

Synchrony vs Causality in the Asynchronous Pi-Calculus*

Kirstin Peters

School of EECS, TU Berlin, Germany
kirstin.peters@tu-berlin.de

Jens-Wolfhard Schicke

Institute for Programming and Reactive Systems, TU Braunschweig, Germany
drahflow@gmx.de

Uwe Nestmann

School of EECS, TU Berlin, Germany
uwe.nestmann@tu-berlin.de

We study the relation between process calculi that differ in their either synchronous or asynchronous interaction mechanism. Concretely, we are interested in the conditions under which synchronous interaction can be implemented using just asynchronous interactions in the π -calculus. We assume a number of minimal conditions referring to the work of Gorla: a “good” encoding must be compositional and preserve and reflect computations, deadlocks, divergence, and success. Under these conditions, we show that it is *not* possible to encode synchronous interactions without introducing additional causal dependencies in the translation.

Keywords: asynchrony, distributed systems, causality, pi-calculus

1 Introduction

We study the relation between process calculi that differ in their either synchronous or asynchronous interaction mechanism. Synchronous and asynchronous interactions are the two basic paradigms of interactions in distributed systems. While synchronous interactions are widely used in specification languages, asynchronous interactions are often better suited to implement real systems. We are interested in the conditions under which synchronous interactions can be implemented using just asynchronous interactions, i.e., in the conditions under which it is possible to encode the synchronous π -calculus into its asynchronous variant. To partially answer this question, we examine the role of causality for encoding synchrony.

Of course, we are not interested in trivial or meaningless encodings. Instead we consider only those encodings that ensure that the original term and its encoding show to some extent the same abstract behaviour. Unfortunately, there is no consensus about what properties make an encoding “good” (compare e.g. [12]). Instead, we find separation results as well as encodability results with respect to very different conditions, which naturally leads to incomparable results. Among these conditions, a widely used criterion is *full abstraction*, i.e. the preservation and reflection of equivalences associated to the two compared languages. There are lots of different equivalences in the range of π -calculus variants. Since full abstraction depends, by definition, strongly on the chosen equivalences, a variation in the respective choice may

*This work was supported by the DFG (German Research Foundation), grants NE-1505/2-1 and GO-671/6-1.

change an encodability result into a separation result, or vice versa. Unfortunately, there is neither a common agreement about what kinds of equivalence are well suited for language comparison—again, the results are often incomparable. To overcome these problems, and to form a more robust and uniform approach for language comparison, Gorla [5, 6] identifies five criteria as being well suited for separation as well as encodability results. In this paper, we rely on these five criteria. Compositionality and name invariance stipulate structural conditions on a good encoding. Operational correspondence requires that a good encoding preserves and reflects the computations of a source term. Divergence reflection states that a good encoding shall not exhibit divergent behaviour, unless it was already present in the source term. Finally, success sensitiveness requires that a source term and its encoding have exactly the same potential to reach successful state.

A discussion on synchrony versus asynchrony cannot be separated from a discussion of choice. When processes communicate via message-passing along channels, they do not only listen to one channel at a time—they concurrently listen to a whole selection of channels. Choice operators just make this natural intuition explicit; moreover, their mutual exclusion property allows us to concisely describe the particular effect of message-passing actions on the process’s local state. Asynchronous send actions make no sense as part of a mutually exclusive selection, as they cannot be prevented from happening. Consequently, the asynchronous calculus only offers input-guarded choice. In contrast, synchronous send actions also allow for the definition of mixed choice: selections of both input and output actions.

It is well known that there is a good encoding from the choice-free synchronous π -calculus into its asynchronous variant [2, 8, 7]. It is also well-known [11, 6, 13] that there is no good encoding from the full π -calculus—the synchronous π -calculus including mixed choice—into its asynchronous variant if the encoding translates the parallel operator homomorphically. Palamidessi was the first to point out that mixed choice strictly raises the absolute expressive power of the synchronous π -calculus compared to its asynchronous variant. Analysing this result [13], we observe that it boils down to the fact that the full π -calculus can break merely syntactic symmetries, where its asynchronous variant can not; there is no need to refer to a semantic problem like the existence of solutions to leader election. Moreover, as already Gorla [6] states, the condition of homomorphic translation of the parallel operator is rather strict. Therefore, Gorla proposes the weaker criterion of compositional translation of the source language operators (see Definition 2.5 at page 94). As claimed in [14], this weakening of the structural condition on the encoding of the parallel operator turns the separation result into an encodability result, i.e., there is an encoding from the synchronous π -calculus (including mixed choice) into its asynchronous variant with respect to the criteria of Gorla¹. Analysing the encoding attempt given in [14], we observe that it introduces additional causal dependencies, i.e., causal dependencies that were not present in the source term and thus introduced by the encoding function. Note, that a step B is considered causally dependent on a previous step A , if B depends on the availability of data produced by A . In this paper, we show that this is a general phenomenon of encoding synchrony.

Thus, as the main contribution of this paper, we show that—in the asynchronous π -calculus—there is a strong connection between synchronous interactions and causal dependencies. More precisely, we show that it is *not* possible to encode synchronous interactions within a completely asynchronous framework without introducing additional causal dependencies in the translation. Moreover, we discuss the role of mixed choice to derive this result for the π -calculus. The companion paper [17] presents a similar result in the context of Petri nets. Hence, this connection between synchronous interactions and causal dependencies is presumably no effect of the representation of concurrent systems in either the π -calculus

¹Note that this encoding is neither prompt nor is the assumed equivalence \simeq strict, so the separation results of [5] and [6] do not apply here.

or Petri nets, but rather a phenomenon of synchronous and asynchronous interactions in general.

Overview of the Paper. In §2, we introduce the synchronous π -calculus and its asynchronous variant. We revisit some notions and results of [13], recall the five criteria of [6] to measure the quality of an encoding, and talk about a definition of causality. In §3, we present our separation result and discuss some observations on its proof. We conclude in §4 with a comparison to a similar result in [17].

2 Technical Preliminaries

2.1 The π -calculus

Our source language is the monadic π -calculus as described for instance in [16]. 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., $\overline{\mathcal{N}} = \{ \bar{n} \mid n \in \mathcal{N} \}$. We use lower case letters $a, a', a_1, \dots, x, y, \dots$ to range over names.

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

$$P ::= (vn)P \quad | \quad P_1 \mid P_2 \quad | \quad !P \quad | \quad \sum_{i \in I} \pi_i.P_i$$

where $\pi ::= y(x) \mid \bar{y}\langle z \rangle$ for some names $n, x, y, z \in \mathcal{N}$ and a finite index set I .

