*Math. Struct. in Comp. Science* (2016), *vol.* 26, *pp.* 1459–1498. © Cambridge University Press 2014
doi:10.1017/S0960129514000644 First published online 23 December 2014

# Synchrony versus causality in distributed systems†

KIRSTIN PETERS‡, JENS-WOLFHARD SCHICKE-UFFMANN§, URSULA GOLTZ§ and UWE NESTMANN‡

‡*School of Electrical Engineering and Computer Science, TU Berlin, Germany*
*Emails:* kirstin.peters@tu-berlin.de, uwe.nestmann@tu-berlin.de
§*Institute for Programming and Reactive Systems, TU Braunschweig, Germany*

*Received 1 January 2012; revised 1 April 2014*

Given a synchronous system, we study the question whether – or, under which conditions – the behaviour of that system can be realized by a (non-trivially) distributed and hence asynchronous implementation. In this paper, we partially answer this question by examining the role of causality for the implementation of synchrony in two fundamental different formalisms of concurrency, Petri nets and the $\pi$-calculus. For both formalisms it turns out that each 'good' encoding of synchronous interactions using just asynchronous interactions introduces causal dependencies in the translation.

## 1. Introduction

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. It would be desirable – from a programming standpoint – to design systems in a synchronous fashion, yet reap the benefits of parallelism by means of an (ideally automatically generated) implementation executed on multiple processing units in parallel, between which only asynchronous communication is possible. Thus, we are interested in the conditions under which synchronous interactions can be implemented using just asynchronous interactions, while maximizing the degree of distribution. To partially answer this question, we examine the role of causality for encoding synchrony. We formalize and study this problem by means of Petri nets (see Section 2) as well as the $\pi$-calculus (see Section 3), as synchronous and asynchronous interactions have already been studied in both models to quite some extent.

Petri nets and the $\pi$-calculus are two fundamentally different models for reactive systems. However, certain basic phenomena show up in both in at least similar ways. Three of them play a crucial role for our investigation. A reactive system exhibits its behaviour by executing various actions. Studying the possible behaviour of a system, two action occurrences must be related in one of three ways: there can either be a *choice* between the two, one of the two can be *causally dependent* on the other, or the two can be entirely *independent*.

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

### 1.1. *Choice*

If there is a choice between two behaviours, the system can only exhibit one. If multiple copies of the same system would be started though, they could each choose a different alternative.

In Petri nets, choice is represented by conflict, i.e. by two transitions sharing a common preplace and competing for tokens such that each transition can fire separately but not both at the same time. The processing of conflicts in Petri nets is inherently synchronous, in particular multiple conflicts transitively connecting an arbitrary number of transitions will always be decided consistently.

In the $\pi$-calculus, choice can be implemented directly via the choice operator or indirectly by offering multiple matching outputs for a single input or vice versa. The choice operator also allows for choices between actions on different channel names. In the literature, different kinds of the choice operator are distinguished depending on what kind of processes are allowed to be combined within a choice. We restrict our attention to guarded choice, i.e. to choice constructs where each summand is guarded by an in- or output prefix.

When processes communicate via message-passing along channels, they do not only listen to one channel at a time – usually, they concurrently listen to a whole selection of channels. Choice operators 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. Because of that, it makes sense to assume a synchronous calculus with the ability of mixed choice, while in its asynchronous variant at most input guarded choice is naturally given.

### 1.2. *Causality*

The second basic concept, which is particularly considered in Petri nets, is the notion of *causal dependence*. A transition occurrence in a Petri net or a step[†] of a process in a process calculus is causally dependent on another one when the first one must necessarily have happened in order to enable the second one. In Petri nets, this notion may be clearly defined using occurrence nets (acyclic nets with only forward branched places) as behavioural representations of the system. Unfortunately, the $\pi$-calculus does not provide such a natural causal semantics for its terms. There are, however, a number of approaches that define causal dependencies, mostly between the actions of $\pi$-terms.

[†] Note that, in Petri nets and process calculi, the notion of steps denotes different concepts.

### 1.3. *Independence*

The third basic notion may then be derived for Petri nets: If two transition occurrences are neither in conflict nor causally dependent then they are *independent*, or *concurrent*. On the system level, we may express independence of transition occurrences by defining a notion of *step*, describing that several transitions may happen together in parallel.

We will avoid the term 'parallel' in this paper, as the $\pi$-calculus features a parallel composition operator, whose standard intuition negatively interferes with the concept of independence: parallel terms need not be independent.

**Example 1.1.** Consider $P = \overline{a} \mid a.\overline{0} + b.\overline{1}$ and $Q = \overline{b} \mid a.\overline{1} + b.\overline{0}$. Then the processes $P$ and $Q$ in the network $(\nu\, a, b)(P \mid Q)$ do not decide independently to emit 0 or 1, but synchronize in order to take that decision.

### 1.4. *Overview of the Paper*

In the following, we consider the two formalisms separately. First, we derive a separation result for Petri nets in Section 2. Then, we derive a similar separation result for the $\pi$-calculus in Section 3. In Section 4, we discuss how these two separation results are related.

## 2. Synchrony versus causality in Petri nets

Petri nets have been introduced as a graphical notation to describe and analyse distributed systems and their behaviour. They are particularly apt at representing distributed state in an intuitive way. When studying distributed implementability in terms of Petri nets we use a semi-structural requirement on Petri nets to represent distribution: As observed in Schicke (2008), consistent outcomes of a choice cannot be assured across different locations, hence each choice must ultimately be decided synchronously and hence on a single location.[†] Hence, if two actions reside on different locations, they must never be in conflict.

However, even where choice exists, a higher degree of distribution can be achieved by introducing some protocol between different locations. In that case, a decision is made before the two conflicting behaviours are executed, thereby separating the synchronous resolution of choice from the distributed performance of the behaviour. To ensure that the protocol used does not introduce incorrect system behaviour, we use equivalence relations on Petri net behaviours to decide whether a candidate implementation is indeed faithful to the synchronous specification.

Many equivalence relations for system behaviour have already been proposed. When comparing the strictness of these equivalences, as done by Glabbeek (1993) or Glabbeek and Goltz (2001), and exploring the resulting lattice, one finds multiple 'dimensions' of features along which such an equivalence may be more or less discriminating. The most

[†] Quantum entanglement would make a distributed yet consistent choice possible though, enabling us to build systems with a higher degree of distribution.

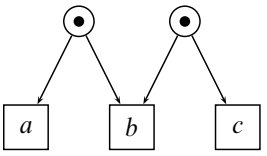



Fig. 1. A fully reached, pure **M**, the core problematic structure.

prominent one is the linear-time branching-time axis, denoting how well the decision structure of a system is captured by the equivalence. Another dimension relevant for this paper is that along which the detail of the causal structure increases. On the first of these two dimensions, we would at the very least like to detect deadlocks introduced by the implementation, on the second one, at least a reduction in concurrency due to the implementation. As every (non-trivial) implementation will introduce internal $\tau$-transitions, a suitable equivalence must abstract from them, as long as they do not allow a divergence.

Glabbeek *et al.* (2008) answers part of the question of distributed implementability for a certain equivalence of this spectrum, namely step readiness equivalence. Step readiness equivalence is one of the weakest equivalences that respects branching time, concurrency and divergence to some degree but abstracts from internal actions. For this equivalence we derived an exact characterization of asynchronously implementable (*distributable*) Petri nets. The main difficulty in implementing arbitrary Petri nets up to step readiness equivalence is a structure called pure **M**, depicted in Figure 1, where two parallel transitions are in pairwise conflict with a common third. By Glabbeek *et al.* (2008) a synchronous net is distributable only if it contains no fully reachable pure **M**, by Glabbeek *et al.* (2012) this characterization is exact, i.e. a net is distributable *iff* it contains no fully reachable pure **M**.

Using the strictly weaker completed step trace equivalence, Schicke (2009) proved any synchronous net to be distributable. Comparing these two results and the given implementation in the latter we made a very interesting observation: we were unable to find an implementation of a synchronous net with a fully reachable pure **M** which did not introduce additional causal dependencies.

A main contribution of the present paper is to show that this drawback holds for any sensible encoding of synchronous interactions, i.e. it is a general phenomenon of encoding synchrony in case of Petri nets. We reach that result by extending the pure **M** of Figure 1 into a repeated pure **M**, depicted in Figure 2. We thereby get a separation result similar to Glabbeek *et al.* (2008) along a different, namely the causal, dimension of the spectrum of behavioural equivalences.

### 2.1. *Overview*

After introducing the standard notions of Petri nets and formally defining distributed nets, we introduce completed pomset trace equivalence to detect newly introduced causalities

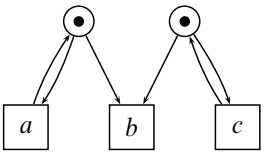



Fig. 2. A repeated pure **M**. A finite, 1-safe, undistributable net used as a running counterexample in case of Petri nets.

in the implementation. Using the concrete example of Figure 2 we then show that for a certain substructure of Petri nets no completed pomset trace equivalent, yet distributed implementation exists.

### 2.2. *Technical preliminaries*

Most material in this section has been taken verbatim or with minimal adaptation from Glabbeek *et al.* (2008) or Schicke (2009).

Where dealing with tuples, we use $\mathrm{pr}_1, \mathrm{pr}_2, \ldots$ as the projection functions returning the first, second, ... element respectively. We extend these functions to sets element-wise.

**Definition 2.1 (labelled net).** Let Act be a set of *visible actions* and $\tau \notin \mathrm{Act}$ be an *invisible action*. A *labelled net* (over Act) is a tuple $N = (S, T, F, M_0, \ell)$ where

— $S$ is a set (of *places*),
— $T$ is a set (of *transitions*),
— $F \subseteq S \times T \cup T \times S$ (the *flow relation*),
— $M_0 \subseteq S$ (the *initial marking*) and
— $\ell : T \to \mathrm{Act} \cup \{\tau\}$ (the *labelling function*).

A net is called *finite* iff $S$ and $T$ are finite.

Petri nets are depicted by drawing the places as circles, the transitions as boxes containing the respective label, and the flow relation as arrows (*arcs*) between them. When a Petri net represents a concurrent system, a global state of such a system is given as a *marking*, a set of places, the initial state being $M_0$. A marking is depicted by placing a dot (*token*) in each of its places. The dynamic behaviour of the represented system is defined by describing the possible moves between markings. A marking $M$ may evolve into a marking $M'$ when a nonempty set of transitions $G$ *fires*. In that case, for each arc $(s,t) \in F$ leading to a transition $t$ in $G$, a token moves along that arc from $s$ to $t$. Naturally, this can happen only if all these tokens are available in $M$ in the first place. These tokens are consumed by the firing, but also new tokens are created, namely one for every outgoing arc of a transition in $G$. These end up in the places at the end of those arcs. A problem occurs when as a result of firing $G$ multiple tokens end up in the same place. In that case $M'$ would not be a marking as defined above. In this paper, we restrict attention to nets in which this never happens. Such nets are called *1-safe*. Unfortunately, in order to formally define this class of nets, we first need to correctly define the firing rule without assuming

1-safety. Below we do this by forbidding the firing of sets of transitions when this might put multiple tokens in the same place. As 1-safety is assumed later, this explicit forbidding will never have any effect.

To help track causality throughout the evolution of a net, we extend the usual notion of marking to *dependency marking*. Within these dependency markings, every token is augmented with the labels of all transitions having causally contributed to its existence. The other basic Petri net notions presented here have been extended in the same manner. While it might seem more natural to annotate the causal history of the tokens by a partial order, we only use a set here in order to keep the number of reachable markings finite for finite nets (a property a later proof will utilize).

We denote the preset and postset of a net element $x \in S \cup T$ by ${}^\bullet x := \{y \mid (y,x) \in F\}$ and $x^\bullet := \{y \mid (x,y) \in F\}$ respectively. These functions are extended to sets in the usual manner, i.e. ${}^\bullet X := \{y \mid y \in {}^\bullet x,\ x \in X\}$.

**Definition 2.2 (steps).** Let $N = (S, T, F, M_0, \ell)$ be a net. Let $M_1, M_2 \subseteq S \times \mathcal{P}(\mathrm{Act})$. $G \subseteq T, G \neq \emptyset$, is called a *dependency step from* $M_1$ *to* $M_2$, $M_1[G\rangle_N M_2$, iff

— all transitions contained in $G$ are enabled, i.e.

$$\forall t \in G. {}^\bullet t \subseteq \mathrm{pr}_1(M_1) \wedge (\mathrm{pr}_1(M_1) \setminus {}^\bullet t) \cap t^\bullet = \emptyset\ ,$$

— all transitions of $G$ are independent, that is not conflicting:

$$\forall t, u \in G, t \neq u. {}^\bullet t \cap {}^\bullet u = \emptyset \wedge t^\bullet \cap u^\bullet = \emptyset\ ,$$

— causalities are extended by the labels of the firing transitions:

$$M_2 = \{p \in M_1 \mid \mathrm{pr}_1(p) \notin {}^\bullet G\} \cup \left\{ \left( s, (\{\ell(t)\} \setminus \{\tau\}) \cup \bigcup_{p \in M_1 \wedge \mathrm{pr}_1(p) \in {}^\bullet t} \mathrm{pr}_2(p) \right) \,\middle|\, t \in G, s \in t^\bullet \right\}\ .$$

Applying $\mathrm{pr}_1$ to a dependency marking results in the classical Petri net notion of marking and similar for the other notions introduced in this section. Note that the enrichment of markings into dependency markings has no impact on the existence of steps, since it neither influences the enabling of transitions nor their independence. We will mainly employ the versions defined here and drop the qualifier 'dependency' most of the time. A token $(s, P) \in M$ is $Q$-dependent iff $Q \subseteq P$ and $Q$-independent iff $P \cap Q = \emptyset$.

To simplify the following argumentation we use some abbreviations.

**Definition 2.3.** Let $N = (S, T, F, M_0, \ell)$ be a labelled net.
We extend the labelling function $\ell$ to (multi)sets element-wise.
$\longrightarrow_N \subseteq \mathcal{P}(S \times \mathcal{P}(\mathrm{Act})) \times \mathbb{N}^{\mathrm{Act}} \times \mathcal{P}(S \times \mathcal{P}(\mathrm{Act}))$ is given by
$M_1 \xrightarrow{A}_N M_2 \Leftrightarrow \exists\, G \subseteq T. M_1\ [G\rangle_N\ M_2 \wedge A = \ell(G)$
$\xrightarrow{\tau}_N \subseteq \mathcal{P}(S \times \mathcal{P}(\mathrm{Act})) \times \mathcal{P}(S \times \mathcal{P}(\mathrm{Act}))$ is defined by
$M_1 \xrightarrow{\tau}_N M_2 \Leftrightarrow \exists t \in T. \ell(t) = \tau \wedge M_1\ [\{t\}\rangle_N\ M_2$
$\Longrightarrow_N \subseteq \mathcal{P}(S \times \mathcal{P}(\mathrm{Act})) \times \mathrm{Act}^* \times \mathcal{P}(S \times \mathcal{P}(\mathrm{Act}))$ is defined by

$$M_1 \xrightarrow{a_1 a_2 \cdots a_n}_N M_2 \Leftrightarrow M_1 \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_1\}}_N \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_2\}}_N \xrightarrow{\tau}{}^*_N \cdots \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_n\}}_N \xrightarrow{\tau}{}^*_N M_2,$$

where $\xrightarrow{\tau}{}^*_N$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}_N$.

We omit the subscript $N$ if clear from context.

