one or more of the bound names in, silently avoiding name clashes P

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$[\, a = a \,]\, P \equiv P \qquad (\nu\, n)\, \mathbf{0} \equiv \mathbf{0} \qquad (\nu\, n)\, (\nu\, m)\, P \equiv (\nu\, m)\, (\nu\, n)\, P$$

$$P \mid (\nu\, n)\, Q \equiv (\nu\, n)\, (P \mid Q) \qquad \text{if } n \notin \mathsf{fn}(P)$$

Fig. 1. Structural congruence.

names M to denote the sequence of names $\tilde{x}$ without the occurrences of name y for all $y \in M$. We also use the tuple notation for other kinds of data, like actions or labels. Of course, two processes can communicate over the same link if and only if the number of parameters send and the number of parameters expected to receive are equal. Note that if we consider the monadic π_{a} instead of its polyadic variant, then the set $\mathcal{P}_{\mathrm{a}}$ of π_{a}-terms is strictly contained in $\mathcal{P}_{\mathrm{sep}}$.

We use capital letters $P, P', P_1, \ldots, Q, R, \ldots$ to range over processes. If we refer to processes without further requirements, we denote elements of $\mathcal{P}_{\mathrm{mix}}$; we sometimes use just $\mathcal{P}$ when the discussion applies to all three calculi.

Let $\mathcal{A} \triangleq \{\, x\, y,\ \overline{x}\, y,\ \overline{x}\, (y) \mid x, y \in \mathcal{N} \,\}$ denote the set of monadic action labels for visible actions, where $x\, y$ denotes *free input*, $\overline{x}\, y$ denotes *free output*, and $\overline{x}\, (y)$ denotes *bound output*, respectively. Let τ denote an internal invisible action whose label is denoted by τ as well. Let $\mathcal{A}_\tau$ be the corresponding set of *labels*, i.e. $\mathcal{A}_\tau = \mathcal{A} \cup \{\, \tau \,\}$. We use $\mu, \mu', \mu_1, \ldots$ to range over labels. Let $\mathsf{fn}(P)$ and $\mathsf{fn}(\mu)$ denote the sets of *free names* in P and μ, respectively. Let $\mathsf{bn}(P)$ and $\mathsf{bn}(\mu)$ denote the sets of *bound names* in P and μ, respectively. Likewise, $\mathsf{n}(P)$ and $\mathsf{n}(\mu)$ denote the sets of all *names* occurring in P and μ. Their definitions are completely standard, i.e. names are bound by restriction and as parameter of input or replicated input and $\mathsf{n}(P) = \mathsf{fn}(P) \cup \mathsf{bn}(P)$ for all $P \in \mathcal{P}$. We assume that there are no clashes between free and bound names in terms, i.e. in any term the set of bound and free names are disjoint.

The *operational semantics* of π_{mix} and π_{sep} are jointly given by the transition rules in Figures 2 and 3, where *structural congruence*, denoted by $\equiv$, is given by the rules in Figure 1. As usual, we use $\equiv_\alpha$ if we refer to alpha-conversion (the first rule of Figure 1) only. Note that for the separation result in Section 3 (according to Palamidessi (2003)) only the following rules of structural congruence are used:

(1) $P \equiv Q$ if $P \equiv_\alpha Q$
(2) $P \mid (\nu\, n)\, Q \equiv (\nu\, n)\, \big(P \mid Q\big)$ if $x \notin \mathsf{fn}(P)$
(3) $P \mid Q \equiv Q \mid P$

We define both the labelled and reduction semantics of π_{mix}, because we conveniently use them for different purposes. The labelled semantics is used in Section 3 to state a separation result in the style of Palamidessi (2003) while the reduction semantics of π_{mix} as well as its counterpart for π_{a} in Figure 4 are used in the style of Gorla (2010) to derive

$$\textsc{O-Sum}\quad \textstyle\sum_{i\in I}\pi_i.P_i \xrightarrow{\overline{x}\,y} P_j \quad \pi_j = \overline{x}\,\langle y\rangle \wedge j \in I$$

$$\textsc{I-Sum}\quad \textstyle\sum_{i\in I}\pi_i.P_i \xrightarrow{x\,y} \{\,{}^{y}/_{z}\,\}\,P_j \quad \pi_j = x\,(z) \wedge j \in I$$

$$\textsc{Rep}\quad x^*\,(z)\,.P \xrightarrow{x\,y} \{\,{}^{y}/_{z}\,\}\,P \mid x^*\,(z)\,.P$$

$$\tau\textsc{-Sum}\quad \textstyle\sum_{i\in I}\pi_i.P_i \xrightarrow{\tau} P_j \quad \pi_j = \tau \wedge j \in I$$

$$\textsc{Com}\quad \frac{P \xrightarrow{x\,y} P' \quad Q \xrightarrow{\overline{x}\,y} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\textsc{Close}\quad \frac{P \xrightarrow{x\,y} P' \quad Q \xrightarrow{\overline{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu\,y)\,(P' \mid Q')} \quad y \notin \mathsf{fn}(P)$$

$$\textsc{Par}\quad \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \quad \mathsf{bn}(\mu) \cap \mathsf{fn}(Q) = \emptyset$$

$$\textsc{Res}\quad \frac{P \xrightarrow{\mu} P'}{(\nu\,z)\,P \xrightarrow{\mu} (\nu\,z)\,P'} \quad z \notin \mathsf{n}(\mu)$$

$$\textsc{Open}\quad \frac{P \xrightarrow{\overline{x}\,y} P'}{(\nu\,y)\,P \xrightarrow{\overline{x}(y)} P'} \quad x \neq y \qquad \textsc{Cong}\quad \frac{P' \equiv P \quad P \xrightarrow{\mu} Q \quad Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$$

Fig. 2. Labelled semantics of π_{mix} and π_{sep}.

$$\textsc{Tau}_{\text{S}}\quad (\ldots + \tau.P + \ldots) \longmapsto_{\text{S}} P$$

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

$$\textsc{Rep}_{\text{S}}\quad y^*\,(x)\,.P \mid (\ldots + \overline{y}\,\langle z\rangle\,.Q + \ldots) \longmapsto_{\text{S}} \{\,{}^{z}/_{x}\,\}\,P \mid y^*\,(x)\,.P \mid Q$$

$$\textsc{Par}_{\text{S}}\quad \frac{P \longmapsto_{\text{S}} P'}{P \mid Q \longmapsto_{\text{S}} P' \mid Q} \qquad \textsc{Res}_{\text{S}}\quad \frac{P \longmapsto_{\text{S}} P'}{(\nu\,n)\,P \longmapsto_{\text{S}} (\nu\,n)\,P'}$$

$$\textsc{Cong}_{\text{S}}\quad \frac{P \equiv P' \quad P' \longmapsto_{\text{S}} Q' \quad Q' \equiv Q}{P \longmapsto_{\text{S}} Q}$$

Fig. 3. Reduction semantics of π_{mix} and π_{sep}.

a good encoding in Section 4. This also explains the indices within the two reduction semantics: S and T refer to source and target language, respectively (see Section 2.2).

A *network* is a process $(\nu\,\tilde{x})\,\big(P_1 \mid \ldots \mid P_n\big)$ for some $n \in \mathbb{N}$, $P_1, \ldots, P_n \in \mathcal{P}$ and $\tilde{x} \in \mathcal{T}\,(\mathcal{N})$. We refer to $P_1, \ldots, P_n$ as the processes of the network.

$$\textsc{Tau}_{\mathrm{T}}\ (\ldots + \tau.P + \ldots) \longmapsto_{\mathrm{T}} P$$

$$\textsc{Com}_{\mathrm{T}}\quad y\,(\tilde{x})\,.P \mid \overline{y}\,\langle\tilde{z}\rangle \longmapsto_{\mathrm{T}} \left\{ {}^{\tilde{z}}/_{\tilde{x}} \right\} P \quad |\tilde{x}| = |\tilde{z}|$$

$$\textsc{Rep}_{\mathrm{T}}\quad y^{*}\,(\tilde{x})\,.P \mid \overline{y}\,\langle\tilde{z}\rangle \longmapsto_{\mathrm{T}} \left\{ {}^{\tilde{z}}/_{\tilde{x}} \right\} P \mid y^{*}\,(\tilde{x})\,.P \quad |\tilde{x}| = |\tilde{z}|$$

$$\textsc{Par}_{\mathrm{T}}\quad \frac{P \longmapsto_{\mathrm{T}} P'}{P \mid Q \longmapsto_{\mathrm{T}} P' \mid Q} \qquad \textsc{Res}_{\mathrm{T}}\quad \frac{P \longmapsto_{\mathrm{T}} P'}{(\nu\, n)\,P \longmapsto_{\mathrm{T}} (\nu\, n)\,P'}$$

$$\textsc{Cong}_{\mathrm{T}}\quad \frac{P \equiv P' \quad P' \longmapsto_{\mathrm{T}} Q' \quad Q' \equiv Q}{P \longmapsto_{\mathrm{T}} Q}$$

Fig. 4. Reduction semantics of π_{a}.

We use σ, σ', σ_1, ... to range over substitutions. A substitution $\left\{ {}^{x_1}/_{y_1},\ldots,{}^{x_n}/_{y_n} \right\}$ is a mapping from names to names. The application of a substitution on a term P, denoted by $\left\{ {}^{x_1}/_{y_1},\ldots,{}^{x_n}/_{y_n} \right\}(P)$, 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 $\mathcal{N} \setminus \{\,y_1,\ldots,y_n\,\}$ the substitution behaves as the identity mapping. Let **id** denote identity, i.e. **id** is the empty substitution $\mathbf{id} = \varnothing$. We sometimes omit the brackets, i.e. $\sigma\,(P) = \sigma P$, and 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.

To avoid confusion, we use $\xrightarrow{\mu}$ with $\mu \in \mathcal{A}_\tau$ for steps within the labelled semantics and $\longmapsto$ within the reduction semantics. Moreover, let $P \longrightarrow (P \not\longrightarrow)$ and $P \longmapsto (P \not\longmapsto)$ respectively denote existence (non-existence) of a step from P, i.e. there is (no) $P' \in \mathcal{P}$ and (no) $\mu \in \mathcal{A}_\tau$ such that $P \xrightarrow{\mu} P'$ and $P \longmapsto P'$. A *(partial) execution* is a sequence of steps $P \xrightarrow{\mu_1,\ldots,\mu_n} P'$ such that $P \xrightarrow{\mu_1} H_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{n-1}} H_{n-1} \xrightarrow{\mu_n} P'$ for some $P, P', H_1, \ldots, H_{n-1} \in \mathcal{P}$ with the sequence $\mu_1,\ldots,\mu_n$ of observable and unobservable actions, i.e. $\mu_1,\ldots,\mu_n \in \mathcal{A}_\tau$. Accordingly, $P \xrightarrow{\tilde{\mu}} P' \not\longrightarrow$ denotes a finite execution from P to P' with the sequence of actions $\tilde{\mu} \in \mathcal{T}\,(\mathcal{A}_\tau)$. Moreover, let $\Longmapsto$ be the reflexive and transitive closure of $\longmapsto$ and let $\longmapsto^{\omega}$ define an infinite sequence of reduction steps, i.e. $P \longmapsto^{\omega}$ means $P \longmapsto\!\!\!\!\longmapsto \ldots$ for an infinite sequence of steps. Let $P\,(\rightsquigarrow)^n\,Q$ denote a sequence of n $\rightsquigarrow$-steps from P to Q for every kind of steps $\rightsquigarrow$; e.g. in case of $\longmapsto$-steps $P\,(\longmapsto)^3\,Q$ denotes a sequence of three reduction steps from P to Q, i.e. $P \longmapsto\!\!\!\!\longmapsto\!\!\!\!\longmapsto Q$.

If $P \xrightarrow{\mu}$ for some $\mu \neq \tau$ then P can perform a visible action. In this case, P has an *input observable* a, denoted by $P \downarrow_a$, if μ is an input action with subject a, i.e. $\mu \in \{\,a\,b \mid a,b \in \mathcal{N}\,\}$, and P has an *output observable* $\overline{a}$, denoted by $P \downarrow_{\overline{a}}$, if μ is an output action with subject a, i.e. $\mu \in \{\,\overline{a}\,b, \overline{a}\,(b) \mid a,b \in \mathcal{N}\,\}$. Observables are often also called *barbs* (Milner and Sangiorgi 1992).

Comparing the three calculi, we observe that at least for their monadic versions their sets of terms form a strict inclusion chain. Moreover, the operational semantics for all three calculi is essentially the same – modulo the usual consistency between labelled τ-steps and reduction steps. Accordingly, these three versions of the π-calculus naturally

form a hierarchy with the full π-calculus at the top and its asynchronous variant at the bottom.

Corollary 2.4 (hierarchy). π_{mix} is at least as expressive as π_{sep} and π_{sep} is at least as expressive as π_{a}.

One main purpose of this paper is to discuss to what extent the expressive power of these calculi are equal and to what extent the inclusions are strict.

2.1.1. *Match.* π_{mix} and its variants are often presented without a match operator. We present all variants including match, because (1) we need the match operator in Section 4 to obtain an encoding from π_{mix} into π_{a} and (2) we want a hierarchy (at least for the monadic versions) of the calculi. However, note that none of the results in Section 3 rely on this decision, i.e. all presented results of Section 3 hold also if match is removed from one or all considered calculi.

2.2. *Quality criteria for encodings*

To compare two languages by means of translational expressiveness, we either distinguish them by a separation result or relate them by an encodability result. A separation result distinguishes two languages by showing that there is a term – or a class of terms – of the source language that cannot be encoded into the target language via some 'good' encoding. An encodability result shows that the target language is as expressive as the source language by showing that there is a 'good' encoding from the source into the target language. In both directions, the notion of 'good' encoding, i.e. the quality of the encoding, is crucial.

To prove separation results, the set of criteria shall be minimal to strengthen the significance of the result. In contrast, encodability results are strengthened by stricter constraints, i.e. the set of criteria shall be maximal. There is no agreement yet about the criteria for language comparison – neither for encodability, nor for separation – which naturally leads to incomparable results (Boer and Palamidessi 1991; Nestmann 2006; Parrow 2008; Shapiro 1989, 1991, 1992).

A widely used criterion is based on the notion of *full abstraction*: the preservation and reflection of equivalences associated to the two compared languages. By definition, the relevance of this notion depends on the involved equivalences. Since there are lots of different equivalences (van Glabbeek 1993, 2001) – as also for the π-calculus and its variants – one may, for the very same encoding, arrive at both encodability and separation results, simply by varying the respective choice of equivalence for the source and target language. As for the set of criteria itself, there is no agreement about what kinds of equivalence are well suited for language comparison.

To overcome this problem, Gorla (2010) identifies five criteria to build a possibly more robust and uniform approach to compare languages. Instead of notions of equivalence, Gorla's approach focuses on the notion of computation and related capabilities. Here, there is no need to provide a notion of equivalence for the source language. There is, though, a need to provide a notion of equivalence for the target language to be able to

abstract from certain low-level artefacts that the encoding may produce. In the following, we recall the five criteria as presented for instance in Gorla (2010).

To distinguish source and target language, the former is denoted by $\mathcal{L}_S$ – as long as no specific source language is given – while the latter is denoted by $\mathcal{L}_T$. The indices S and T are then deployed along the definitions of relations and rules on the languages to distinguish those of the source from those of the target language. Both languages are defined by their set of terms possibly up to some notion of structural congruence as we did for π_{mix} and π_a in Section 2.1 and their reduction semantics. Moreover, we require that both languages contain a special operator ✓ called *success* within their syntax. As explained further below, this operator allows to compare the behaviour of source and target terms by means of testing in a very general, i.e. not domain-specific, way. We also require that the parallel composition operator | is binary and subject to the same operational semantics in both languages, as is the case in our setting. Let $S, S', S_1, \ldots$ range over processes of the source language $\mathcal{L}_S$ and $T, T', T_1, \ldots$ range over processes of the target language $\mathcal{L}_T$.

Additionally, on the target language, we assume a behavioural equivalence $\asymp_T$. Its purpose is to describe the abstract behaviour of a target process, where abstract basically means with respect to the behaviour of the source terms. Therefore, it should abstract from 'junk' left over by the encoding while mimicking the behaviour of the source term (compare to Definition 2.8). Note that the choice of this equivalence is still crucial in the sense that choosing a trivial form of equivalence allows for meaningless encodings. By Gorla (2010) a 'good' equivalence $\asymp_T$ is often defined in the form of a barbed equivalence (as described e.g. in Milner and Sangiorgi (1992)) or can be derived directly from the reduction semantics and is often a congruence, at least with respect to parallel composition.

The five conditions on a good encoding are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*, i.e. that the encoding does not depend on specific names within the respective source term. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection* and (5) *success sensitiveness*.

Intuitively, an encoding is compositional if the translation of an operator is the same for all possible sets of parameters. To mediate between the translations of the parameters the encoding defines a unique context for each operator, whose arity is the arity of the operator. Moreover the context can be parametrized on the free names of the respective source term.

Definition 2.5 (criterion 1: compositionality). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ is *compositional* if, for every k-ary operator **op** of $\mathcal{L}_S$ and for every subset of names N, there exists a k-ary context $\mathcal{C}^N_{\mathbf{op}}(\_1; \ldots; \_k)$ 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 $\triangleleft \mathbf{op}(S_1, \ldots, S_k) \triangleright = \mathcal{C}^N_{\mathbf{op}}(\triangleleft S_1 \triangleright; \ldots; \triangleleft S_k \triangleright)$.

In Section 3, we refer to a separation result of Palamidessi (Palamidessi 2003) where, instead of compositionality, the stronger condition of homomorphic translation of the parallel operator is assumed. In the context of variants of the π-calculus, the homomorphic translation of the parallel operator enjoys the pleasant property of preserving the *degree of*

distribution, i.e. encoding the parallel operator homomorphically ensures that the degree of distribution of the source and the target term are the same. However, it is also a possibly too limiting condition as it rules out some well-accepted encodings; Gorla (2010) specifically mentions (Baldamus *et al.* 2005; Bugliesi and Giunti 2007; Nestmann 2000). Compositionality, as defined above, may allow to change the degree of distribution but it does not enforce this. Thus, we rely on this weaker condition and additionally require, whenever needed, an argumentation on a case-by-case basis that the encoding does not significantly change the degree of distribution.

The second structural criterion states that the encoding should not depend on specific names used within the respective source term. Of course, an encoding that translates each name to itself simply preserves this condition. However, it is sometimes necessary and meaningful to translate a name into a sequence of names or to reserve a couple of names for the encoding, i.e. to give them a special function within the encoding. To ensure that there are no conflicts between (i) the names used by the encoding function for special purposes and (ii) the names used by the source term, the encoding is enriched with a renaming policy, i.e. a substitution from names into sequences of names. If we allow a name to be translated into a sequence of names, we have to require that (1) each such sequence is finite – such that the translated names can be handled by the target language – and that (2) the sequences associated to two different names are of the same length and do not have names in common – such that the encoding cannot link the translations of different names nor build up a hierarchy on the translated names.

Definition 2.6 (renaming policy). Let $n \in \mathbb{N}$ be a positive natural number. A substitution $\varphi^n_{\triangleleft\cdot\triangleright} : \mathcal{N} \to \mathcal{N}^n$ is a *renaming policy* if, for all $u, v \in \mathcal{N}$ such that $u \neq v$, it holds that $\varphi^n_{\triangleleft\cdot\triangleright}(u) \cap \varphi^n_{\triangleleft\cdot\triangleright}(v) = \varnothing$, where $\varphi^n_{\triangleleft\cdot\triangleright}(\cdot)$ is considered as a set.

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.

Definition 2.7 (criterion 2: name invariance). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ is *name invariant* if, for every S and σ, it holds that

$$\triangleleft \sigma(S) \triangleright \begin{cases} \equiv \sigma'(\triangleleft S \triangleright) & \text{if } \sigma \text{ is injective} \\ \asymp_T \sigma'(\triangleleft S \triangleright) & \text{otherwise} \end{cases}$$

where σ' is such that $\varphi^n_{\triangleleft\cdot\triangleright}(\sigma(a)) = \sigma'\left(\varphi^n_{\triangleleft\cdot\triangleright}(a)\right)$ for every $a \in \mathcal{N}$.

Note that substitutions may always induce alpha-conversion to avoid name capture. Since we have no control of the names used within such an alpha-conversion $\triangleleft \sigma(S) \triangleright$ and $\sigma'(\triangleleft S \triangleright)$ in the first case can only be compared up to alpha-conversion which explains the use of structural congruence instead of equality[†]. The use of $\asymp_T$ in the second case is included by Gorla to allow for non-injective substitutions.

[†] In Gorla (2010) we find equality at this position but Gorla states before at pages 3 and 4 that processes are usually identified up to some notion of structural congruence.