The interpretation of the defined process terms is as usual. Since all examples and counterexamples within this paper are CCS-like we omit the objects of actions. Moreover we denote the empty sum with $\mathbf{0}$ and omit it in continuations. As usual we often notate a sum $\sum_{i \in \{i_1, \dots, i_n\}} \pi_i.P_i$ by $\pi_{i_1}.P_{i_1} + \dots + \pi_{i_n}.P_{i_n}$.

As target language, we use π_a , the asynchronous π -calculus (see [8] or [2]).

Definition 2.2 (π_a). The set of process terms of the asynchronous π -calculus, denoted by \mathcal{P}_a , is given by

$$P ::= (vn)P \quad | \quad P_1 \mid P_2 \quad | \quad !P \quad | \quad \mathbf{0} \quad | \quad \bar{y}\langle z \rangle \quad | \quad y(x).P \quad | \quad [a = b]P$$

for some names $n, a, b, x, y, z \in \mathcal{N}$.

Here, we equip the target language with the match operator, because it is used in [14], which introduces the only good encoding that we are aware of between the synchronous π -calculus (with mixed choice) and its asynchronous variant. With [14], this encoding depends on the availability of the match operator in the target language; we do not yet know whether there is such an encoding without match. Since matching do increase the expressive power of the asynchronous π -calculus (see [3]), answering this question is an important task for future work. However, note that the proof of our main result in Theorem 3.8 does not depend on this decision.

As shown by the encoding in [10] one could also use separate choice within an asynchronous variant of the calculus without a significant effect on its expressive power. We claim, that our main result does not depend on the decision whether to allow separate choice or not. In Section 3.1 we give some hints on how to change the proof to capture separate choice in the target language.

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

$$\begin{array}{l}
P \equiv Q \text{ if } Q \text{ can be obtained from } P \text{ by renaming one or more of the bound names in } P, \\
\text{silently avoiding name clashes} \\
P | \mathbf{0} \equiv P \quad P | Q \equiv Q | P \quad P | (Q | R) \equiv (P | Q) | R \quad [a = a] P \equiv P \quad !P \equiv P | !P \\
(\nu n) \mathbf{0} \equiv \mathbf{0} \quad (\nu n) (\nu m) P \equiv (\nu m) (\nu n) P \quad P | (\nu n) Q \equiv (\nu n) (P | Q) \text{ if } n \notin \text{fn}(P)
\end{array}$$

Figure 1: Structural Congruence.

to both calculi. Let $\text{fn}(P)$ denote the set of *free names* in P . Let $\text{bn}(P)$ denote the set of *bound names* in P . Likewise, $\text{n}(P)$ denotes the set of all *names* occurring in P . Their definitions are completely standard.

The *reduction semantics* of π_{mix} and π_a are jointly given by the transition rules in Figure 2, where *structural congruence*, denoted by \equiv , is given by the rules in Figure 1. Note that the rule COM_a for communication in π_a is a simplified version of the rule COM_{mix} for communication in π_{mix} . The differences between these two rules result from the differences in the syntax, i.e. the lack of choice and the fact that only input can be used as guard in π_a . As usual, we use \equiv_α if we refer to alpha-conversion (the first rule of Figure 1) only.

$$\begin{array}{l}
\text{COM}_{\text{mix}} \quad (\dots + y(x).P + \dots) | (\dots + \bar{y}\langle z \rangle.Q + \dots) \mapsto \{z/x\} P | Q \\
\text{COM}_a \quad y(x).P | \bar{y}\langle z \rangle \mapsto \{z/x\} P \\
\text{PAR} \quad \frac{P \mapsto P'}{P | Q \mapsto P' | Q} \quad \text{RES} \quad \frac{P \mapsto P'}{(\nu n) P \mapsto (\nu n) P'} \\
\text{CONG} \quad \frac{P \equiv P' \quad P' \mapsto Q' \quad Q' \equiv Q}{P \mapsto Q}
\end{array}$$

Figure 2: Reduction Semantics of π_{mix} and π_a .

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

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

The first quality criteria presented in Section 2.3 is compositionality. It induces the definition of a

context parametrised on a set of names for each operator of π_{mix} . A context $\mathcal{C}([\cdot]_1, \dots, [\cdot]_n)$ is simply a π -term, i.e. a π_a -term in case of Definition 2.5, with n holes. Putting some π_a -terms P_1, \dots, P_n in this order into the holes $[\cdot]_1, \dots, [\cdot]_n$ of the context, respectively, gives a term denoted $\mathcal{C}(P_1, \dots, P_n)$. Note that a context may bind some free names of P_1, \dots, P_n . The arity of a context is the number of its holes.

2.2 Symmetry in the π -calculus

A *network* of degree n is a process $(\nu \tilde{x})(P_1 \mid \dots \mid P_n)$ for some $n \in \mathbb{N}$, some $P_1, \dots, P_n \in \mathcal{P}$ and a sequence of names \tilde{x} . We refer to P_1, \dots, P_n as the processes of the network. Note that the processes of a network can be networks itself. A *symmetric network* of degree n is a network of degree n such that $P_i = \sigma^{i-1}(P_1)$ for all $i \in \{2, \dots, n\}$ and some substitution σ , called *symmetry relation*, such that $\sigma^n = \text{id}$. The minimal degree of a symmetry relation σ is the smallest $n > 0$ such that $\sigma^n = \text{id}$. A *symmetric execution* is an execution starting at a symmetric network of degree n , returning to a symmetric network of the same degree after any n 'th step, and which is either infinite or terminates in a symmetric network of degree n . Let π_{sep} be the subcalculus of π_{mix} with separate but no mixed choice, i.e. there is no sum with both input- and output-guarded summands, and let \mathcal{P}_{sep} be its set of processes. The first and the last author [13] prove that it is not possible in π_{sep} to break the symmetry of a symmetric network. More precisely, in Theorem 4.4 in [13], it is shown that any symmetric network in \mathcal{P}_{sep} has at least one symmetric execution. Since \mathcal{P}_a is a subset of \mathcal{P}_{sep} , we obtain the following lemma.

Lemma 2.3. *Every symmetric network in \mathcal{P}_a has at least one symmetric execution.*

Moreover, by Lemma 5.4 in [13], we know that if the minimal degree of the symmetry relation is smaller than the degree of the symmetric network, then not only the network can be subdivided into a network of symmetric networks but also its symmetric execution. As already done in [13], we use this Lemma in the context of a symmetric network $P \mid P$ for some arbitrary $P \in \mathcal{P}_a$. $P \mid P$ is a symmetric network of degree 2 with the symmetry relation id . The minimal degree of id is 1. Let $P \mid P \Longrightarrow (\nu \tilde{x})(P' \mid \sigma(P'))$ be a symmetric execution of $P \mid P$ for some $P' \in \mathcal{P}_a$, a sequence of names \tilde{x} , and some symmetry relation σ with $\sigma^2 = \text{id}$. By Lemma 5.4 in [13], this symmetric execution can be subdivided such that there is an execution $P \Longrightarrow (\nu \tilde{x}')P'$ for some sequence of names \tilde{x}' .