We write $M_1 \xrightarrow{A}_N$ for $\exists M_2.M_1 \xrightarrow{A}_N M_2$, $M_1 \not\xrightarrow{A}_N$ for $\nexists M_2.M_1 \xrightarrow{A}_N M_2$, and similar for the other two relations. Likewise the term $M_1[G\rangle_N$ abbreviates $\exists M_2.M_1[G\rangle_N M_2$. A marking $M_1$ is said to be *reachable* iff there is a sequence of labels $\sigma \in \mathrm{Act}^*$ such that $M_0 \times \{\varnothing\} \overset{\sigma}{\Longrightarrow}_N M_1$. The set of all reachable markings is denoted by $[M_0\rangle_N$.

As said before, here we only want to consider 1-safe nets. Formally, we restrict ourselves to *contact-free nets*, where in every reachable marking $M_1 \in [M_0\rangle$ for all $t \in T$ with ${}^\bullet t \subseteq \mathrm{pr}_1(M_1)$

$$(\mathrm{pr}_1(M_1) \setminus {}^\bullet t) \cap t^\bullet = \varnothing \ .$$

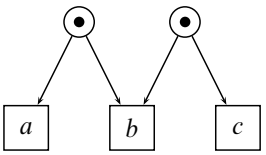
For such nets, in Definition 2.2 we can just as well consider a transition $t$ to be enabled in $M$ iff ${}^\bullet t \subseteq \mathrm{pr}_1(M)$, and two transitions to be independent when ${}^\bullet t \cap {}^\bullet u = \varnothing$.

The later proof that Figure 2 is non-implementable depends crucially on this 1-safety assumption. We conjecture, however, that the result itself will hold, even if non-safe implementations will be allowed.

### 2.3. *Asynchronous Petri nets*

Petri nets are inherently synchronous and we have to use some additional requirements to define asynchronous Petri nets. As already mentioned in the Introduction (Section 1) the synchronous nature of Petri nets mainly manifests in the processing of conflicts.

**Example 2.4.** Consider the fully reached, pure **M** already given in Figure 1:




Here, the transitions $a$ and $c$ can independently be performed in a single step. However, if $b$ fires, then both $a$ and $c$ are disabled. To ensure this behaviour $b$ has to consume both tokens simultaneously, i.e. there is no intermediate state of the system in which only one token is removed.

As explained by Glabbeek *et al.* (2008) we can define asynchronous Petri nets by restricting the existence of such conflicts. Of course, we do not want to forbid all kind of conflicts, but only those that cannot be implemented asynchronously. To do so, we assign to each net element a *location*, place sensible restrictions on arrows crossing location borders, and restrict the sets of net elements being allowed to reside on the same location.

We regard locations as sequential computation units of the underlying system, each one able to execute at most one action during each step. This necessitates that no pair

of transitions firing in the same step can reside on the same location. Additionally, if locations are indeed physically apart as their name suggests, communication between them can only proceed asynchronously.

We discussed a very similar notion of distribution in Glabbeek *et al.* (2008), whence the following description and definition of the present version have been derived. The central insight from that paper is that the synchronous removal of tokens from preplaces of a transition is essential to the conflict resolution taking place between multiple enabled transitions and that hence transitions must reside on the same location as their preplaces.

So, to achieve asynchrony, we basically require that, for each transition $t$, $t$ and all of its preplaces, ${}^\bullet t$, have to placed on the same location. Thus, only outgoing arcs of transitions can cross location borders. That meets our intuition that in an asynchronous setting the consumption of a token takes time, while in the production of tokens a delay cannot have any effect. Since locations are considered as sequential computation units, conflicts within a location are not critical under the assumption of asynchronous interactions between locations. By placing the preplaces of a transition at the same location as the transition itself, we rule out any potential conflict between transitions on different locations.

We model the association of locations to the places and transitions in a net $N = (S, T, F, M_0, \ell)$ as a function $D : S \cup T \to \mathrm{Loc}$, with Loc a set of possible locations. We refer to such a function as a *distribution* of $N$. Since the identity of the locations is irrelevant for our purposes, we can just as well abstract from Loc and represent $D$ by the equivalence relation $\equiv_D$ on $S \cup T$ given by $x \equiv_D y$ iff $D(x) = D(y)$.

**Definition 2.5 (distributed net).** Let $N = (S, T, F, M_0, \ell)$ be a net. The *concurrency relation* $\smile \subseteq T^2$ is given by $t \smile u \Leftrightarrow t \neq u \wedge \exists M \in [M_0\rangle M[\{t,u\}\rangle$. $N$ is *distributed* iff it has a distribution $D$ such that

— $\forall s \in S,\ t \in T . s \in {}^\bullet t \implies t \equiv_D s$,
— $t \smile u \implies t \not\equiv_D u$.

It is easy to see, that the fully reached, pure **M** of Figure 1 is not distributed. It is straightforward to give a semi-structural† characterization of this class of nets.

**Observation 2.6.** A net is distributed iff there is no sequence $t_0, \ldots, t_n$ of transitions with $t_0 \smile t_n$ and ${}^\bullet t_{i-1} \cap {}^\bullet t_i \neq \varnothing$ for $i = 1, \ldots, n$.

### 2.4. *Quality of Petri net implementations*

We consider an implementation of a Petri net $N$ as a variant of $N$ that is achieved by changing the structure of $N$ and introducing invisible transitions, i.e. transitions labelled with the action $\tau$. To rule out trivial or meaningless implementations, i.e. to identify 'good' implementations, we compare $N$ and its implementation by an equivalence relation. In the following, we motivate the properties of this equivalence relation by means of highlighting some possible shortcomings of implementations one would intuitively like to avoid.

† mainly structural, but with a reachability side-condition.

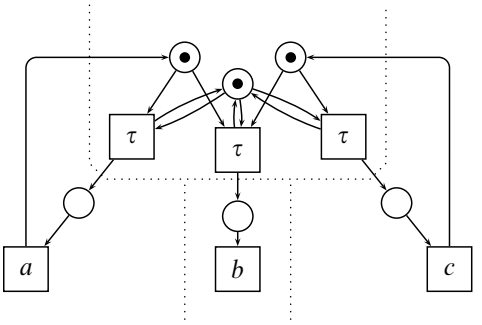



Fig. 3. A centralized implementation of the net in Figure 2, location borders dotted.

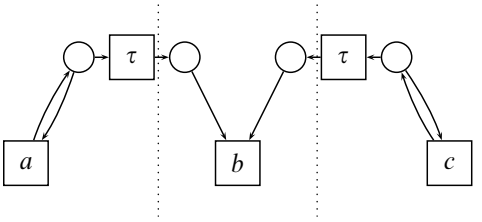



Fig. 4. A locally deadlocking implementation of Figure 2, location borders dotted.

When trying to implement a synchronous Petri net by a distributed one, one of the easiest approaches is central serialization of the entire original net by introduction of a single new place connected with loops to every transition, thereby vacuously fulfilling the requirement that no parallel transitions may reside on the same location. This clearly loses parallelism. We illustrate in Figure 3 the result of applying a slightly more intricate variant of this scheme, where every visible step of the original still exists in the implementation, to the repeated pure **M**. Nonetheless, this approach is intuitively not scalable and highly inefficient, as all decisions made concurrently in the original net are now made in sequence. In particular, the parts of the net firing *a* were completely independent of those parts firing *c* in the specification, while being connected through the central place in the implementation. Such new dependencies can be detected if the causal dependencies between events are included in the behavioural description of a net. Apart from the obvious implications for scalability, if a Petri net is used as an abstract description of a more concrete system, a new dependency might enable interactions between different parts of the system the designer did not take into account. Hence, we would like to disallow such a strategy by means of the equivalence between specification and implementation.

No such causalities are introduced by the implementation in Figure 4. There however, one of the cycles of *a*'s or *c*'s may spontaneously decide to commit to the *b* action and wait until the other does likewise, resulting in what is essentially a local deadlock. Compared to the original net, where *a* stayed enabled until *b* was fired, such behaviour is new. Trying to resolve this deadlock by adding a $\tau$-transition in the reverse direction would introduce a diverging computation not present in the original net.

All these deviations from the original behaviour can elegantly be captured by the causal equivalence from Schicke (2009), called completed pomset trace equivalence. It extends the pomset trace equivalence of Pratt (1985) as to detect local deadlocks, which can be regarded as unjust computations in the sense of Reisig (1984). This equivalence is presented in Section 2.5.

In addition we require that the implementation of a finite Petri net is again finite.

### 2.5. *Completed pomset trace equivalence*

Pomset trace equivalence is obtained by unrolling a Petri net into a process as defined by Petri (1977). Such a process can be understood to be an account of one particular way to decide all conflicts which occurred while proceeding from one marking to the next. The behaviour of the net is hence a set of these processes, covering all possible ways to decide conflicts.

Unrolling a net $N$ intuitively proceeds as follows: The initially marked places of $N$ are copied into a new net $\mathtt{N}$ and their correspondence to the original places recorded in a mapping $\pi$. Then, whenever in $N$ a transition $t$ is fired, this is replayed in $\mathtt{N}$ by a new transition connected to places corresponding by $\pi$ to the original preplaces of $t$ and which are not yet connected to any other post-transition. A new place of $\mathtt{N}$ is created for every token produced by $t$. Again all correspondences are recorded in $\pi$. Every place of $\mathtt{N}$ has, thus, at most one post-transition. If it has none, this place represents a token currently being placed on the corresponding original place.

As a shorthand notation to gather these places, we introduce the *end* of a net.

**Definition 2.7 (end of a net).** Let $N = (S, T, F, M_0, \ell)$ be a labelled net. The *end* of the net is defined as $N^\circ := \{s \in S \mid s^\bullet = \emptyset\}$.

**Definition 2.8 (process).**
A pair $\mathtt{P} = (\mathtt{N}, \pi)$ is a *process* of a net $N = (S, T, F, M_0, \ell)$ iff

— $\mathtt{N} = (\mathtt{S}, \mathtt{T}, \mathtt{F}, \mathtt{M}_0, \mathtt{l})$ is a net, satisfying
  – $\forall s \in \mathtt{S}.|{}^\bullet s| \leqslant 1 \geqslant |s^\bullet| \wedge s \in \mathtt{M}_0 \Leftrightarrow {}^\bullet s = \emptyset$
  – $\mathtt{F}$ is acyclic, i.e. $\forall x \in \mathtt{S} \cup \mathtt{T}.(x, x) \notin \mathtt{F}^+$, where $\mathtt{F}^+$ is the transitive closure of $\mathtt{F}$,
  – and $\{t \mid (t, u) \in \mathtt{F}^+\}$ is finite for all $u \in \mathtt{T}$.

— $\pi : \mathtt{S} \cup \mathtt{T} \to S \cup T$ is a function with $\pi(\mathtt{S}) \subseteq S$ and $\pi(\mathtt{T}) \subseteq T$, satisfying
  – $s \in M_0 \Leftrightarrow |\pi^{-1}(s) \cap \mathtt{M}_0| = 1$ for all $s \in S$,
  – $\pi$ is injective on $\mathtt{M}_0$,
  – $\forall t \in \mathtt{T}, s \in S.\ sF\pi(t) \Leftrightarrow \pi^{-1}(s) \cap {}^\bullet t \neq \emptyset \wedge \pi(t)Fs \Leftrightarrow \pi^{-1}(s) \cap t^\bullet \neq \emptyset$, and
  – $\forall t \in \mathtt{T}.\mathtt{l}(t) = \ell(\pi(t))$.[†]

$\mathtt{P}$ is called *finite* if $\mathtt{N}$ is finite.

$\mathtt{P}$ is *maximal* iff $\pi(\mathtt{N}^\circ) \not\rightarrow_N$. The set of all maximal processes of a net $N$ is denoted by $MP(N)$.

[†] While $\ell$ and $\mathtt{l}$ look nearly identical, the authors see no problem in that, given the close correspondence.

To disambiguate between a not-yet-occurred firing of a transition $a$ and the impossibility of firing an $a$, we restrict the set of processes relevant for the behavioural description to maximal processes. We thereby obtain a just semantics in the sense of Reisig (1984) [†], i.e. a transition which remained enabled infinitely long must ultimately fire.

To abstract from the $\tau$-actions introduced in an implementation, we extract from the maximal processes the causal structure between the fired visible events in the form of a partially ordered multiset (*pomset*). Formally, a pomset is an isomorphism class of a partially ordered multiset of action labels.

**Definition 2.9 (labelled partial order).** A *labelled partial order* is a structure $(V, T, \leqslant, l)$ where

— $V$ is a set (of *vertices*),
— $T$ is a set (of *labels*),
— $\leqslant \subseteq V \times V$ is a partial order relation and
— $l : V \to T$ (the *labelling function*).

Two labelled partial orders $o = (V, T, \leqslant, l)$ and $o' = (V', T, \leqslant', l')$ are *isomorphic*, $o \approxeq o'$, iff there exists a bijection $\varphi : V \to V'$ such that

— $\forall v \in V.l(v) = l'(\varphi(v))$ and
— $\forall u, v \in V.u \leqslant v \Leftrightarrow \varphi(u) \leqslant' \varphi(v)$.

**Definition 2.10 (pomset).** Let $o = (V, T, \leqslant, l)$ be a partial order. The *pomset* of $o$ is its isomorphism class $[o] := \{o' \mid o \approxeq o'\}$.

By hiding the unobservable transitions of a process, we gain a pomset which describes causality relations of all participating visible transitions.

**Definition 2.11 (pomset of maximal processes).** Let $\mathtt{P} = ((\mathtt{S}, \mathtt{T}, \mathtt{F}, \mathtt{M}_0, \mathtt{l}), \pi)$ be a process. Let $\mathtt{O} := \{t \in \mathtt{T} \mid \mathtt{l}(t) \neq \tau\}$, i.e. the visible transitions of the process. The *visible pomset* of $\mathtt{P}$ is the pomset $VP(\mathtt{P}) := [(\mathtt{O}, \mathrm{Act}, \mathtt{F}^* \cap \mathtt{O} \times \mathtt{O}, \mathtt{l} \cap (\mathtt{O} \times \mathrm{Act}))]$ where $\mathtt{F}^*$ is the transitive and reflexive closure of the flow relation $\mathtt{F}$.

$\mathrm{MVP}(N) := \{VP(\mathtt{P}) \mid \mathtt{P} \in MP(N)\}$ is the set of visible pomsets of all maximal processes of $N$.

Using this notion we can now define completed pomset trace equivalence.

**Definition 2.12 (completed pomset trace equivalence).** Two nets $N$ and $N'$ are *completed pomset trace equivalent*, $N \simeq_{CPT} N'$, iff $\mathrm{MVP}(N) = \mathrm{MVP}(N')$.

### 2.6. *Implementing synchronous Petri nets*

In the following, we show that it is in general impossible to distribute 1-safe nets while preserving finiteness of nets and ensuring that the source and the target nets are completed pomset trace equivalent. In particular we give a counterexample and prove that no finite distributed implementation of it can exist.

[†] or in modern terms, a 'weakly fair' semantics.

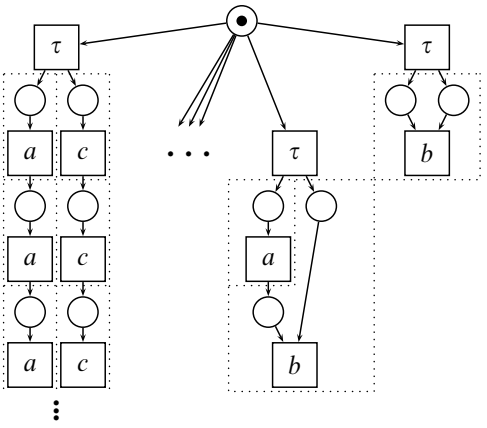



Fig. 5. An infinite implementation of Figure 2, constructed by taking every maximal process and initially choosing one, location borders dotted.