The first semantic criterion is operational correspondence, which consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be mimicked by its translation, i.e. the translation does not reduce the computations of the source term. *Soundness* requires that every computation of a target term corresponds to some computation of the respective source term, i.e. the translation does not introduce new computations.

Definition 2.8 (criterion 3: (weak) operational correspondence). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ is *(weak) operationally corresponding* if it is

Complete: for all $S \Longmapsto_S S'$, it holds that $\triangleleft S \triangleright \Longmapsto_T \asymp_T \triangleleft S' \triangleright$;
Sound: for all $\triangleleft S \triangleright \Longmapsto_T T$, there exists an S' such that $S \Longmapsto_S S'$ and $T \Longmapsto_T \asymp_T \triangleleft S' \triangleright$.

Note that the Definition of operational correspondence relies on the equivalence $\asymp_T$ to get rid of junk possibly left over within computations of target terms. For instance in (Fu and Lu 2010) a slightly different formulation of operational correspondence is used.

Definition 2.9 (strong operational correspondence). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ is *strong operationally corresponding* if it is

Complete: for all $S \Longmapsto_S S'$, it holds that $\triangleleft S \triangleright \Longmapsto_T \asymp_T \triangleleft S' \triangleright$;
Sound: for all $\triangleleft S \triangleright \Longmapsto_T T$, there exists an S' such that $S \Longmapsto_S S'$ and $T \asymp_T \triangleleft S' \triangleright$.

The only difference between these two versions is the requirement on T within the soundness condition. Here, the second definition is significantly stricter, because it rules out the existence of 'intermediate states', as we discuss in Section 4.5. Sometimes, we refer to the completeness criterion of weak or strong operational correspondence as (weak) or strong operational completeness and, accordingly, for the soundness criterion we use the terms (weak) or strong operational soundness.

The next criterion concerns the role of infinite computations in encodings.

Definition 2.10 (criterion 4: divergence reflection). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ reflects divergence if, for every S, $\triangleleft S \triangleright \longmapsto_T^\omega$ implies $S \longmapsto_S^\omega$.

With the last criterion, the behaviour of the source terms is linked to the behaviour of the target terms in order to except unreasonable encodings. With Gorla (2010), we assume a *success* operator ✓ to be part of the syntax of both the source and the target language. Likewise, for the encoding presented in Section 4 we add ✓ to the syntax of π_{mix} in Definition 2.1 and of π_a in Definition 2.3. Since ✓ cannot be further reduced, the operational semantics is left unchanged in both cases. Moreover, note that $\mathsf{n}(\checkmark) = \mathsf{fn}(\checkmark) = \mathsf{bn}(\checkmark) = \varnothing$, so also interplay of ✓ with the $\equiv$-rules is smooth and does not require explicit treatment. The test for reachability of success is standard.

Definition 2.11 (success). A process $P \in \mathcal{L}$ *may lead to success*, denoted as $P \Downarrow$, if (and only if) it is reducible to a process containing a top-level unguarded occurrence of ✓, i.e.

$$\exists P', P'' \in \mathcal{L} \,.\, P \Longmapsto P' \wedge P' \equiv P'' \mid \checkmark$$

Note that we use the modal operator may. With that we get may testing. Other modal operators lead to different forms of testing and, thus, to possibly different results concerning translational expressiveness. So, again, the choice of this modal operator is crucial. Each result, regardless whether it is an encodability or a separation result, needs a discussion of the involved modal operator within the definition of test for success. At best, the quality of the encoding or the separation result is robust w.r.t. different modal operators.

Finally, an encoding preserves the behaviour of the source term if it and its respective target term answer the tests for success in exactly the same way.

Definition 2.12 (criterion 5: success sensitiveness). A translation $\triangleleft \cdot \triangleright : \mathcal{L}_S \to \mathcal{L}_T$ is *success sensitive* if, for every S, $S \Downarrow$ if and only if $\triangleleft S \triangleright \Downarrow$.

Note that this criterion only links the behaviours of source and target terms but not of their continuations. To do so we relate success sensitiveness and operational correspondence by requiring that the equivalence on the target language never relates two processes P and Q such that $P \Downarrow$ and $Q \not\Downarrow$.

Definition 2.13 (success respecting). An equivalence $\asymp \subseteq \mathcal{L} \times \mathcal{L}$ is *success respecting* if, for every P and Q such that $P \Downarrow$ and $Q \not\Downarrow$, it holds that $P \not\asymp Q$.

3. The role of symmetry

In this section, we discuss the role of symmetry for the expressive power of π_{mix} in comparison to π_{sep}. Therefore, we first revisit the result of Palamidessi (2003) and generalise its absolute part. Then, we show that we can prove several translational separation results due to different definitions of reasonableness as simple consequences of our absolute result.

3.1. *Semantic versus syntactic symmetry*

Palamidessi (2003) proved that π_{mix} is strictly more expressive than π_{sep} by proving that the former can solve leader election in symmetric networks while the latter cannot. The leader election problem consists of choosing a leader among the processes of a network. In Palamidessi (2003), a special channel *out* is assumed to propagate the index of the winning process, i.e. the leader. The leader election problem is solved by a network iff in each of its executions each process propagates the same process index over *out* and no other index is propagated.

As already Bougé did for *CSP* in Bougé (1988), Palamidessi uses a *semantic* definition of symmetry. Intuitively, the *syntactic component* of the symmetry definition in Bougé (1988), Palamidessi (2003) and Vigliotti *et al.* (2007) states two processes as symmetric iff they are identical modulo some renaming according to a permutation σ on their free names. Bougé (1988) argues why a syntactic notion of symmetry does not suffice[†] considering the

[†] Compare Johnson and Schneider: 'Symmetry means different things to different people'. (Johnson and Schneider 1985).

leader election problem to distinguish $CSP_{\mathrm{i/o}}$, i.e. *CSP* where input and output commands may appear in guards, and CSP_{in}, i.e. *CSP* where only input commands may appear in guards. He presents two networks in CSP_{in} each solving leader election although each should be considered as syntactically symmetric. The following example presents such a syntactically symmetric network solving leader election in π_{sep}:

$$N \triangleq P \mid \sigma(P) \text{ with } P = \overline{x} \mid \left(x.\overline{out}\,\langle 1\rangle + y.\overline{out}\,\langle 2\rangle\right) \text{ and } \sigma = \{\, {}^{x}/_{y}, {}^{y}/_{x} \,\} \tag{1}$$

N is syntactically symmetric with respect to the permutation σ, i.e. $N = P_1 \mid P_2$ and P_2 is equal to P_1 modulo the exchange of x and y according to σ. Moreover N solves the leader election problem. To rule out such examples, the *semantic component* of the symmetry definition is designed to be strongly connected to the problem considered, i.e. leader election in this case. Intuitively, its purpose is to ensure that the only way to solve the leader election problem is to break the initial symmetry of the given network. Note that N does *not* break the initial syntactic symmetry, because e.g. in the execution $N \xrightarrow{\tau} P \mid \overline{out}\,\langle 1\rangle \xrightarrow{\tau} \overline{out}\,\langle 1\rangle \mid \overline{out}\,\langle 1\rangle \xrightarrow{\overline{out}\,1} \mathbf{0} \mid \overline{out}\,\langle 1\rangle \xrightarrow{\overline{out}\,1} \mathbf{0} \mid \mathbf{0} \not\longrightarrow$ each second step results in a network that is syntactically symmetric with respect to σ. So, without this semantic part in the definition of symmetry, the leader election problem cannot be used to distinguish π_{mix} and π_{sep} (or $CSP_{\mathrm{i/o}}$ and CSP_{in}).

3.1.1. *Semantic symmetry.* We revisit Palamidessi's notion of symmetry for the π-calculus as of (Palamidessi 2003). Note that the involved definitions are based on the ones introduced by Bougé (1988) for *CSP*. According to Palamidessi (2003), a hypergraph is a tuple $H = \langle N, X, t\rangle$, where N and X are finite sets whose elements are called nodes and edges, and t, called type, is a function assigning to each edge the set of nodes connected by this edge. An automorphism on a hypergraph is a pair $\sigma = \langle \sigma_N, \sigma_X\rangle$ such that $\sigma_N : N \to N$ and $\sigma_X : X \to X$ are permutations which preserve the type of edges. Given a hypergraph H and σ on H the orbit of a name n is the set of nodes in which the iterations of σ map n.

A network $P \equiv (\nu\, \tilde{x})\left(P_1 \mid \ldots \mid P_k\right)$ of k processes solves the leader election problem if for every computation of P there exists an extension of the computation and there exists an index $n \in \{\, 1,\ldots,k \,\}$ such that for each process the extended computation contains one output action of the form $\overline{out}\, n$ and no other action $\overline{out}\, m$ with $m \neq n$. The hypergraph associated to a network P is the hypergraph $H(P) = \langle N, X, t\rangle$ with $N = \{\, 1,\ldots,k \,\}$, $X = \mathsf{fn}\left(P_1 \mid \ldots \mid P_k\right) \setminus \{\, out \,\}$, and for each $x \in X$, $t(x) = \{\, n \mid x \in \mathsf{fn}(P_n) \,\}$. Given a network P and the hypergraph $H(P)$ associated to P, an automorphism on P is any automorphism $\sigma = \langle \sigma_N, \sigma_X\rangle$ on $H(P)$ such that σ_X coincides with σ_N on $N \cap X$ and σ_X preserves the distinction between free and bound names.

A network P with the associated hypergraph $H(P) = \langle N, X, t\rangle$ and an automorphism σ on P is symmetric with respect to σ iff for each node $i \in N$, $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$[†] holds. To distinguish π_{mix} and π_{sep} Palamidessi shows that a network $P \in \mathcal{P}_{\mathrm{sep}}$ which is symmetric

[†] In Bougé (1988) and Vigliotti *et al.* (2007) formally slightly different conditions but with the same effect are used.

with respect to an automorphism σ on P with only one orbit cannot solve the leader election problem while this is possible in π_{mix}.

The main point of the semantic component of symmetry is that the special channel *out* cannot be renamed by σ while the indices of the processes of the network must be permuted by σ. With that, the network N in (1) above is *not* symmetric according to (Palamidessi 2003). This allows Palamidessi to prove that for each execution of a network in $\mathcal{P}_{\text{sep}}$, which is symmetric with respect to an automorphism σ, whenever there is an output action $\overline{out}\, i$ there is an output $\overline{out}\, \sigma(i)$ with $\sigma(i) \neq i$ as well, which contradicts the leader election problem. This explains why in Bougé (1988), Palamidessi (2003) and Vigliotti *et al.* (2007) such an effort is spent to define symmetry.

Nevertheless it turns out that we do not need the leader election problem to distinguish π_{mix} and π_{sep}. The main argument in the proof of Palamidessi (2003) that there is no symmetric network in $\mathcal{P}_{\text{sep}}$ solving leader election is that it is impossible in π_{sep} to break symmetries.

3.1.2. *Syntactic symmetry.* As mentioned in the introduction, we directly focus on the problem of breaking symmetries instead of concentrating on leader election. Thus, we can release most of the above conditions for symmetry. Moreover, we abandon the notion of hypergraphs and automorphisms. Instead, we use a simple syntactic definition of symmetry that, as mentioned above, states two processes as symmetric iff they are identical modulo some renaming according to a permutation σ on their free names.

Definition 3.1 (symmetry relation). A *symmetry relation of degree n* is a permutation $\sigma : \mathcal{N} \to \mathcal{N}$, such that $\sigma^n = \mathbf{id}$.

Let $\text{Sym}(n, \mathcal{N})$ denote the set of symmetry relations of degree n over $\mathcal{N}$ and let $\sigma^0 = \mathbf{id}$.

Note that this definition does not require that n is the minimal degree of σ; consequently, the condition that σ is an automorphism with only one orbit is released. A symmetric network is then a network of n processes that are equal except for some renaming according to a symmetry relation σ.

Definition 3.2 (symmetric network). Let $P \in \mathcal{P}$. Let sequence $\tilde{x}$ contain only free names of P. Let $n \in \mathbb{N}$. Let σ be a symmetry relation of degree n over $\mathcal{N} \setminus \mathsf{bn}(P)$. Let $\tilde{x}$ be closed under σ. Then

$$\left[P\right]_{\sigma}^{n,\tilde{x}} = (\nu\, \tilde{x}) \left(\sigma^0(P) \mid \ldots \mid \sigma^{n-1}(P) \right)$$

is a *symmetric network of degree n.*

In contrast to (Palamidessi 2003), we consider the network N of (1) as symmetric network, because $\sigma = \{ {}^{x}/_{y}, {}^{y}/_{x} \}$ is a symmetry relation of degree 2 and thus $N = \left[P\right]_{\sigma}^{2}$. Note that, in the following proofs, we make use of the fact that names bound in P are bound in each other process of $\left[P\right]_{\sigma}^{n,\tilde{x}}$ as well, so we explicitly forbid alpha-conversion here. In the following, whenever we assume some symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$, we implicitly assume the respectively quantified parameters: a process $P \in \mathcal{P}$, a sequence $\tilde{x}$ of free names of P, a network size $n \in \mathbb{N}$, and a symmetry relation σ of degree n over $\mathcal{N} \setminus \mathsf{bn}(P)$.

The main difference of our definition to the definition of a symmetric network in Palamidessi (2003) is that, in Palamidessi (2003), the processes of a symmetric network are numbered consecutively and for each process P_i within the symmetric network $P_{\sigma(i)} \equiv \sigma(P_i)$ holds, i.e. the symmetry relation *additionally* has to permute the indices of the processes. Accordingly, to obtain a symmetric network in the sense of Palamidessi (2003) from (1), we have to unify σ with the permutation $\{ {}^{1}/_{2}, {}^{2}/_{1} \}$. But then, of course, N does not solve leader election anymore. Thus, each symmetric network in (Palamidessi 2003) is a symmetric network for our definition, but not vice versa. Our definition of symmetry is weaker.

We use an index-guided form of substitution to replace single processes within a symmetric network.

Definition 3.3 (indexed substitution). Let $[P]_{\sigma}^{n,\tilde{x}}$ be a symmetric network. An *indexed substitution* of some processes within a symmetric network, denoted by the term $\{ i_1 \mapsto Q_1, \ldots, i_m \mapsto Q_m \} [P]_{\sigma}^{n,\tilde{x}}$ for some processes $Q_1, \ldots, Q_m \in \mathcal{P}$ and $i_1, \ldots, i_m \in \{ 0, \ldots, n-1 \}$ such that for all $j,k \in \{ 1, \ldots, m \}$, $j \neq k$ implies $i_j \neq i_k$, is the result of exchanging $\sigma^{i_k}(P)$ in $[P]_{\sigma}^{n,\tilde{x}}$ by Q_k for all $k \in \{ 1, \ldots, m \}$.

Obviously $\{ i_1 \mapsto Q_1, \ldots, i_m \mapsto Q_m \} [P]_{\sigma}^{n,\tilde{x}}$ is a network; in general, however, it is not symmetric with respect to σ.

3.2. *Symmetric executions*

We explicitly prove that in π_{sep} it is not possible to break initial symmetries, i.e. starting with a symmetric network there is always at least one execution preserving the symmetry. We refer to such an execution as *symmetric execution*. Let us consider a symmetric network $[P]_{\sigma}^{n,\tilde{x}}$ of degree n. Of course, if only one process does a step on its own, then all the other processes of the network can mimic this step and thus restore symmetry. So, there is a symmetry preserving execution if there is no communication between the processes of the network. The most interesting case is how the symmetry is restored after a communication between two processes of the network has temporarily destroyed it. Both cases are reflected in the proof of Theorem 3.7.

Apart from symmetric networks, we use the notion of a symmetric sequence of actions. Similarly to symmetric networks, in which a symmetry relation is applied to processes to derive symmetric processes, a symmetric sequence of actions is the result of applying a symmetry relation to action labels. It is sometimes necessary to translate a bound output action to an according unbound output action because a network can send a bound name several times but only the first of this outputs will be bound.

Definition 3.4 (symmetric sequence of actions). Let $\mu \in \mathcal{A}_{\tau}$ be an action label, let $\tilde{x} \in \mathcal{T}(\mathcal{N})$ be a sequence of names and σ a symmetry relation of degree $n \in \mathbb{N}$. Then $[\mu]_{\sigma}^{n,\tilde{x}}$ denotes

the sequence $\mu_1, \ldots, \mu_n$ of n labels such that $\mu_1, \ldots, \mu_n \in \mathcal{A}_\tau$, $\mu_1 = \mu$ and for $i \in \{ 2, \ldots, n \}$:

$$\mu_i = \begin{cases} \tau, & \text{if } \mu = \tau \\ \sigma^i(a)\, b, & \text{if } \mu = a\, b \\ \overline{\sigma^i(a)}\, \sigma^i(b), & \text{if } \mu = \overline{a}\, b \text{ or } \big(\mu = \overline{a}(b) \text{ and } \\ & \qquad \sigma^i(b) \notin \tilde{x} \setminus \{ b, \sigma(b), \ldots, \sigma^{i-1}(b) \}\big) \\ \overline{\sigma^i(a)}\, \big(\sigma^i(b)\big), & \text{if } \mu = \overline{a}(b) \text{ and } \sigma^i(b) \in \tilde{x} \setminus \{ b, \sigma(b), \ldots, \sigma^{i-1}(b) \} \end{cases}$$

Sometimes we refer to $\mu_2, \ldots, \mu_n$ as the symmetric counterparts of μ.

Intuitively, a symmetric execution is an execution starting from a symmetric network returning to a symmetric network after any nth step, and which is either infinite or terminates in a symmetric network. Thereby, each sequence of n steps is labelled by a symmetric sequence of actions.

Definition 3.5 (symmetric execution). A *symmetric execution* is either a finite execution of length $m \cdot n \in \mathbb{N}$

$$\big[P\big]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \big[P_1\big]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \ldots \xrightarrow{[\mu_m]_{\sigma_m}^{n,\tilde{x}_{m-1}}} \big[P_m\big]_{\sigma_m}^{n,\tilde{x}_m} \not\longrightarrow$$

for some $P_1, \ldots, P_m \in \mathcal{P}$, $\mu_1, \ldots, \mu_m \in \mathcal{A}_\tau$, $\tilde{x}_1, \ldots, \tilde{x}_m \in \mathcal{T}(\mathcal{N})$ and some $\sigma_1, \ldots, \sigma_m \in \mathrm{Sym}(n, \mathcal{N})$ such that $\sigma \subseteq \sigma_1 \subseteq \ldots \subseteq \sigma_m$ or an infinite execution

$$\big[P\big]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \big[P_1\big]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \big[P_2\big]_{\sigma_2}^{n,\tilde{x}_2} \xrightarrow{[\mu_3]_{\sigma_3}^{n,\tilde{x}_2}} \ldots.$$

for some $P_1, P_2, \ldots \in \mathcal{P}$, $\mu_1, \mu_2, \ldots \in \mathcal{A}_\tau$, $\tilde{x}_1, \tilde{x}_2, \ldots \in \mathcal{T}(\mathcal{N})$ and $\sigma_1, \sigma_2, \ldots \in \mathrm{Sym}(n, \mathcal{N})$ such that $\sigma \subseteq \sigma_1 \subseteq \sigma_2 \subseteq \ldots$.

Note that because of $\sigma \subseteq \sigma_1 \subseteq \ldots$ the symmetry relation can only increase during a symmetric execution such that existing symmetries are preserved. Moreover – as shown in Lemma 3.8 – the symmetry relation does only grow in the presence of bound output to capture the renaming done by alpha-conversion. In the absence of bound output we have $\sigma = \sigma_1 = \ldots = \sigma_m$ and $\sigma = \sigma_1 = \sigma_2 = \ldots$ respectively.

Palamidessi proved that π_{sep} enjoys a *confluence* property (Palamidessi 2003). Let $\overline{x}\,[y]$ denote an output action, i.e. $\overline{x}\,[y]$ is either a bound output $\overline{x}\,(y)$ or an unbound output $\overline{x}\,y$.

Lemma 3.6 (confluence). Let $P \in \mathcal{P}_{\mathrm{sep}}$ be a process. If P can make two steps $P \xrightarrow{\overline{x}[y]} Q$ and $P \xrightarrow{z\,w} R$ then there exists S such that $Q \xrightarrow{z\,w} S$ and $R \xrightarrow{\overline{x}[y]} S$.

The proof of this lemma is by analysis of the possible rules used to derive the steps and by the fact that an input and an output guarded term cannot be combined within a sum in π_{sep}.