Lemma 2.4. *Let $P, P' \in \mathcal{P}_a$, \tilde{x} be a sequence of names, and σ be a symmetry relation with $\sigma^2 = \text{id}$. Every symmetric execution $P \mid P \Longrightarrow (\nu \tilde{x})(P' \mid \sigma(P'))$ can be subdivided such that $P \Longrightarrow (\nu \tilde{x}')P'$ for some sequence of names \tilde{x}' .*

2.3 Quality Criteria for Encodings

Gorla presented in [6] a small framework of five criteria well suited for language comparison. We use this five criteria to measure the quality of an encoding $\llbracket \cdot \rrbracket$ from π_{mix} into π_a , i.e. an encoding $\llbracket \cdot \rrbracket$ is “good” if it fulfils the five criteria proposed by Gorla. Note that for the definition of these criteria a behavioural equivalence \simeq on the target language is assumed. Its purpose is to describe the abstract behaviour of a target process, where abstract basically means with respect to the behaviour of the source term.

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection* and (5) *success sensitiveness*. In the following we use S, S', S_1, \dots to range over terms of the source language and T, T', T_1, \dots to range over terms of the target language.

Intuitively, an encoding is compositional if the translation of an operator depends only on the translation of its 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 parametrised on the free names of the corresponding source term. Note that our result is independent of this parametrisation.

Definition 2.5 (Criterion 1: Compositionality). The encoding $\llbracket \cdot \rrbracket$ is *compositional* if, for every k-ary operator \mathbf{op} of π_{mix} and for every subset of names N , there exists a k-ary context $\mathcal{C}_{\mathbf{op}}^N([\cdot]_1, \dots, [\cdot]_k)$ such that, for all S_1, \dots, S_k with $\text{fn}(S_1) \cup \dots \cup \text{fn}(S_k) = N$, it holds that

$$\llbracket \mathbf{op}(S_1, \dots, S_k) \rrbracket = \mathcal{C}_{\mathbf{op}}^N(\llbracket S_1 \rrbracket, \dots, \llbracket S_k \rrbracket).$$

The second structural criterion states that the encoding should not depend on specific names used in the 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 the names used by the encoding function for special purposes and the source term names, the encoding is enriched with a renaming policy $\varphi_{\llbracket \cdot \rrbracket}$, i.e., a substitution from names into sequences of names². 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 2.6 (Criterion 2: Name Invariance). The encoding $\llbracket \cdot \rrbracket$ is *name invariant* if, for every S and σ , it holds that

$$\llbracket \sigma(S) \rrbracket \begin{cases} \equiv_{\alpha} \sigma'(\llbracket S \rrbracket) & \text{if } \sigma \text{ is injective} \\ \asymp \sigma'(\llbracket S \rrbracket) & \text{otherwise} \end{cases}$$

where σ' is such that $\varphi_{\llbracket \cdot \rrbracket}(\sigma(a)) = \sigma'(\varphi_{\llbracket \cdot \rrbracket}(a))$ for every $a \in \mathcal{N}$.

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 simulated 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 corresponding source term, i.e., the translation does not introduce new computations.

Definition 2.7 (Criterion 3: Operational Correspondence). The encoding $\llbracket \cdot \rrbracket$ is *operationally corresponding* if it is

$$\begin{aligned} \text{Complete:} & \quad \text{for all } S \Longrightarrow S', \text{ it holds that } \llbracket S \rrbracket \Longrightarrow \asymp \llbracket S' \rrbracket; \\ \text{Sound:} & \quad \text{for all } \llbracket S \rrbracket \Longrightarrow T, \text{ there exists an } S' \text{ such that } S \Longrightarrow S' \text{ and } T \Longrightarrow \asymp \llbracket S' \rrbracket. \end{aligned}$$

Note that the Definition of operational correspondence relies on the equivalence \asymp to get rid of junk possibly left over within computations of target terms. Sometimes, we refer to the completeness criterion of operational correspondence as operational completeness and, accordingly, for the soundness criterion as operational soundness.

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

Definition 2.8 (Criterion 4: Divergence Reflection). The encoding $\llbracket \cdot \rrbracket$ reflects divergence if, for every S , $\llbracket S \rrbracket \mapsto^{\omega}$ implies $S \mapsto^{\omega}$.

²To keep distinct names distinct Gorla assumes that $\forall n, m \in \mathcal{N} . n \neq m$ implies $\varphi_{\llbracket \cdot \rrbracket}(n) \cap \varphi_{\llbracket \cdot \rrbracket}(m) = \emptyset$, where $\varphi_{\llbracket \cdot \rrbracket}(x)$ is simply considered as set here.

The last criterion links the behaviour of source terms to the behaviour of target terms. With Gorla [6], we assume a *success* operator \surd to be part of the syntax of both the source and the target language. Likewise, we add \surd to the syntax of π_{mix} in Definition 2.1 and of π_a in Definition 2.2. Since \surd can not be further reduced, the operational semantics is left unchanged in both cases. Moreover, note that $n(\surd) = \text{fn}(\surd) = \text{bn}(\surd) = \emptyset$, so also interplay of \surd with the \equiv -rules is smooth and does not require explicit treatment. The test for reachability of success is standard.

Definition 2.9 (Success). A process $P \in \mathcal{P}$ 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 \surd , i.e. $\exists P', P'' \in \mathcal{P} . P \Longrightarrow P' \wedge P' \equiv P'' \mid \surd$.

Note that we choose may-testing here. However, as we claim, our main result in Theorem 3.8 holds for must-testing, as well.

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

Definition 2.10 (Criterion 5: Success Sensitiveness). The encoding $\llbracket \cdot \rrbracket$ is *success sensitive* if, for every $S, S \Downarrow$ if and only if $\llbracket S \rrbracket \Downarrow$.

Note that this criterion only links the behaviours of source terms and their literal translations but not of their continuations. To do so, Gorla relates 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 2.11 (Success Respecting). $\simeq \subseteq \mathcal{P}_a \times \mathcal{P}_a$ is *success respecting* if, for every P and Q with $P \Downarrow$ and $Q \not\Downarrow$, it holds that $P \not\approx Q$.

2.4 Causality

Analysing the five criteria of the last section, we observe that there are two structural criteria to ensure that a good encoding is implementable, i.e., is of practical interest, and there are three criteria to ensure that the encoding preserves and reflects the main behaviour of source terms. However, there is no criterion requiring the preservation or reflection of causal dependencies.