An infinite implementation always exists, as completed pomset trace equivalence is a very linear-time equivalence, and disregards the decision structure of a system. Hence an implementation like the one of Figure 5, which simply provides a separate branch for each possible maximal process of the original net, would be distributed and completed pomset trace equivalent. In practice though, such an infinite implementation is unwieldy to say the least. If, however, infinite implementations are ruled out, our main result shows that no valid implementation of the repeated pure **M** of Figure 2 exists.

Before we consider this main theorem of the paper, let us concentrate on two auxiliary lemmata. The first states that the careful introduction of a $\tau$-transition before an arbitrary transition of a net, as described below, does not significantly influence the properties of that net.

**Lemma 2.13.** Let $N = (S, T, F, M_0, \ell)$ be a finite, 1-safe, distributed net with the distribution function $D$. Let $t \in T$. The net $N' = (S', T', F', M_0, \ell')$ with
— $S' = S \cup \{s_t\}$,
— $T' = T \cup \{\tau_t\}$,
— $F' = (F \setminus (S \times \{t\})) \cup \{(s, \tau_t) \mid s \in {}^\bullet t\} \cup \{(\tau_t, s_t), (s_t, t)\}$, and
— $\ell'(x) = \begin{cases} \tau & \text{if } x = \tau_t \\ \ell(x) & \text{otherwise} \end{cases}$
is finite, 1-safe, distributed and completed pomset trace equivalent to $N$.

*Proof.* (Sketch)
$N'$ is finite as only two new elements were introduced.

$N'$ is completed pomset trace equivalent to $N$. Given a process $(\mathtt{N}, \pi)$ of $N$, a process of $N'$ can be constructed by refining in $\mathtt{N}$ every transition $u$ in the same manner as $\pi(u)$ was in $N$. For the reverse direction, note that in every maximal processes of $N'$, $\pi(u) = t \implies \pi({}^\bullet u) = \{s_t\} \wedge \pi({}^\bullet s_t) = \{\tau_t\}$. By fusing $u$, ${}^\bullet u$, and ${}^{\bullet\bullet} u$ into a single transition

$v$ whenever $\pi(u) = t$ and setting the process mapping of $v$ to $t$, a maximal process of $N'$ can be transformed into a maximal process of $N$.

For the same reason, $N'$ is also 1-safe.

$N'$ is distributed with the distribution function:

$$D'(x) := \begin{cases} D(t) & \text{if } x = s_t \vee x = \tau_t \\ D(x) & \text{otherwise.} \end{cases}$$

The places in ${}^\bullet\tau_t$ are on $D(t) = D'(\tau_t)$. $D'(s_t) = D(t) = D'(t)$. Hence all transitions are on the same location as their preplaces. No new parallelism is introduced, as a parallel firing of either $\tau_t$ or $t$ with some other transition $u$ can only occur if $t$ and $u$ could already fire in parallel in $N$. □

Next we show that if a marking is reached twice during a computation, the dependencies of all tokens consumed and produced by a transition firing in such a cycle are equal.

**Lemma 2.14.** Let $N = (S, T, F, M_0, \ell)$ be a finite, 1-safe net. Let $t_s, t_{s+1}, \ldots, t_{e-1}, t_e \in T$ be a sequence of transitions leading from a reachable marking $M_{base}$ to the same, i.e. $M_{base} \xrightarrow{\{t_s\}} \cdots \xrightarrow{\{t_e\}} M_{base}$. Then every $t_i$ produced tokens that were dependent on the same labels as the tokens on its preplaces.

*Proof.* Assume the opposite, i.e. there is a $t_i$ for $s \leqslant i \leqslant e$ such that $t_i$ consumed an $L$-independent token from one of its preplaces (for some $L \subseteq \mathrm{Act}$), but produced no $L$-independent tokens. This $L$-independent token needs to be replaced to again reach $M_{base}$. However, the replacement token needs to be $L$-independent as otherwise a dependency marking different from $M_{base}$ would be reached. This token can, thus, not depend on any of the tokens produced by $t_i$, as it would then not be $L$-independent. In other words, had $t_i$ not fired, a new $L$-independent token could also have been produced on its preplaces, i.e. $N$ would not be 1-safe, violating the assumptions. Hence no such $t_i$ can be fired, or equivalently, every $t_i$ produced tokens that were dependent on the same labels as the tokens on its preplaces (which hence all have the same dependencies). □

We will now show that, given an arbitrary finite, 1-safe net, it is not possible in general to find a finite, 1-safe, and distributed net which is completed pomset trace equivalent to the original. As a counterexample, consider the repeated pure **M** of Figure 2. It is a simple net allowing to perform several transitions of $a$ and $c$ in parallel, and terminating with a single transition $b$. The main argument of the following proof proceeds as follows: To perform an arbitrary number of $a$- and $c$-transitions within a finite net there has to be a loop. To terminate with $b$ the process has to escape from that loop by disabling all transitions leading to $a$ or $c$. Therefore, either a single token is consumed that is dependent on $a$ as well as on $c$, or two different tokens – one $a$-dependent and one $c$-dependent – are consumed. In the first case an additional iteration of the loop results in an additional causal dependency, i.e. in a causal dependency between $a$ and $c$. In the second case the net is not distributed in the sense of Definition 2.5.

**Theorem 2.15.** It is in general impossible to find for a finite, 1-safe net a distributed, completed pomset trace equivalent, finite, 1-safe net.

*Proof.* Via the counterexample given in Figure 2. Suppose a finite, 1-safe, distributed net $N_{impl}$, which is completed pomset trace equivalent to the net of Figure 2, would exist. By refining every $b$-labelled transition in $N_{impl}$ into two transitions in the manner of Lemma 2.13, a new net $N = (S, T, F, M_0, \ell)$ is derived. By Lemma 2.13 this new net is finite, 1-safe, distributed and completed pomset trace equivalent to the net in Figure 2 since $N_{impl}$ is.

$N$ has $|S|$ places and 3 different visible labels, every place can hold either no token, or a token dependent on any possible combination of the three labels. Since $N$ is finite so is $|S|$. Hence $N$ has at most $9^{|S|}$ reachable dependency markings. Let $m := 9^{|S|}$. $N$ is able to fire $(ac)^m b$ without any step containing more than a single transition since the net of Figure 2 is and the two are assumed to be completed pomset trace equivalent. Let $G_1, G_2, \ldots G_n$ be the steps fired while doing so. $|G_i| = 1$ for all $i$. In the course of firing that sequence, at least one dependency marking is bound to be reached at least twice. Of all those dependency markings which occur twice or more, we take the one occurring last while firing $(ac)^m b$ and call it $M_{base}$. Let $G_s, G_{s+1}, \ldots, G_{e-1}, G_e$ be a sequence of steps between two occurrences of $M_{base}$, i.e. $M_0 \times \{\varnothing\} \xrightarrow{G_1} \xrightarrow{G_2} \cdots M_{base} \xrightarrow{G_s} \cdots \xrightarrow{G_e} M_{base} \cdots \xrightarrow{G_n}$.

Using 2.14 the transitions of the steps $G_s$ to $G_e$ can be partitioned into subsets $T_X$ based on the dependencies of the tokens they produced and consumed. A set $T_X$ includes all transitions producing $X$-dependent, $\mathrm{Act} \setminus X$-independent tokens. By firing $G_s \cap T_{\{a\}}, G_{s+1} \cap T_{\{a\}}, \ldots, G_e \cap T_{\{a\}}$ (skipping empty steps) repeatedly, $M_{base} \stackrel{a^m}{\Longrightarrow}$. By firing $G_s \cap T_{\{c\}}, G_{s+1} \cap T_{\{c\}}, \ldots, G_e \cap T_{\{c\}}$ (skipping empty steps) repeatedly, $M_{base} \stackrel{c^m}{\Longrightarrow}$.

We now search for the marking where the decision to fire $b$ is made.

Assume a reachable marking $M''$ of $N$ with $M'' \stackrel{a^m}{\Longrightarrow}$. If $M'' \stackrel{c^m}{\not\Longrightarrow}$ this holds for all $M'''$ reachable from $M''$ since $c$ cannot be enabled using tokens produced by a transition labelled $a$ or $b$. Otherwise there would exist a pomsets of $N$ in which a $c$ is causally dependent on an $a$ or $b$. Such a pomset, however, does not exist for the net of Figure 2, thereby violating the assumption of completed pomset trace equivalence. If, however, $c$ is not re-enabled after $M''$, a maximal process including finitely many $c$ but infinitely many $a$'s can be produced also leading to a pomset not present in the net of Figure 2. The same argument can be applied with the roles of $a$ and $c$ reversed, hence $M'' \stackrel{a^m}{\Longrightarrow}$ iff $M'' \stackrel{c^m}{\Longrightarrow}$.

We start from $M_{base}$ and start to fire the steps $G_s, G_{s+1}, \ldots, G_n$ until $a^m$ cannot be fired any more for the first time. This step always exists as after $b$ no further $a$'s or $c$'s may be fired. Call the single transition in that step $t_b$. The marking right before that transition fired we call $M$, the one right after it $M'$. Not only $M \stackrel{a^m}{\Longrightarrow}$ but also $M \stackrel{c^m}{\Longrightarrow}$ and not only $M' \stackrel{a^m}{\not\Longrightarrow}$ but also $M' \stackrel{c^m}{\not\Longrightarrow}$, as both $M$ and $M'$ are reachable markings.

$t_b$ is not itself labelled $b$, as the refined net has a $\tau$-transition before the $b$, and once a token resides on the intermediate place, no $a$-transitions can be fired any more, as otherwise a pomset where an $a$ which is not a causal predecessor to a $b$ would be produced, again not existing for the net of Figure 2.

To disable the trace $a^m$, the transition $t_b$ needed to consume a token. If $t_b$ had not fired, some $G_i \cap T_{\{a\}}$, $s \leqslant i \leqslant e$ could have consumed that token, hence that token must be $a$-dependent, $c$-independent. Similarly, $t_b$ must have consumed a token which could have led to $c^m$. This token needs to be $c$-dependent, $a$-independent. Hence $t_b$ has at least two preplaces, which in turn are also preplaces to two different transitions, call them $t_a$ and

$t_c$, which then lead to $a^m$ and $c^m$, respectively.† As they have common preplaces, $t_a$, $t_b$ and $t_c$ are on the same location.

From $M$ the net can fire $a^m$ consuming only $a$-dependent, $c$-independent tokens. It can also fire $c^m$ consuming only $c$-dependent, $a$-independent tokens.

Hence there is a sequence of steps leading from $M$ to a marking where $t_a$ is enabled, yet only $a$-dependent, $c$-independent tokens have been removed or added. Similarly there is a firing sequence leading from $M$ to a marking where $t_c$ is enabled, yet only $c$-dependent, $a$-independent tokens have been removed or added. As they change disjunct sets of tokens, these two firing sequences can be concatenated, thereby leading to a marking where $t_a$ and $t_c$ are concurrently enabled, yet they are on the same location, thereby violating the implementation requirements. □

Note that the self-loops of the counterexample are not critical to the success of the proof. We can in fact generalize the $a$- and $c$-transitions to arbitrary transition sequences $aa_0a_1 \dots a_n$ (none of which labelled $c$) and $cc_0c_1 \dots c_n$ (none of which labelled $a$). The proof goes through when replacing $a$ and $c$ with $\ell(a)\ell(a_0)\ell(a_1)\dots\ell(a_n)$ and $\ell(c)\ell(c_0)\ell(c_1)\dots\ell(c_n)$ in all firing sequences and adjusting the number of visible labels.

Unfortunately a characterization of such critical structures, of which Figure 2 depicts only an example, in semi-structural terms as done, e.g. by Glabbeek *et al.* (2008) is not possible here: In the end, the two firing sequences forming the loops then would need to be identifiable from the semi-structural properties. To ensure that both exist and can actually be fired independently, there must exist a sufficiently long sequence of reachable markings where both places loose and receive tokens sufficiently often while never getting the wrong causalities. This, however, ceases to be a property of a single marking and should no longer be considered semi-structural.

This paper only considered 1-safe nets as possible implementations. We conjecture, however, that the proof of Theorem 2.15 can be extended to non-safe implementations as well, as from a place where tokens of different dependency mix, a transition can always choose the most-dependent token. In particular a transition intended to produce independent tokens cannot have such a place as a preplace. Hence every part of the net providing independent tokens can do so without depending on firings of labelled transitions. The number of independent tokens produced on a place where a labelled transition consumes them is, thus, either finite over every run of the system, or unbounded even without any labelled transition ever firing. In both cases that place is unsuitable for disabling a potentially infinitely often occurring loop. If only finitely many tokens are produced, the loop can no longer happen infinitely often, if an unbounded number of tokens can be produced, no disabling can be guaranteed.

Comparing the proof of Theorem 2.15 with the proof by Glabbeek *et al.* (2008), we observe that the counterexample in both proofs is based on two conflicts overlapping the same transition, i.e. on what is therein called a fully reachable pure **M**. In the synchronous

† The removal of the token leading to $a^m$ and the one leading to $c^m$ must indeed be done by a single transition $t_b$ as only a single transition was fired between $M$ and $M'$ and both traces were possible in $M$ but impossible in $M'$.

setting such an overlapping conflict is solved by the simultaneous removal of tokens on different places in the preset. In an asynchronous setting these two conflicts have to be distributed over at least two locations. Intuitively, the problem with such a distribution is that it prevents the simultaneous solution of the original overlapping conflicts. Instead these two conflicts have to be solved in some order. This order must, as done within the encoding presented by Schicke (2009), be enforced by the encoding, leading to additional causal dependencies.

## 3. Synchrony versus causality in the $\pi$-calculus

Similar to the work in the previous section, we show that in the $\pi$-calculus it is *not* possible to encode synchronous interactions within a completely asynchronous framework without introducing additional causal dependencies in the translation.

It is debatable how well a discussion on synchrony versus asynchrony can be separated from a discussion of choice when considering the $\pi$-calculus. 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* Nestmann (1998), as motivated in the join calculus Fournet and Gonthier (1996), represents another natural and interesting generalization.) Likewise, as soon as a calculus offers synchronous output, one may generalize this primitive to output-guarded choice. This generalization 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.

We are interested in the conditions under which it is possible to encode the synchronous $\pi$-calculus into its asynchronous variant. 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. Parrow (2008)). 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 $\pi$-calculus variants. Since full abstraction depends, by definition, strongly on the chosen equivalences, a variation in the

respective choice may 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 (2008, 2010) identifies five criteria as being well suited for separation as well as encodability results. Here, we rely on these five criteria to measure the quality of encodings between variants of the $\pi$-calculus. Compositionality and name invariance stipulate structural conditions on a good encoding. Operational correspondence requires that a good encoding preserves and reflects the executions 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 a successful state.

It is well known that there is a good encoding from the choice-free synchronous $\pi$-calculus into its asynchronous variant (see Boudol (1992); Honda and Tokoro (1991); Honda (1992)). It is also well-known Palamidessi (2003); Gorla (2010); Peters and Nestmann (2010) that there is no good encoding from the full $\pi$-calculus – the synchronous $\pi$-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 $\pi$-calculus compared to its asynchronous variant. Analysing this result Peters and Nestmann (2010), we observe that it boils down to the fact that only the full $\pi$-calculus can break syntactic symmetries, whereas its asynchronous variant cannot. Moreover, as already Gorla (2010) 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 3.5 at page 1481). As proved by Peters and Nestmann (2012), 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 $\pi$-calculus (including mixed choice) into its asynchronous variant with respect to the criteria of Gorla[†]. Analysing the encoding given by Peters and Nestmann (2012), 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. Next, we show that this is a general phenomenon of encoding synchrony in the $\pi$-calculus.