Proof of Lemma 3.6. See proof of Lemma 4.1 in Palamidessi (2003) at pages 17–18. □

Intuitively, the confluence lemma states the impossibility in π_{sep}, that an output-step immediately withdraws the possibility to perform a formerly alternative input-step, and

vice versa. Obviously, this is not a problem in case the output and the input are combined in a mixed choice. Hence, the confluence lemma is itself an absolute result distinguishing π_{mix} and π_{sep}. Moreover, it is crucial for both of the other absolute results: leader election in symmetric networks in Palamidessi (2003) as well as for breaking symmetries in the following proof. It ensures that a communication of two processes of a network cannot immediately withdraw the possibility of all other network processes to mimic this communication. So, why do we need an absolute result on top of confluence? The answer is, it is not necessary but it is extremely helpful to derive translational results. To use an absolute result in a translational result, we derive a problem instance from the absolute result that we can use as counterexample in the translational result and ensure that the discriminating properties of this example are preserved by the criteria required for 'good' encodings (compare to Section 3.3). It is not easy to obtain such an example directly from confluence and, in case of the criteria used in Section 3.3, it is very hard and intricate to argue for the preservation of its relevant properties. Deriving more complex absolute results on top of confluence may need some effort, but it makes the derivation of translational results a magnitude easier.

So, we use confluence to prove that it is not possible to break symmetries in π_{sep}. Intuitively, we show that there is at least one symmetric execution by proving that whenever there is a step destroying symmetry we can restore it in $n-1$ more steps mimicking the first step. The respective existence relies on the standard lemma in process calculi like the π-calculus that transitions are preserved under substitution. As conclusion, it is not possible in π_{sep} to break an initial symmetry in all executions.

Theorem 3.7 (symmetric execution). No symmetric network in $\mathcal{P}_{\text{sep}}$ can break its symmetry within a single step, i.e. every symmetric network in $\mathcal{P}_{\text{sep}}$ has at least one symmetric execution.

Proof of Theorem 3.7 First we prove that given an arbitrary symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$ in $\mathcal{P}_{\text{sep}}$, whenever $\left[P\right]_{\sigma}^{n,\tilde{x}}$ can perform a step then there are exactly $n-1$ more steps that restore symmetry, i.e. that lead to a symmetric network again and the corresponding n steps are labelled by a sequence of symmetric actions. Note that the main line of argumentation of this lemma is very similar to the proof of Theorem 4.2 in Palamidessi (2003) at pages 18–23, although we prove a completely different statement. Nevertheless, due to the different formulations of the statements, also the proofs differ in technical details.

Lemma 3.8.

$$\begin{aligned}
&\forall n \in \mathbb{N} \,.\, \forall \tilde{x} \in \mathcal{T}(\mathcal{N}) \,.\, \forall P \in \mathcal{P}_{\text{sep}} \,.\, \forall \sigma \in \text{Sym}\left(n, \mathcal{N} \setminus \mathsf{bn}(P)\right) \,.\, \forall \mu \in \mathcal{A}_{\tau} \,.\\
&\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P} \text{ implies}\\
&\quad \exists P' \in \mathcal{P}_{\text{sep}} \,.\, \exists \tilde{x}' \in \mathcal{T}(\mathcal{N}) \,.\, \exists \mu_2, \ldots, \mu_n \in \mathcal{A}_{\tau} \,.\, \exists \sigma' \in \text{Sym}(n, \mathcal{N}) \,.\\
&\quad \widehat{P} \xrightarrow{\mu_2, \ldots, \mu_n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'} \text{ and } \mu, \mu_2, \ldots, \mu_n = [\mu]_{\sigma'}^{n,\tilde{x}} \text{ and } \sigma \subseteq \sigma'
\end{aligned}$$

Proof of Lemma 3.8. $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P}$ can be the result of either an internal μ-step of one process of the network, i.e. it can be produced without the rules Com or Close, or of a communication between two processes of the network, i.e. be produced by one of the rules Com or Close. In the first case, only one process performs a step and the rest of the network remains equal, i.e.:

$$\begin{aligned} &\exists i \in \{\, 0,\ldots,n{-}1 \,\} \;.\; \exists H \in \mathcal{P}_{\text{sep}} \;.\; \exists \tilde{x}_1 \in \mathcal{T}\left(\mathcal{N}\right) \;.\; \sigma^i\left(P\right) \xrightarrow{\mu} H \\ &\qquad \text{and } \widehat{P} \equiv \{\, i \mapsto H \,\} \left[P\right]_{\sigma}^{n,\tilde{x}_1} \end{aligned} \tag{C1}$$

In the second case, $\mu = \tau$ and two processes of the network change, i.e.:

$$\begin{aligned} &\exists i,j \in \{\, 0,\ldots,n{-}1 \,\} \;.\; \exists H_1,H_2 \in \mathcal{P}_{\text{sep}} \;.\; \exists z,z' \in \mathcal{N} \;.\; i \neq j \\ &\quad \text{and } \left(\sigma^i\left(P\right) \mid \sigma^j\left(P\right) \xrightarrow{\tau} H_1 \mid H_2 \;\text{ or }\; \sigma^i\left(P\right) \mid \sigma^j\left(P\right) \xrightarrow{\tau} \left(\nu\, z,z'\right)\left(H_1 \mid H_2\right)\right) \\ &\quad \text{and } \widehat{P} \equiv \{\, i \mapsto H_1, j \mapsto H_2 \,\} \left[P\right]_{\sigma'}^{n,\tilde{x}'} \end{aligned} \tag{C2}$$

We proceed with a case split.

Case (C1). Within the symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$, for $i \in \{\, 0,\ldots,n{-}1 \,\}$ all processes $\sigma^i\left(P\right)$ are equal except for some renaming of free names according to σ. Thus, whenever a process $\sigma^i\left(P\right)$ can perform a step $\xrightarrow{\mu}$ then each other process $\sigma^k\left(P\right)$ of the network can mimic this step by $\xrightarrow{\mu'}$, where μ' is the result of applying σ^{k-i+n} to μ possibly by changing bound output to unbound output as described in Definition 3.4.[†] The case of a bound output action μ is slightly tricky, so we consider the other cases first.

If μ is no bound output, then we can choose the labels $\mu_2,\ldots,\mu_n$ such that the sequence $\mu,\mu_2,\ldots,\mu_n$ is equal to $[\mu]_{\sigma}^{n,\tilde{x}}$. Moreover, by symmetry $\sigma^i\left(P\right) \xrightarrow{\mu} H$ implies $\sigma^k\left(P\right) \xrightarrow{\mu'} \sigma^{k-i+n}\left(H\right)$ for a $H \in \mathcal{P}_{\text{sep}}$. With it, we can restore symmetry by mimicking the μ-step of process $\sigma^i\left(P\right)$ by the $n{-}1$ steps $\sigma^{i+1}\left(P\right) \xrightarrow{\mu_2} \sigma\left(H\right)$, …, $\sigma^{n-1}\left(P\right) \xrightarrow{\mu_{n-i}} \sigma^{n-1-i}\left(H\right)$, $\sigma^0\left(P\right) \xrightarrow{\mu_{n-i+1}} \sigma^{n-i}\left(H\right)$, …, $\sigma^{i-1}\left(P\right) \xrightarrow{\mu_n} \sigma^{n-1}\left(H\right)$. These n steps build the chain

$$\begin{aligned} \left[P\right]_{\sigma}^{n,\tilde{x}} \;&\xrightarrow{\mu}\; \left(\nu\,\tilde{x}_1\right)\left(\sigma^0\left(P\right) \mid \ldots \mid \sigma^{i-1}\left(P\right) \mid \sigma^0\left(H\right) \mid \sigma^{i+1}\left(P\right) \mid \ldots \right. \\ &\qquad\qquad \left. \mid \sigma^{n-1}\left(P\right)\right) \\ &\;\;\vdots \\ &\xrightarrow{\mu_{n-i}}\; \left(\nu\,\tilde{x}_{n-i}\right)\left(\sigma^0\left(P\right) \mid \ldots \mid \sigma^{i-1}\left(P\right) \mid \sigma^0\left(H\right) \mid \ldots \right. \\ &\qquad\qquad \left. \mid \sigma^{n-1-i}\left(H\right)\right) \\ &\xrightarrow{\mu_{n-i+1}}\; \left(\nu\,\tilde{x}_{n-i+1}\right)\left(\sigma^{n-i}\left(H\right) \mid \sigma\left(P\right) \mid \ldots \mid \sigma^{i-1}\left(P\right) \mid \sigma^0\left(H\right) \right. \\ &\qquad\qquad \left. \mid \ldots \mid \sigma^{n-1-i}\left(H\right)\right) \\ &\;\;\vdots \\ &\xrightarrow{\mu_n}\; \left(\nu\,\tilde{x}_n\right)\left(\sigma^{n-i}\left(H\right) \mid \ldots \mid \sigma^{n-1}\left(H\right) \mid \sigma^0\left(H\right) \mid \ldots \right. \\ &\qquad\qquad \left. \mid \sigma^{n-1-i}\left(H\right)\right) \end{aligned}$$

[†] Note that n is added in $k{-}i{+}n$ and $k{-}j{+}n$ just to ensure that both values are positive. Because $\sigma^n = \mathbf{id}$ if $k{-}i \geqslant 0$ we have $\sigma^{k-i+n} = \sigma^{k-i}$.

with $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathcal{T}(\mathcal{N})$ and $\tilde{x}' = \tilde{x}_n$. Because of $\sigma^n = \mathbf{id}$ after the last step, we result in a network which is again symmetric with respect to σ, i.e. we choose $\sigma' = \sigma$. With that, we can choose $P' = \sigma^{n-i}(H)$ such that $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P} \xrightarrow{\mu_2,\ldots,\mu_n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'}$.

If μ is an input or an unbound output action, then so are its symmetric counterparts $\mu_2, \ldots, \mu_n$. We choose $\tilde{x}' = \tilde{x}_1 = \ldots = \tilde{x}_n = \tilde{x}$ and are done.

If μ is a bound output action $\overline{y}(z)$, then we have to consider two cases.

Case $z \notin \mathsf{bn}\left(\sigma^i(P)\right)$**.** Here, $z \in \mathsf{fn}\left(\sigma^i(P)\right)$ and because μ is a bound output z must be in $\tilde{x}$. So we have to choose $\tilde{x}_1 = \tilde{x} \setminus \{ z \}$. Then, by Definition 3.4 some of the actions $\mu_2, \ldots, \mu_n$ might be bound and some might be unbound outputs depending on whether $\sigma^{j-1}(z)$ was already the subject of an earlier bound output of this sequence or not. If $\sigma^{j-1}(z)$ of μ_j was already the subject of an bound output within $\mu, \mu_2, \ldots, \mu_{j-1}$, then μ_j is an unbound output and we choose $\tilde{x}_j = \tilde{x}_{j-1}$, else μ_j is a bound output and we choose $\tilde{x}_j = \tilde{x}_{j-1} \setminus \sigma^{j-1}(z)$ for all $j \in \{ 2, \ldots, n \}$. Again, we can choose $\sigma' = \sigma$ and $P' = \sigma^{n-i}(H)$ and proceed as in the case where μ is no bound output.

Case $z \in \mathsf{bn}\left(\sigma^i(P)\right)$**.** Here, by symmetry, $\sigma^j(z) = z$ is bound in $\sigma^{i+j}(P)$ for all $j \in \{ 0, \ldots, n-1 \}$. By the above assumption that there are no name clashes (except for the duplicate binding of names), we conclude $z \notin \tilde{x}$. Then, by Definition 3.4, $\mu, \mu_2, \ldots, \mu_n$ is a sequence of n bound output actions. Each of these actions μ_j changes the scope of $\sigma^{i+j-1}(P)$ (in a symmetric way to the other processes) but the scope of the network is left unchanged. So, we can again choose $\tilde{x}' = \tilde{x}_1 = \ldots = \tilde{x}_n = \tilde{x}$. The crux is that performing the first bound output with label μ may force an alpha-conversion to avoid name clashes to the other bound instances of z in the other processes of the network such that the symmetry is destroyed. To illustrate this problem, let us consider an example:

Example 3.9. Let

$$N \triangleq (\nu\, x)\, \overline{a}\langle x \rangle.\overline{x} \mid (\nu\, x)\, \overline{a}\langle x \rangle.\overline{x} = \left[(\nu\, x)\, \overline{a}\langle x \rangle.\overline{x} \right]_{\mathbf{id}}^{2}.$$

N can perform two bound outputs $\overline{a}(x)$. To avoid name capture we have to apply alpha-conversion such that we have $N \xrightarrow{\overline{a}(x)} \overline{x} \mid \left(\nu\, x'\right) \overline{a}\langle x' \rangle.\overline{x'} \xrightarrow{\overline{a}(x')} \overline{x} \mid \overline{x'}$. Because of this alpha-conversion, we result in a network which is not symmetric with respect to **id**. Nevertheless, the first step is mimicked by the second step and thus both parts of the network behave symmetrically. As consequence to our intuition, the resulting network should be again considered as symmetric network. To overcome this problem, we record the renaming done by alpha-conversion in $\sigma' = \left\{ {}^{x}/_{x'}, {}^{x'}/_{x} \right\}$ such that $\overline{x} \mid \overline{x'} = \left[\overline{x}\right]_{\sigma'}^{2}$. Note that because of this σ' can only increase by adding permutations on formerly bound names and fresh names.

That is why we have to increase the symmetry relation in this case to keep track of the renaming done by alpha-conversion. Thereto, we enforce the alpha-conversion after the first bound output to rename all instances of z (except the first one) to a different fresh name for each process of the network and add the respective permutations of z to σ in order to obtain σ' such that $\sigma \subseteq \sigma'$. Afterwards, we can

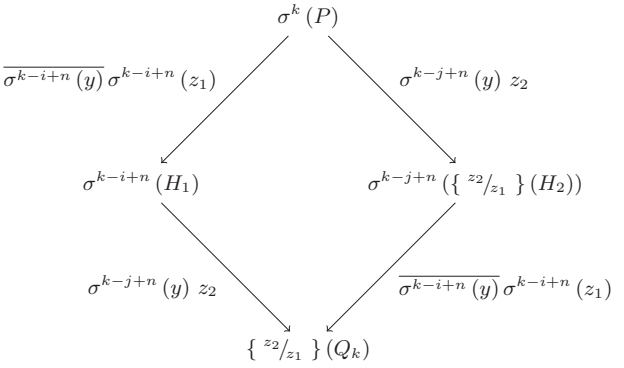

Fig. 5. Local confluence of receiving and sending actions.

choose $\mu_2, \ldots, \mu_n$ such that $\mu, \mu_2, \ldots, \mu_n = [\mu]^{n,\tilde{x}}_{\sigma'}$ and $P' = \sigma'^{n-i}(H)$ and proceed as in the case where μ is no bound output.

Case (C2). In this case, there is a communication between $\sigma^i(P)$ and $\sigma^j(P)$ as result of one of the rules Com or Close. Without loss of generality, let us assume that $\sigma^i(P)$ is the sender and $\sigma^j(P)$ is the receiver of this communication, i.e. there are $y, z_1, z_2 \in \mathcal{N}$ such that $\sigma^i(P) \xrightarrow{\overline{y}\,z_1} H_1$ (or $\sigma^i(P) \xrightarrow{\overline{y}(z_1)} H_1$) and $\sigma^j(P) \xrightarrow{y\,z_2} \{\ ^{z_2}/_{z_1}\ \}(H_2)$. Because of symmetry, each process $\sigma^k(P)$ for $0 \leqslant k \leqslant n-1$ can perform an output action $\mu_{\mathsf{out},k} = \overline{\sigma^{k-i+n}(y)}\,\sigma^{k-i+n}(z_1)$ (or $\mu_{\mathsf{out},k} = \overline{\sigma^{k-i+n}(y)}\,(\sigma^{k-i+n}(z_1))$ for bound output) and an input action $\mu_{\mathsf{in},k} = \sigma^{k-j+n}(y)\,z_2$ such that for each process $\sigma^k(P) \xrightarrow{\mu_{\mathsf{out},k}} \sigma^{k-i+n}(H_1)$ and $\sigma^k(P) \xrightarrow{\mu_{\mathsf{in},k}} \sigma^{k-j+n}(\{\ ^{z_2}/_{z_1}\ \}(H_2))$. Because of the confluence Lemma 3.6, i.e. without mixed choice an output action cannot block an alternatively input action (within one step) and vice versa, as depicted in Figure 5 (case of unbound output) process $\sigma^k(P)$ must be able to perform both actions consecutively in arbitrary order resulting in the same term which we denote by Q_k. Indeed without mixed choice the only possibility for $\sigma^k(P)$ to be able to perform both actions is that these two actions are composed in parallel, so $\sigma^k(P)$ can perform both actions in an arbitrary order and it is not possible that performing one of these actions alone prevents $\sigma^k(P)$ form performing the other one next. To restore symmetry, we build a chain of n steps such that each process $\sigma^k(P)$ performs the output action $\mu_{\mathsf{out},k}$ in step $((k-i+n) \bmod n)+1$ and the input action $\mu_{\mathsf{in},k}$ in step $((k-j+n) \bmod n)+1$, i.e. each process is once a sender and once a receiver and $\mu = \mu_2 = \cdots = \mu_n = \tau$ and with that $\mu, \mu_2, \ldots, \mu_n = [\mu]^{n,\tilde{x}}_{\sigma}$. Again, we consider the case of unbound outputs first. Then, we have $\left[P\right]^{n,\tilde{x}}_{\sigma} \xrightarrow{\tau} \{\ i \mapsto H_1, j \mapsto H_2\ \}\left[P\right]^{n,\tilde{x}}_{\sigma}$ as first step with $\sigma^i(P) \xrightarrow{\overline{y}\,z_1} H_1$, $\sigma^j(P) \xrightarrow{y\,z_2} \{\ ^{z_2}/_{z_1}\ \}(H_2)$ and $\sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} H_1 \mid H_2$. Depending on the values of i and j, some of the processes perform the corresponding input action first while others perform at first the corresponding output action. Because of Lemma 3.6, both is possible. We let each process perform exactly these two actions (compare to Figure 5). We choose $P' = Q_0$ and $\tilde{x}' = \tilde{x}$. We start with a symmetric network and all processes behave symmetrically, i.e. each process mimic the behaviour of its neighbour, so we have $Q_k = \sigma^k(Q_0)$ for all k with $0 \leqslant k \leqslant n-1$ such that can

choose $\sigma' = \sigma$ and have

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\tau}{}^{n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'}.$$

Now, we consider the case of bound outputs. Note that $\sigma^i(P)$ and $\sigma^j(P)$ perform a communication step *within* the network, so if $\sigma^i(P)$ performs a bound output z_1 must be bound in $\sigma^i(P)$, i.e. $z_1 \notin \tilde{x}$. By symmetry $\sigma^l(z_1) = z_1$ is bound in $\sigma^{i+l}(P)$ for all $l \in \{\, 0, \ldots, n{-}1 \,\}$. With that either all output actions are bound or all are unbound. In case of bound output we have $\sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} \left(\nu\, z, z'\right)(H_1 \mid H_2)$, because first we have to apply alpha-conversion to rename the instance of z_1 bound in $\sigma^j(P)$ and then the bound output by $\sigma^i(P)$ leads to a scope extrusion such that $z = z_1$ and z' is the renaming of z_1 in $\sigma^j(P)$. Again we use alpha-conversion after the first communication step to rename all instances of z_1 (except the first) to a different fresh name for each process of the network and add the respective permutations of z_1 to σ in order to obtain σ' such that $\sigma \subseteq \sigma'$. Let $z_{1,2}, \ldots, z_{1,n}$ denote the sequence of names used to rename z_1 according to σ'. We proceed as in the case of unbound outputs with the $n{-}1$ communication steps as described above. Of course we have to replace the processes $\sigma^k(P)$ by $\left\{ {}^{z_{1,2}}/_{z_1}, \ldots, {}^{z_1}/_{z_{1,n}} \right\}^{k-i}(P)$ and $\mu_{\mathsf{out},k}$ by $\overline{\sigma^{k-i+n}(y)}\left[\left\{ {}^{z_{1,2}}/_{z_1}, \ldots, {}^{z_1}/_{z_{1,n}} \right\}^{k-i}(z_1)\right]$ for $0 \leqslant k \leqslant n{-}1$. After completing these n communication steps the names $z_1, z_{1,2}, \ldots, z_{1,n}$ are pulled outwards by scope extrusion, i.e. we have

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\tau} \{\, i \mapsto H_1, j \mapsto H_2 \,\} \left[P\right]_{\sigma}^{n,\tilde{x},z_1,z_{1,2}} \xrightarrow{\tau}{}^{n-1} \left[R\right]_{\sigma'}^{n,\tilde{x}'},$$

where $\tilde{x}' = \tilde{x}, z_1, z_{1,2}, \ldots, z_{1,n}$ and $P \xrightarrow{\overline{\sigma^{n-i}(y)}(z_1),\sigma^{n-j}(y)\, z_2} R$. With that we can choose $P' = R$ and are done.