For the π -calculus usually two kinds of causal dependencies are distinguished (see [15, 1]). The first one, called structural or subject dependencies, originates from the nesting of prefixes, i.e., from the structure of processes. A typical example of such a dependency is given by $(\nu b)(\bar{a}.b \mid b.\bar{c}) \mid a \mid c \mapsto (\nu b)(\bar{b} \mid b.\bar{c}) \mid c \mapsto \bar{c} \mid c \mapsto \mathbf{0}$. The second step on channel b is causally dependent on the first step, because it unguards \bar{b} . So b is causally dependent on a . Similarly, c is causally dependent on b , and by transitivity c is causally dependent on a . The other kind of dependencies are called link or object dependencies and originate from the binding mechanisms on names. Here a typical example is $(\nu x)(\bar{y}(x) \mid \bar{x})$. In a labelled semantics the output on x is causally dependent on the extrusion of x by an output on y , i.e. x is causally dependent on y .

We observe that causal dependencies are defined as a condition between actions or names of actions. In the context of encodings this view is problematic, because steps are often translated into sequences of steps and names may be translated into sequences of names. Moreover a sequence of steps simulating a single source term step may be interleaved with another such sequence or some target term steps used to prepare the simulation of another source term step, whose simulation may never be completed. So, what precisely does it mean for an encoding to preserve or respect causal dependencies? If source term names are translated into sequences of names should one consider the causal dependencies between all

such translated names or only between some of them? Moreover how should an encoding handle names reserved for some special purposes of the encoding function, i.e., target term names that do not result from the translation of a source term name?

We have no final answer to these questions yet. However, in the next section we prove a separation result, which does not require a thorough answer to the questions above. Instead, we use a definition of causal dependencies that is based only on direct subject dependencies. So within this paper, a step B is considered causally dependent on a previous step A , if B depends on the availability of a capability produced by A . More precisely, step B is causally dependent on step A , if A unguards some capability, i.e., some input or output prefix, which is consumed by step B . An encoding preserves causal dependencies, if for any causal dependency between two steps of the source term there is a causal dependency between some steps of their simulations, and an encoding reflects causal dependencies, if for any causal dependency between two steps of different, completed simulations there is a causal dependency of the corresponding source term steps.

3 Synchrony vs Causality

In this section, we show that any good encoding from π_{mix} into π_a introduces causal dependencies, i.e., is not causality respecting.

3.1 Encoding Synchrony introduces Causal Dependencies

To prove our main result we analyse the context introduced to encode the parallel operator and examine how this context has to interact with the encodings of its parameters to allow for a simulation of a source term step. We start with some observations concerning the three process terms

$$P \triangleq \bar{a} + a \mid \bar{b} + b.\checkmark, \quad Q \triangleq P \mid P, \quad \text{and} \quad R \triangleq \bar{a} + b + b.\checkmark \mid \bar{b} + a + a.\checkmark,$$

which are used in the following lemmata as counterexamples. Note that we choose Q , and R such that each of them is a symmetric network of degree 2 with either $\sigma = \{a/b, b/a\}$ or id as symmetry relation. Moreover to fix the context used to encode the parallel operator we choose P , Q , and R such that $\text{fn}(P) = \text{fn}(Q) = \text{fn}(R) = \{a, b\}$. Hence by compositionality for each of these three terms the outermost parallel operator is translated by exactly the same context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$.

Observation 3.1. There exists a context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ such that $\llbracket P \rrbracket = \mathcal{C}_1^{\{a,b\}}(\llbracket \bar{a} + a \rrbracket, \llbracket \bar{b} + b.\checkmark \rrbracket)$, $\llbracket Q \rrbracket = \mathcal{C}_1^{\{a,b\}}(\llbracket P \rrbracket, \llbracket P \rrbracket)$, and $\llbracket R \rrbracket = \mathcal{C}_1^{\{a,b\}}(\llbracket \bar{a} + b + b.\checkmark \rrbracket, \llbracket \bar{b} + a + a.\checkmark \rrbracket)$.

We choose P such that none of its executions lead to success, i.e., $P \not\rightarrow$ and $P \not\Downarrow$. $P \not\rightarrow$ implies, by operational soundness, that $\llbracket P \rrbracket$ can not perform a step that changes its state modulo \simeq , i.e. $\llbracket P \rrbracket \Longrightarrow T_P$ implies $T_P \Longrightarrow \simeq \llbracket P \rrbracket$ for all $T_P \in \mathcal{P}_a$. By success sensitiveness, $P \not\Downarrow$ implies $\llbracket P \rrbracket \not\Downarrow$. Because of that and since \simeq is success respecting we have $T_P \not\Downarrow$ for all $T_P \in \mathcal{P}_a$ such that $\llbracket P \rrbracket \Longrightarrow T_P$.

Observation 3.2. $\forall T_P \in \mathcal{P}_a. \llbracket P \rrbracket \Longrightarrow T_P$ implies $T_P \not\Downarrow$

Hence, any occurrence of \checkmark —if there is any—in the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ is input guarded (since π_a forbids output guards) and the context can not remove such a guard on its own. In opposite to P we choose Q such that Q reaches an unguarded occurrence of success in any of its executions, i.e. $Q \Longrightarrow Q'$ implies $Q' \Downarrow$ for all $Q' \in \mathcal{P}_{\text{mix}}$. By operational completeness, any execution $Q \rightarrow Q_1 \rightarrow Q_2 \not\rightarrow$ of Q can be simulated by its encoding, i.e. $\llbracket Q \rrbracket \Longrightarrow Q'_1$, $\llbracket Q \rrbracket \Longrightarrow Q'_2$, and $\llbracket Q_1 \rrbracket \Longrightarrow Q''_2$, where $Q'_i \simeq \llbracket Q_i \rrbracket$ for

$i \in \{1, 2\}$ and $Q_2'' \asymp \llbracket Q_2 \rrbracket$. Note that any (maximal) execution of Q is such that $Q \mapsto Q_1 \mapsto Q_2 \not\mapsto$ for some $Q_1, Q_2 \in \mathcal{P}_{\text{mix}}$. By operational soundness for each $T_Q \in \mathcal{P}_a$ such that $\llbracket Q \rrbracket \Longrightarrow T_Q$, there is some $Q' \in \mathcal{P}_{\text{mix}}$ such that $Q \Longrightarrow Q'$ and $T_Q \Longrightarrow \asymp \llbracket Q' \rrbracket$, i.e. there is some $T'_Q \in \mathcal{P}_a$ such that $T_Q \Longrightarrow T'_Q$ and $T'_Q \asymp \llbracket Q' \rrbracket$. By success sensitiveness, $\llbracket Q' \rrbracket \Downarrow$ and since \asymp is success respecting, we have $T'_Q \Downarrow$. Thus, by Definition 2.9, we have $T_Q \Downarrow$ for all $T_Q \in \mathcal{P}_a$ with $\llbracket Q \rrbracket \Longrightarrow T_Q$.