Thus, as a main contribution of this section, we show that – in the asynchronous $\pi$-calculus – there is a strong connection between synchronous interactions and causal dependencies. More precisely we show – analogue to the separation result on Petri nets of the previous section – that no encoding from the synchronous $\pi$-calculus with mixed choice into the asynchronous $\pi$-calculus preserves causal independence and satisfies all the criteria of Gorla (2010).

[†] Note that this encoding is neither prompt nor is the assumed equivalence $\asymp$ strict, so the separation results of Gorla 2010 do not apply here.

### 3.1. *Overview*

In Section 3.2, we introduce the variants of the $\pi$-calculus we want to compare. Section 3.3 introduces the notion of encoding and defines the set of criteria we assume to hold to call an encoding 'good'. Causality – or more precisely causal independence – is then defined for the $\pi$-calculus in Section 3.4. In Section 3.5, we present our separation result for the $\pi$-calculus.

### 3.2. *Technical preliminaries*

We study the relation between process calculi that differ in their either synchronous or asynchronous interaction mechanism. Our source language – in case of the $\pi$-calculus – is the monadic $\pi$-calculus as described for instance by Sangiorgi and Walker (2001). Since the main reason for the differences in the expressiveness of the full $\pi$-calculus compared to the asynchronous $\pi$-calculus is the power of mixed choice we denote the full $\pi$-calculus also by $\pi_{\text{mix}}$.

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.

**Definition 3.1** ($\pi_{\text{mix}}$**).** The set of process terms of the *$\pi$-calculus (with mixed choice)*, denoted by $\mathcal{P}_{\text{mix}}$, is given by

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

where $\pi \;::=\; y\,(x) \;\mid\; \overline{y}\,\langle z \rangle$ for some names $a, b, n, x, y, z \in \mathcal{N}$ 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 $P$. $P_1 \mid P_2$ defines *parallel composition*, i.e. the process in which $P_1$ and $P_2$ may proceed independently, possibly interacting using shared links. $!P$ denotes *recursion*. The *match prefix* $[\, a = b \,]\, P$ works as a conditional guard. It can be removed iff $a$ and $b$ are equal. The process term $\sum_{i \in I} \pi_i.P_i$ represents *finite guarded choice*; as usual, the sum $\sum_{i \in \{\, 1, \ldots, n \,\}} \pi_i.P_i$ is sometimes written as $\pi_1.P_1 + \cdots + \pi_n.P_n$ and $\mathbf{0}$ abbreviates the empty sum, i.e. where $I = \varnothing$. 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 $y$. We denote $y$ as the subject of an action prefix $\overline{y}\,\langle z \rangle$ or $y\,(x)$ and $z$ or $x$ as its object. All branches of a choice are guarded by one of these prefixes. Since some examples and our counterexample in Section 3.5 are CCS-like we often omit the objects of actions. Moreover, we denote the empty sum with $\mathbf{0}$ and omit it in continuations, e.g. $\overline{y}\,\langle z \rangle .\mathbf{0}$ is abbreviated as $\overline{y}$. The term $\checkmark$ denotes *success* (or *successful termination*). It is introduced in order to compare the abstract behaviour of terms in different process calculi as described in Section 3.3.

The asynchronous $\pi$-calculus ($\pi_{\text{a}}$) was introduced independently in 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 $\pi_{\text{a}}$, output actions are not allowed to guard a process different from $\mathbf{0}$. Accordingly, the interpretation of output guards within

a choice construct is delicate. Here, we use the standard variant of $\pi_a$ , where choice is not allowed at all. Since $\mathcal{P}_a$ has no choice, and thus no nullary choice, we include $\mathbf{0}$ as a primitive.

**Definition 3.2 ($\pi_a$).** The set of process terms of the *asynchronous $\pi$-calculus*, denoted by $\mathcal{P}_a$, is given by

$$P ::= (\nu\, n)\, P \quad | \quad P_1 \mid P_2 \quad | \quad !P \quad | \quad \mathbf{0} \quad | \quad \overline{y}\langle z\rangle \quad | \quad y(x).P \quad | \quad [\,a = b\,]\, P \quad | \quad \checkmark$$

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

As shown by the encoding by Nestmann (2000) one could also use separate choice within an asynchronous variant of the calculus without a significant effect on its expressive power. Accordingly, our separation result holds already for $\pi_{sep} - \pi_{mix}$ restricted to separate choice – as target language.

**Definition 3.3 ($\pi_{sep}$).** The set of process terms of the *$\pi$ with separate choice*, denoted by $\mathcal{P}_{sep}$, is given by

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

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

As expected, the definitions of $\pi_{sep}$and $\pi_{mix}$ differ in the definition of choice only. In the following we use $\pi_{sep}$as target language. Since $\pi_a$ is a subcalculus of $\pi_{sep}$the following results hold also for $\pi_a$ as target language.

In the literature there are different variants of the 'full' $\pi$-calculus. It can be defined without the match prefix, with different variants of replication or recursion, or with generalized choice $P_1 + P_2$ instead of guarded choice. Note that neither the match prefix nor any form of replication or recursion occurs in our counterexample in Section 3.5. Moreover, the definition of causality in Section 3.4 is such that the presence of the match prefix does not influence Lemmas 3.18 and 3.19. Hence, because our separation result in Section 3.5 relies only on these two lemmata for causality and the counterexample as only source term, the presence of the match prefix does not influence our results. In fact our separation result holds even if we remove the match prefix from our source language $\pi_{mix}$ but let it remain in our target languages $\pi_{sep}$ and $\pi_a$ , i.e. even with the power of matching – which as shown in Carbone and Maffeis (2003) increases the expressive power of the $\pi$-calculus – no 'good' encoding from $\pi_{mix}$ into $\pi_a$ preserves causal independence. The same holds for replication or recursion. Causality can be defined for different variants of replication or recursion. Priami (1996) e.g. defines causality for a variant of the $\pi$-calculus with recursive definitions of process constants instead of replication. Again for all these variants we can show conditions similar to Lemmas 3.18 and 3.19. Hence our results are also not influenced by the choice of the operator for replication or recursion.

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

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

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

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

Fig. 6. Structural congruence.

Similarly we can replace guarded choice by generalized choice $P_1 + P_2$ in our source language $\pi_{\mathrm{mix}}$, because guarded choice is a special case of generalized choice and, thus, our counterexample would still belong to the source language. On the other side replacing separate choice by generalized choice or adding generalized choice to $\pi_{\mathrm{a}}$ would invalidate our separation result. However, since already the interpretation of output guards within a choice construct is delicate for an asynchronous calculus, we do not consider a variant of the $\pi$-calculus with generalized choice as asynchronous.

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 $\mathsf{fn}(P)$ denote the set of *free names* in $P$. Let $\mathsf{bn}(P)$ denote the set of *bound names* in $P$. Likewise, $\mathsf{n}(P)$ denotes the set of all *names* occurring in $P$. Their definitions are completely standard, i.e. names are bound by restriction and as parameter of input and $\mathsf{n}(P) = \mathsf{fn}(P) \cup \mathsf{bn}(P)$ for all $P$.

Let $P \in \mathcal{P}$. A term $M$ is a *top-level subterm of* $P$ if $M$ is a subterm that is unguarded in $P$ such that the outermost operator of $M$ is choice if $\mathcal{P} \in \{\, \mathcal{P}_{\mathrm{mix}}, \mathcal{P}_{\mathrm{sep}} \,\}$; and else if $\mathcal{P} \in \mathcal{P}_{\mathrm{a}}$ then either $M = y\,(x).P'$ or $M = \overline{y}\,\langle z \rangle$ for some $x, y, z \in \mathcal{N}$ and $P' \in \mathcal{P}_{\mathrm{a}}$. In both cases there is a sequence of names $\tilde{n}$ and $P' \in \mathcal{P}$ such that $P \equiv (\nu\, \tilde{n})\, (M \mid P')$.

The *reduction semantics* of $\pi_{\mathrm{mix}}$, $\pi_{\mathrm{sep}}$and, $\pi_{\mathrm{a}}$ are jointly given by the transition rules in Figure 7, where *structural congruence*, denoted by $\equiv$, is given by the rules in Figure 6. Note that the rule $\textsc{Com}_{\mathrm{a}}$ for communication in $\pi_{\mathrm{a}}$ is a simplified version of the rule Com for communication in $\pi_{\mathrm{mix}}$ and $\pi_{\mathrm{sep}}$. 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 $\pi_{\mathrm{a}}$. As usual, we use $\equiv_\alpha$ if we refer to alpha-conversion (the first rule of Figure 6) only.

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

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

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

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

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

Fig. 7. Reduction semantics of $\pi_{\mathrm{mix}}$ and $\pi_{\mathrm{a}}$.

Let $P \longmapsto$ (and $P \not\longmapsto$) denote the existence (and non-existence) of a step from $P$, i.e. there is (no) $P' \in \mathcal{P}$ such that $P \longmapsto P'$. Moreover, let $\Longmapsto$ be the reflexive and transitive closure of $\longmapsto$. We write $P \longmapsto^{\omega}$ if $P$ can perform an infinite sequence of reduction steps. A sequence of reduction steps starting in a term $P$ is called *execution* of $P$. An execution is either finite as $P_0 \longmapsto P_1 \longmapsto \cdots \longmapsto P_n$ or infinite. A finite execution $P_0 \Longmapsto P_n$ is maximal if it cannot be further extended, i.e. if $P_n \not\longmapsto$, otherwise it is partial. We denote a process as convergent, if it has no infinite execution.

In order to conveniently identify which occurrences of actions contributed to the reduction steps in an execution at hand, we define a *labelling* on it: more precisely, to each of the prefixes occurring in the terms of the execution, we assign a unique label. Therefore, let $\mathcal{L}$ be a set of labels such that $\mathcal{L} \cap \mathcal{N} = \varnothing$ and $\mathcal{L} \cap \overline{\mathcal{N}} = \varnothing$. The labelled variants of the sets $\mathcal{P}_{\mathrm{mix}}$, $\mathcal{P}_{\mathrm{sep}}$, and $\mathcal{P}_{\mathrm{a}}$ are obtained by replacing prefixes of the form $\pi.P$ simply by $\pi_l.P$, where $l \in \mathcal{L}$. But, for the purpose of uniqueness, all prefixed actions of a process term must be equipped with pairwise different labels.

In order to state that a labelled execution consists of steps that are deduced via a reduction semantics, we define a labelled version of both the reduction semantics and structural congruence. They are obtained in a mostly straightforward way, as follows:

1. With the exception of the rule $!P \equiv P \mid !P$, the labelled variants of the structural congruence rules are unchanged, but just operate on labelled terms $P$, $Q$, and $R$. The labelled variant of $!P \equiv P \mid !P$ is $!P \equiv P' \mid !P$, where again $P$ is a labelled term and $P'$ is obtained from $P$ by replacing all labels by fresh labels.
2. The labelled variants of the rules Com and $\textsc{Com}_{\mathrm{a}}$ are

$$\underbrace{\left(\cdots + y\,(x)_{l_1}.P + \cdots\right)}_{M_1} \mid \underbrace{\left(\cdots + \overline{y}\,\langle z\rangle_{l_2}.Q + \cdots\right)}_{M_2} \overset{\{l_1,l_2\}}{\longmapsto} \{\,{}^{z}/_{x}\,\}\,P \mid Q$$

and

$$y\,(x)_{l_1}.P \mid \overline{y}\,\langle z\rangle_{l_2} \overset{\{l_1,l_2\}}{\longmapsto} \{\,{}^{z}/_{x}\,\}\,P$$

with $P$ and $Q$ being labelled terms and $l_1, l_2 \in \mathcal{L}$. Abbreviations for $X \overset{\{l_1,l_2\}}{\longmapsto} Y$ are $X \overset{l_1,l_2}{\longmapsto} Y$ and $X \overset{L}{\longmapsto} Y$ (with $L = \{l_1, l_2\}$).

Accordingly, the labelled variants of PAR, RES, and CONG are

$$\frac{P \overset{L}{\longmapsto} P'}{P \mid Q \overset{L}{\longmapsto} P' \mid Q} \qquad \frac{P \overset{L}{\longmapsto} P'}{(\nu\, n)\, P \overset{L}{\longmapsto} (\nu\, n)\, P'}$$

and

$$\frac{P \equiv Q \quad Q \overset{L}{\longmapsto} Q' \quad Q' \equiv P'}{P \overset{L}{\longmapsto} P'},$$

where $P$, $P'$, $Q$, and $Q'$ are labelled terms and $\equiv$ denotes the labelled variant of structural congruence.

By this adaptation of the definition, a labelled term has a labelled reduction step iff its corresponding unlabelled term has a standard reduction step.

A reduction step $P \overset{L}{\longmapsto} P'$ *reduces the prefixed action* $\pi_1$ *and* $\pi_2$ or *the corresponding subterms* $M_1$ *and* $M_2$ *that are top-level in* $P$ if these are – identifiable through their labels – the prefixed actions (or the subterms containing them) that are used to instantiate the only application of the labelled COM/$\text{COM}_\text{a}$-rule in the proof tree of this step.

A labelled (and potentially partial) execution is then an execution of labelled terms. Due to the freshness condition on applications of the replication law, we need to construct the labelling of terms in a given execution in an on-the-fly manner. For our purpose, we only need to define labelling for finite executions.

**Definition 3.4 (labelled execution).** Let $E : P_0 \longmapsto \cdots \longmapsto P_n$ be a finite execution and let $\mathcal{L}$ be a set of labels. To obtain a labelled variant of $E$

1. Assign a unique label $l \in \mathcal{L}$ to each prefix in $P_0$.
2. For all $i \in \{ 1, \ldots, n \}$ the labels of $P_i$ are obtained from the labels of $P_{i-1}$ by replacing in the proof tree of $P_{i-1} \longmapsto P_i$ the term $P_{i-1}$ and the rules of structural congruence and operational semantics by their corresponding labelled variant. Thereby for each application of $!P \equiv P' \mid !P$ the fresh labels introduced by this rule do not occur in the labelled variant of the execution so far, i.e. they are distinct from all labels in $P_0, \ldots, P_{i-1}$ and all other labels in $P_i$.

Note that, because reduction steps consume prefixed actions, $P_i$ does not contain all the labels of $P_{i-1}$. On the other side, because of the introduction of fresh labels by the structural congruence rule $!P \equiv P \mid !P$, the term $P_i$ can contain some labels that do not yet occur in $P_{i-1}$. Moreover, note that an unlabelled finite execution can have different labelled variants. However, by definition, all these variants only differ in the choice of distinct labels, i.e. all these variants are pairwise equivalent modulo some bijection on labels.

The first quality criterion to compare process calculi presented in Section 3.3 is compositionality. It induces the definition of a context parameterized on a set of names for each operator of $\pi_{\text{mix}}$ . A context $\mathcal{C}([\cdot]_1, \ldots, [\cdot]_n) : \mathcal{P}^n \to \mathcal{P}$ is simply a $\pi$-term, i.e. a $\pi_{\text{sep}}$-term (or $\pi_\text{a}$ -term) in case of Definition 3.5, with $n$ holes. Putting some $\pi_{\text{sep}}$-terms (or $\pi_\text{a}$ -terms) $P_1, \ldots, P_n$ in this order into the holes $[\cdot]_1, \ldots, [\cdot]_n$ of the context, respectively,

gives a term denoted $\mathcal{C}(P_1, \ldots, P_n)$. Note that a context may bind some free names of $P_1, \ldots, P_n$. The arity of a context is the number of its holes.