□

With Lemma 3.8, we can now construct the symmetric execution. We start with an arbitrary symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$. If $\left[P\right]_{\sigma}^{n,\tilde{x}} \not\longrightarrow$ we have a symmetric execution of length 0. Otherwise, if $\left[P\right]_{\sigma}^{n,\tilde{x}}$ can perform a step labelled by μ_1 by Lemma 3.8 we can perform $n{-}1$ more steps such that $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1}$. Now we can proceed alike with $\left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1}$ and result either in a finite symmetric execution of length n or we have $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \left[P_2\right]_{\sigma_2}^{n,\tilde{x}_2}$. By recursively repeating this argument, we either get a finite or an infinite symmetric execution. □

3.2.1. *Breaking symmetries.* Note that Theorem 3.7 does not state anything about encodability and it does not need a notion of reasonableness either. Instead, it just states without any precondition that every symmetric network in $\mathcal{P}_{\mathsf{sep}}$ has at least one symmetric execution. In contrast, there are symmetric networks in $\mathcal{P}_{\mathsf{mix}}$ without such a symmetric execution, as the following example shows. Consider the network

$$(\nu\, x, y)\left(P \mid \sigma\,(P)\right) \quad \text{with} \quad P = \overline{x}.\overline{1} + y.\overline{2} \quad \text{and} \quad \sigma = \left\{ {}^{x}/_{y}, {}^{y}/_{x}, {}^{1}/_{2}, {}^{2}/_{1} \right\}$$

with $\sigma^2 = \mathbf{id}$, i.e. $(\nu\, x, y)\left(P \mid \sigma(P)\right)$ is a symmetric network in $\mathcal{P}_{\text{mix}}$. It has, modulo structural congruence, exactly the two following executions

$$(\nu\, x, y)\left(P \mid \sigma(P)\right) \xrightarrow{\tau} \overline{1} \mid \overline{1} \xrightarrow{\overline{1}} \overline{1} \xrightarrow{\overline{1}} \mathbf{0}$$

$$(\nu\, x, y)\left(P \mid \sigma(P)\right) \xrightarrow{\tau} \overline{2} \mid \overline{2} \xrightarrow{\overline{2}} \overline{2} \xrightarrow{\overline{2}} \mathbf{0}$$

and even none of them is symmetric; the initial symmetry is broken. So Theorem 3.7 proves a difference in the absolute expressive power between π_{sep} and π_{mix}.[†]

Conclusion 1. The π-calculus with mixed choice (π_{mix}) is strictly more expressive than the π-calculus without mixed choice (π_{sep}).

3.3. *Non-existence of uniform encodings*

As done by Palamidessi (2003) and also by Gorla (2008b), we now also prove that there is no uniform and reasonable encoding from π_{mix} into π_{sep}, but here using Theorem 3.7 which states a difference in the absolute expressive power of the two calculi. It is no real surprise that this absolute result leads to differences in the translational expressiveness of the languages. Because uniform encodings preserve symmetries – or at least enough of the symmetric nature of the terms – , the non-existence of a uniform and reasonable encoding is a natural consequence of the difference in their absolute expressiveness. Unfortunately, there is no agreement on the minimal requirements of a reasonable encoding, so we cannot formally prove this result in general, although we believe that it holds for any meaningful definition of reasonableness. Instead, to underpin our assertion, we prove it in the settings of Palamidessi (2003) and Gorla (2008b).

According to Palamidessi (2003), an encoding is uniform if it translates the parallel operator homomorphically and preserves renamings, i.e. for all permutations of names σ there exists a permutation of names θ such that $\triangleleft \sigma(P) \triangleright = \theta(\triangleleft P \triangleright)$. Vigliotti *et al.* (2007) additionally require that the permutations σ and θ are compatible on observables. Gorla (2008b) does not use the notion of uniformity, but in his first setting the separation result between π_{mix} and π_{sep} does also assume homomorphic translation of the parallel operator. Moreover, he specifies name invariance as a criterion for a good encoding, which is a more complex condition than Palamidessi's second condition. It turns out that, in our setting, we do not need a second condition like renaming preservation or name invariance, because we base our counterexamples in the following separation results on symmetric networks of the form $P \mid P$ as already Gorla did in Gorla (2008b). For us, an encoding is uniform iff it translates the parallel operator homomorphically.

Definition 3.10 (uniform encoding). An encoding $\triangleleft \cdot \triangleright$ from π_{mix} into an other language is a *uniform encoding* if and only if for all $P, Q \in \mathcal{P}_{\text{mix}}$

$$\triangleleft P \mid Q \triangleright = \triangleleft P \triangleright \mid \triangleleft Q \triangleright \tag{U}$$

[†] Remember that $\mathcal{P}_{\text{sep}}$ is a subset of $\mathcal{P}_{\text{mix}}$, so π_{mix} is at least as expressive as π_{sep}.

Actually, Theorem 3.7 should suffice to prove that there cannot be a uniform and reasonable encoding from π_{mix} into π_{sep}, because uniform encodings preserve symmetries and it is possible to break symmetries in π_{mix} while this is not possible in π_{sep}. The crux is that there is no commonly accepted notion of reasonableness. For separation results, we seek a definition of reasonableness that is as weak as possible. But, without any notion of reasonableness, the theorem would not hold, because there are uniform encodings from π_{mix} into π_{sep}. For instance, we could simply translate everything to $\mathbf{0}$ (modulo $\equiv$). Of course such an encoding makes no sense and so hardly anyone would call it reasonable. Usually, an encoding is called reasonable if it preserves some kind of behaviour or the ability to solve some kind of problem so to ensure that the purpose of the original term is preserved. In the following, we consider three different notions of reasonableness.

3.3.1. *Version 1.* For Palamidessi, an encoding is reasonable if it preserves the relevant observables and termination properties (Palamidessi 2003). Implicitly, she requires that a reasonable encoding should at least preserve the ability to solve leader election. We do alike but with a different interpretation of what it means to solve leader election that is more closely related to the definition used by Bougé (1988): a network is said to solve leader election iff in each execution exactly one process propagates itself as leader while all the other processes propagate themselves as slaves. We assume the existence of two different predetermined output actions, one to propagate as leader (μ_l) and the other to propagate as slave (μ_s). Moreover, we require that for both output actions neither the channel names nor the sent values are bound within the network[†].

Definition 3.11 (solving leader election). Let $\mu_l, \mu_s \in \mathcal{A}$ be two different output action labels, i.e. $\mu_l \neq \mu_s$. A network N of size n *solves leader election* if every maximal execution of N contains exactly one step labelled by μ_l and $n-1$ steps labelled by μ_s and all names of μ_l and μ_s are free in N.

The main difference to the definition of leader election used in Palamidessi (2003) is that here the slaves do not have to know the identity, i.e. the index, of the leader. So, this definition is usually considered as a weaker notion of the leader election problem. However, the leader may inform all its slave about its identity after election. An encoding is now said to be reasonable iff it preserves the ability to solve the leader election problem.

Definition 3.12 (1-reasonableness). An encoding $\triangleleft \cdot \triangleright : \mathcal{P}_{\text{mix}} \to \mathcal{P}_{\text{sep}}$ is *1-reasonable*, if $\triangleleft P \triangleright$ solves leader election if and only if P solves leader election for all $P \in \mathcal{P}_{\text{mix}}$.

To prove that there is no uniform and reasonable encoding, we force our encoding to lead to a network of two processes that is symmetric with respect to identity. By Theorem 3.7, this network has at least one symmetric execution. Because we use the identity as symmetry relation, in the symmetric execution both processes behave exactly the same such that if one of them propagates himself as leader then the other one does alike, which contradicts leader election.

[†] Note that if we allow bound names in these output actions, we could hardly predetermine them.

Theorem 3.13 (separation result). There is no uniform and 1-reasonable encoding from π_{mix} into π_{sep}.

Proof of Theorem 3.13 Let us assume the contrary, i.e. there is a uniform and 1-reasonable encoding $\triangleleft \cdot \triangleright$ from π_{mix} into π_{sep}. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P \triangleq a.\overline{slave} + \overline{a}.\overline{leader}$$

Obviously $\sigma = \mathbf{id}$ is a symmetry relation of degree 2. With that the network $N = \left[a.\overline{slave} + \overline{a}.\overline{leader}\right]^2_\sigma$ is a symmetric network. Moreover, N solves leader election, because the leader sends an empty message over channel *leader* and all slaves send an empty message over channel *slave*. By Definition 3.10 of uniformity, we have $\triangleleft P \mid P \triangleright \overset{(U)}{=} \triangleleft P \triangleright \mid \triangleleft P \triangleright = \left[\triangleleft\ P\ \triangleright\right]^2_{\mathbf{id}}$, i.e. $\triangleleft N \triangleright$ is again a symmetric network of degree 2 with **id** as symmetry relation. By Theorem 3.7, $\triangleleft N \triangleright$ has at least one symmetric execution and by reasonableness $\triangleleft N \triangleright$ must solve leader election, i.e. there is exactly one process that propagates itself as leader by an output action. Let μ_l denote this send action. By Definition 3.5, a symmetric execution has symmetric sequences of actions, i.e. the action μ_l is coupled to its symmetric counterpart building the sequence $[\mu_l]^{2,\tilde{z}'}_{\sigma'}$ for some $\tilde{z}' \in \mathcal{T}(\mathcal{N})$ and $\sigma' \in \text{Sym}(2,\mathcal{N})$. By construction in the proof of Lemma 3.8, and because we start with **id**, we know that σ' consists of (permutations of) names that are bound in $\triangleleft N \triangleright$ or fresh. Because, by Definition 3.11, μ_l can neither contain fresh nor bound names, we conclude $[\mu_l]^{2,\tilde{z}'}_{\sigma'} = \mu_l, \mu_l$, i.e. the output action appears twice in the symmetric execution. With that two processes propagate themselves as leader, which is a contradiction. □

Note that, in contrast to the proof of Palamidessi (Palamidessi 2003; Vigliotti *et al.* 2007), we do not have to assume that the encoding is renaming preserving.

3.3.2. *Version 2.* Here, we first introduce a technical lemma. Intuitively, it states that the symmetric execution of a symmetric network of degree n, where n is *not* the minimal degree of the corresponding symmetry relation, can be subdivided into symmetric executions on symmetric subnetworks of the original network.

Lemma 3.14. Let $\left[P_0\right]^{n,\tilde{x}}_\sigma$ be a symmetric network in $\mathcal{P}_{\text{sep}}$. If the degree of σ is not minimal, i.e. if there is a $n' \in \mathbb{N}$ with $0 < n' < n$ such that $\sigma^{n'} = \mathbf{id}$, then $\left[P_0\right]^{n,\tilde{x}}_\sigma$ has a finite or an infinite symmetric execution

$$\left[P_0\right]^{n,\tilde{x}}_\sigma \xrightarrow{[\mu_1]^{n,\tilde{x}}_{\sigma_1}} \left[P_1\right]^{n,\tilde{x}_1}_{\sigma_1} \xrightarrow{[\mu_2]^{n,\tilde{x}_1}_{\sigma_2}} \ldots \xrightarrow{[\mu_m]^{n,\tilde{x}_{m-1}}_{\sigma_m}} \left[P_m\right]^{n,\tilde{x}_m}_{\sigma_m} \not\longrightarrow$$

$$\text{or} \quad \left[P_0\right]^{n,\tilde{x}}_\sigma \xrightarrow{[\mu_1]^{n,\tilde{x}}_{\sigma_1}} \left[P_1\right]^{n,\tilde{x}_1}_{\sigma_1} \xrightarrow{[\mu_2]^{n,\tilde{x}_1}_{\sigma_2}} \ldots$$

for a $m \in \mathbb{N}$, $P_1, \ldots, P_m \in \mathcal{P}_{\text{sep}}$, $\sigma_1, \ldots, \sigma_m \in \text{Sym}(n,\mathcal{N})$ with $\sigma \subseteq \sigma_1 \subseteq \ldots \subseteq \sigma_m$, $\tilde{x}_1, \ldots, \tilde{x}_m \in \mathcal{T}(\mathcal{N})$ and $\mu_1, \ldots, \mu_m \in \mathcal{A}_\tau$ or some $P_1, P_2, \ldots \in \mathcal{P}_{\text{sep}}$, $\sigma_1, \sigma_2, \ldots \in \text{Sym}(n,\mathcal{N})$ with $\sigma \subseteq \sigma_1 \subseteq \sigma_2 \subseteq \ldots$, some $\tilde{x}_1, \tilde{x}_2, \ldots \in \mathcal{T}(\mathcal{N})$ and $\mu_1, \mu_2, \ldots \in \mathcal{A}_\tau$ respectively such that $\left[P_0\right]^{n',\tilde{x}}_\sigma$ has the

finite or infinite symmetric execution

$$\left[P_0\right]_{\sigma}^{n',\tilde{x}'} \xrightarrow{\left[\mu_1'\right]_{\sigma_1'}^{n',\tilde{x}'}} \left[P_1\right]_{\sigma_1'}^{n',\tilde{x}_1'} \xrightarrow{\left[\mu_2'\right]_{\sigma_2'}^{n',\tilde{x}_1'}} \cdots \xrightarrow{\left[\mu_m'\right]_{\sigma_m'}^{n',\tilde{x}_{m-1}'}} \left[P_m\right]_{\sigma_m'}^{n',\tilde{x}_m'} \not\longrightarrow$$

$$\text{or} \quad \left[P_0\right]_{\sigma}^{n',\tilde{x}} \xrightarrow{\left[\mu_1'\right]_{\sigma_1'}^{n',\tilde{x}}} \left[P_1\right]_{\sigma_1'}^{n',\tilde{x}_1'} \xrightarrow{\left[\mu_2'\right]_{\sigma_2'}^{n',\tilde{x}_1'}} \cdots$$

for some $\tilde{x}_1', \ldots, \tilde{x}_m' \in \mathcal{T}(\mathcal{N})$, $\mu_1', \ldots, \mu_m' \in \mathcal{A}_\tau$ and $\sigma_1', \ldots, \sigma_m' \in \mathrm{Sym}\left(n', \mathcal{N}\right)$ with $\sigma \subseteq \sigma_1' \subseteq \ldots \subseteq \sigma_m'$ or some $\tilde{x}_1', \tilde{x}_2', \ldots \in \mathcal{T}(\mathcal{N})$, $\mu_1', \mu_2', \ldots \in \mathcal{A}_\tau$ and $\sigma_1', \sigma_2', \ldots \in \mathrm{Sym}\left(n', \mathcal{N}\right)$ with $\sigma \subseteq \sigma_1' \subseteq \sigma_2' \subseteq \ldots$ respectively such that $\tilde{x}'$ is a subsequence of $\tilde{x}$, $\tilde{x}_i'$ is a subsequence of $\tilde{x}_i$ and either μ_i' or if μ_i' is a bound output its unbound variant is in $[\mu_i]_{\sigma_i}^{n,\tilde{x}_{i-1}}$ for all $i \in \{ 1, \ldots, m \}$ or $i \in \mathbb{N}$ respectively.

Note that, like Theorem 3.7, this result is absolute in the sense that it holds independently of any notion of uniformity or reasonableness.

The proof is based on the following observation: every network of degree n that is symmetric with respect to a symmetry relation σ such that n is *not* the minimal degree of σ can be subdivided into several identical symmetric networks with respect to σ. Then, an induction on the number of sequences of n steps from a symmetric network to a symmetric network is performed. The inductive step is proved by a case analysis on whether the first step of such a sequence is due to an action of only one process of the network or to a communication between two processes.

Proof of Lemma 3.14 Assume there is a $0 < n' < n$ such that $\sigma^{n'} = \mathbf{id}$. Then because $\sigma^n = \mathbf{id}$ there must be a $k \in \mathbb{N}$ such that $n = k * n'$. Because $\sigma^0 = \sigma^{n'} = \sigma^{i*n'}$ for each $i \in \{ 1, \ldots, k \}$ we have $\sigma^j = \sigma^{j+n'}$. So $\left[P'\right]_{\sigma}^{n,\tilde{x}}$ can be divided into k identical symmetric networks such that $\left[P'\right]_{\sigma}^{n,\tilde{x}} = \left[P\right]_{\sigma}^{n',\tilde{x}} \mid \ldots \mid \left[P\right]_{\sigma}^{n',\tilde{x}}$ and $\left[\mu'\right]_{\sigma}^{n,\tilde{x}}$ can be divided in k identical sequences such that $\left[\mu'\right]_{\sigma}^{n,\tilde{x}} = \left[\mu'\right]_{\sigma}^{n',\tilde{x}}, \ldots, \left[\mu'\right]_{\sigma}^{n',\tilde{x}}$ for each $P' \in \mathcal{P}_{\mathrm{sep}}$ and each $\mu' \in \mathcal{A}_\tau$.

If $\left[P_0\right]_{\sigma}^{n,\tilde{x}}$ has a symmetric execution of length 0, i.e. $\left[P_0\right]_{\sigma}^{n,\tilde{x}} \not\longrightarrow$, then of course we have $\left[P_0\right]_{\sigma}^{n',\tilde{x}} \not\longrightarrow$ as well and so $\left[P_0\right]_{\sigma}^{n',\tilde{x}}$ has a symmetric execution of length 0.

Else we consider an arbitrary sequence of n steps

$$\left[P_k\right]_{\sigma_k}^{n,\tilde{x}_k} \xrightarrow{\left[\mu_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_k}} \left[P_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_{k+1}}$$

of the given symmetric execution for $k \in \{ 0, \ldots, m \}$ in the case of a finite symmetric execution and $k \in \mathbb{N}$ for an infinite symmetric execution. As constructed in Theorem 3.7 P_{k+1} is either the result of a step of $\sigma_k^i(P_k)$ realized without the rules Com and Close (C1) or it is the result of two communications of $\sigma_k^i(P_k)$ and $\sigma_k^j(P_k)$ realized by one of the rules Com or Close (C2). We proceed with a case split.

Case (C1): Let $\sigma_k^i(P_k)$ with $i \in \{ 0, \ldots, n-1 \}$ be the process which performs the first of the n steps labelled μ_{k+1}. We choose μ_{k+1}' as the $n-i$th action in $[\mu_{k+1}]_{\sigma_{k+1}}^{n,\tilde{x}_k}$, i.e. we choose the label of the action performed by process P_k. If μ_{k+1} is a bound output and μ_{k+1}' is not then we choose the bound output variant of μ_{k+1}'. By construction in the proof of Lemma 3.8 there are n' steps performed by the processes $\sigma_k^0(P_k), \ldots,$

$\sigma_k^{n'-1}(P_k)$ and labelled by the first n' labels of $\left[\mu'_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_k}$. Note that because σ'_k differs from σ_k only on permutations on formerly bound names we can perform these steps by ${\sigma'_k}^0(P_k), \ldots, {\sigma'_k}^{n'-1}(P_k)$, too. If μ_k is no bound output we can choose $\tilde{x}'_{k+1} = \tilde{x}'_k$ and $\sigma'_{k+1} = \sigma'_k$ and are done. Else if $\mu_k = \overline{y}\,z$ and $z \notin \mathsf{bn}\left(\sigma_k^i(P_k)\right)$ we can choose $\sigma'_{k+1} = \sigma'_k$ and $\tilde{x}'_{k+1}$ as the sequence of names in $\tilde{x}'_k, z_1, \ldots, z_l$, where $z_1, \ldots, z_l$ are the values of the bound outputs in $\left[\mu'_{k+1}\right]_{\sigma'_{k+1}}^{n',\tilde{x}_k}$. Else if $z \in \mathsf{bn}\left(\sigma_k^i(P_k)\right)$ we can choose $\tilde{x}'_{k+1} = \tilde{x}'_k$ and we add the permutations of z done by alpha-conversion as described in Lemma 3.8 to σ'_k to obtain σ'_{k+1}. Again by construction in Lemma 3.8 performing these n' steps we have $\left[P_k\right]_{\sigma'_k}^{n',\tilde{x}'_k} \xrightarrow{\left[\mu'_{k+1}\right]_{\sigma'_{k+1}}^{n',\tilde{x}'_k}} \left[P_{k+1}\right]_{\sigma'_{k+1}}^{n',\tilde{x}'_{k+1}}$.