Observation 3.3. $\forall T_Q \in \mathcal{P}_a . \llbracket Q \rrbracket \Longrightarrow T_Q$ implies $T_Q \Downarrow$

At last we choose R such that some of its executions lead to success while some do not. R can reduce either to \checkmark or to $\mathbf{0}$. By operational completeness, $\llbracket R \rrbracket$ can simulate both steps, i.e. $\llbracket R \rrbracket \Longrightarrow \asymp \llbracket \checkmark \rrbracket$ and $\llbracket R \rrbracket \Longrightarrow \asymp \llbracket \mathbf{0} \rrbracket$. Since $\llbracket \checkmark \rrbracket \Downarrow$ and $\llbracket \mathbf{0} \rrbracket \not\Downarrow$, and since \asymp is success respecting, we have $\llbracket \checkmark \rrbracket \not\asymp \llbracket \mathbf{0} \rrbracket$. By operational soundness, for all $T_R \in \mathcal{P}_a$ such that $\llbracket R \rrbracket \Longrightarrow T_R$ there is some $R' \in \mathcal{P}_{\text{mix}}$ such that $R \Longrightarrow R'$ and $T_R \Longrightarrow \asymp \llbracket R' \rrbracket$.

Observation 3.4. $\exists T_{R,1}, T_{R,2} \in \mathcal{P}_a . \llbracket R \rrbracket \Longrightarrow T_{R,1} \wedge \llbracket R \rrbracket \Longrightarrow T_{R,2} \wedge T_{R,1} \Downarrow \wedge T_{R,2} \not\Downarrow$

Our last observation concerns the structure of the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$. Because there is no choice operator in \mathcal{P}_a , the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ has to place its parameters in parallel, as this is the only binary operator for processes. However, even if we allow separate choice in the target language the encodings of the parameters have to be placed in parallel, because placing them within a choice would not allow to use the encodings of both parameters to simulate target term steps. Consequently, there must be some \mathcal{P}_a -contexts $\mathcal{C}_1([\cdot]), \mathcal{C}_2([\cdot]), \mathcal{C}_3([\cdot])$ with $\llbracket S_1 \mid S_2 \rrbracket \equiv \mathcal{C}_1^{\{a,b\}}(\llbracket S_1 \rrbracket, \llbracket S_2 \rrbracket) \equiv \mathcal{C}_1(\mathcal{C}_2(\llbracket S_1 \rrbracket) \mid \mathcal{C}_3(\llbracket S_2 \rrbracket))$, for all source terms $S_1, S_2 \in \mathcal{P}_{\text{mix}}$ with $\text{fn}(S_1 \mid S_2) = \{a, b\}$.

Observation 3.5. $\exists \mathcal{C}_1([\cdot]), \mathcal{C}_2([\cdot]), \mathcal{C}_3([\cdot]) . \mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2) \equiv \mathcal{C}_1(\mathcal{C}_2([\cdot]_1) \mid \mathcal{C}_3([\cdot]_2))$

Learning from the separation result in [13], we know that any good encoding from π_{mix} into π_a must break source term symmetries. To do so, we show that the context introduced by the encoding of the parallel operator (which is allowed in weakly compositional as opposed to homomorphic translations) must interact with the encodings of its parameters.

Lemma 3.6. *To simulate a source term step, $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ and the encodings of its parameters have to interact.*

Intuitively we show, that if there is no such interaction, then since Q is a symmetric network its encoding behaves as a symmetric network again. Since any execution of $\llbracket Q \rrbracket$ leads to an unguarded occurrence of success, by symmetry and by Lemma 2.4 there is an execution of $\llbracket P \rrbracket$ leading to an unguarded occurrence of success, which contradicts Observation 3.2.

Proof. Assume the opposite, i.e. assume the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ is such that possibly after some preprocessing steps of the context on its own, e.g. to unguard the parameters, the source term steps can be simulated without any interaction with the context. In this case, we have

$$\llbracket Q \rrbracket \stackrel{3.1}{\equiv} \mathcal{C}_1^{\{a,b\}}(\llbracket P \rrbracket \mid \llbracket P \rrbracket) \stackrel{3.5}{\equiv} \mathcal{C}_1(\mathcal{C}_2(\llbracket P \rrbracket) \mid \mathcal{C}_3(\llbracket P \rrbracket)) \Longrightarrow (v\tilde{y})(\sigma_1(\llbracket P \rrbracket) \mid \sigma_2(\llbracket P \rrbracket) \mid T_C)$$

for some constant term T_C , a sequence of names \tilde{y} , and two substitutions σ_1 and σ_2 . Note that σ_1 and σ_2 capture renaming done by alpha conversion possibly necessary to pull restriction outwards. Since there is no need for an interaction, i.e. for a communication, with T_C to simulate source term steps, we can ignore it.

If $\sigma_1 = \sigma_2$, then since these substitutions result from alpha conversion $\sigma_1 = \sigma_2 = \text{id}$. Then $\llbracket P \rrbracket \mid \llbracket P \rrbracket$ is a symmetric network of degree 2 with id as symmetry relation. By Lemma 2.3, $\llbracket P \rrbracket \mid \llbracket P \rrbracket$ has a

symmetric execution. By Observation 3.3, $\llbracket Q \rrbracket$ reaches an unguarded occurrence of success in any of its executions. Since the context and with it T_C can not reach success on its own and there is no interaction, $\llbracket P \rrbracket \mid \llbracket P \rrbracket$ reaches success in its symmetric execution. Then there is some $T_Q'' \in \mathcal{P}_a$ such that $\llbracket P \rrbracket \mid \llbracket P \rrbracket \Longrightarrow (v\tilde{x}) \left(T_Q'' \mid \sigma_3 \left(T_Q'' \right) \right)$ is a symmetric execution for some sequence of names \tilde{x} and some symmetry relation σ_3 of degree 2 and $(v\tilde{x}) \left(T_Q'' \mid \sigma_3 \left(T_Q'' \right) \right)$ has an unguarded occurrence of success. By symmetry and since $n(\checkmark) = \emptyset$, this implies that T_Q'' as well as $\sigma \left(T_Q'' \right)$ has an unguarded occurrence of success. Since 2 is not the minimal degree of identity, by Lemma 2.4, this symmetric execution can be subdivided such that $\llbracket P \rrbracket \Longrightarrow (v\tilde{x}') T_Q''$ for some sequence of names \tilde{x}' . Then $\llbracket P \rrbracket \Downarrow$, because of the unguarded occurrence of \checkmark in T_Q'' . That contradicts Observation 3.2.