### 3.3. *Quality criteria for encodings in the π-calculus*

Gorla (2010) presented a small framework of five criteria well suited for language comparison in case of process calculi. We use these five criteria to measure the quality of an encoding $[\![ \cdot ]\!]$ from $\pi_{\text{mix}}$ into $\pi_{\text{a}}$ or $\pi_{\text{sep}}$, i.e. such an encoding $[\![ \cdot ]\!]$ is 'good' if it fulfils the criteria proposed by Gorla. Note that for the definition of these criteria a behavioural equivalence $\asymp$ on the target language is assumed. Its purpose is to describe the abstract behaviour of a target process, where abstract basically means with respect to the behaviour of the source term.

The five conditions are divided into two structural and three semantic criteria. The structural criteria comprise (1) *compositionality* and (2) *name invariance*. The semantic criteria comprise (3) *operational correspondence*, (4) *divergence reflection* and (5) *success sensitiveness*. In the following we use $S, S', S_1, \ldots$ to range over terms of the source language and $T, T', T_1, \ldots$ 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 parameterized on the free names of the corresponding source term. Note that our result is independent of this parameterization.

**Definition 3.5 (criterion 1: compositionality).** The encoding $[\![ \cdot ]\!]$ is *compositional* if, for every k-ary operator **op** of the source language and for every subset of names $N$, there exists a k-ary context $\mathcal{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_k)$ in the target language such that, for all $S_1, \ldots, S_k$ with $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_k) = N$, it holds that

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

The second structural criterion of Gorla states that the encoding should not depend on specific names used in the source term. We do not need this criterion for our separation result. Thus, we omit it.

The first semantic criterion is operational correspondence, which consists of a soundness and a completeness condition. *Completeness* requires that every execution of a source term can be simulated by its translation, i.e. the translation does not omit any executions of the source term. *Soundness* requires that every execution of a target term corresponds to some execution of the corresponding source term, i.e. the translation does not introduce new executions.

**Definition 3.6 (criterion 3: operational correspondence).** The encoding $[\![ \cdot ]\!]$ is *operationally corresponding* if it is

*Complete*: for all $S \Longmapsto S'$, it holds that $[\![ S ]\!] \Longmapsto \asymp [\![ S' ]\!]$;

*Sound*: for all $[\![ S ]\!] \Longmapsto T$, there exists an $S'$ such that $S \Longmapsto S'$ and $T \Longmapsto \asymp [\![ S' ]\!]$.

Note that the Definition of operational correspondence relies on the equivalence $\asymp$ to get rid of junk possibly left over within executions 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 executions in encodings.

**Definition 3.7 (criterion 4: divergence reflection).** The encoding $[\![ \cdot ]\!]$ reflects divergence if, for every $S$, $[\![ S ]\!] \longmapsto^{\omega}$ implies $S \longmapsto^{\omega}$.

The last criterion links the behaviour of source terms to the behaviour of target terms. With Gorla (2010), we assume a *success* operator ✓ to be part of the syntax of both the source and the target language. Since ✓ cannot be further reduced, the operational semantics is left unchanged. 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 3.8 (success).** A process $P \in \mathcal{P}$ *may lead to success*, denoted as $P \Downarrow_{\checkmark}$, iff it is reducible to a process containing a top-level unguarded occurrence of ✓, i.e. $\exists P', P'' \in \mathcal{P} \,.\, P \Longrightarrow P' \wedge P' \equiv P'' \mid \checkmark$.

Note that we choose may-testing here. However, as we claim, our main result in Theorem 3.24 holds for must-testing, as well. Moreover, we write $P \Downarrow_{\checkmark!}$, if $P$ has only finite executions and reaches success in every finite maximal execution.

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 3.9 (criterion 5: success sensitiveness).** The encoding $[\![ \cdot ]\!]$ is *success sensitive* if, for every $S$, $S \Downarrow_{\checkmark}$ iff $[\![ S ]\!] \Downarrow_{\checkmark}$.

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 3.10 (success respecting).** $\asymp$ *respects success* if, for every $P$ and $Q$ with $P \Downarrow$ and $Q \not\Downarrow$, it holds that $P \not\asymp Q$.

By Gorla (2010) a 'good' equivalence $\asymp$ 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. For the separation results presented in this paper, we require only that $\asymp$ is a success respecting reduction bisimulation.

**Definition 3.11 (reduction bisimulation).** The equivalence $\asymp$ is a *(weak) reduction bisimulation* if, for every $T_1, T_2$ in the target language such that $T_1 \asymp T_2$, for all $T_1 \Longrightarrow T_1'$ there exists a $T_2'$ such that $T_2 \Longrightarrow T_2'$ and $T_1' \asymp T_2'$.

In this case, a good encoding respects also the ability to reach success in all finite maximal executions.

**Lemma 3.12.** For all success respecting reduction bisimulations $\asymp$ and all convergent target terms $T_1, T_2$ such that $T_1 \asymp T_2$, it holds $T_1 \Downarrow_{\checkmark!}$ iff $T_2 \Downarrow_{\checkmark!}$.

*Proof.* Let us assume the opposite, i.e. there is some success respecting bisimulation $\asymp$ and two convergent target terms $T_1, T_2$ such that $T_1 \asymp T_2$ and $T_1 \Downarrow_{\checkmark!}$ but not $T_2 \Downarrow_{\checkmark!}$. Then, for all $T_1'$ with $T_1 \Longrightarrow T_1'$, we have $T_1' \Downarrow_{\checkmark}$ but there exists some $T_2'$ such that $T_2 \Longrightarrow T_2'$ and $T_2' \not\Downarrow_{\checkmark}$. By Definition 3.11, $T_1 \asymp T_2$ and $T_2 \Longrightarrow T_2'$ imply that there exists some $T_1''$ such that $T_1 \Longrightarrow T_1''$ and $T_2' \asymp T_1''$. By Definition 3.10, $T_2' \asymp T_1''$ and $T_2' \not\Downarrow_{\checkmark}$ imply $T_1'' \not\Downarrow_{\checkmark}$. This violates the requirement that $T_1 \Downarrow_{\checkmark!}$, i.e. contradicts the assumption that for all $T_1'$ with $T_1 \Longrightarrow T_1'$ we have $T_1' \Downarrow_{\checkmark}$. We conclude that $T_1 \Downarrow_{\checkmark!}$ iff $T_2 \Downarrow_{\checkmark!}$. □

Moreover, in this case success sensitiveness preserves also the ability to reach success in all finite maximal executions.

**Lemma 3.13.** For all operationally sound, divergence reflecting, and success sensitive encodings $[\![ \cdot ]\!]$ with respect to some success respecting equivalence $\asymp$ and for all convergent source terms $S$, if $S \Downarrow_{\checkmark!}$ then $[\![ S ]\!] \Downarrow_{\checkmark!}$.

*Proof.* Assume the opposite, i.e. there is an encoding that satisfies the criteria operational soundness, divergence reflection, and success sensitiveness, $\asymp$ respects success, and there is some convergent $S$ such that $S \Downarrow_{\checkmark!}$, but $[\![ S ]\!] \not\Downarrow_{\checkmark!}$. By divergence reflection, all encodings of a convergent source term are convergent. Thus, $[\![ S ]\!] \not\Downarrow_{\checkmark!}$ implies that there is some $T$ such that $[\![ S ]\!] \Longrightarrow T$ and $T \not\Downarrow_{\checkmark}$. By Definition 3.6, $[\![ S ]\!] \Longrightarrow T$ implies that there exists some $S''$ and some $T'$ such that $S \Longrightarrow S''$ and $T \Longrightarrow T' \asymp [\![ S'' ]\!]$. By Definition 3.8, then $T \not\Downarrow_{\checkmark}$ and $T \Longrightarrow T'$ imply $T' \not\Downarrow_{\checkmark}$. By Definition 3.10, $T' \asymp [\![ S'' ]\!]$ and $T' \not\Downarrow_{\checkmark}$ imply $[\![ S'' ]\!] \not\Downarrow_{\checkmark}$. By Definition 3.9, then also $S'' \not\Downarrow_{\checkmark}$, which contradicts the assumption that $S \Downarrow_{\checkmark!}$. We conclude that if $S \Downarrow_{\checkmark!}$ then $[\![ S ]\!] \Downarrow_{\checkmark!}$. □

### 3.4. *Causality in the π-calculus*

As explained above, the extraction of causal information from Petri nets is rather unambiguous. This is not so in the $\pi$-calculus. Here, a number of approaches have been pursued to extract causal information from process terms and their semantics, leading to various notions of dependencies mostly defined on actions, inducing notions of dependency on transitions or reduction steps. Following Boreale and Sangiorgi (1998) two kinds (or sources) of causal dependencies can be distinguished.

The first kind of causal dependencies, called *structural* or *subject dependency*, originates from the nesting of prefixes, i.e. from the structure of processes. Therefore, it is a notion that already occurs in non-name-passing calculi like CCS. A typical example of such a dependency is given by the term $(\nu\, b)(\overline{a}.\overline{b} \mid b.\overline{c})$, in which – according to the operational semantics – the action $\overline{a}$ must happen before the action $\overline{b}$ can take place; likewise does the action $\overline{c}$ depend on the action $b$ having happened before. There is another, less explicit dependency involved, here, due to the restriction on name $b$. Having again the operational semantics in mind, $\overline{c}$ also causally depends on $\overline{a}$ happening before, because

only then the required interaction between $\overline{b}$ and $b$ becomes *enabled*[†]. Note, however, that this observation is only valid due to the restriction on $b$: without the restriction on $b$, the action $b$ in the term $(\overline{a}.\overline{b} \mid b.\overline{c})$ might also find some partner $\overline{b}$ from elsewhere, which is usually expressed by means of a standard labelled transition semantics. On the other hand, if we only have reduction semantics in mind – as we do in this paper – then we implicitly assume that we ignore such communication possibilities to some extent. With reduction semantics, one would need to supply this partner explicitly, as in $(\overline{a}.\overline{b} \mid b.\overline{c}) \mid \overline{b}$, to let reduction semantics observe this causal independence. In fact, it depends on the actual execution leading to $\overline{c}$ being enabled, whether we want to state that, in this particular execution, the $\overline{c}$ was actually causally depending on $\overline{a}$, or not. Let us now also add a partner for the action $c$ and analyse the execution $\left(\overline{a}.\overline{b} \mid b.\overline{c}\right) \mid a \mid c \longmapsto \left(\overline{b} \mid b.\overline{c}\right) \mid c \longmapsto \overline{c} \mid c \longmapsto \mathbf{0}$. In what follows, we no longer observe the dependencies between actions in a term, but between steps in a reduction sequence. Here, the second step on channel $b$ is causally dependent on the first step on $a$, because this particular first step unguards $\overline{b}$. Similarly, the step on $c$ is causally dependent on the step on $b$, and by transitivity the step on $c$ is – in this particular execution – causally dependent on the step on $a$.

The second kind of causal dependencies in the $\pi$-calculus is called *link* or *object dependency* and originates from the binding mechanisms on names; thus, it is specific to name-passing calculi. Here, a typical example is $(\nu\, x)\,(\overline{y}\,\langle x\rangle \mid \overline{x})$. With a labelled semantics in mind, the action $\overline{x}$ causally depends on the extrusion of $x$ by the action $\overline{y}\,\langle x\rangle$, because none but a receiver along $y$ may possibly know and use the name $x$. In contrast, with a reduction semantics in mind, and as sketched above the complementary actions would need to be supplied explicitly to see the extrusion effect and observe the object dependency. So, let us analyse the execution $(\nu\, x)\left(\overline{y}\,\langle x\rangle \mid \overline{x}\right) \mid y\,(z)\,.z \longmapsto (\nu\, x)\left(\overline{x} \mid x\right) \longmapsto \mathbf{0}$. Note that, here, the reduction step on $x$ causally depends on the reduction step on $y$. However, the dependency ultimately arises from the required structural dependency of the partner. In fact, this is no real surprise, as within reduction semantics all scope extrusions result from transformations within structural congruence. Hence, as we only consider reduction semantics in this paper, we safely restrict our attention to subject dependencies.

In general, and as already indicated above, different executions of the same term can exhibit different causal dependencies between steps, although they arise from the same actions. There is, for instance, a causal dependency in the execution $(\overline{a} \mid a.\overline{b} \mid b \mid \overline{b}) \longmapsto (\overline{b} \mid b \mid \overline{b}) \longmapsto \overline{b}$ – if the second step reduces the $\overline{b}$ that is originally guarded by $a$, then the second step is causally dependent on the first – whereas there is no causal dependency between the first and second step in $(\overline{a} \mid a.\overline{b} \mid b \mid \overline{b}) \longmapsto (\overline{a} \mid a.\overline{b}) \longmapsto \overline{b}$. There are two problems in the detection of causal dependencies within an execution. First, in order to detect causal dependencies we have to distinguish between equivalent parts of a term as the two instances of $\overline{b}$ in the example above. Second, in order to examine whether a step $s_2$ is causally dependent on a step $s_1$ in an execution, we have to consider all steps that occur in this execution after the step $s_1$ and before the step $s_2$.

[†] Note that the notions of causality and enabling are sometimes kept separate Priami (1996).

Technically, in order to detect causal dependencies in executions, we use the labelled counterparts introduced in Section 3.2. Before we delve into this, we first exploit the labelling to define notions of conflict between reduction steps. Intuitively, steps are in conflict if each of them can happen but not both together. Technically, steps are in conflict when they compete for some of the subterms they reduce.

**Definition 3.14.** Let $P, P_{12}, P_{34} \in \mathcal{P}$ with $s_{12} : P \overset{l_1,l_2}{\longmapsto} P_{12}$ and $s_{34} : P \overset{l_3,l_4}{\longmapsto} P_{34}$.

The reduction steps $s_{12}$ and $s_{34}$ are in *conflict*, if there is some top-level subterm $M \in \mathcal{P}$ that is reduced by both steps, i.e. such that one of the labels $\{ l_1, l_2 \}$ and one of the labels $\{ l_3, l_4 \}$ both occur in $M$.

Consider for example the labelled term $P = \overline{a}_{l_1} \mid (a_{l_2} + b_{l_3}) \mid \overline{b}_{l_4}$ and the steps $s_{12} : P \overset{l_1,l_2}{\longmapsto} \overline{b}_{l_4}$ and $s_{34} : P \overset{l_3,l_4}{\longmapsto} \overline{a}_{l_1}$. These two steps compete for the top-level subterm $a_{l_2} + b_{l_3}$. Hence $s_{12}$ and $s_{34}$ are in conflict. But in the example $Q = \overline{a}_{l_5} \mid a_{l_6} \mid b_{l_7} \mid \overline{b}_{l_8} \mid a_{l_9}$ there is no top-level subterm that is reduced in both of the steps $s_{56} : Q \overset{l_5,l_6}{\longmapsto} b_{l_7} \mid \overline{b}_{l_8} \mid a_{l_9}$ and $s_{78} : Q \overset{l_7,l_8}{\longmapsto} \overline{a}_{l_5} \mid a_{l_6} \mid a_{l_9}$. Thus, $s_{56}$ and $s_{78}$ are not in conflict. However $s_{56}$ and the step $Q \overset{l_5,l_9}{\longmapsto} a_{l_6} \mid b_{l_7} \mid \overline{b}_{l_8}$ are in conflict, because they compete for $\overline{a}_{l_5}$.

Note that if a potentially successful term can remove all possibilities to reach success in a step then there is another step that does not remove the possibility to reach success such that these two steps are in conflict.