Case (C2): Then $\left[\mu_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_k}$ is a sequence of n times τ. We choose $\mu'_{k+1} = \mu_{k+1} = \tau$ and $\left[\mu'_{k+1}\right]_{\sigma_k}^{n',\tilde{x}_k}$ is a sequence of n' times τ. Let $\sigma_k^i(P_k)$ and $\sigma_k^j(P_k)$ with $i, j \in \{0, \ldots, n-1\}$ be the processes which perform the first of the n steps. Without loss of generality let $\sigma_k^i(P_k)$ be the sender and $\sigma_k^j(P_k)$ be the receiver, i.e. $\sigma_k^i(P_k)$ performs an output action $\overline{\gamma}$ and $\sigma_k^j(P_k)$ performs the complementary receiving action γ. By construction in the proof of Theorem 3.7 the first n' steps within $\left[P_k\right]_{\sigma_k}^{n,\tilde{x}_k} \xrightarrow{\left[\mu_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_k}} \left[P_{k+1}\right]_{\sigma_{k+1}}^{n,\tilde{x}_{k+1}}$ are performed by the senders $\sigma_k^i(P_k), \ldots, \sigma_k^{i+n'-1}(P_k)$ in this order sending $[\overline{\gamma}]_{\sigma_{k+1}}^{n',\tilde{x}_k}$ respectively and by the receivers $\sigma_k^j(P_k), \ldots, \sigma_k^{j+n'-1}(P_k)$ in this order receiving $[\gamma]_{\sigma_{k+1}}^{n',\tilde{x}_k}$. Now because of $\sigma^{n'} = \mathbf{id}$ and σ'_k differs from σ only by formerly bound names and their renamings according to alpha-conversion for each $g \in \{i, \ldots, i+n'-1\}$ and for each $h \in \{j, \ldots, j+n'-1\}$ we have $\sigma_k^g(P_k)$ and ${\sigma'_k}^{g \bmod n'}(P_k)$, and $\sigma_k^h(P_k)$ and ${\sigma'_k}^{h \bmod n'}(P_k)$ respectively are equal modulo the renaming performed by formerly alpha-conversion. With that we can again close the cycle as in the proof of Lemma 3.8 leading to $\left[P_k\right]_{\sigma_k}^{n',\tilde{x}'_k} \xrightarrow{\left[\mu'_{k+1}\right]_{\sigma'_{k+1}}^{n',\tilde{x}'_k}} \left[P_{k+1}\right]_{\sigma'_{k+1}}^{n',\tilde{x}'_{k+1}}$, where $\tilde{x}'_{k+1}$ and σ'_{k+1} are obtained from $\tilde{x}'_k$ and σ'_k as described in Lemma 3.8.

Because we can subdivide an arbitrary sequence of n steps we can subdivide each such sequence in the symmetric execution and with it the symmetric execution. □

Gorla (2008b) defines the reasonableness of an encoding by the properties operational correspondence, divergence reflection and success sensitiveness. We use just the last of his properties instantiated with must testing. So we implicitly require divergence reflection. According to Gorla (2008b), success is represented by a process ✓, that is a part of the source and the target language of the encoding and always appears unbound. More precisely, a process must-succeeds if it *always* reduces to a process containing a top-level unguarded occurrence of ✓. The fact that P must-succeeds is denoted by $P \downarrow\downarrow$. With it, an encoding is reasonable if the encoding of a term must-succeeds iff the term itself must-succeeds.

Definition 3.15 (2-reasonableness). An encoding $\triangleleft \cdot \triangleright : \mathcal{P}_{\mathrm{mix}} \to \mathcal{P}_{\mathrm{sep}}$ is *2-reasonable*, if $P \downarrow\downarrow$ iff $\triangleleft P \triangleright \downarrow\downarrow$ for all $P \in \mathcal{P}_{\mathrm{mix}}$.

Again, we choose a term such that the encoding results in a network of the form $Q \mid Q$ in $\mathcal{P}_{\text{sep}}$ that is symmetric with respect to identity. In this case, we take advantage of the fact that the minimal degree of **id** is less than the degree of the network such that we can use Lemma 3.14 to subdivide the symmetric execution. With it already Q can perform the same sequence of steps as each process in $Q \mid Q$ performs in the symmetric execution.

Theorem 3.16 (separation result). There is no uniform and 2-reasonable encoding from π_{mix} into π_{sep}.

Proof of Theorem 3.16 Let us assume the contrary, i.e. there is a uniform and 2-reasonable encoding $\triangleleft \cdot \triangleright$ from π_{mix} into π_{sep}. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P \triangleq a.\mathbf{0} + \overline{a}.\checkmark$$

Obviously, $\sigma = \mathbf{id}$ is a symmetry relation of degree 2 and so $N = \left[a.\mathbf{0} + \overline{a}.\checkmark\right]^2_\sigma$ is a symmetric network. Moreover, we have $N \downdownarrows$ but $P \not\downdownarrows$. We have $\triangleleft P \mid P \triangleright \overset{(U)}{=} \triangleleft P \triangleright \mid \triangleleft P \triangleright = \left[\triangleleft P \triangleright\right]^2_{\mathbf{id}}$, i.e. $\triangleleft N \triangleright$ is again a symmetric network of degree 2 with **id** as symmetry relation. By Theorem 3.7, $\triangleleft N \triangleright$ has at least one symmetric execution and by success sensitiveness and must testing $\triangleleft N \triangleright$ must reduce to a process containing a top-level unguarded occurrence of $\checkmark$ within this symmetric execution, i.e. there is a sequence of actions $\tilde{\mu} \in \mathcal{T}(\mathcal{A}_\tau)$, a process $P' \in \mathcal{P}_{\text{sep}}$, a $\sigma' \in \text{Sym}(2, \mathcal{N})$ and a sequence of names $\tilde{x}$ such that $\triangleleft P \triangleright \mid \triangleleft P \triangleright \xrightarrow{\tilde{\mu}} \left[P'\right]^{2,\tilde{x}}_{\sigma'}$ and P' or $\sigma'(P')$ contain a top-level unguarded occurrence of $\checkmark$. Then, by symmetry, both processes of $\left[P'\right]^{2,\tilde{x}}_{\sigma'}$ contain a top-level unguarded occurrence of $\checkmark$. By Lemma 3.14, there is a sequence of actions $\tilde{\mu}' \in \mathcal{T}(\mathcal{A}_\tau)$ and an execution $\triangleleft P \triangleright \xrightarrow{\tilde{\mu}'} (\nu\, \tilde{x}')\, P'$ for a subsequence $\tilde{x}'$ of $\tilde{x}$. With it, $\triangleleft P \triangleright \downdownarrows$, and with success sensitiveness $P \downdownarrows$, which is a contradiction. □

Note that, reconsidering the proofs of this separation result in Gorla (2008b), we managed to omit one of Gorla's additional assumptions. Namely, we do not need the assumption that $\asymp_{\text{T}}$ is exact (first setting in Gorla (2008b)) or reduction sensitive (second setting in Gorla (2008b)) and we do not need to assume the stronger version of operational correspondence in the third setting in Gorla (2008b). On the other side Gorla does not need to assume homomorphic translation of $\mid$ in his second and third setting. He uses the weaker notion of compositional translation of $\mid$ instead. But, as we conjecture, there is an encoding from π_{mix} into π_{a} for this weaker structural assumption (compare to the presented attempt of such an encoding in Section 4 and the full encoding in (Peters and Nestmann 2012a)). Summarizing, this separation result is weaker as the result in the first setting of Gorla but incomparable to the results in the other two settings. Moreover, note that because we focus on breaking symmetries instead of leader election, we can apply Theorem 3.7 to problem instances different from leader election.

Version 3. In his proofs of this separation result in Gorla (2008b), he uses may testing to show that there are terms $P \in \mathcal{P}_{\text{mix}}$ such that $P \not\longmapsto$, $P \not\downdownarrows$ and $(P \mid P) \downdownarrows$, but there are no such terms in $\mathcal{P}_{\text{sep}}$. Implicitly, he uses the fact that $P \not\downdownarrows$ and $(P \mid P) \downdownarrows$ implies $P \mid P \longmapsto$

and that there are no terms P in $\mathcal{P}_{\text{sep}}$ such that $P \not\longmapsto$ and $P \mid P \longmapsto$. By proving this fact directly, we do not need any notion of testing to prove the separation result.

Definition 3.17 (3-reasonableness). An encoding $\triangleleft \cdot \triangleright : \mathcal{P}_{\text{mix}} \to \mathcal{P}_{\text{sep}}$ is *3-reasonable* if $P \longmapsto$ if and only if $\triangleleft P \triangleright \longmapsto$ for all $P \in \mathcal{P}_{\text{mix}}$.

To our knowledge, only few intuitively reasonable encodings are not also 3-reasonable. Note, however, that the encoding in Section 4 is *not* 3-reasonable.

Theorem 3.18 (separation result). There is no uniform and 3-reasonable encoding from π_{mix} into π_{sep}.

Again, for the separation proof, we enforce that the encoding results in a symmetric network $Q \mid Q$. By subdividing the symmetric execution of this network, we prove that $Q \xrightarrow{\tau}$ iff $Q \mid Q \xrightarrow{\tau}$, which does not necessarily hold in π_{mix}.

Proof of Theorem 3.18 Let us assume the contrary, i.e. there is a uniform and 3-reasonable encoding $\triangleleft \cdot \triangleright$ from π_{mix} into π_{sep}. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P \triangleq a + \overline{a}$$

Obviously, $\sigma = \mathbf{id}$ is a symmetry relation of degree 2 and so $N = \left[a + \overline{a}\right]^2_\sigma$ is a symmetric network. Moreover, we have $N \xrightarrow{\tau}$ but $P \not\xrightarrow{\tau}$ and thus $N \longmapsto$ but $P \not\longmapsto$. We have $\triangleleft P \mid P \triangleright \overset{(U)}{=} \triangleleft P \triangleright \mid \triangleleft P \triangleright = \left[\triangleleft\ P\ \triangleright\right]^2_{\mathbf{id}}$, i.e. $\triangleleft N \triangleright$ is again a symmetric network of degree 2 with **id** as symmetry relation. By Theorem 3.7 $\triangleleft N \triangleright$ has at least one symmetric execution and by 3-reasonableness we have $\triangleleft P \triangleright \mid \triangleleft P \triangleright \longmapsto$ and $\triangleleft P \triangleright \not\longmapsto$ and thus $\triangleleft P \triangleright \mid \triangleleft P \triangleright \xrightarrow{\tau}$ and $\triangleleft P \triangleright \not\xrightarrow{\tau}$. By Lemma 3.8, $\triangleleft P \triangleright \mid \triangleleft P \triangleright \xrightarrow{\tau}$ implies that there is at least one step in the symmetric execution, i.e. there is a process $P' \in \mathcal{P}_{\text{sep}}$, a $\sigma' \in \text{Sym}(2, \mathcal{N})$, and a sequence of names $\tilde{x} \in \mathcal{T}(\mathcal{N})$ such that $\triangleleft P \triangleright \mid \triangleleft P \triangleright \xrightarrow{\tau,\tau} \left[P'\right]^{2,\tilde{x}}_{\sigma'}$. By Lemma 3.14, there is an execution $\triangleleft P \triangleright \xrightarrow{\tau} \left(\nu\, \tilde{x}'\right) P'$ for a subsequence $\tilde{x}'$ of $\tilde{x}$, i.e. $\triangleleft P \triangleright \longmapsto$, which is a contradiction. □

Note that – in contrast to both Palamidessi and Gorla – we do not even assume divergence reflection in this argumentation.

3.4. *Intermezzo*

We prove that π_{mix} is strictly more expressive than π_{sep} by means of an absolute separation result about the ability to break initial symmetries. This result is independent of any notion of encodability, uniformity and reasonableness. By choosing the problem of breaking initial symmetries instead of leader election, we significantly weaken the underlying definition of symmetry in comparison to Palamidessi (2003). Moreover, we could still apply our absolute separation result to derive that there is no uniform and reasonable encoding from π_{mix} into π_{sep} considering three different definitions of reasonableness. It turns out that the concentration on the underlying problem of breaking initial symmetries allows us to use counterexamples different from leader election in translational separation results. Likewise, the separation result in the setting of Gorla (2008b) can be derived by our absolute result as well. Besides that, our absolute separation result allows us to

weaken the definition of uniformity in comparison to the translational separation result of Palamidessi (2003), and also to weaken the definition of reasonableness in comparison to the translational separation result in the first setting of Gorla (2008b). Moreover, considering our last translational separation result, we can even withdraw the assumption of divergence reflection.

Our own translational separation results, i.e. the proofs of the non-existence of a uniform and reasonable encoding for different definitions of reasonableness, follow similar lines of argument. The proofs argue by contradiction. First, a symmetric network of the form $P \mid P$ in $\mathcal{P}_{\text{mix}}$ with special features is presented. Second, we use the fact that uniformity, i.e. the homomorphic translation of the parallel operator, preserves essential parts of the symmetric nature of $P \mid P$. Third, we apply Theorem 3.7 to conclude with the existence of a symmetric execution. In two proofs, we then apply Lemma 3.14 to subdivide this symmetric execution. At last, we derive a contradiction between the additional information provided by the symmetric execution (and its subdivision) and the respective definition of reasonableness.

Note that, we prove the absolute result without any precondition. We use different definitions of reasonableness for the translational results. The only constant precondition of the translational separation results is the definition of uniformity, i.e. the homomorphic translation of the parallel operator. This condition is crucial. Without it, we could not apply our absolute separation result. To the best of our knowledge, only Gorla ever managed to prove such a separation result between π_{mix} and π_{sep} without the homomorphic translation of the parallel operator, using compositionality, operational correspondence, divergence reflection, success sensitiveness and either a reduction sensitive version of $\asymp_{\text{T}}$ or the stronger version of operational correspondence of his third setting. However, Gorla believes that the result also holds for the general formulation of his criteria as presented in Section 2.2.

We may also turn the non-existence of a uniform *and* reasonable encoding around and rephrase it as a weakened existence statement. Recall that any uniform encoding from π_{mix} into π_{sep} preserves symmetries. While it is possible to break such symmetries in π_{mix}, this is not possible in π_{sep}. Thus, should there be a non-uniform (at least: 'weakly compositional') *but* reasonable encoding from π_{mix} into π_{sep}, then it would have to be the encoding itself to break these symmetries.

Conclusion 2. A reasonable, divergence reflecting encoding from π_{mix} into π_{sep} must be able to break initial source term symmetries.

This tells us much about how such an encoding, if it exists, has to look like. The homomorphic translation of the parallel operator preserves the symmetry of the source term. Hence the encoding is sure not homomorphic. Instead, by compositionality in Definition 2.5, it has to introduce a context $\mathcal{C}^{N}_{\mid}(\_1;\_2)$ to translate the parallel operator. To break symmetries the context cannot treat the left and the right side in exactly the same way, i.e. it cannot look like $(\nu\,\tilde{x})\left(\mathcal{C}'(\_1) \mid \mathcal{C}'(\_2)\right)$ for some sequence of names $\tilde{x}$ and a subcontext $\mathcal{C}'(\_)$. But of course it has to ensure, that both sides can mimic all behaviours of source terms.

Finding a reasonable, divergence reflecting encoding from π_{mix} into π_{sep} is an open problem. A uniform and 'almost reasonable' divergent encoding was already presented in (Nestmann 2000). It shows that, if divergence reflection is not required, the encoding can ensure that all undesired symmetric executions are divergent such that it is not necessary for the encoding function to break symmetry. In the next section, we present an encoding that, as we strongly conjecture, meets all five of Gorla's criteria. In turn, this would counter Gorla's conjecture that it is possible to prove a separation result for the general formulation of his criteria without further assumptions.

4. Encoding synchrony by breaking symmetry

In this section, we present the main idea of an encoding from π_{mix} into π_{a} that we conjecture to be correct with respect to the five criteria of Gorla presented in Section 2.2. We start with an encoding from π_{sep} into π_{a} of Nestmann (2000). Based on it, we illustrate the main idea to design an encoding from π_{mix} into π_{a}, i.e. how to break possible source term symmetries by means of an encoding. In the following, we explain some concepts auxiliary and notions that we use within the presented encoding functions.

4.1. *Locks and tests*

A *lock* is a special channel used by the encoding function to block some further behaviour. Therefore the term we want to block is guarded by an input on the lock channel such that the term is blocked until an output on this channel is available. Moreover, we use a special kind of locks called *boolean locks*. A boolean lock is a channel on which only the boolean values $\top$ (true) or $\bot$ (false) are transmitted. An output over a boolean lock with value $\top$ is called a positive instantiation of the respective lock while sending $\bot$ is denoted as negative instantiation. At the receiving end of such a channel, the boolean value can be used to make a binary decision, which is done here within a test-*construct*. The test-construct and accordingly positive and negative instantiations of boolean locks are implemented using restriction (compare to Nestmann and Pierce (2000)).

Definition 4.1 (test). The test operator and *positive* or *negative instantiations* of boolean locks, denoted by $\bar{l}\langle\top\rangle$ and $\bar{l}\langle\bot\rangle$ for a boolean lock l, are abbreviations of the terms:

$$\begin{aligned}
\bar{l}\langle\top\rangle &\triangleq l(t,f).\bar{t}\\
\bar{l}\langle\bot\rangle &\triangleq l(t,f).\bar{f}\\
\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q &\triangleq (\nu\, t,f)\left(\bar{l}\langle t,f\rangle \mid t.P \mid f.Q\right) \qquad \text{for some } t,f \notin \mathsf{fn}\left(P \mid Q\right)
\end{aligned}$$

Note that with this definition the test predicate operates as guard for its subterms P and Q. Moreover, we observe that the boolean values $\top$ and $\bot$ are realized by a pair of links without parameters. With that a boolean lock is indeed a lock with two parameters and the distinction between true and false is given by the order of these parameters.

Note that to simplify the following argumentation we omit, for now, the encoding of matching, success, and the τ-prefix. Moreover, we omit the usage of the renaming policy.

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

Fig. 6. Encoding from π_{sep} into π_{a}.

It is used to ensure that there are no conflicts between the names used by the encoding function for special purposes and the names used by the source term. We explain how to extend the encoding function by these concepts after the presentation of its main idea.

4.2. *An encoding from π_{sep} into π_{a}*

Nestmann (2000) presents an encoding from π_{sep} into π_{a} (compare to Figure 6) that encodes the parallel operator homomorphically.

Here $\prod_{i\in I} [\![\pi_i.P_i]\!]_l$ denotes the parallel composition of the encodings of all summands $\pi_i.P_i$ in the respective sum. Remember that, in contrast to π_{sep}, there is no choice operator in the syntax of π_{a}. Analysing the rules $\textsc{Com}_{\mathrm{S}}$ and $\textsc{Rep}_{\mathrm{S}}$ (see Figure 3) we observe that a communication with a summand of a sum automatically removes the other summands of that sum. To compensate for the lacking choice operator, the encodings of the summands are put in parallel. To ensure that a communication on one summand of a sum disables the other summands of that sum the encoding of a sum introduces a boolean lock l and exactly one positive instantiation of it to ensure that at most one of its branches can be chosen. Input and output are tucked away behind such boolean locks. The receivers take control over the locks. If there are a source term sender and receiver on the same channel name, the translated sender sends the names of its sum lock and its sender lock to the receiver. The sender lock s guards the encoded continuation of a sender. The receiver then checks for its own and the sum lock of the sender. If both locks are instantiated with true then he instantiates the sender lock and performs its subprocess. Moreover, it sets both sum locks to false such that no other summand of the respective sums can be used for communication.

The receiver lock r allows to restart a test on the corresponding receiver if a former test failed due to a negative instantiation of the senders sum lock. To do so, in this case, the receiver lock is reinstantiated. To allow the first test it is initially instantiated. Moreover, it blocks the search of a matching output partner for communication to avoid multiple concurrent tests on the same encoded input guarded term. The receiver lock is not reinstantiated in the case of a successful completion of a test nor of a test failing due to a negative instantiation of the sum lock of the receiver because, in these cases, a

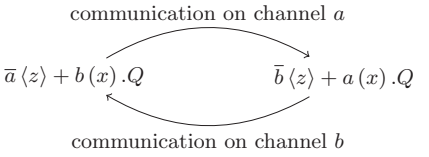

Fig. 7. Cyclic sums.

corresponding test will never succeed any more. Note that in each case the completion of a test either reinstantiates the instantiations of all consumed sum locks or changes them into false instantiations.

The sender lock is implemented as a simple lock and is instantiated by the positive outcome of the test-construct within the encoding of a matching receiver. With that, of course, it is possible that it is never instantiated. In that case, the sender lock blocks the continuation of the respective output for ever.

Note that, in the case of an empty sum (i.e. $I = \varnothing$), the encoding yields $(\nu\, l)\left(\overline{l}\langle\top\rangle \mid \mathbf{0}\right)$ which is semantically equivalent to $\mathbf{0}$. With that $\mathbf{0}$ is translated semantically equal to $\mathbf{0}$. We refer to Nestmann (2000) for a more exhaustive explanation of this encoding.

The problem with mixed choice are cyclic dependencies within a single sum or a set of sums. In both situations, the encoding introduces a deadlock. That is why the encoding above is a good encoding from π_{sep} into π_{a}, but no good encoding from π_{mix} into π_{a}. We explain the problem using two examples.