The argumentation for $\sigma_1 \neq \sigma_2$ is similar but more difficult. In this case $\sigma_1(\llbracket P \rrbracket) \mid \sigma_2(\llbracket P \rrbracket)$ is still a symmetric network whose symmetric execution leads to an unguarded occurrence of success. But since its symmetry relation is not id we can not apply Lemma 2.4. However, because σ_1 and σ_2 result from alpha conversion, they rename free names of $\llbracket P \rrbracket$ to fresh names. If $\sigma_1(\llbracket P \rrbracket)$ and $\sigma_2(\llbracket P \rrbracket)$ want to interact on such a fresh name, then they have first to exchange this fresh name over a channel known to both. Let us denote this channel by z . So either $\sigma_1(\llbracket P \rrbracket)$ receives a fresh name from $\sigma_2(\llbracket P \rrbracket)$ over z or vice versa. By symmetry both terms have an unguarded input as well as an unguarded output on z , so—instead of a communication between these two processes— $\sigma_1(\llbracket P \rrbracket)$ can as well reduce on its own. Adding this argumentation to the argumentation in the proof of Lemma 2.4 in [13] we can prove again that the symmetric execution of $\sigma_1(\llbracket P \rrbracket) \mid \sigma_2(\llbracket P \rrbracket)$ can be subdivided such that $\sigma_1(\llbracket P \rrbracket) \Downarrow$. Because $n(\checkmark) = \emptyset$, this implies $\llbracket P \rrbracket \Downarrow$. Hence, to simulate a source term step, the context necessarily has to interact with its parameters. \square

Note that the only possibility for the context to interact with its parameters is by communication. So the context contains at least one capability, i.e., input or output prefix, that needs to be consumed to simulate a source term step. Without loss of generality let us assume that indeed only a single capability needs to be consumed to simulate a step, i.e. a single communication step of the context with (one of) its parameters suffices to enable the simulation of a source term step. The argumentation for a couple of necessary communication steps is similar. Let us denote this capability by μ^3 .

Next we show, that it is not possible to simulate two different source term steps between the parameters of $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ at the same time.

Lemma 3.7. *At most one simulation of a source term step can be enabled concurrently by the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$.*

Here we use R as a counterexample. R can reduce either to $\mathbf{0}$ or \checkmark , so the choice operator introduces mutual exclusion. Without choice mutual exclusion is not that easy to implement, because of its ability to immediately block an alternative reduction. We show that the simulation of these blocking introduces either deadlock or divergence.

Proof. Assume the opposite, i.e. assume that the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ provides several instances of μ , e.g. by replication, and with them it enables the simulation of different alternative source term steps concurrently. Consider the source term R . Since $\{a/b, b/a\}(\bar{a} + b + b.\checkmark) = \bar{b} + a + a.\checkmark$, by name

³In case of a sequence of necessary steps, choose μ such that it denotes the capability consumed at last in this sequence. In case there are different ways to enable the simulation of a step, consider a set of those capabilities with one μ_i for each such way.

invariance, there is some substitution σ' such that $\sigma'(\llbracket \bar{a} + b + b.\checkmark \rrbracket) \equiv_{\alpha} \llbracket \bar{b} + a + a.\checkmark \rrbracket$, i.e. these two terms are equal except to some renamings of free names.

Note that R can perform either a step on channel a or b . Since there is no choice operator in π_a , the encodings of the capabilities of the sums $\bar{a} + b + b.\checkmark$ and $\bar{b} + a + a.\checkmark$ have to be placed somehow in parallel such that the simulation of the source term step on a does not immediately withdraw the encodings of the capabilities on b and vice versa. Thus, since the simulation of both steps of R are enabled concurrently, there is some point in the simulation of one source term step that disables the completion of the simulation of the respective other source term step. Therefore, one simulation has to consume some capability that is necessary to complete the other simulation. Remember, that we assume that the only capability of the context $\mathcal{C}_1^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ necessary to be consumed to simulate a source term step is μ . Hence, to allow the simulation of one step of R , to disable the simulation of the respective other step of R , and since $\sigma'(\llbracket \bar{a} + b + b.\checkmark \rrbracket) \equiv_{\alpha} \llbracket \bar{b} + a + a.\checkmark \rrbracket$, there is some capability in $\llbracket \bar{a} + b + b.\checkmark \rrbracket$ as well as in $\llbracket \bar{b} + a + a.\checkmark \rrbracket$ and, to simulate a source term step, both of these capabilities have to be consumed. Moreover, since $\sigma'(\llbracket \bar{a} + b + b.\checkmark \rrbracket) \equiv_{\alpha} \llbracket \bar{b} + a + a.\checkmark \rrbracket$, both capabilities are of the same kind, i.e. both are either input prefixes or both are output prefixes. Note that there is no possibility in π_a to consume two capabilities of the same kind within the same target term step. Then it can not be avoided that for each of the simulations of the two steps exactly one of these two capabilities is consumed. In this case, none of the simulations can be completed, i.e., there is some local deadlock.

Considering $\llbracket Q \rrbracket$, such a deadlock leads to a term T_Q with $\llbracket Q \rrbracket \Longrightarrow T_Q$ and, since none of the source term steps is simulated, no unguarded occurrence of \checkmark is reached, i.e. $T_Q \not\Downarrow$. With that, such a deadlock leads to a contradiction.

The only way to circumvent a deadlock in this situation, is that one of these capabilities is released by one of these simulations. To complete the simulation of this step later on, it has to be possible that the released capability is consumed again. But then it can not be avoided that this is done before the other simulation is finished, i.e. that leads back to the situation before. Then we introduce divergence, i.e., contradicts divergence reflection. Thus, it is not possible that the simulation of alternative source term steps is enabled concurrently. \square

Note that, even if we allow separate choice within the target language, the simulation of the source term step on a can not immediately withdraw the encodings of the capabilities on b and vice versa. Because either we have to split each of these sums into an input and an output guarded sum or we have to convert each of them into a single sum with only separate choice. In the second case, since by compositionality both parameters have to be encoded in exactly the same way, we either result in two input guarded or two output guarded sums. But then we can not simulate a communication between these to sums within a single step and, moreover, we can not decide within a single step whether a considered capability can be used to successfully simulate a source term step. Unfortunately, the first step used to try whether we can use this capability to simulate a source term step removes all the other encoded capabilities of that sum, which violates operational correspondence. So Lemma 3.7 holds as well for separate choice in the target language.

Now we can prove the main result of this paper.