**Lemma 3.15.** Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{mix}}, \mathcal{P}_{\mathrm{sep}}, \mathcal{P}_{\mathrm{a}} \}$. Let $Q, Q_1 \in \mathcal{P}$ such that $Q \Downarrow_{\checkmark}$, $s_1 : Q \longmapsto Q_1$, and $Q_1 \not\Downarrow_{\checkmark}$. Then there exists $Q_2 \in \mathcal{P}$ such that $s_2 : Q \longmapsto Q_2$, $Q_2 \Downarrow_{\checkmark}$, and $s_1$ and $s_2$ are in conflict.

*Proof.* We consider the labelled variants of terms and steps.

Because of $s_1 : Q \overset{l_1,l_2}{\longmapsto} Q_1$, there are two different top-level subterms $M_{11}$ and $M_{12}$ of $Q$ that are reduced in $s_1$. By Definition 3.8, $Q \Downarrow_{\checkmark}$, and $Q_1 \not\Downarrow_{\checkmark}$, all occurrences of $\checkmark$ in $Q$ are guarded and it is possible to unguard one such occurrence. Since $Q_1 \not\Downarrow_{\checkmark}$, the step $s_1$ removes all possibilities to reach success. To do so $s_1$ has to either reduce a sum such that a (guarded) summand is discarded, whose reduction does allow to reach success, or reduce a prefix, that could if it is reduced with another communication partner lead to success. In both cases the step $s_1$ removes a prefix $\pi_l$ such that $l$ is a label in $Q$ but not in $Q_1$, i.e. $l$ is a label in either $M_{11}$ or $M_{12}$, and there is another step $s_2 : Q \overset{l_3,l_4}{\longmapsto} Q_2$ such that $l \in \{ l_3, l_4 \}$ and $Q_2 \Downarrow_{\checkmark}$. Let $M_{21}$ and $M_{22}$ be the top-level subterms of $Q$ that are reduced in $s_2$. Since $l$ is a label in either $M_{11}$ or $M_{12}$ and $l \in \{ l_3, l_4 \}$, the steps $s_1$ and $s_2$ are in conflict, i.e. $\{ M_{11}, M_{12} \} \cap \{ M_{21}, M_{22} \} \neq \varnothing$. □

To detect causal dependencies we derive a partial order on the labels of an execution.

**Definition 3.16.** Let $E : P_0 \longmapsto \cdots \longmapsto P_n$ be a finite labelled execution, and let $\mathcal{L}_E$ be the set of labels that are used in $E$. The *partial order* $\leqslant_{\mathcal{L}} \subseteq \mathcal{L}_E \times \mathcal{L}_E$ *of labels in* $E$ is the smallest reflexive and transitive relation such that

1. For all subterms $\pi_l.P'$ of $P_0$ with label $l \in \mathcal{L}_E$, and for all labels $l' \in \mathcal{L}_E$ that occur in $P'$, also $l \leqslant_{\mathcal{L}} l'$.
2. For all $l_1 \leqslant_{\mathcal{L}} l_2$ with $l_1, l_2 \in \mathcal{L}_E$, and for all applications of $!P \equiv P' \mid !P$ such that $l_1$ does not occur in $P$ or $P'$ and $l_2$ labels a $\pi_{l_2}$ in $P$ whose copy $\pi_{l'_2}$ in $P'$ is labelled by $l'_2 \in \mathcal{L}_E$, also $l_1 \leqslant_{\mathcal{L}} l'_2$.
3. For all $l_1 \leqslant_{\mathcal{L}} l_2$ with $l_1, l_2 \in \mathcal{L}_E$, and for all applications of $!P \equiv P' \mid !P$ such that $l_1$ labels a $\pi_{l_1}$ in $P$ and $l_2$ labels a $\pi'_{l_2}$ in $P$ whose copies $\pi_{l'_1}$ and $\pi'_{l'_2}$ in $P'$ are labelled by $l'_1, l'_2 \in \mathcal{L}_E$, also $l'_1 \leqslant_{\mathcal{L}} l'_2$.
4. For all $l_1 \leqslant_{\mathcal{L}} l_2$ and $l_3 \leqslant_{\mathcal{L}} l_4$ with $l_1, l_2, l_3, l_4 \in \mathcal{L}_E$ such that there is a step in the execution that is labelled by $l_2, l_3$ or $l_3, l_2$, also $l_1 \leqslant_{\mathcal{L}} l_4$.

Condition 1 collects all causal dependencies in $P_0$. Since the labels of $P_i$ are derived from the labels of $P_{i-1}$ this suffices to detect all causal dependencies that directly result from the nesting of prefixes in all terms of the execution. Conditions 2 and 3 copy the known inequalities for fresh labels that result from $!P \equiv P' \mid !P$. With the last condition causal dependencies are inherited symmetrically in communication steps such that a cause for a step $s$ is also a cause for all steps of subterms that are unguarded in $s$. Moreover, note that by design for all labelled variants of an unlabelled finite execution the corresponding partial order of labels is the same modulo the corresponding bijection of labels.

Now, we may call two reduction steps causally dependent if the reduced labels are contained in $\leqslant_{\mathcal{L}}$. To capture this information, we define $L_i \leqslant_{\mathcal{L}} L_j$ if there are $l_1 \in L_1$ and $l_2 \in L_2$ with $l_1 \leqslant_{\mathcal{L}} l_2$.

**Definition 3.17 (causal (In)dependence).** Let $E : P_0 \overset{L_1}{\longmapsto} \cdots \overset{L_n}{\longmapsto} P_n$ be a finite labelled execution, let $\leqslant_{\mathcal{L}}$ be the partial order of labels in $E$, let $i, j \in \{ 1, \ldots, n \}$ such that $i \neq j$, and let $s_i : P_{i-1} \overset{L_i}{\longmapsto} P_i$ and $s_j : P_{j-1} \overset{L_j}{\longmapsto} P_j$ denote the $i$'th and the $j$'th step of $E$. Then,

— $s_j$ is *causally dependent* on $s_i$ in $E$, written $s_i \leqslant_E s_j$, if $L_i \leqslant_{\mathcal{L}} L_j$;
— $s_i$ and $s_j$ are *causally independent* if neither $s_i \leqslant_E s_j$ nor $s_j \leqslant_E s_i$.

Two steps of an unlabelled finite execution are causally dependent (independent) iff the corresponding steps of a labelled variant of this execution are causally dependent (independent).

Note that the above defined variant of causal dependencies coincides with a reduction-based variant of the definition of so-called enabling in Priami (1996) and Degano and Priami (1999), where causality is distinguished from enabling. The main difference there between these two notions is that for causality the cause of a step is inherited only by the continuation of the receiver of a step, because there is no information flow from a receiver to a sender, whereas for enabledness – as in our definition of causality – causes are inherited symmetrically by both sender and receiver. By Degano and Priami (1999), enabling coincides with causality in Boreale and Sangiorgi (1998) and Busi and Gorrieri (1995). Moreover – as already proved in Priami (1996) and Degano and Priami (1999) – the here used version of causality, where causes are inherited symmetrically, has two nice properties: (1) Two causally independent steps of an execution can be swapped and (2) if an execution has two causally independent steps, then there exists an execution in

which these two steps occur consecutively. In fact, in order to derive our separation result for the $\pi$-calculus we make use of these two conditions. Hence, we adapt the proofs of these two statements in Priami (1996) and Degano and Priami (1999) to our definition of causality with respect to reduction semantics. To prove that two consecutive steps of an execution can be swapped if they are causally independent we need to show that a step can neither consume nor unguard the subterms that are reduced in a consecutive causally independent step. Hence all subterms that are reduced in one of these two steps are already unguarded and composed in parallel before the first of the two step occurs.

**Lemma 3.18.** Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{mix}}, \mathcal{P}_{\mathrm{sep}}, \mathcal{P}_{\mathrm{a}} \}$. For all $P_1, P_2, P_3, M_1, M_2, M_3, M_4 \in \mathcal{P}$ such that $P_1 \longmapsto P_2 \longmapsto P_3$, where $s_1 : P_1 \longmapsto P_2$ reduces $M_1$ and $M_2$ and $s_2 : P_2 \longmapsto P_3$ reduces $M_3$ and $M_4$, and $s_1$ and $s_2$ are causally independent, there is $P_2' \in \mathcal{P}$ such that $P_1 \longmapsto P_2' \longmapsto P_3$, where $s_1' : P_1 \longmapsto P_2'$ reduces $M_3$ and $M_4$ and $s_2' : P_2' \longmapsto P_3$ reduces $M_1$ and $M_2$.

*Proof.* Let $P_1 \overset{L_1}{\longmapsto} P_2 \overset{L_2}{\longmapsto} P_3$ be the labelled variant of $P_1 \longmapsto P_2 \longmapsto P_3$, i.e. $s_1 : P_1 \overset{L_1}{\longmapsto} P_2$ and $s_2 : P_2 \overset{L_2}{\longmapsto} P_3$. Since $s_1$ and $s_2$ are causally independent (Definition 3.17), neither $L_1 \leqslant_{\mathcal{L}} L_2$ nor $L_2 \leqslant_{\mathcal{L}} L_1$. Since $s_1$ reduces $M_1$ and $M_2$, these two subterms are top-level in $P_1$. Similarly, $M_3$ and $M_4$ are top-level in $P_2$.

To show that also $M_3$ and $M_4$ are top-level in $P_1$ assume the opposite, i.e. assume that $M_3$ or $M_4$ are guarded in $P_1$. Then, since they are unguarded in $P_2$ they are unguarded by $s_1$. But the only prefixed actions that are removed in $s_1$ without removing their subterms are labelled by one of the two labels in $L_1$. Thus, because each of the terms $M_3$ and $M_4$ contains a label in $L_2$, and because of Condition 1 of Definition 3.16, there is $L_1 \leqslant_{\mathcal{L}} L_2$. But this inequality contradicts the assumption that $s_1$ and $s_2$ are causally independent. Hence $M_3$ and $M_4$ are unguarded and, thus, top-level in $P_1$.

If $M_3 = M_1$ then $M_3$ and, thus, one of the two labels in $L_2$ is removed in $s_1$, i.e. by Definition 3.4 this label does not occur in $P_2$ and no such label can be produced modulo structural congruence. This contradicts the assumption that $s_2$ reduces the two prefixes that are labelled by the labels in $L_2$. Repeating the same argument we have $M_3 \notin \{ M_1, M_2 \}$ and $M_4 \notin \{ M_1, M_2 \}$. Note that, because of the definition of top-level, neither $M_3$ nor $M_4$ are unguarded parts of the same sum as $M_1$ or $M_2$.

We conclude that modulo structural congruence the $M_1, \ldots, M_4$ are all unguarded and composed in parallel within $P_1$, i.e.

$$\begin{aligned} P_1 &\equiv (\nu\, \tilde{n})\left((\nu\, \tilde{n_1})\left(M_1 \mid M_2\right) \mid (\nu\, \tilde{n_2})\left(M_3 \mid M_4\right) \mid R\right) \\ P_2 &\equiv (\nu\, \tilde{n})\left(D_1 \mid (\nu\, \tilde{n_2})\left(M_3 \mid M_4\right) \mid R\right), \text{ and} \\ P_3 &\equiv (\nu\, \tilde{n})\left(D_1 \mid D_2 \mid R\right) \end{aligned}$$

for some sequences of names $\tilde{n}$, $\tilde{n_1}$, and $\tilde{n_2}$, and some $R, D_1, D_2 \in \mathcal{P}$. Thus, the steps $s_1$ and $s_2$ can be swapped, i.e. there is some $P_2' \in \mathcal{P}$ such that $P_1 \longmapsto P_2' \longmapsto P_3$, $P_2' \equiv (\nu\, \tilde{n})\left((\nu\, \tilde{n_1})\left(M_1 \mid M_2\right) \mid D_2 \mid R\right)$, $s_1' : P_1 \longmapsto P_2'$ reduces $M_3$ and $M_4$, and $s_2' : P_2' \longmapsto P_3$ reduces $M_1$ and $M_2$. □

Of course, $s_1'$ and $s_2'$ are again causally independent.

Repeating the above lemma we can turn each execution that contains two causally independent steps into an execution in which these two steps occur consecutively.

**Lemma 3.19.** Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{mix}}, \mathcal{P}_{\mathrm{sep}}, \mathcal{P}_{\mathrm{a}} \}$. For all $P, P_1, P_2, P_3, P_4, M_1, M_2, M_3, M_4 \in \mathcal{P}$ such that $P \Longmapsto P_1 \longmapsto P_2 \Longmapsto P_3 \longmapsto P_4$, where $s_1 : P_1 \longmapsto P_2$ reduces $M_1$ and $M_2$ and $s_2 : P_3 \longmapsto P_4$ reduces $M_3$ and $M_4$, and $s_1$ and $s_2$ are causally independent, there are $P_1', P_2', P_3' \in \mathcal{P}$ such that $P \Longmapsto P_1' \longmapsto P_2' \longmapsto P_4'$, where $s_1' : P_1' \longmapsto P_2'$ reduces $M_1$ and $M_2$ and $s_2' : P_2' \longmapsto P_3'$ reduces $M_3$ and $M_4$.

*Proof.* The proof is by induction on the number $n$ of steps between $s_1$ and $s_2$. If $n = 0$ then the lemma holds trivially. Otherwise, assume as inductive hypothesis that the lemma holds for any $k \leqslant n$ and consider the case of $k+1$ steps $t_1, \ldots, t_{k+1}$ between $s_1$ and $s_2$. Let $h \in \{ 1, \ldots, k+1 \}$ denote the minimal index such that $s_1$ and $t_h$ are causally independent.

We show that for all steps $t_g$ such that $g < h$ the steps $t_g$ and $t_h$ are causally independent. Consider the labelled variant of $P \Longmapsto P_1 \longmapsto P_2 \Longmapsto P_3 \longmapsto P_4$. If there is some $g \in \{ 1, \ldots, h-1 \}$ such that $t_h$ is causally dependent on $t_g$ then, by Definition 3.17, there are $l_g, l_h \in \mathcal{L}$ such that $t_g$ reduces a prefixed action labelled by $l_g$, $t_h$ reduces a prefixed action labelled by $l_h$, and $l_g \leqslant_{\mathcal{L}} l_h$. Since $h$ is the minimal index such that $s_1$ and $t_h$ are causally independent and because $g < h$, $t_g$ is causally dependent on $s_1$. Thus, there are $l_1, l_g' \in \mathcal{L}$ such that $s_1$ reduces a prefixed action labelled by $l_1$, either $l_g = l_g'$ or the other prefixed action reduced in $t_g$ is labelled by $l_g'$, and $l_1 \leqslant_{\mathcal{L}} l_g'$. In both cases, either by transitivity or by Condition 4 of Definition 3.16, $l_1 \leqslant_{\mathcal{L}} l_h$. But then, by Definition 3.17, $t_h$ is causally dependent on $s_1$ which contradicts our assumption that $s_1$ and $t_h$ are causally independent. Hence $t_g$ and $t_h$ are causally independent for all $g \in \{ 1, \ldots, h-1 \}$.

If $h = 1$, i.e. if $s_1$ and $t_h$ are consecutive steps, then, by Lemma 3.19 they can be swapped. Else if $h > 1$ then, again by Lemma 3.19, the steps $t_{h-1}$ and $t_h$ can be swapped, because the $t_{h-1}$ and $t_h$ are causally independent. Repeating this argument $h$ times we obtain an execution where $t_h$ occurs before $s_1$, i.e. where there are no more than $k$ steps between $s_1$ and $s_2$. Thus, we can conclude with the induction hypothesis. □

Note that the two lemmas above do not hold for the definition of causality in Priami (1996) and Degano and Priami (1999). Hence, our separation result between $\pi_{\mathrm{mix}}$ and $\pi_{\mathrm{sep}}$in the next section might not hold for their definition of causality but only enabling. However in $\pi_{\mathrm{a}}$ , because outputs cannot guard a process different from **0**, the two notions of causality coincide. Thus, our argumentation in the next section suffices even for the definition of causality in Priami (1996) and Degano and Priami (1999) to separate $\pi_{\mathrm{mix}}$ from $\pi_{\mathrm{a}}$ , i.e. at least the separation result between $\pi_{\mathrm{mix}}$ and $\pi_{\mathrm{a}}$ holds for both definitions of causality.