Example 4.2 (incestuous sum). Consider the sum $\overline{a}\langle z\rangle + a(x)$. It is called an *incestuous sum* because it contains two potential communication partners, i.e. there is an output and a matching input within this sum. The encoding of this sum

$$\begin{aligned}(\nu\, l)\,\big(\,&\overline{l}\langle\top\rangle \mid (\nu\, s)\left(\overline{a}\langle l,s,z\rangle \mid s.[\![\,\mathbf{0}\,]\!]\right)\\ &\mid (\nu\, r)\left(\overline{r} \mid r^{*}.a\left(l',s,x\right).\mathsf{test}\ l\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\,\mathbf{0}\,]\!]\right.\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r}\\ &\qquad\qquad\qquad\qquad\qquad\quad\ \ \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{y}\langle l',s,x\rangle\,\big)\big)\end{aligned}$$

deadlocks while performing the nested test-construct because it tries to check twice for the same lock, i.e. the first part of the nested test-construct consumes the sum lock and so the second part – which tests for the same lock – is deadlocked. Since the source term $\overline{a}\langle z\rangle + a(x)$ cannot perform a step as well this is not a problem. But consider the term $P = \overline{a}\langle z\rangle + a(x) \mid a(x).Q$. It reduces to $Q\{^{z}/_{x}\}$. In this case, the deadlock which may occur by first testing the communication within the incestuous sum leads to different behaviour of the target term, i.e. the target term may deadlock without reaching the encoding of $Q\{^{z}/_{x}\}$ while the source term reaches $Q\{^{z}/_{x}\}$ in any execution (of reductions).

Example 4.3 (cyclic sums). With *cyclic sums* we denote a set of sums with cyclic dependencies of their potential communication partners as in $P = \overline{a}\langle z\rangle + b(x).Q \mid \overline{b}\langle z\rangle + a(x).Q$. The cyclic dependencies of P are depicted in Figure 7. Obviously P can either reduce to a communication on channel a or on channel b. The encoding of P in

$$
\begin{aligned}
&(\nu\, l_1)\,\big(\overline{l_1}\,\langle\top\rangle \mid (\nu\, s_1)\,(\overline{a}\,\langle l_1, s_1, z\rangle \mid s_1.[\![\,\mathbf{0}\,]\!]\big)\\
&\quad \mid (\nu\, r_1)\,\big(\overline{r_1} \mid r_1^*.b\,\big(l', s, x\big)\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l'\ \mathsf{then}\ \overline{l_1}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s} \mid [\![\,Q\,]\!]\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{r_1}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\,\langle\bot\rangle \mid \overline{b}\,\langle l', s, x\rangle\,\big)\big)\\
&\mid (\nu\, l_2)\,\big(\overline{l_2}\,\langle\top\rangle \mid (\nu\, s_2)\,(\overline{b}\,\langle l_2, s_2, z\rangle \mid s_2.[\![\,\mathbf{0}\,]\!]\big)\\
&\quad \mid (\nu\, r_2)\,\big(\overline{r_2} \mid r_2^*.a\,\big(l', s, x\big)\,.\mathsf{test}\ l_2\ \mathsf{then\ test}\ l'\ \mathsf{then}\ \overline{l_2}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s} \mid [\![\,Q\,]\!]\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_2}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{r_2}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_2}\,\langle\bot\rangle \mid \overline{a}\,\langle l', s, x\rangle\,\big)\big)
\end{aligned}
$$

Fig. 8. Encoding of P.

Figure 8 will deadlock if the two nested test-construct are performed simultaneously, i.e. if the first nested test-construct consumes the lock l_1 and before its second part can be performed the second nested test-construct consumes the lock l_2. In this situation the process is deadlocked because both sum locks are consumed and with it none of the remaining test-constructs can be resolved. Again the target term may deadlock without reaching the encoding of $Q\{{}^{z}/_{x}\}$ while the source term reaches $Q\{{}^{z}/_{x}\}$ in any execution, i.e. there is a difference in the behaviour of the target and the source term.

Both cases result in a deadlock that is induced by the encoding function and not intended by the underlying source term. Note that such deadlocks are detected by operational soundness provided $\asymp_{\mathrm{T}}$ is not trivial. In other words, not dealing with this problem adequately violates the operational correspondence criteria in Definition 2.8. In Nestmann (2000) different attempts to overcome these deadlocks are discussed. The simplest way to resolve them is to implement the possibility to roll back a test. Unfortunately this directly leads to divergence introduced by the encoding and not the source term and with it violates the divergence reflection criterion in Definition 2.10. Another attempt is to assume a total order among the threads or processes of the system, such that within the test statements the order of the locks tested can be determined by the order of the according threads. Since, we have no such total ordering on the source terms the ordering must be constructed by the encoding function. That can be done for instance by a two-level encoding or by an encoding with global knowledge about the source term. Both violate the compositionality criterion in Definition 2.5. A third attempt is to choose the order of the locks tested at random. Again this violates the divergence reflection criterion although divergence may occur only with a very low probability. This approach was formally investigated by Herescu and Palamidessi (2002) in the context of the probabilistic π-calculus. Nevertheless, as we will show in the following, there is a way to circumvent both problems, incestuous and cyclic sums, without referring to randomization within the framework of a good encoding presented in Section 2.2.

4.3. *An Encoding from π_{mix} into π_{a}*

First in Palamidessi (2003), and later as well in Gorla (2008b, 2010), and by us in Section 3, it is proved that there is no encoding from π_{mix} into π_{sep} and, thus, no encoding from π_{mix} into π_{a} that translates the parallel operator homomorphically. So, we have to abandon

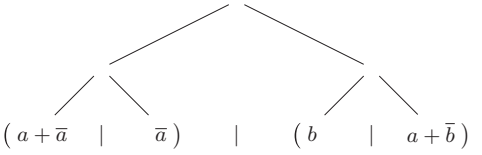

Fig. 9. Parallel structure.

this condition. Moreover, in Section 3, we state that an encoding from π_{mix} into π_{sep} or π_{a} – if there is any – has to break symmetries.

Already Nestmann (2000) proposed some attempts to break symmetries, albeit at a global level: he proposed (1) a two-level encoding, where the outermost level ensures to break symmetries at a global level by the provision of a globally restricted channel – so two encoded terms could not be run in parallel, as the global control would get lost – and (2) extended target languages that allowed to impose some global ordering on names to prevent from deadlock situations.

4.3.1. *Principle.* We propose a novel single-level encoding, in which the *symmetry is broken locally* at each parallel operator, while still allowing for an unconstrained composability of encoded terms. By doing so, we can avoid the problem with cyclic dependencies in sums. To explain the main idea of such an encoding, let us consider the example, $S = \left(a + \overline{a} \mid \overline{a}\right) \mid \left(b \mid a + \overline{b}\right)$ and its *parallel structure* depicted in Figure 9. As parallel structure, we denote the binary tree induced by the nesting of the parallel operator of a term, where the leafs are formed by its capabilities. Analysing the operational semantics of π_{mix}, given by Figure 3, we observe that communication steps in π_{mix} are always due to an input and an output guarded term on two different sides of a node within the parallel structure of a term. Moreover, we observe that each matching pair of communication partners is left and right of exactly one node of that binary tree, i.e. their closest common parent node. Since there is no sum operator in π_{a}, the encoding of sum forces us to represent its summands as parallel terms; otherwise, there would be no way for them to be concurrently enabled. Obviously, and unfortunately, this changes the parallel structure of the original term. The sum lock is introduced to restore the lost information about the correspondence of summands to a sum. That suffices to encode separate choice, but, as shown in the Examples 4.2 and 4.3 above, it does not suffice to encode mixed sums. The main idea to overcome these problems is to exploit the parallel structure of the originating source term when enabling or disabling interactions in the translation at the target-level. More precisely: (1) to avoid the problem of incestuous sums, the encoding will guarantee that target-level communications will only be possible for requests emanating from two different sides of a parallel operator in the source term; (2) to avoid the problem of cyclic sums, the encoding guarantees that there cannot be two different tests being concurrently enabled at the level of the same parallel operator, i.e. at the same node in the parallel structure of the source term. This is the main effect of breaking symmetries locally.

In essence, the detection of matching communication partners is ceded to the nodes of the parallel structure of the source term; more precisely, each parent node is equipped

$$
\begin{aligned}
[\![\, (\nu\, x)\, P \,]\!] &\triangleq (\nu\, x)\, [\![\, P \,]\!] \\
[\![\, P \mid Q \,]\!] &\triangleq (\nu\, c, p_{i,up}, p_{o,up}, \mathsf{i}, \mathsf{o})\, (\\
&\qquad (\nu\, p_i, p_o)\, ([\![\, P \,]\!] \\
&\qquad\quad \mid {p_i}^*\, (y, r)\, .\, (\overline{y \cdot \mathsf{i}}\, \langle r \rangle \mid \overline{p_{i,up}}\, \langle y, r \rangle) \\
&\qquad\quad \mid {p_o}^*\, (y, l, s_1, s_2, z)\, .\, (\overline{y \cdot \mathsf{o}}\, \langle l, s_1, s_2, z \rangle \mid \overline{p_{o,up}}\, \langle y, l, s_1, s_2, z \rangle)) \\
&\qquad \mid (\nu\, p_i, p_o)\, ([\![\, Q \,]\!] \mid \overline{c} \\
&\qquad\quad \mid {p_i}^*\, (y, r)\, .\, (y \cdot \mathsf{o}\, (l, s_1, s_2, z)\, .c.\overline{r}\, \langle l, s_1, s_2, z, c \rangle \mid \overline{p_{i,up}}\, \langle y, r \rangle) \\
&\qquad\quad \mid {p_o}^*\, (y, l, s_1, s_2, z)\, .\, (y \cdot \mathsf{i}\, (r)\, .c.\overline{r}\, \langle l, s_1, s_2, z, c \rangle \mid \overline{p_{o,up}}\, \langle y, l, s_1, s_2, z \rangle)) \\
&\qquad \mid {p_{i,up}}^*\, (y, r)\, .\overline{p_i}\, \langle y, r \rangle \mid {p_{o,up}}^*\, (y, l, s_1, s_2, z)\, .\overline{p_o}\, \langle y, l, s_1, s_2, z \rangle\,) \\
[\![\, \textstyle\sum_{i \in I} \pi_i.P_i \,]\!] &\triangleq (\nu\, l) \left(\overline{l}\, \langle \top \rangle \mid \prod_{i \in I} [\![\, \pi_i.P_i \,]\!]_l \right) \\
[\![\, \overline{y}\, \langle z \rangle\, .P \,]\!]_l &\triangleq (\nu\, s_1, s_2)\, (\overline{s_1} \mid s_1^*.\overline{p_o}\, \langle y, l, s_1, s_2, z \rangle \mid s_2.[\![\, P \,]\!]) \\
[\![\, y\, (x)\, .P \,]\!]_l &\triangleq (\nu\, r)\, (\overline{p_i}\, \langle y, r \rangle \\
&\qquad\quad \mid r^*\, (l', s_1, s_2, x, c)\, .\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\, \langle \bot \rangle \mid \overline{l'}\, \langle \bot \rangle \mid \overline{s_2} \mid [\![\, P \,]\!] \mid \overline{c} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l}\, \langle \top \rangle \mid \overline{l'}\, \langle \bot \rangle \mid \overline{c} \mid \overline{p_i}\, \langle y, r \rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l}\, \langle \bot \rangle \mid \overline{c} \mid \overline{s_1}\,)
\end{aligned}
$$

Fig. 10. (Colour online) Encoding from π_{mix} into π^2_{a}.

with a *coordinator* process. This is explicitly allowed by weak compositionality, and it is here that the source-term-level symmetry is broken. Now, as opposed to the previous globally-breaking proposals by Nestmann (2000), the overall exercise is here much more difficult: from the point of view of a single coordinator, communication may not only occur between its left- and right-hand subterms, but also between either of these two and some outer – unknown – communication partner in the environment. All coordinators residing at the various nodes in the binary parallelism-reflecting tree must play together, and the encoding function must treat them all alike (to avoid the term 'symmetrically') to keep the encoding truly compositional. To this aim, the encoding of input and output capabilities announce their requests along special channels to their parent coordinator nodes within that binary tree. If, at the level of a node, a matching pair of communication partners is identified, the (nested) test-construct are checked as described for the encoding in Figure 6. At the same time, to keep up with communication possibilities with 'external' partners, the requests are passed on to the potential parent of this coordinator.

4.3.2. *Implementation.* The encoding in Figure 10 implements the above idea to translate π_{mix} into π^2_{a}, the asynchronous π-calculus augmented with polyadic (here: 2-adic) synchronization, as proposed by Carbone and Maffeis (2003). This mechanism is convenient to focus on the essence of the encoding; as we use it only in a limited manner, its usage can be expanded into the standard target calculus and is thus not critical for the intended result. We comment on their intuition below, when explaining their usage in the encoding itself.

For simplicity, in this paper, we leave out the encoding of replicated input. The encoding of recursive behaviour, however it is defined, is subtle. We explain the problems and how to overcome them at the end of this section.

For each edge in the binary tree, there are two upward-directed channels named p_o and p_i. We use the same names in every node, but communications along such edges are explicitly restricted in order to avoid interference. Thus, each (term at a) node of the binary tree is assumed to dispose of the free names p_o and p_i, intended for communication with its parent node; the (term at the) parent node is, in turn, responsible for restricting the respective access to this child. Consequently, also the top-level node disposes of these two free names. For simplicity, we systematically do not display the indices p_o, p_i on expressions $[\![P]\!]_{p_o,p_i}$ and leave them implicit.

The differences in comparison with the encoding in Figure 6 are coloured blue and red. Input and output guarded terms announce their ability to send or to receive a value to the respective environment of the next parallel operator, i.e. to the next parent node in the parallel structure of the source term, by an output over the channel p_o or p_i. These links are bound twice within the encoding of each parallel operator; once for each side. We call such output messages *requests*, i.e. an output on channel p_o is an *output request* and an output on channel p_i is an *input request*. If there is no parallel operator, then the encoding of the term cannot perform any step except by mimicking a source term step of a term guarded by τ.

Coordinators are to break symmetries locally. To deal with potential cyclic dependencies (see Example 4.3), we allow coordinators to do only one thing at a time, so we control their behaviour by *coordinator locks* c. To implement communications, coordinators shall connect the requests from the left- and right-hand sides of parallel terms. In our encoding, we choose to transfer requests from the left-hand side of a parallel operator (LHS requests) into outputs, and requests from the right-hand side of a parallel operator (RHS requests) into inputs. This choice of direction is not essential. However, it is essential that the resulting outputs and inputs must allow to be checked for matching requests.

In order to guarantee that only requests from its children are taken into account, a coordinator uses the local names i/o, which at the same time serve to distinguish the two kinds of request. Here, we use polyadic synchronization as syntactic sugar to bind the search for matching requests on channel y to a specific node (referred to via i/o). Polyadic synchronization in the case of π^2_{a} allows structured channels as combinations of any two names $n, m \in \mathcal{N}$, denoted by $n \cdot m$. Moreover, two polyadic channels are equal if and only if they have the same length and are composed of the same names in the same order. In contrast to the usual π-calculus, polyadic synchronization allows us to restrict parts of a channel such that $\left((\nu\, n)\, \overline{n \cdot m}\right) \mid n \cdot m \not\longrightarrow$. See further below how we can avoid polyadic synchronization by working in the pure target language.

Now, by means of polyadic synchronization (cf. the red channel names in Figure 10), LHS requests are translated into outputs on channel $y \cdot \mathsf{i}$ for an input request and on channel $y \cdot \mathsf{o}$ for an output request, where y is the corresponding source term link, and i/o are used as distinguishing tags for the kind of request. RHS requests are translated into the according inputs on these channels.

If a matching pair of communication partners is found at a node, then the respective sum lock l of the sender, then the sender locks s_1, s_2, the sent value z, and the coordinator lock c are transmitted by the respective right-hand communication partner back along channel r to the encoding of the respective receiver. This step enables a test on the involved sum locks. Note that to perform that step an instantiation of the coordinator lock c has to be consumed. There is exactly one coordinator lock for each encoding of a parallel operator, i.e. each node of the parallel structure of the source term, and there is at most one instantiation on each coordinator lock. Moreover, we observe that, by finishing a test-construct regardless of its outcome, exactly one instantiation on the respective consumed coordinator lock is restored. Thus, the node is blocked until this check is finished by the coordinator lock c, i.e. at each node only one test can be enabled concurrently.

The test-constructs are mainly the same as in the encoding in Figure 6. One difference is the emission of an instantiation of a coordinator lock in each case. Considering the nested test-construct, in case of a negative instantiation of the sum lock of the encoded sender, we observe that the input request of the encoded receiver is retransmitted instead of the receiver lock. Moreover, in case of a negative instantiation of the sum lock of the encoded receiver, the sender lock s_1 is instantiated. This ensures that the retransmission of the corresponding output request starts at the same leaf as the original output request of that encoded output.

Analysing the parallel structure of S in Figure 9, we observe that the output on a in the second leaf can communicate with the corresponding input in the first or the last leaf. Therefore, it does not suffice to search for a matching communication partner at the immediate parent node but we have to search at each parent node in the parallel structure. Therefore, each node *also* pushes all its requests upwards within the parallel structure to its subsequently parent node. The channels $p_{i,up}$ and $p_{o,up}$ are used to push all left or right requests over the restriction on p_i and p_o. Then, in the last line of the encoding of a parallel operator, the requests are relayed by performing the respective output over p_i or p_o such that they can be bound by a surrounding parallel operator if there is any. Otherwise, i.e. at the top-level of the parallel structure of the source term, the requests remain unbounded.

There are two deficiencies of the encoding presented so far. First, it uses polyadic synchronization, which is not a part of the source language π_a. Second, there is no encoding of replicated input. In the following, we sketch the main ideas to overcome these issues.

4.3.3. *Avoiding polyadic synchronization.* Since π_a does not allow for polyadic communication, we have to encode the binding of the search for a matching pair of communication partners to a node in the parallel structure of the source term different to obtain an encoding from π_{mix} into π_a. Unfortunately this requires match and further blows up the already rather complicated encoding of the parallel operator. Therefore, we only give some hints on how to obtain such an encoding. Intuitively, we have to ensure that at the level of each node each left request is combined with each matching right request without introducing divergence or deadlock. In Figure 10, we use polyadic synchronization to

synchronize on the respective source term link name and at the same time to restrict the communication to the respective node. Instead of using a communication step and synchronization to find a matching pair of requests, we can transmit all LHS requests to all RHS requests and use the matching operator to check for a matching of the respective source term links. The transmission of requests from the left to the right can be performed on a fresh link which can be bound more easily to the respective node than a source term name.

We have to be careful that indeed each left request can be combined with each right request without the introduction of divergence as for instance a careless broadcast would induce. Moreover notice that the unguarding of the encoding of the continuation of some guarded source term may add further requests to a side of a node or even further subtrees to the parallel structure of an encoded term. With that it is necessary that the possibility to transmit requests from left to the right is no pure preprocessing but remains during the hole execution of the target term. Therefore, the requests on the right can be ordered dynamically into two chains – one for input requests and one for output requests – such that the first member of each chain receives all left requests of the opposite kind, checks itself for the matching of the respective source term links, and regardless of the outcome of that check forwards all left requests to the respective next member in that chain. With that all left requests are pushed along that chains and are combined with each right request of the opposite kind. In that case the test-constructs should not retransmit the requests in case of a fail of the test as it is done by the encoding in Figure 10. Note that such a retransmission of requests in case of a failed test is necessary in Figure 10 to be able to perform a test on the respective request that was not the reason for the failure and a possibly other matching request. Obviously such retransmitting leads to duplications of requests within the parallel structure. Since in the case of chains as explained all possible combinations of requests are checked there is no need for these retransmitting of requests.

To explain why these chains – or at least a comparable construct that ensures that each left request is combined with each right request of the opposite kind and not only with a single one – are indeed necessary let us consider an example without chains. We assume a couple of fresh links m_i and m_o. To combine left and right requests we exchange $y \cdot \mathsf{i}$ by m_i and $y \cdot \mathsf{o}$ by m_o and send y as first parameter to compare it within some matching. Then, the right side (for right input requests) becomes:

$$m_o\left(y', l, s_1, s_2, z\right).\left[\, y = y' \,\right] c.\overline{r}\left\langle l, s_1, s_2, z, c\right\rangle$$

Unfortunately, if matching fails then not only the left request is lost, but also the ability of the corresponding right request to receive a left request. Alternatively, we might use the mismatch operator to restore the lost capabilities; mismatch, however, might have an unexpected impact on the expressive power of the respective calculus. Thus, the introduction of chains seems to be the better way. Another alternative would be to retransmit the left request regardless of whether matching fails or not and to restore the ability to receive left requests. With this, we would get something like:

$$m_o{}^*\left(y', l, s_1, s_2, z\right).\left(\left[\, y = y' \,\right] c.\overline{r}\left\langle l, s_1, s_2, z, c\right\rangle \mid \overline{m_o}\left\langle y', l, s_1, s_2, z\right\rangle\right)$$

Obviously, this solution introduces divergence since, if it is possible to reduce the replicated input on m_o once then it can reduce infinity often.