Theorem 3.8. *Any good encoding from π_{mix} into π_a introduces additional causal dependencies.*

By Lemma 3.6 and Lemma 3.7 the context must contain some capability, i.e. some kind of lock, that must be consumed to enable the simulation of a source term step. Moreover, to enable the simulation of subsequent source term steps, the capability that was consumed from the context must be restored. Then, a subsequent simulating sequence is enabled by the consumption of a capability that was produced by a

former simulating sequence; this interplay imposes a causal dependence between subsequent simulations of source term steps.

Proof. By Lemma 3.6 and Lemma 3.7 the context $\mathcal{C}_{\perp}^{\{a,b\}}([\cdot]_1, [\cdot]_2)$ provides exactly one instance of μ , i.e., one capability, that needs to be consumed to simulate a source term step. Since $\llbracket Q \rrbracket \Longrightarrow Q'_1$, $\llbracket Q \rrbracket \Longrightarrow Q'_2$, and $\llbracket Q_1 \rrbracket \Longrightarrow Q''_2$, where $Q'_i \asymp \llbracket Q_i \rrbracket$ for $i \in \{1, 2\}$ and $Q''_2 \asymp \llbracket Q_2 \rrbracket$, i.e. $\llbracket Q \rrbracket$ simulates two subsequently source term steps, the instance of μ consumed by the simulation of the first source term step has to be restored during this first simulation such that the second step can be simulated. Thus, the simulation of the second step has to consume some capability μ produced by the simulation of the first step. Then the simulation of the second step causally depends on the simulation of the first step, although any pair of subsequent steps of Q are causal independent. We conclude that the encoding function adds additional causal dependencies. \square

3.2 Discussion

It is no fortuity that the counterexamples in the proof of Theorem 3.8 rely on mixed choices. It is the power of mixed choice that allows π_{mix} to break initial symmetries. Note that the separation results of [11] and [13] are based on that absolute difference in the expressive power of π_{mix} compared to π -calculus variants without mixed choice. In [6], the role of breaking symmetries for the separation result is not equally obvious. Nevertheless, the counterexample used there also relies on mixed choices and their ability to break symmetries. In summary, the difference in the expressive power of π_{mix} compared to π_a essentially—if not exclusively—relies on the expressive power of mixed choice.

Synchrony versus Guarded Choice. It is debatable in how far a discussion on synchrony versus asynchrony can be separated from a discussion of choice. In fact, even from a pragmatic point of view within our model of distributed reactive systems, it cannot. It is part of the nature of reactive systems—in our case: systems communicating via message-passing along channels—that agents do not only listen to one channel at a time; they concurrently listen to a whole selection of channels. In this respect, as soon as a calculus offers a synchronous (blocking) input primitive, it is natural to extend this primitive to an input-guarded choice. Having mutual exclusion on concurrently enabled inputs is useful when thinking of a process's local state that may be influenced differently by any received information along the competing input channels. (*Joint input* [9], as motivated in the join calculus [4], represents another natural and interesting generalisation.) Likewise, as soon as a calculus offers synchronous output, one may generalise this primitive to output-guarded choice. This generalisation seems less natural, though, as the process's state would hardly be influenced by a continuation of one of the branches after an output. However, having both input- and output-guards in the calculus, mixed choice becomes expressible. Mixed choice is again also natural, as the successful execution of an output may prevent a competing input, including the effect of the latter on the local state. These pragmatic arguments support the point of view that, in a message-passing scenario, any discussion of synchronous versus asynchronous interaction must consider a competitive context, as expressed by means of choice operators.

The Role of Mixed Choice. In the proof of Lemma 3.6 a counterexample based on mixed choice is used not only to rule out that the parallel operator is translated homomorphically (compare to the argumentation of the separation results in [11], [6], and [13]) but also to prove that, in order to break symmetries, the context introduced to encode the parallel operator must interact with the encodings of its parameters. Moreover, only with mixed choice it is possible to give an example of a symmetric network

with two conflicting steps on two different channels. This is necessary in the proof of Lemma 3.7 to show that the context can not enable the simulation of more than one source term step at a time. Since the two source term steps are on different channels, their simulation is on the encoding of different capabilities, i.e. the simulations do not interfere by interaction on the same encoded capabilities. Because the two source term steps are in conflict, operational completeness in combination with success sensitivity forbids that both simulations are completed successfully. We conclude that the simulation of one source term step must inevitably disable the successful completion of the conflicting source term step by the consumption of some necessary capability. Since the source terms are on different channels, this capability is not due to the encoding of a source term capability but added to the encoding of the corresponding process of the source term network. The symmetry of the source term allows to apply the name invariance criterion to conclude that both processes of that symmetric source term net are encoded symmetrically. Hence, there is one instance of the respective capability in each encoded process. Now, the simulation of one of the two conflicting steps has to consume both such capabilities. As explained above two concurrently enabled simulations then compete for these capabilities, which leads to deadlock or divergence. Note that deadlock would violate operational soundness.

Interestingly, the necessity of such a capability to rule out conflicting steps as well as the associated danger to introduce deadlock or divergence was already pointed out by Nestmann [10], presenting a good encoding from π_{sep} into π_a . To rule out conflicting steps on the same sum, the encoding of a sum introduces a so-called sum lock: a boolean-valued lock that is initially instantiated with true, but is set to false as soon as some summand of the sum commits to successfully simulate a source term step. Using this sum lock, the encoding function does not only forbid to use that summand twice but also to use any other summand of that sum to simulate subsequently source term steps. Hence, this sum lock of the encoding in [10] does exactly occupy the role of the capability consumed by the simulation of a step to rule out the simulation of a conflicting step as described above. Note that the encoding presented in [10] translates the parallel operator homomorphically and thus is no good encoding from π_{mix} into π_a . [10] explains that the application of that encoding function to terms with mixed choices—or, more precisely, in the presence of mixed choices with cyclic dependencies between the links of matching capabilities as in R or Q —leads to exactly the deadlocked situation described in the proof above.

The proof of Theorem 3.8 reveals the solution to circumvent this problem with the so-called cyclic sums. The encoding of the parallel operator has to ensure that there is at most one simulation of a source term step between the two parameters of that parallel operator at the same time. In fact, it is exactly this required blocking of alternative steps that leads to additional causal dependencies. As claimed in [14], the introduction of a lock at the level of the encoding of a parallel operator indeed suffices to circumvent the problem of cyclic sums. Note that, in the appendix of [14], the main line of argumentation of a respective proof is given. Moreover, it is explained in more detail how—in the context of that particular encoding—this lock leads to additional causal dependencies. It is the temporal blocking of the simulation of source term steps, necessary to avoid deadlock or divergence in case of conflicting source term steps, that leads to additional causal dependencies in case of concurrent source term steps.