With the definition of causality in mind we define preservation of causal independence in the context of encodings. Remember that steps are often translated into sequences of steps. Hence, we lift our definition of causal independence between steps to sequences of steps.

**Definition 3.20.** Two sequences $A$ and $B$ of steps within an execution are causally independent iff each pair of a step in $A$ and a step in $B$ is causally independent.

Moreover, a sequence of steps simulating a single source term step may be interleaved with another such sequence or some other target term steps. For example, there can be target term steps used to prepare the simulation of a source term step, whose simulation may never be completed. Note that the sequences $A$ and $B$ can be interleaved within the execution. An encoding then preserves causal independence if the simulations of causally independent source term steps are causally independent.

**Definition 3.21.** An encoding preserves causal independence of source term steps iff for each simulation of a source term execution and for each pair of causally independent steps within this source term execution the simulations of these steps are causally independent.

At a first glimpse this definition might seem strict, since every pair of steps of causally independent simulations has to be causally independent. But in fact the above definition is rather relaxed, because we do not specify when a step belongs to a simulation. Hence, we still allow for pre- and post-processing steps to belong to no simulation at all. The only requirement we use in the next section is that if a step marks a conflict between two different simulations, i.e. if a step ensures that the execution simulates a source term step $s_x$ and rules out that a source term step – say $s_y$ – is simulated then this step has to belong to the simulation of $s_x$ and is, thus, no pre- or post-processing step.

### 3.5. *Encoding the synchronous Pi-calculus*

In this section, we show that no good encoding from $\pi_{\text{mix}}$ into $\pi_{\text{sep}}$preserves causal independence. Since $\pi_{\text{a}}$ is a subcalculus of $\pi_{\text{sep}}$, the same holds for any good encoding from $\pi_{\text{mix}}$ into $\pi_{\text{a}}$.

As counterexample we consider the (family of) source term(s)

$$\mathsf{P}^\star \triangleq (\overline{a} + b.P_b) \mid \left(\overline{b} + c.P_c\right) \mid (\overline{c} + d.P_d) \mid \left(\overline{d} + e.P_e\right) \mid (\overline{e} + a.P_a)$$

for $P_a, \ldots, P_e \in \{\, \mathbf{0}, \checkmark \,\}$, i.e. we consider all variants of $\mathsf{P}^\star$ that satisfy the above description and differ only in the subterms $P_a, \ldots, P_e$ which are either equal to $\mathbf{0}$ or $\checkmark$, respectively. Moreover, let $P_i'$ denote the result of a step on channel $i$ for all $i \in \{\, a, \ldots, e \,\}$, i.e. for example $P_a' = P_a \mid (\overline{b} + c.P_c) \mid (\overline{c} + d.P_d) \mid (\overline{d} + e.P_e)$. $\mathsf{n}(\mathbf{0}) = \varnothing = \mathsf{n}(\checkmark)$ and, thus, all the variants of our counterexample $\mathsf{P}^\star$ have the same free names. Because of that and because of compositionality (Definition 3.5), the encodings of different variants of $\mathsf{P}^\star$ differ in the encodings of the respective subterms $P_i$ only.

**Observation 3.22.** There exists a context $\mathcal{C}([\cdot]_1, \ldots, [\cdot]_5)$ in $\pi_{\text{sep}}$such that for all $P_a, \ldots, P_e \in \{\, \mathbf{0}, \checkmark \,\}$ we have $[\![\, \mathsf{P}^\star \,]\!] = \mathcal{C}\left([\![\, P_a \,]\!], \ldots, [\![\, P_e \,]\!]\right)$.

In particular this means, that all executions of $[\![\, \mathsf{P}^\star \,]\!]$ that do not reduce some part of $[\![\, P_a \,]\!]$, $[\![\, P_b \,]\!]$, $[\![\, P_c \,]\!]$, $[\![\, P_d \,]\!]$, or $[\![\, P_e \,]\!]$ are exactly the same for all variants of $\mathsf{P}^\star$ with $P_a, \ldots, P_e \in \{\, \mathbf{0}, \checkmark \,\}$.

Moreover, we observe that – in opposite to $\pi_{\text{mix}}$ – in $\pi_{\text{sep}}$each reduction step reduces exactly one (possibly unary) input-guarded sum and one (possibly unary) output-guarded sum. To show that there is no good and independency-preserving encoding, we show that the mixed sums of our counterexample $\mathsf{P}^\star$ cannot be translated into a separate choice

term that satisfies all the conditions of a good encoding. In particular, we derive a conflict on the type of the required sums, i.e. one such sum in the translated term has to be input- as well as output-guarded which is impossible. In order to formulate this problem, let $\mathcal{P}^{+}_{\mathrm{sep}}$ be the subset of $\mathcal{P}_{\mathrm{sep}}$-terms where the outermost operator is separate choice. Moreover, if $T_1, T_2 \in \mathcal{P}^{+}_{\mathrm{sep}}$ then let $\mathrm{t}(T_1) = \mathrm{t}(T_2)$ iff the sums $T_1$ and $T_2$ have the same type, i.e. if they are either both input-guarded or both output-guarded.

In our counterexample $\mathsf{P}^{\star}$ there are five different ways to perform a (first) step, whereby steps on neighbouring channels – as for instance $a$ and $b$ – are in conflict. Because of operational correspondence, the translation $[\![\, \mathsf{P}^{\star} \,]\!]$ has to be able to simulate all these source term steps. Moreover, because of success sensitiveness, the conflicts between the source term steps have to be translated into conflicts between the respective simulations. Here we use Observation 3.22 to switch between different variants of our counterexample $\mathsf{P}^{\star}$ and, thus, choose for all pairs of neighbouring source term steps a variant of $\mathsf{P}^{\star}$ that constitutes the conflict between these steps by a difference in the reachability of success.

**Lemma 3.23.** Any good encoding $[\![ \cdot ]\!]$ from $\pi_{\mathrm{mix}}$ into $\pi_{\mathrm{sep}}$ has to translate the conflicts in $\mathsf{P}^{\star}$ into conflicts of some corresponding simulations, i.e. for all pairs $(i,j) \in \{(a,b),(b,c),(c,d),(d,e),(e,a)\}$ there are $T, T_i, T_j \in \mathcal{P}_{\mathrm{sep}}$ and there are $M^{ij}, M^{ij}_i, M^{ij}_j \in \mathcal{P}^{+}_{\mathrm{sep}}$ such that

$$[\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto T \longmapsto T_i \Longmapsto\asymp [\![\, P'_i \,]\!], \tag{$S_i$}$$

$$[\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto T \longmapsto T_j \Longmapsto\asymp [\![\, P'_j \,]\!], \tag{$S_j$}$$

$\mathrm{t}(M^{ij}) \neq \mathrm{t}(M^{ij}_i) = \mathrm{t}(M^{ij}_j)$, the step $t_i : T \longmapsto T_i$ reduces $M^{ij}$ and $M^{ij}_i$, and the step $t_j : T \longmapsto T_j$ reduces $M^{ij}$ and $M^{ij}_j$.

*Proof.* The source term $\mathsf{P}^{\star}$ has the five different executions $\mathsf{P}^{\star} \longmapsto P'_i$ with $i \in \{\, a, \ldots, e \,\}$. Each execution consists of a single step, which we denote by $s_i$. By operational completeness (Definition 3.6), all these source term steps have to be simulated modulo $\asymp$, i.e. for all $i \in \{\, a, \ldots, e \,\}$ there is a simulation $S_i : [\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto\asymp [\![\, P'_i \,]\!]$. Because all executions of $\mathsf{P}^{\star}$ are finite and because of divergence reflection (Definition 3.7), all executions of $[\![\, \mathsf{P}^{\star} \,]\!]$ are finite. Let us fix $i$ and $j$ such that $(i,j) \in \{\, (a,b),(b,c),(c,d),(d,e),(e,a) \,\}$. Thus, there are $T_1, T_2 \in \mathcal{P}_{\mathrm{sep}}$ such that $S_i : [\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto T_1 \asymp [\![\, P'_i \,]\!]$ and $S_j : [\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto T_2 \asymp [\![\, P'_j \,]\!]$.

Consider the case $P_j = \checkmark$ and $P_i = P_k = \mathbf{0}$ for all $k \in \big(\{\, a, \ldots, e \,\} \setminus \{\, j \,\}\big)$. Then $P'_j \Downarrow_{\checkmark!}$ and $P'_i \not\Downarrow_{\checkmark}$. Because of success sensitiveness (Definition 3.9) and Lemma 3.13, $P'_i \not\Downarrow_{\checkmark}$ implies $[\![\, P'_i \,]\!] \not\Downarrow_{\checkmark}$ and $P'_j \Downarrow_{\checkmark!}$ implies $[\![\, P'_j \,]\!] \Downarrow_{\checkmark!}$. Since $\asymp$ respects success (Definition 3.10) and by Lemma 3.12, we have $[\![\, P'_i \,]\!] \asymp T_1 \not\asymp T_2 \asymp [\![\, P'_j \,]\!]$, $T_1 \not\Downarrow_{\checkmark}$, and $T_2 \Downarrow_{\checkmark!}$. Thus, $[\![\, \mathsf{P}^{\star} \,]\!] \Downarrow_{\checkmark}$ and there are $T, T_i \in \mathcal{P}_{\mathrm{sep}}$ such that $S_i : [\![\, \mathsf{P}^{\star} \,]\!] \Longmapsto T \longmapsto T_i \Longmapsto T_1 \asymp [\![\, P'_i \,]\!]$ with $t_i : T \longmapsto T_i$, $T \Downarrow_{\checkmark}$, and $T_i \not\Downarrow_{\checkmark}$. By Lemma 3.15, then there is $T_j \in \mathcal{P}_{\mathrm{sep}}$ such that $t_j : T \longmapsto T_j$, $T_j \Downarrow_{\checkmark}$, and $t_i$ and $t_j$ are in conflict.

Because $T \in \mathcal{P}_{\mathrm{sep}}$, the step $t_i$ reduces exactly one input- and one output-guarded sum. Since $t_i$ and $t_j$ are in conflict, $t_j$ either reduces the same input-guarded or the same output-guarded or the same two sums. If these steps reduce the same two sums, choose $M^{ij}$ as one of these sums and let $M^{ij}_i = M^{ij}_j$ be the respective other sum. Else (if $t_i$ and $t_j$ reduce three different sums), let $M^{ij}$ be the sum that is reduced in both steps, let $M^{ij}_i$ be

the respective other sum that is reduced in $t_i$, and $M_j^{ij}$ be the respective other sum that is reduced in $t_j$. In both cases we have $\mathrm{t}(M^{ij}) \neq \mathrm{t}(M_i^{ij}) = \mathrm{t}(M_j^{ij})$.

Since $T_1 \not\Downarrow_{\checkmark}$ and $T_2 \Downarrow_{\checkmark!}$, the conflict between $S_i$ and $S_j$ is such that all possible ways to reach success have to be eliminated before $T_1$ is reached and all possible ways that do not lead to success have to be eliminated before $T_2$ is reached. Since $[\![ \checkmark ]\!] \Downarrow_{\checkmark!}$ and $[\![ \mathbf{0} ]\!] \not\Downarrow_{\checkmark}$ and by compositionality, the continuations $[\![ P_i ]\!]$ and $[\![ P_j ]\!]$ cannot be unguarded before the conflict between $t_i$ and $t_j$ is ruled out, i.e. $[\![ P_i ]\!]$ and $[\![ P_j ]\!]$ are guarded in $T$. Of course before the first such conflict between two simulations is ruled out all continuations $[\![ P_x ]\!]$ with $x \in \{ a, \ldots, e \}$ are guarded. Thus, by Observation 3.22, for each $(i,j) \in \{ (a,b), (b,c), (c,d), (d,e), (e,a) \}$ and all $P_a, \ldots, P_e \in \{ \mathbf{0}, \checkmark \}$ there are $T, T_i, T_j \in \mathcal{P}_{\mathrm{sep}}$ and $M^{ij}, M_i^{ij}, M_j^{ij} \in \mathcal{P}_{\mathrm{sep}}^{+}$ as required. □

Adding the preservation of causal independence to the requirements of a good encoding we derive a contradiction on the requirements for the sums $M^{ij}, M_i^{ij}, M_j^{ij} \in \mathcal{P}_{\mathrm{sep}}^{+}$: Every maximal execution of our counterexample $\mathsf{P}^{\star}$ contains two causally independent steps. Thus, for any good and independence-preserving translation of $\mathsf{P}^{\star}$ every maximal execution contains two causally independent simulations of source term steps, each containing a step that is in conflict with the simulations of the respective neighbouring source term steps. Hence each maximal execution of $[\![ \mathsf{P}^{\star} ]\!]$ has two causally independent steps each reducing some $M^{ij}$ and some $M_i^{ij}$ or $M_j^{ij}$. The side conditions on these sums that result from the conflicts as described in Lemma 3.23 and from the causal independence of these steps leads to a contradiction with the constraints on the type $\mathrm{t}(\cdot)$ of these sums.

**Theorem 3.24.** No good encoding $[\![ \cdot ]\!]$ from $\pi_{\mathrm{mix}}$ into $\pi_{\mathrm{sep}}$ preserves causal independence.

*Proof.* Assume the contrary, i.e. assume that there is a good and independence-preserving encoding $[\![ \cdot ]\!]$ from $\pi_{\mathrm{mix}}$ into $\pi_{\mathrm{sep}}$.

$\mathsf{P}^{\star}$ is such that each maximal execution of $\mathsf{P}^{\star}$ consists of exactly two steps that are causally independent as for example $\mathsf{P}^{\star} \longmapsto P'_a \longmapsto P_a \mid P_c \mid (\overline{d} + e.P_e)$. By Definition 3.21, an encoding $[\![ \cdot ]\!]$ that preserves causal independence ensures that the simulation of each such execution contains two causally independent sequences of steps representing the respective simulations of the two source term steps, i.e. each maximal execution of $[\![ \mathsf{P}^{\star} ]\!]$ contains two causally independent simulations of two causally independent source term steps. These two sequences can be interleaved, but by Lemma 3.23 they cannot be empty and within each maximal execution each pair of steps of different simulations is causally independent. By Lemma 3.23 there are $M^{ab}, M_a^{ab}, M_b^{ab}, \ldots, M^{ea}, M_e^{ea}, M_a^{ea} \in \mathcal{P}_{\mathrm{sep}}^{+}$ such that

$$\forall \{ i,j \} \in \{ \{ a,b \}, \{ b,c \}, \{ c,d \}, \{ d,e \}, \{ e,a \} \}. \quad \mathrm{t}\left(M^{ij}\right) \neq \mathrm{t}\left(M_i^{ij}\right) = \mathrm{t}\left(M_j^{ij}\right) \tag{1}$$

and each maximal execution reduces the sums corresponding to two causally independent simulations. Note that we consider labelled variants of terms here and that for all $i,j \in \{ a,b,c,d,e \}$ the reduction of $M^{ij}$ and $M_i^{ij}$ (or $M^{ij}$ and $M_j^{ij}$) belong to the simulation of the source term step $s_i$ (or $s_j$) on channel $i$ (or $j$), because these steps mark which source term step is simulated.