4.3.4. *Encoding replication.* So far, the encoding takes advantage of the parallel structure of the source term to circumvent problems with cyclic sums. Unfortunately, the use of the parallel structure further complicates the encoding of replication. Consider the rule REP_S for replicated inputs in the source language:

$$\text{REP}_\text{S} \quad y^*(x).P \mid \left(\cdots + \overline{y}\langle z\rangle.Q + \cdots\right) \longmapsto_\text{S} \{\,{}^{z}/_{x}\,\} P \mid y^*(x).P \mid Q$$

It states that the communication of a replicated input with a matching output results not only in the respective continuations, but also in a copy of the original replicated input in parallel to its continuation. The outcome of a communication for the side of an output is encoded by a guard on the encoding of the respective continuation. This guard can be removed after the respective communication is mimicked within the target term (compare to the second sender lock s_2 in Figure 10). Revisiting the encoding of an input guarded term, we observe that the guard of the continuation in this case is the test-construct that is used to mimic a step on that capability. This suffices to mimic the COM_S rule. However, mimicking REP_S is not that easy. Intuitively, the problem is that its conclusion has three terms in parallel, but there are only two parallel terms within its precondition. In the encoding of an input guarded term, after mimicking a step on the respective capability, the continuation is placed at the same position as the original input capability. Since this capability cannot be used once more, this is not a problem. On the other side, we cannot simply place the encoding of the continuation of a replicated input in the same place at this replicated input, because this would prevent the encodings of several such continuations from communicating among themselves, or with the replicated input.

As a solution, we propose the introduction of a new kind of node into the parallel structure of the source term that, in contrast to the nodes of the parallel structure, is not binary, but needs at least two children to enable a test, i.e. to mimic a source term step. Instead of placing the encoding of a replicated input within a leaf, as the other capabilities, we place them within a non-binary node. Therefore, the encoding restricts not only its own receiver lock, but also a coordinator lock. The encoding starts as the encoding of an input guarded term with the respective input request and a single test-construct guarded by the receiver lock similar to the second test-construct of an encoded input guarded term. Thus, in the beginning, this node behaves as a leaf, but whenever a source term step on that replicated input capability is mimicked the node places the encoding of the continuation of that replicated input as a new child. To enable communication among them, these children are constructed in the same way as the right side of the encoding of a parallel operator, i.e. the children look like the encoding of a parallel operator without the left side and with the encoding of the continuation of the replicated input instead of $[\![\, Q \,]\!]$, but without a restriction on the coordinator lock. As a result, the respective node can have several right children, one for each mimicked source term step of that replicated input. To organize communication on these right children they are dynamically ordered into a chain just as the right requests are ordered within the right side of a parallel operator. Moreover, to enable the communication with the encoded replicated input its

$$\begin{aligned} [\![\, [\, a = b \,]\, P \,]\!] &\triangleq [\, \varphi_{[\![\,]\!]}(a) = \varphi_{[\![\,]\!]}(b) \,]\, [\![\, P \,]\!] \\ [\![\, \tau.P \,]\!]_l &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \left(\overline{l}\,\langle\perp\rangle \mid [\![\, P \,]\!]\right)\ \mathsf{else}\ \overline{l}\,\langle\perp\rangle \\ [\![\, \checkmark \,]\!] &\triangleq \checkmark \end{aligned}$$

Fig. 11. Encoding of matching, τ-prefix, and success.

input request is put as the only left child of that node. The request from the left child is transported to the first such right child in that chain, but, in contrast to the chains of right requests, a right child forwards not only the received requests, but also all its own requests to the respective next member in that chain. To ensure that there is no problem with cyclic sums, the hole node, i.e. all of its right children, share the same coordinator lock. Moreover, to allow for communication with the environment, each child pushes all of its own (but not received) requests further upwards along the channels p_i and p_o. Finally, these requests are added to the request of the encoded replicated input and are further propagated by a surrounding parallel operator.

4.3.5. *Renaming policy*. Analysing the encoding function in Figure 10, we observe that the encoding uses several names for special purposes. For instance, l is used to denote a sum lock. A renaming policy ensures that there are no conflicts with respect to the names of the source term.

Definition 4.4 (renaming policy). $\varphi_{[\![\,]\!]} : \mathcal{N} \to \mathcal{N}$ is an arbitrary substitution such that:

$$\forall n \in \mathcal{N}\,.\, \varphi_{[\![\,]\!]}(n) \notin \{\, l, s_1, s_2, r, c, p_i, p_o, p_{i,up}, p_{o,up} \,\} \tag{R1}$$

$$\forall n_1, n_2 \in \mathcal{N}\,.\, n_1 \neq n_2 \text{ implies } \varphi_{[\![\,]\!]}(n_1) \neq \varphi_{[\![\,]\!]}(n_2)\,. \tag{R2}$$

Obviously, we do not completely specify $\varphi_{[\![\,]\!]}$ because we do not want to make special assumptions about the set $\mathcal{N}$ of possible names. Nevertheless, any substitution satisfying conditions (R1) and (R2) suffices as renaming policy. To augment the encoding in Figure 10 by a renaming policy, use $\varphi_{[\![\,]\!]}(n)$ instead of n in the encodings for each source term name n, i.e. in case of Figure 10 for $n \in \{\, x, y, z \,\}$ (cf. the renaming policy to encode matching in Figure 11).

In addition to the operators in the encodings above, we give an encoding of success $\checkmark$, matching $[\, a = b \,]\, P$, and prefix $\tau.P$ in Figure 11, which are straightforward and therefore should not need further explanation.

4.4. *Encoding example*

To illustrate the encoding and especially the resulting computations, we consider an example. Let us consider the source term:

$$S = a(b).\mathbf{0} + \overline{a}\,\langle a\rangle.\mathbf{0} \mid a(b).\mathbf{0} + \overline{a}\,\langle a\rangle.\mathbf{0}$$

Note that this is a version of the term $a + \overline{a} \mid a + \overline{a}$ used in the Proof of Theorem 3.18, now without abbreviations, i.e. without omitting unnecessary parameters and trailing **0**'s. Since a and b are no special names of the encoding function we can assume without loss

$$
\begin{array}{l}
(\nu\, c, p_{i,up}, p_{o,up}, \mathsf{i}, \mathsf{o})\,\big(\\
\quad (\nu\, p_i, p_o)\,\big(\,(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \\
\qquad \mid (\nu\, r)\,\big(\,\overline{p_i}\,\langle a, r\rangle \mid r^*\,\big(l', s_1, s_2, b, c\big)\,. \\
\qquad\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_2} \mid (\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big) \mid \overline{c} \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r\rangle \\
\qquad\qquad\quad \mathsf{else}\ \overline{l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\big) \\
\qquad \mid (\nu\, s_1, s_2)\,\big(\overline{s_1} \mid s_1^*.\overline{p_o}\,\langle a, l, s_1, s_2, a\rangle \mid s_2.(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big)\big)\,\big) \\
\qquad \mid p_i{}^*\,(y, r)\,.\,\big(\overline{y\cdot \mathsf{o}}\,\langle r\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle\big) \\
\qquad \mid p_o{}^*\,(y, l, s_1, s_2, z)\,.\,\big(\overline{y\cdot \mathsf{i}}\,\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle\big)\,\big) \\
\quad \mid (\nu\, p_i, p_o)\,\big(\,(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \\
\qquad \mid (\nu\, r)\,\big(\,\overline{p_i}\,\langle a, r\rangle \mid r^*\,\big(l', s_1, s_2, b, c\big)\,. \\
\qquad\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_2} \mid (\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big) \mid \overline{c} \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r\rangle \\
\qquad\qquad\quad \mathsf{else}\ \overline{l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\big) \\
\qquad \mid (\nu\, s_1, s_2)\,\big(\overline{s_1} \mid s_1^*.\overline{p_o}\,\langle a, l, s_1, s_2, a\rangle \mid s_2.(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big)\big)\,\big) \\
\qquad \mid \overline{c} \mid p_i{}^*\,(y, r)\,.\,(y\cdot \mathsf{i}\,(l, s, z)\,.c.\overline{r}\,\langle l, s, z, c\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle) \\
\qquad \mid p_o{}^*\,(y, l, s_1, s_2, z)\,.\,(y\cdot \mathsf{o}\,(r)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle)\big) \\
\quad \mid p_{i,up}{}^*\,(y, r)\,.\overline{p_i}\,\langle y, r\rangle \mid p_{o,up}{}^*\,(y, l, s_1, s_2, z)\,.\overline{p_o}\,\langle y, l, s_1, s_2, z\rangle\,\big)
\end{array}
$$

Fig. 12. (Colour online) Encoding example: $[\![\, S \,]\!]$.

of generality that $\varphi_{[\![\,]\!]}(a) = a$ and $\varphi_{[\![\,]\!]}(b) = b$. The corresponding target term $[\![\, S \,]\!]$ is given by the term in Figure 12.

First we observe that the encoding of $\mathbf{0}$ – coloured blue in S and $[\![\, S \,]\!]$ – is simply $(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big)$ which is semantically equal to $\mathbf{0}$ because we have $(\nu\, l)\,\big(\overline{l}\,\langle\top\rangle \mid \mathbf{0}\big) \not\longrightarrow$. Secondly we observe that although the source term is a symmetric network of degree 2 (with respect to identity) the resulting target term is not a symmetric network. Note that the source term symmetry is broken because of the different encodings of the left and the right side of the parallel operator, i.e. mainly by the single instance of $\overline{c}$, and not by changing the degree of the source network. If we want to make sure that the encoding function does not change the degree of distribution by translating a network into a network of the same degree, we duplicate the last line of the encoding of the parallel operator and assign one instance of it to each side of the encoding of the parallel operator within different scopes of the names $p_{i,up}$ and $p_{o,up}$. Then performing alpha-conversion to push the restrictions on the different versions of the names o, i, and c outwards restores the degree of the original network. We observe that there are initially four requests within the target term; one for each input or output capability of the source term. Moreover, since the corresponding capabilities are unguarded in S, the requests are unguarded in $[\![\, S \,]\!]$ or can become so by a single target term step on the respective first sender lock.

Apart from the requests, there are three more unguarded outputs. Two of them are the positive instantiations of the sum locks $\overline{l}\,\langle\top\rangle$ to which no matching inputs are unguarded. The same holds for the instantiation of the coordinator lock $\overline{c}$. With that initially there are two steps on sender locks and then four steps, one for each request.

$$
\begin{aligned}
&(\nu\, p_i, p_o, l_l, r_l, s_{1,l}, s_{2,l})\,\big(\,\overline{l_l}\,\langle\top\rangle\\
&\quad \mid r_l^{*}\,\big(l', s_1, s_2, b, c\big)\,.\mathsf{test}\ l_l\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_l}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s_2} \mid [\![\,\mathbf{0}\,]\!] \mid \overline{c}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_l}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_l\rangle\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\\
&\quad \mid s_{1,l}^{*}.\overline{p_o}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle \mid s_{2,l}.[\![\,\mathbf{0}\,]\!]\\
&\quad \mid p_i{}^{*}\,(y, r)\,.\,\big(\overline{y\cdot \mathsf{i}}\,\langle r\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle\big) \mid \overline{a\cdot \mathsf{i}}\,\langle r_l\rangle \mid \overline{p_{i,up}}\,\langle a, r_l\rangle\\
&\quad \mid p_o{}^{*}\,(y, l, s_1, s_2, z)\,.\,(\overline{y\cdot \mathsf{o}}\,\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle)\\
&\quad \mid \overline{a\cdot \mathsf{o}}\,\langle l_l, s_{1,l}, s_{2,l}, a\rangle \mid \overline{p_{o,up}}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle\big)
\end{aligned}
$$

Fig. 13. (Colour online) Consumption of left requests.

At first we take a look on the left side of the encoding of the parallel operator. The consumption of the left requests leads to the term in Figure 13. Analysing this term we observe that the requests were not completely consumed but instead copied into a new version for each request with the same parameters but on different channel names, namely $\overline{p_{i,up}}\,\langle a, r_l\rangle$ and $\overline{p_{o,up}}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle$. The purpose of those copies is to push the content of the request over the restriction on p_i and p_o such that it can be pushed upwards in the parallel structure to enable communications with other parts of the binary tree. Note that the replicated inputs on the links p_i and p_o remain. So some of the requests might be processed at the beginning while other requests might be processed later. That allows us to handle the requests of the encoding of a continuation of some input or output guarded term as soon as the completion of a corresponding source term reduction step removing such a guard is mimicked within the encoding. Therefore, note that the encodings of continuations of guarded terms, i.e. the $[\![\,\mathbf{0}\,]\!]$ in our case, appear guarded within the encoding of the source term, where the guard is either a receiver lock in case of an input guarded source or a sender lock in case of an output guarded source. Moreover, we observe that these guards cannot be removed by reduction steps on requests. We also observe that the two reduction steps cause a scope extrusion of the restrictions on r, l, s_1 and s_2. Since in the current case there is only one instance of each of these locks no alpha-conversion is necessary. Multiple receiver/sender locks stem from multiple input/output guarded summands in the respective source term or from the case that at the corresponding side of the parallel operator a subtree of the parallel structure of the source term is encoded which can also leads to multiple sum locks. Later on we combine the requests of the left with the requests of the right in order to mimic a reduction step of the source term. Therefore, since on the right side there are different versions of these locks we perform alpha-conversion to avoid ambiguity later, i.e. we index the locks on the left side by l and the locks on the right side by r.

The processing of the requests on the right side of the encoding of a parallel operator as visualized in Figure 14 is similar. We observe that to enable a test there are some necessary informations: the receiver lock of the corresponding encoded input capability and the sum lock, the sender locks, and the sent value of the corresponding encoded output capability. The requests cover all these information. If a right request is processed the already gathered information are filled in (compare to $c.\overline{r_r}\,\langle l, s_1, s_2, z, c\rangle$ for right output

$$
\begin{aligned}
&(\nu\, p_i, p_o, l_r, r_r, s_{1,r}, s_{2,r})\,\big(\,\overline{l_r}\,\langle\top\rangle \mid \overline{c}\\
&\quad \mid r_r^*\,\big(l', s_1, s_2, b, c\big)\,.\mathsf{test}\ l_r\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s_2} \mid [\![\,\mathbf{0}\,]\!] \mid \overline{c}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_r}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_r\rangle\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\\
&\quad \mid s_{1,r}^*.\overline{p_o}\,\langle a, l_r, s_{1,r}, s_{2,r}, a\rangle \mid s_{2,r}.[\![\,\mathbf{0}\,]\!]\\
&\quad \mid {p_i}^*\,(y, r)\,.\,(y\cdot \mathsf{o}\,(l, s_1, s_2, z)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle)\\
&\quad \mid a\cdot \mathsf{o}\,(l, s_1, s_2, z)\,.c.\overline{r_r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{i,up}}\,\langle a, r_r\rangle\\
&\quad \mid {p_o}^*\,(y, l, s_1, s_2, z)\,.\,(y\cdot \mathsf{i}\,(r)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle)\\
&\quad \mid a\cdot \mathsf{i}\,(r)\,.c.\overline{r}\,\langle l_r, s_{1,r}, s_{2,r}, a, c\rangle \mid \overline{p_{o,up}}\,\langle a, l_r, s_{1,r}, s_{2,r}, a\rangle\big)
\end{aligned}
$$

Fig. 14. (Colour online) Consumption of right requests.

$$
\begin{aligned}
&(\nu\, c, p_{i,up}, p_{o,up}, \mathsf{i}, \mathsf{o}, l_l, r_l, s_{1,l}, s_{2,l}, l_r, r_r, s_{1,r}, s_{2,r})\,\big(\\
&(\nu\, p_i, p_o)\,\big(\overline{l_l}\,\langle\top\rangle\\
&\quad \mid r_l^*\,\big(l', s_1, s_2, b, c\big)\,.\mathsf{test}\ l_l\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_l}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s_2} \mid [\![\,\mathbf{0}\,]\!] \mid \overline{c}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_l}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_l\rangle\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\\
&\quad \mid s_{1,l}^*.\overline{p_o}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle \mid s_{2,l}.[\![\,\mathbf{0}\,]\!]\\
&\quad \mid {p_i}^*\,(y, r)\,.\,\big(\overline{y\cdot \mathsf{i}}\,\langle r\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle\big)\\
&\quad \mid {p_o}^*\,(y, l, s_1, s_2, z)\,.\,\big(\overline{y\cdot \mathsf{o}}\,\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle\big)\,\big)\\
&(\nu\, p_i, p_o)\,\big(\overline{l_r}\,\langle\top\rangle \mid \overline{c}\\
&\quad \mid r_r^*\,\big(l', s_1, s_2, b, c\big)\,.\mathsf{test}\ l_r\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s_2} \mid [\![\,\mathbf{0}\,]\!] \mid \overline{c}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_r}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_r\rangle\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1}\\
&\quad \mid s_{1,r}^*.\overline{p_o}\,\langle a, l_r, s_{1,r}, s_{2,r}, a\rangle \mid s_{2,r}.[\![\,\mathbf{0}\,]\!]\\
&\quad \mid {p_i}^*\,(y, r)\,.\,(y\cdot \mathsf{o}\,(l, s_1, s_2, z)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle)\\
&\quad \mid c.\overline{r_r}\,\langle l_l, s_{1,l}, s_{2,l}, a, c\rangle\\
&\quad \mid {p_o}^*\,(y, l, s_1, s_2, z)\,.\,(y\cdot \mathsf{i}\,(r)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle)\\
&\quad \mid c.\overline{r_l}\,\langle l_r, s_{1,r}, s_{2,r}, a, c\rangle\,\big)\\
&\mid {p_{i,up}}^*\,(y, r)\,.\overline{p_i}\,\langle y, r\rangle \mid \overline{p_i}\,\langle a, r_l\rangle \mid \overline{p_i}\,\langle a, r_r\rangle\\
&\mid {p_{o,up}}^*\,(y, l, s_1, s_2, z)\,.\overline{p_o}\,\langle y, l, s_1, s_2, z\rangle\\
&\mid \overline{p_o}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle \mid \overline{p_o}\,\langle a, l_r, s_{1,r}, s_{2,r}, a\rangle\big)
\end{aligned}
$$

Fig. 15. (Colour online) Encoding example: $[\![\,S\,]\!]\,(\longmapsto_{\mathrm{T}})^{12}$.

requests and $c.\overline{r}\,\langle l_r, s_{1,r}, s_{2,r}, a, c\rangle$ for right input requests). The missing details are gathered by the communication with a matching left request.

Now there are two concurrently enabled steps, at channel $a\cdot\mathsf{i}$ and at $a\cdot\mathsf{o}$. One for each possible step of the source term. Note that the two steps of the source term are in conflict, whereas the two steps here are not conflicting. The result of these two steps and the four steps on the channels $p_{i,up}$ and $p_{o,up}$ is given in Figure 15. We observe that the missing details are filled in. The results are two outputs on receiver locks each guarded by the same coordinator lock c. Since there is only one instantiation of that lock, we

$$
\begin{aligned}
&(\nu\, p_i, p_o)\,\big(\,\overline{l_r}\,\langle\top\rangle \\
&\quad \mid r_r^{*}\,\big(l', s_1, s_2, b, c\big)\,.\mathsf{test}\ l_r\ \mathsf{then\ test}\ l'\ \mathsf{then}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{s_2} \mid [\![\,\mathbf{0}\,]\!] \mid \overline{c} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_r}\,\langle\top\rangle \mid \overline{l'}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_r\rangle \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_1} \\
&\quad \mid \mathsf{test}\ l_r\ \mathsf{then\ test}\ l_l\ \mathsf{then}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{l_l}\,\langle\bot\rangle \mid \overline{s_{2,l}} \mid \{\,{}^{a}\!/_{b}\,\}\,([\![\,\mathbf{0}\,]\!]) \mid \overline{c} \\
&\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_r}\,\langle\top\rangle \mid \overline{l_l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{p_i}\,\langle a, r_r\rangle \\
&\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_r}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_{1,l}} \\
&\quad \mid s_r.[\![\,\mathbf{0}\,]\!] \\
&\quad \mid p_i{}^{*}\,(y, r)\,.\,(y \cdot \mathsf{o}\,(l, s_1, s_2, z)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{i,up}}\,\langle y, r\rangle) \\
&\quad \mid p_o{}^{*}\,(y, l, s_1, s_2, z)\,.\,(y \cdot \mathsf{i}\,(r)\,.c.\overline{r}\,\langle l, s_1, s_2, z, c\rangle \mid \overline{p_{o,up}}\,\langle y, l, s_1, s_2, z\rangle) \\
&\quad \mid c.\overline{r_l}\,\langle l_r, s_{1,r}, s_{2,r}, a, c\rangle\,\big)
\end{aligned}
$$

Fig. 16. (Colour online) Unguarding a test-construct.

can only reduce one of these terms by now. The consumption of a coordinator lock and a following step on the receiver lock enables a test of the sum locks of the respective found pair of matching communication partners. Note that since such a pair consists of a communication partner left and a partner right to the encoding of the respective parallel operator these two sum locks are always different. With that the problem of incestuous sums described in Example 4.2 is avoided.