True Concurrency. Let us take a closer look at the kind of additional causal dependencies an encoding function must, according to the above proof, introduce to encode synchronous interactions. We observe that the proof induces a causal dependency between the simulation of all subsequent source term steps between the two sides of the same parallel operator. Moreover, we observe that a later such step is causally dependent of a former one, but that no fixed order on the simulations of steps is induced. The same observation holds for the encoding presented in [14]: it is forced to block some simulations of

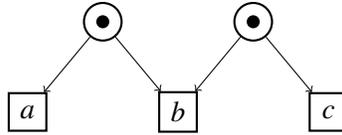


Figure 3: A fully reached, pure \mathbf{M} [17].

source term steps until another simulation is finished, but there is no directive on which kind of steps have to be simulated first. Hence, the simulation of two concurrent source term steps can still appear in either order but, if both steps are due to a communication over the same parallel operator, i.e. between the same processes of a network, the corresponding simulations can not be performed truly concurrently. Note that the proof above does not state that the described additional causal dependency is the only kind of causal dependency any good encoding of synchrony has to introduce. This might be an interesting question for further research.

4 Conclusion

We show that, in the context of the π -calculus, any good encoding of synchronous interaction within a purely asynchronous setting introduces additional causal dependencies and, thus, reduces the number of truly parallel steps of the original term. Moreover, the proof of this result further illustrates the importance of the role of mixed choice to distinguish the expressive power of π_{mix} and π_a . To tighten that view on the importance of mixed choice for the introduction of causal dependencies in encoding synchrony, we have to show that the encoding from π_{sep} into π_a presented in [10] does not introduce additional causal dependencies. This statement seems intuitively correct; we leave its formal proof for further research.

As already mentioned, there is a companion paper [17] that proves a result similar to Theorem 3.8 in the context of Petri nets. More precisely, they show that it is not always possible to find a finite, 1-safe, distributed net which is completed pomset trace equivalent to a given net. Note that completed pomset trace equivalence is sensitive to (local) deadlocks and causal dependencies. Hence, the connection between synchronous interactions and causalities, i.e., that any good encoding of synchrony changes the causal semantics of the source, is no effect of the representation of concurrent systems in either the π -calculus or Petri nets, but seems to be a phenomenon of synchronous and asynchronous interactions in general.

A closer look at the proof in [17] reveals that this proof depends on a counterexample including a so-called *fully reached pure M* (see Figure 3). Similarly, our result depends on counterexamples that are symmetric networks including mixed choices. In both cases, the counterexample refers to a situation in the synchronous setting in which there are two distinct but conflicting steps. To solve this conflict, two simultaneous activities are necessary—in case of the π -calculus the reduction of two sums, in case of Petri nets the removal of two tokens. In the asynchronous setting, this simultaneous solution must be serialised, e.g., by means of some kind of lock. It blocks the enabling of the asynchronous simulations of source term steps, such that no two simulations of conflicting source steps are enabled concurrently. In both formalisms, Petri nets and the π -calculus, it is this temporal blocking of the simulation of source term steps—necessary to avoid deadlock or divergence in case of conflicting source term steps—that leads to additional causal dependencies.

However, apart from this apparent similarity, the relation between the two results leaves us with a

number of open problems and incites further directions of research. To begin with, the requirements imposed on π -calculus implementations and Petri net implementations take rather different forms. Additionally, in contrast to the Petri net result, the present paper has no need to employ a (pomset based) equivalence to compare source and target terms and also does not need to deal with infinite implementations specifically. On the other hand, the Petri net result does not impose any restrictions on the encoding itself, but it connected source and target nets by means of behaviour only without any reference to the net structure. Finally, the Petri nets considered in [17] are not Turing-complete. So is it possible to derive the same result considering a Turing-complete formalism as for instance Petri nets with inhibitor arcs? We hope to answer some of these questions in future work.

References

- [1] Michele Boreale & Davide Sangiorgi (1998): *A fully abstract semantics for causality in the π -calculus*. *Acta Inf.* 35(5), pp. 353–400, doi:10.1007/s002360050124.
- [2] Gérard Boudol (1992): *Asynchrony and the π -calculus (note)*. Note, INRIA.
- [3] Marco Carbone & Sergio Maffei (2003): *On the Expressive Power of Polyadic Synchronisation in π -Calculus*. *Nordic Journal of Computing* 10(2), pp. 70–98.
- [4] Cédric Fournet & Georges Gonthier (1996): *The Reflexive Chemical Abstract Machine and the Join-Calculus*. In: *Proceedings of POPL '96*, ACM, pp. 372–385, doi:10.1145/237721.237805.
- [5] Daniele Gorla (2008): *Comparing Communication Primitives via their Relative Expressive Power*. *Inf. & Comp.* 206(8), pp. 931–952, doi:10.1016/j.ic.2008.05.001.
- [6] Daniele Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Inf. & Comp.* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.
- [7] Kohei Honda (1992): *Notes on Soundness of a Mapping from π -calculus to ν -calculus*. With comments added in October 1993.
- [8] Kohei Honda & Mario Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In: *ECOOP'91, LNCS 512*, Springer Berlin / Heidelberg, pp. 133–147, doi:10.1007/BFb0057019.
- [9] Uwe Nestmann (1998): *On the Expressive Power of Joint Input*. In Catuscia Palamidessi & Iliaria Castellani, editors: *Proceedings of EXPRESS '98, ENTCS 16.2*, Elsevier Science Publishers, doi:10.1016/S1571-0661(04)00123-9.
- [10] Uwe Nestmann (2000): *What is a "Good" Encoding of Guarded Choice?* *Inf. & Comp.* 156(1-2), pp. 287–319, doi:10.1006/inco.1999.2822.
- [11] Catuscia Palamidessi (2003): *Comparing the Expressive Power of the Synchronous and the Asynchronous π -calculi*. *MSCS* 13(5), pp. 685–719, doi:10.1017/S0960129503004043.
- [12] Joachim Parrow (2008): *Expressiveness of Process Algebras*. *ENTCS* 209, pp. 173–186, doi:10.1016/j.entcs.2008.04.011.
- [13] Kirstin Peters & Uwe Nestmann (2010): *Breaking Symmetries*. In: *EXPRESS'10, EPTCS 41*, pp. 136–150, doi:10.4204/EPTCS.41.10.
- [14] Kirstin Peters & Uwe Nestmann (2011): *Breaking Symmetries*. Submitted to MSCS.
- [15] Corrado Priami (1996): *Enhanced Operational Semantics for Concurrency*. Ph.D. thesis, Università di Pisa-Genova-Udine.
- [16] Davide Sangiorgi & David Walker (2001): *The π -calculus: A Theory of Mobile Processes*. Cambridge University Press, NY, USA.
- [17] Jens-Wolfhard Schicke, Kirstin Peters & Ursula Goltz (2011): *Synchrony vs. Causality in Asynchronous Petri Nets*. To appear in EXPRESS'11.