We reduce the first occurrence of the guard of the coordinator lock. The other case is similar. A second step removes the receiver lock – r_r in this case – that guards the right nested test-construct as depicted in Figure 16. Since, both sum lock are positive instantiated – the instantiation of the left sum lock l_l can be found on the left side – the unguarded nested test-construct reduce to its first case, i.e. it reduces to:

$$\overline{l_r}\,\langle\bot\rangle \mid \overline{l_l}\,\langle\bot\rangle \mid \overline{s_{2,l}} \mid \{\,{}^{a}\!/_{b}\,\}\,([\![\,\mathbf{0}\,]\!]) \mid \overline{c}$$

The reduction to the first case shows that our communication attempt on the identified pair of communication partners was successful, i.e. at this point we mimic the corresponding source term step. Both sum locks are changed to false instantiations. This outlines that already a summand of each of these two sums was used for communication. Then there is an unguarded instantiation of the left sender lock $s_{2,l}$. With that we can remove the guard of the encoding of the continuation of the output guarded left source term within one more reduction step. The encoding of the continuation of the respective input guarded source term is the term $\{\,{}^{a}\!/_{b}\,\}\,([\![\,\mathbf{0}\,]\!]) = [\![\,\mathbf{0}\,]\!]$. As we can observe it is already unguarded. Moreover, the value send by the respective left source term output was received at the encoding of the right input guarded term as depicted by the substitution $\{\,{}^{a}\!/_{b}\,\}$. The last new subterm is an instantiation of the coordinator lock. Since the test of the two sum locks is finished another test can be enabled.

S can perform only a single step, which we have mimicked within its encoding. The encoded term can perform some post-processing steps. Since there is an instantiation of the coordinator lock the second test of sum locks on the second pair of matching communication partners can be performed. Again the guarding coordinator lock is reduced

by a reduction step. Then an additional step is performed to remove the left receiver lock guarding the left nested test-construct. The first sum lock, which is l_l in this case, is tested. There is only one instantiation of the lock and that is a negative instantiation. With that the test is finished immediately without testing the second lock and the nested test-construct reduce to its last case, i.e. to $\overline{l_l}\,\langle\bot\rangle \mid \overline{c} \mid \overline{s_{1,r}}$. With that a subsequently step results in a duplicate version of the right output request.

Moreover, the completion of the second nested test-construct results in an other instantiation of the coordinator lock. Indeed, analysing the encoding function we observe that each completion of a test-construct block restores the coordinator lock consumed to enable this test regardless of its outcome. In our case we found no other matching pair of communication partners, so no further tests can be enabled.

Note that the resulting target term contains five unguarded and not restricted, i.e. free, requests: $\overline{p_i}\,\langle a, r_l\rangle$, $\overline{p_i}\,\langle a, r_r\rangle$, $\overline{p_o}\,\langle a, l_l, s_{1,l}, s_{2,l}, a\rangle$, and twice $\overline{p_o}\,\langle a, l_r, s_{1,r}, s_{2,r}, a\rangle$. They can be bound by a surrounding parallel operator, i.e. by the next parent node. Since in our example there is no such surrounding parallel operator they remain free.

4.5. *Properties of the encoding*

Abstracting from the instantiation of sender locks we observe that the encoding function translates source term observables into *translated observables*, i.e. into requests containing the name of the corresponding observable as first parameter augmented with positive instantiations of sum locks. A source term step $S \longmapsto_{\mathrm{S}} S'$ is translated into a sequence of target term steps. Most of these steps are pre- or post-processing steps, which we call *administrative steps*. Indeed, each source term step can be mapped to exactly one non-administrative target step and vice versa, namely to a step from the second test-construct of a nested test-construct or from a single test-construct to its then-case. All steps of the target term before or after such a step are administrative steps.

Based on the notion of translated observables we define *translated barbed bisimilarity* (denoted by $\bowtie$) as weak reduction bisimilarity augmented with a check for the same set of translated barbs. The main purpose of this relation is to capture our intuition of the connection between source and target terms. Moreover, it turns out that $\bowtie$ is well suited to prove the missing three criteria on the quality of the encoding; it identifies terms with the same behaviour with respect to translated observables. Interestingly, the relation $\bowtie$ identifies structural congruent terms, but it does not *preserve* the structural congruence of source terms. The reason is that the encoding obviously treats the two sides of the rule $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$ quite differently.

To circumvent problems with cyclic sums (compare to Example 4.3), the encoding function introduces a so-called coordinator lock. It ensures that at each node of the parallel structure of the source term, and at any moment in time, at most one test can be enabled. Remember that the reduction of a test-construct to its then-case is a non-administrative step. Here, the enabling of a non-administrative step at a node may block the enabling of an alternative non-administrative step at the same node. This blocking of non-administrative steps leads to (1) *intermediate states*, in case the blocked non-administrative step is in conflict to the enabled step, and (2) *additional causal*

dependencies, in case the blocked non-administrative step is independent of the enabled step. Intuitively, an intermediate state is a state of the target term that cannot be mapped to a state of the source term but can be placed in between a source term state and some, but not all, of its subsequent states. In comparison to the encoding of the former state, some of the alternative conflicting non-administrative steps are ruled out, while there is still some choice on conflicting non-administrative steps leading to the encoding of different subsequent source term states.

Of course, these effects – the introduction of intermediate states and additional causal dependencies and the fact that associativity of the parallel operator in the source terms is not preserved – rely on the fact that the encoding function takes such strong advantage of the parallel structure of the source term. However, we believe that any encoding from π_{mix} into π_{sep} (and likewise into π_{a}) will introduce intermediate states and additional causal dependencies, although we have up to now only proven the second claim in Peters *et al.* (2011). Note that causality is often defined as the opposite of concurrency, i.e. two actions are concurrent only if they are not causal dependent (Boreale and Sangiorgi 1998; Charron-Bost *et al.* 1996; Priami 1996). Hence Peters *et al.* (2011) prove that there is no good encoding from π_{mix} into π_{a} that preserves the degree of distribution of source terms, because each such encoding introduces additional causal dependencies and thus reduces the amount of concurrent enabled steps. We particularized this fact in Peters and Nestmann (2012a), hence – restoring the original intention of Palamidessi (2003) – there is no distribution preserving encoding from π_{mix} into π_{a} with respect to the criteria of Gorla.

4.5.1. *Monadic versus polyadic communication.* To reduce the complexity of the encoding function, we chose to use a monadic version of π_{mix}, but a polyadic version of π_{a}. There are decent encodings from the polyadic π-calculus into its monadic variant, especially when the polyadic usage is limited. Such an encoding is described for the case of two parameters in Nestmann and Pierce (2000). Here, the encoding uses at most four parameters. It is straightforward to adapt the encoding function in Figure 10 using an encoding similar to the one in Nestmann and Pierce (2000) to reduce each input, replicated input, and output to one parameter, i.e. to translate the encoding into an encoding from π_{mix} into the monadic version of π_{a}.

4.5.2. *Proof of correctness.* We will not present a formal proof of the correctness of our encoding here, but in Peters and Nestmann (2012b) there is an exhaustive argumentation, why the encoding presented in Peters and Nestmann (2012a) is correct with respect to the five criteria presented in Section 2.2. The main difference of the encoding presented above and the encoding of Peters and Nestmann (2012a) is that the former uses *coordinator looks* to rule out potential deadlocks caused by cyclic sums, while the later implements an algorithm to compute an *order on the sum locks* at runtime without global knowledge, i.e. in a compositional way. To do so both encodings rely on the idea of requests, presented above. Hence much of the argumentation in Peters and Nestmann (2012b) can be simply adapted or used verbatim for a similar argumentation here.

Since it is the main difference of our encoding and the encoding in Peters and Nestmann (2012a), we shortly explain how our encoding avoids the deadlocks caused by cyclic sums. In the encoding given in Nestmann (2000), several nested test-constructs can lead to deadlocks, because they consume some sum locks while waiting for other sum locks (see Section 4.2). The coordinator locks restrict the number of test-constructs that can be evaluated concurrently. More precisely, for each related pair of input and output requests the instantiation of the coordinator lock of the closest common parent node is consumed. Therefore, the consumed coordinator locks and the tested sum locks form a binary tree with the sum locks as leafs and the coordinator locks as remaining nodes. For each consumed coordinator lock a nested test-construct is enabled, that checks two sum locks of leafs of the corresponding subtree. Hence there is always at least one more sum lock than enabled test-constructs, i.e. at least one sum lock is tested by at most one test-construct. Because of that it is not possible to close the cycle, i.e. there are no deadlocks on the test-constructs.

4.5.3. *Match.* For the presented idea of an encoding from π_{mix} into π_{a} as well as for the encoding presented in Peters and Nestmann (2012a), the match operator seems crucial, i.e. we strongly believe that there is no encoding from π_{mix} into π_{sep} satisfying all criteria presented in Section 2.2 without match. We have not formally proven this claim yet, but in Peters *et al.* (2011) and Peters and Nestmann (2012a) some argumentation can be found. The main idea is, that according to the proof in Peters *et al.* (2011) each encoding has to utilize some kind of requests, that are combined at the level of an encoded parallel operator. Here, matching is necessary to identify requests that belong to the same source term step. To avoid this kind of matching, related requests have to be identified by communication on restricted channels. This restriction requires global knowledge – e.g. to cover source term steps that result from scope extrusion – which is not available in compositional encodings (see Definition 2.5). Note that the only proof that π_{a} is strictly more expressive than π_{a} without match in Carbone and Maffeis (2003) we are aware of, relies on criteria harder than Gorla's criteria, but we believe that the proof can be generalized to the criteria presented in Section 2.2.

5. Summary and future work

We prove without any further assumption that π_{sep} lacks – in contrast to π_{mix}– the ability to break initial symmetries. For this reason, we state an absolute separation result proving that π_{mix} is strictly more expressive than π_{sep}. Moreover, since homomorphic translation of the parallel operator preserves initial symmetries, this absolute result turns out to be well suited to prove several translational separation results for respectively different definitions of reasonableness. These results support the conjecture that there is no reasonable encoding from π_{mix} into π_{sep} that translates the parallel operator homomorphically, where the notion of reasonableness covers at least reflection of divergence and deadlock, and some suitable notion of preservation of behaviour. Moreover, comparing the presented translational separation results we observe that:

1. the absolute separation result plays a central role in each of the above-presented translational results,
2. the use of the absolute separation result allows us to weaken the assumptions under which the translational separation results hold in comparison to earlier proposals,
3. the use of the absolute separation result induces an intuitive way to prove quite different translational separation results.

In summary, these arguments emphasize the central role of absolute separation results for language comparison, even when considering translational results. Note that even with the help of match, π_{sep} cannot break symmetries and there is no uniform and reasonable encoding from π_{mix} into π_{sep} or π_{a}.

As shown in Palamidessi (2003), leader election serves to derive a translational result, but even input-output confluence suffices to separate π_{mix} from π_{sep} by an absolute result. Absolute separation results like confluence, leader election, and breaking symmetries can be used to obtain translational separation results. Therefore, typically an example is chosen that illustrates the main discriminating features of the absolute result and that can be used in the translational separation result as counterexample. To do so, the main features of this example have to be preserved by the encoding function, i.e. by the criteria required for the encoding. Thus, confluence is not an adequate choice to derive a translational separation result as the above, since it is very difficult to find a discriminating counterexample based on confluence; even if such an example is found, it is intricate to argue for the preservation of its properties. In this sense, leader election is much more suitable, because its main conditions are preserved under uniform encodings that preserve substitutions. However, as shown above, breaking symmetries is even better suited because its properties are preserved by weaker requirements on reasonable encodings. Accordingly, confluence can be considered as a too weak property, while leader election is a little bit too specific. In short, breaking symmetries serves as a 'sweet spot'. It allows for the formulation of a general result: there is no reasonable encoding from π_{mix} into π_{sep} that translates the parallel operator homomorphically, where the notion of reasonableness covers at least reflection of divergence and deadlock, and some suitable notion of preservation of behaviour.

By abandoning the condition of homomorphic translation of the parallel operator in favour of weak compositionality, we propose an encoding from π_{mix} into π_{a} that, as we strongly conjecture, meets all five of Gorla's criteria. As a novelty, our new encoding overcomes the previous non-compositional attempts to break symmetries globally by providing a principle that breaks symmetries locally, saving true compositionality. Because of the complexity of this encoding, a complete proof of its correctness with respect to Gorla's criteria is rather intricate (compare to Peters and Nestmann (2012b)). However, it should reveal greater insight on the relation between π_{mix} and π_{a} and thus, hopefully, on the relation between synchronous and asynchronous interactions in general. For instance, it preserves structural congruence of source terms except for the associativity of the parallel operator, which at first may seem rather unexpected. Even more interesting is the introduction of intermediate states and additional causal dependencies. Note that the necessity of the latter is already proved Peters *et al.* (2011). We are convinced that also

the former is no effect of the particularities of the chosen encoding; rather, we conjecture that any encoding from π_{mix} into π_{sep} that is correct with respect to Gorla's criteria also introduces intermediate states. We also conjecture that the match operator is required for such encodings.

Symmetry is also a concept of practical relevance. Likewise, from the early days of CSP (Hoare 1978), researchers were looking for convincing ways to practically implement mixed choice – then called *generalized input-output construct* or *generalized alternative command* (Bernstein 1980; Buckley and Silberschatz 1983; Kieburtz and Silberschatz 1979; van de Snepshout 1981) – which turned out to be hard to do correctly (Kumar and Silberschatz 1997). However, in practice, systems are never truly symmetric when considered as sitting on top of some network architecture. In order to achieve global ordering information, one can always use IP addresses, process IDs, etc and exploit them to break symmetries. Ideas along this path have been pursued to implement mixed choice in the context of a programming language (Knabe 1993) beyond mere proof-of-concept prototyping. On the other hand, the relevance of symmetry is also a matter of the level of abstraction. Not referring to the underlying technical architecture in all low-level details, we are forced to abandon solutions that require global knowledge in favour of those that only require local knowledge. Finding a local-knowledge solution is then practically useful for issues like scalability and fault-tolerance.

References

Baldamus, M., Parrow, J. and Victor, B. (2005) A fully abstract encoding of the i-calculus with data terms. In: Proceedings of ICALP. *Springer Lecture Notes in Computer Science* **3580** 1202–1213.

Bernstein, A. (1980) Output guards and nondeterminism in 'communicating sequential processes'. *ACM Transactions on Programming Languages and Systems* **2** (2) 234–238.

Boer, F. S. and Palamidessi, C. (1991) Embedding as a tool for Language Comparison: On the CSP hierarchy. In: Proceedings of CONCUR. *Springer Lecture Notes in Computer Science* **527** 127–141.

Boreale, M. and Sangiorgi, D. (1998) A fully abstract semantics for causality in the π-calculus. *Acta Informatica* **35** (5) 353–400.

Boudol, G. (1992) Asynchrony and the π-calculus (note). Note, INRIA.

Bougé, L. (1988) On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Informatica* **25** (4) 179–201.

Buckley, G. and Silberschatz, A. (1983) An effective implementation for the generalized input-output construct of CSP. *ACM Transactions on Programming Languages and Systems* **5** (2) 223–235.

Bugliesi, M. and Giunti, M. (2007) Secure implementations of typed channel abstractions. In: Proceedings of POPL. *SIGPLAN-SIGACT* **42**, ACM 251–262.

Busi, N., Gorrieri, R. and Zavattaro, G. (2000) On the expressiveness of linda coordination primitives. *Information and Compututation* **156** (1–2) 90–121.

Carbone, M. and Maffeis, S. (2003) On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing* **10** (2) 70–98.

Charron-Bost, B., Mattern, F. and Tel, G. (1996) Synchronous, asynchronous, and causally ordered communication. *Distributed Computing* **9** (4) 173–191.

Fu, Y. and Lu, H. (2010) On the expressiveness of interaction. *Theoretical Computer Science* **411** (11–13) 1387–1451.

Gorla, D. (2008a) Comparing communication primitives via their relative expressive power. *Information and Computation* **206** (8) 931–952.

Gorla, D. (2008b) Towards a unified approach to encodability and separation results for process calculi, *Technical Report*, Dip. di Informatica, Univ. di Roma 'La Sapienza', 2008. (An extended abstract appeared in the Proceedings of CONCUR'08. *Springer Lecture Notes in Computer Science* **5201** 492–507.)

Gorla, D. (2010) Towards a unified approach to encodability and separation results for process calculi. *Information and Computation* **208** (9) 1031–1053.

Hoare, C. A. R. (1978) Communicating sequential processes. *Communications of the ACM* **21** (8) 666–677.

Herescu, O. M. and Palamidessi, C. (2002) A randomized distributed encoding of the pi-calculus with mixed choice. In: Baeza-Yates, R. A., Montanari, U. and Santoro, N. (eds.) IFIP TCS. *IFIP Conference Proceedings*, Kluwer 537–549.

Honda, K. and Tokoro, M. (1991) An object calculus for asynchronous communication. In: Proceedings of ECOOP. *Springer Lecture Notes in Computer Science* **512** 133–147.

Johnson, R. E. and Schneider, F. B. (1985) Symmetry and similarity in distributed systems. In: *Proceedings of PODC*, ACM 13–22.

Knabe, F. (1993) A distributed protocol for channel-based communication with choice. *Computers and Artificial Intelligence* **12** (5) 475–490.

Kieburtz, R. and Silberschatz, A. (1979) Comments on 'communicating sequential processes'. *ACM Transactions on Programming Languages and Systems* **1** (2) 218–225.

Kumar, D. and Silberschatz, A. (1997) A counter-example to an algorithm for the generalized input-output construct of CSP. *Information Processing Letters* **61** 287.

Lipton, R., Snyder, L. and Zalcstein, Y. (1974) A comparative study of models of parallel computation. In: *15th Annual Symposium on Switching and Automata Theory, New Orleans* 145–155.

Milner, R., Parrow, J. and Walker, D. (1992) A calculus of mobile processes, part I and II. *Information and Computation* **100** (1) 1–77.

Milner, R. and Sangiorgi, S. (1992) Barbed sisimulation. In: Proceedings of ICALP. *Springer Lecture Notes in Computer Science* **623** 685–695.

Nestmann, U. (2000) What is a 'Good' encoding of guarded choice? *Information and Computation* **156** (1-2) 287–319.

Nestmann, U. (2006) Welcome to jungle: A subjective guide to mobile process calculi. In: Proceedings of CONCUR. *Springer Lecture Notes in Computer Science* **4137** 52–63.

Nestmann, U. and Pierce, B. C. (2000) Decoding choice encodings. *Information and Computation* **163** (1) 1–59.

Palamidessi, C. (2003) Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science* **13** (5) 685–719.

Parrow, J. (2008) Expressiveness of process algebras. *Electronic Notes in Theoretical Computer Science* **209** 173–186.

Peters, K. and Nestmann, U. (2010) Breaking symmetries. In: Proceedings of EXPRESS. *Electronic Proceedings in Theoretical Computer Science* **41** 136–150.

Peters, K. and Nestmann, U. (2012a) Is it a 'Good' encoding of mixed choice? In: Proceedings of FoSSaCS. *Lecture Notes in Computer Science* **7213** 210–224.

Peters, K and Nestmann, U. (2012b) Is it a 'Good' encoding of mixed choice? *Technical Report*, TU Berlin, Germany. http://arxiv.org/corr/home.

Peters, K., Schicke-Uffmann, J.-W. and Nestmann, U. (2011) Synchrony versus causality in the asynchronous Pi-calculus. In: Proceedings of EXPRESS. *Electronic Notes in Theoretical Computer Science* **64** 89–103.

Priami, C. (1996) *Enhanced Operational Semantics for Concurrency*, Ph.D. thesis, Università di Pisa-Genova-Udine.

Sangiorgi, D. and Walker, D. (2001) *The π-Calculus: A Theory of Mobile Processes*, Cambridge University Press New York, NY, USA.

Shapiro, E. (1989) The family of concurrent logic programming languages. *ACM Computing Surveys (CSUR)* **21** (3) 413–510.

Shapiro, E. (1991) Separating concurrent languages with categories of language embeddings. In: *Proceedings of STOC*, ACM 198–208.

Shapiro, E. (1992) Embeddings among concurrent programming languages. In: Proceedings of CONCUR. *Springer Lecture Notes in Computer Science* **630** 486–503.

van de Snepshout, J. (1981) Synchronous communication between asynchronous components. *Information Processing Letters* **13** (3) 127–130.

van Glabbeek, R. J. (1993) The linear time - branching time spectrum II. In: Proceedings of CONCUR. *Springer Lecture Notes in Computer Science* **715** 66–81.

van Glabbeek, R. J. (2001) The linear time – branching time spectrum I: The semantics of concrete, sequential processes. In: Bergstra, J.A., Ponse, A. and Smolka, S. A. (eds.) *Handbook of Process Algebra*, Elseveier Science B.V. 3–99.

Vigliotti, M. G., Phillips, I. and Palamidessi, C. (2007) Tutorial on separation results in process calculi via leader election problems. *Theoretical Computer Science* **388** (1–3) 267–289.