Hence each maximal execution of $[\![ \mathsf{P}^\star ]\!]$ has (at least) two causally independent steps such that one step reduces $M^{ij}, M_x^{ij}$ and the other step reduces $M^{kl}, M_y^{kl}$, where $i,j,k,l \in \{ a,\ldots,e \}$, $x \in \{ i,j \}$, and $y \in \{ k,l \}$. For example in the simulation of $\mathsf{P}^\star \longmapsto\!\!\!\!\rightarrow P_a \mid P_c \mid (\overline{d} + e.P_e)$ we have $(M^{ij}, M_x^{ij}) \in \{ (M^{ab}, M_a^{ab}), (M^{ea}, M_a^{ea}) \}$ and $(M^{kl}, M_y^{kl}) \in \{ (M^{bc}, M_c^{bc}), (M^{cd}, M_c^{cd}) \}$, or vice versa. By Lemma 3.19, there are executions such that these two steps are consecutive and, by Lemma 3.18, in such executions these two steps can occur in either order. Thus, there are $T, T_1, T_1', T_2 \in \mathcal{P}_{\text{sep}}$ such that $[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_1 \longmapsto T_2$ and $[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_1' \longmapsto T_2$, where $T \longmapsto T_1$ and $T_1' \longmapsto T_2$ reduce $M^{ij}$ and $M_x^{ij}$, and $T \longmapsto T_1'$ and $T_1 \longmapsto T_2$ reduce $M^{kl}$ and $M_y^{kl}$. Thus $M^{ij}, M_x^{ij}, M^{kl}$, and $M_y^{kl}$ are unguarded in $T$. By Definition 3.4, the step $T \longmapsto T_1$ cannot consume the unguarded instance of the labelled term $M^{kl}$ and unguard another instance with the same labels. Hence $M^{kl} \notin \{ M^{ij}, M_x^{ij} \}$. Analogously, $M_y^{kl} \notin \{ M^{ij}, M_x^{ij} \}$ and $M^{ij}, M_x^{ij} \notin \{ M^{kl}, M_y^{kl} \}$.

$$\begin{aligned} &\forall (i,j),(k,l) \in \{ (a,b),(b,c),(c,d),(d,e),(e,a) \} \,.\, \forall x \in \{ i,j \} \,. \\ &\quad \forall y \in \{ k,l \} \,.\, \{ x,y \} \in \{ \{a,c\},\{a,d\},\{b,d\},\{b,e\},\{c,e\} \} \implies \\ &\qquad M^{kl}, M_y^{kl} \notin \{ M^{ij}, M_x^{ij} \} \wedge M^{ij}, M_x^{ij} \notin \{ M^{kl}, M_y^{kl} \}. \end{aligned} \tag{2}$$

Next let $i,j,k \in \{ a,\ldots,e \}$ be such that the steps on $i$ and $k$ are causally independent in $\mathsf{P}^\star$ but the step on $j$ is in conflict with both the step on $i$ as well as the step on $k$, i.e. $(i,j,k) \in \{ (a,b,c),(b,c,d),(c,d,e),(d,e,a),(e,a,b) \}$. Let $s_i, s_j, s_k$ denote the respective steps on channel $i,j,k$. By Lemma 3.18, Lemma 3.19, and Lemma 3.23, there are (at least) two maximal executions containing the causally independent simulations of $s_i$ and $s_k$ and all such executions are in conflict with all executions containing the simulation of $s_j$. Thus, there are $T, T_{i,1}, T_{i,2}, T_{j,1}, T_{j,2}, T_{k,1}, T_{k,2} \in \mathcal{P}_{\text{sep}}$ and $M^{ij}, M_i^{ij}, M_j^{ij}, M^{jk}, M_j^{jk}, M_k^{jk} \in \mathcal{P}_{\text{sep}}^+$ such that

$$\begin{aligned} &[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_{i,1} \longmapsto T_{k,1} \Longmapsto\!\!\!\!\!\!\rightarrow, \\ &[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_{j,1} \Longrightarrow, \\ &[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_{k,2} \longmapsto T_{i,2} \Longmapsto\!\!\!\!\!\!\rightarrow, \\ &[\![ \mathsf{P}^\star ]\!] \Longrightarrow T \longmapsto T_{j,2} \Longrightarrow, \end{aligned}$$

$\mathrm{t}(M^{ij}) \neq \mathrm{t}(M_i^{ij}) = \mathrm{t}(M_j^{ij})$, $\mathrm{t}(M^{jk}) \neq \mathrm{t}(M_j^{jk}) = \mathrm{t}(M_k^{jk})$, $T \longmapsto T_{i,1}$ and $T_{k,2} \longmapsto T_{i,2}$ reduce $M^{ij}$ and $M_i^{ij}$, $T \longmapsto T_{j,1}$ reduces $M^{ij}$ and $M_j^{ij}$, $T \longmapsto T_{k,2}$ and $T_{i,1} \longmapsto T_{k,1}$ reduce $M^{jk}$ and $M_k^{jk}$, and $T \longmapsto T_{j,2}$ reduces $M^{jk}$ and $M_j^{jk}$. Then $T$ has an unguarded instance of $M_j^{jk}$ but $T_{i,1}$ cannot have such an unguarded instance, because the conflict to the simulation of $s_j$ is already ruled out in $T \longmapsto T_{i,1}$. Hence $T \longmapsto T_{i,1}$ removes $M_j^{jk}$. Because $T \longmapsto T_{i,1}$ reduces only $M^{ij}$ and $M_i^{ij}$, we conclude that $M_j^{jk} \in \{ M^{ij}, M_i^{ij} \}$. Similarly, $M_j^{ij} \in \{ M^{jk}, M_k^{jk} \}$

$$\begin{aligned} &\forall (i,j,k) \in \{ (a,b,c),(b,c,d),(c,d,e),(d,e,a),(e,a,b) \} \,. \\ &\quad M_j^{jk} \in \left\{ M^{ij}, M_i^{ij} \right\} \wedge M_j^{ij} \in \left\{ M^{jk}, M_k^{jk} \right\}. \end{aligned} \tag{3}$$

From the Conditions 1 and 3 we obtain

$$\forall (i,j,k) \in \{ (a,b,c),(b,c,d),(c,d,e),(d,e,a),(e,a,b) \}. \\ \left(M_j^{jk} = M^{ij} \wedge M_j^{ij} = M^{jk}\right) \vee \left(M_j^{jk} = M_i^{ij} \wedge M_j^{ij} = M_k^{jk}\right). \tag{4}$$

Consider $(i,j,k,l) \in \{ (a,b,c,d),(b,c,d,e),(c,d,e,a),(d,e,a,b),(e,a,b,c) \}$.

Then, by applying Condition 4 twice, we have $M_j^{jk} = M^{ij} \wedge M_j^{ij} = M^{jk}$ or $M_j^{jk} = M_i^{ij} \wedge M_j^{ij} = M_k^{jk}$ as well as $M_k^{kl} = M^{jk} \wedge M_k^{jk} = M^{kl}$ or $M_k^{kl} = M_j^{jk} \wedge M_k^{jk} = M_l^{kl}$. Of the resulting four cases only one remains, because the other cases lead to $M^{ij} = M_k^{kl}$, $M_j^{ij} = M^{kl}$, or $M_i^{ij} = M_k^{kl}$ and, thus, violate Condition 2

$$\forall (i,j,k,l) \in \{ (a,b,c,d),(b,c,d,e),(c,d,e,a),(d,e,a,b),(e,a,b,c) \}. \\ M_j^{jk} = M^{ij} \wedge M_j^{ij} = M^{jk} = M_k^{kl} \wedge M_k^{jk} = M^{kl}. \tag{5}$$

From Condition 5 we derive, by checking all possible assignments for $i,j,k,l$, that $M^{ab} = M_a^{ea} = M_b^{bc}$, $M^{bc} = M_b^{ab} = M_c^{cd}$, $M^{cd} = M_c^{bc} = M_d^{de}$, $M^{de} = M_d^{cd} = M_e^{ea}$, and $M^{ea} = M_e^{de} = M_a^{ab}$.[†] But then, by Condition 1, we derive the contradiction $M_b^{ab} = M^{bc} \wedge M_a^{ab} = M^{ea} \implies \mathrm{t}(M^{bc}) = \mathrm{t}(M^{ea})$ but $M_c^{cd} = M^{bc} \wedge M^{de} = M_d^{cd} \wedge M_e^{de} = M^{ea} \implies \mathrm{t}(M^{bc}) \neq \mathrm{t}(M^{ea})$. □

## 4. Summary

In Section 2, we proved that it is in general impossible to find for a finite, 1-safe net a distributed, completed pomset trace equivalent, finite, 1-safe net. Then in Section 3 we proved that no good encoding from the synchronous $\pi$-calculus with mixed choice into the asynchronous $\pi$-calculus can preserves causal independence. In summary, both within Petri nets and within the $\pi$-calculus it is in general impossible to translate a synchronous system into a purely asynchronous one without introducing additional causal dependencies. We believe that the existence of these results in two fundamentally different concurrency formalisms reveals a general phenomenon. The similarities between the results as outlined below, in particular in the structure of the counterexamples used, point to a general problem when trying to implement synchronization or distributed choices under preservation of the causal structure.

### 4.1. *Comparison of the two results*

Let us have a look at the different criteria that are used to show the two separation results. In Petri nets we require a good implementation to be distributed, finite, complete pomset trace equivalent, and thereby also divergence-free. An encoding between two $\pi$-calculi is considered good if it satisfies compositionality, operational correspondence, success sensitiveness, and divergence reflection.

Note that the first criterion on the $\pi$-calculus side, i.e. the structural criterion compositionality, basically ensures that the encoding is of practical use. In the case of a

[†] Note that in our counterexample $\mathsf{P}^\star$ we can indeed choose $\overline{a} + b.P_b = M^{ab}$, $\overline{b} + c.P_c = M^{bc}$, $\overline{c} + d.P_d = M^{cd}$, $\overline{d} + e.P_e = M^{de}$, and $\overline{e} + a.P_a = M^{ea}$ such that theses equations hold.

non-compositional encoding, it is often already hard to write the encoding down and even more difficult to implement it. Multi-level encodings, where each level represents a compositional encoding, are an exception. However, in that case it is possible to implement a global coordinator like the one in the centralized solution of the Petri net implementation in Figure 3 (compare also to the discussion of multi-level encodings in Gorla (2010)). So, compositionality rules out encodings that solve all conflicts by centralization. Similarly, in the case of Petri nets, it is required that the implementation net is distributed and transitions firing in the source net can also fire in parallel in the implementation. Note that the second requirement is ensured by the requirement that the original net and its implementation are completed pomset trace equivalent. We also observe that compositionality rules out the case that the encoding function simply translates every possible execution into a sequential process, which is the only reason to forbid infinite implementations in the Petri net setting.

Operational correspondence in combination with success sensitiveness ensures that, in the case of the $\pi$-calculus, the source term and its translation have the same abstract behaviour. Note that these criteria – similar to completed pomset trace equivalence – also forbid the introduction of (local) deadlocks not present in the source. By these two criteria, the source and the target term are at least equated by some kind of testing equivalence. The equivalence $\asymp$, which is assumed on the target language, influences how strict the abstract behaviours of the source and the target terms have to coincide. Therefore, we do not presume any requirements on $\asymp$ except that it is a reduction bisimulation. Similarly, in the case of Petri nets, with completed pomset trace equivalence we choose one of the weakest behavioural equivalences which is sensitive to local deadlocks, divergence, causal independence, and amount of parallelism.

The most obvious difference between the two separation results is presumably the different expressivity of the source languages. Petri nets are, contrary to the synchronous $\pi$-calculus, not Turing-complete. This makes the similarities between both results even more surprising. It turns out that in both results the critical structure – if represented as a Petri net[†] – is a more complex variant of the pure **M** of Figure 1. In both cases, the counterexample refers to a situation in the synchronous setting in which there are two causally independent Petri net transitions or $\pi$-calculus steps that are both in conflict to a third one. To mimic this behaviour, the Petri net implementation as well as the $\pi$-calculus encoding have to introduce a causal dependence that is not present in the source. In Peters (2012) a good encoding from $\pi_{\text{mix}}$ into $\pi_{\text{a}}$ (with matching) is presented and it is proved that this encoding satisfies the criteria of Section 3.3. Thus, our separation result indeed results from the additional criterion on the preservation of causal independence. Note that also the notions of causality used to derive these two results are comparable, because the two conditions on causally independent steps – (1) two consecutive causally independent steps can be swapped and (2) for each execution with two causally independent steps there is some execution in which these steps are consecutive – used to derive the result for the $\pi$-calculus hold similarly for transitions in Petri nets.

[†] In Peters *et al.* (2013), we show that the counterexample of Section 3.5 corresponds to a Petri net that looks like a star and consists of three overlapping pure **M**'s.

However, apart from these apparent similarities, the relation between the two results leaves us with a number of open problems and gives possible directions of future research. To begin with, the requirements imposed on Petri net implementations and $\pi$-calculus encodings take rather different forms. Note that the $\pi$-calculus setting seems to utilize stronger requirements on the structure of an encoding function, while the Petri net setting requires a stronger connection between source and target by means of an equivalence. One possibility to obtain the same preconditions for both proofs, would be to augment Petri nets with a notion of success and then to apply the criteria of Gorla to the Petri net setting. Similarly, we could examine what kind of equivalence between source and target terms is induced by the combination of the five criteria of Gorla and compare this equivalence with completed pomset trace equivalence. However, we believe that both modifications would not change the main statement of our proofs. Also, in contrast to the $\pi$-calculus, the Petri nets considered here 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.

### 4.2. *Related work*

4.2.1. *In Petri nets.* A review of existing literature in related areas of Petri nets research can be found in Glabbeek *et al.* (2008), nonetheless we wish to refer the reader explicitly to Hopkins (1991), where instead of requiring the equivalence between specification and implementation to preserve parallelism, more structural resemblance of the implementation to the specification is required.

A paper not covered by Glabbeek *et al.* (2008) is Badouel *et al.* (2002), where an algorithm for the automated synthesis of distributed implementations of protocols is presented. The notion of distributed Petri nets employed therein differs from ours by not requiring formally that no parallelism may occur on the same location. The authors, however, finally generate a finite automaton for each location, again serializing all actions on a single location. In contrast to the present paper and similar to Hopkins (1991), the authors start with a user-supplied map from events to locations, and answer the concrete problem of whether that specific distribution is realizable or not instead of requiring the maximal possible parallelism to be realized.

The present paper adds another patch to the emerging map of the separation plane between those equivalences from the spectrum of behavioural equivalences which allow asynchronous Petri net implementation in general and those which do not. Glabbeek *et al.* (2008) show that Petri nets cannot in general be implemented up to step readiness equivalence, thereby giving an upper bound for distributability along the branching-time dimension. The present paper provided an upper bound on the dimension of causality. We did not formally proof that this bound is tight, and one might imagine that a behavioural equivalence closer to the notion of dependency markings exists. However, we were unable to find an equivalence which is sensitive to the local deadlock problem outlined in Figure 4 and is not based on processes. The implementation by Schicke (2009) can serve as a lower bound on both dimensions. It would be interesting to answer the implementability

question for systems which feature real-valued time, thereby enabling timeout detection and simultaneous action without co-locality.

The question up to which behavioural equivalence *general* Petri nets are implementable can also be reversed into the question what properties or substructures of a Petri net make it unimplementable. One problematic structure for causal equivalences, identified in this paper, is the net of Figure 2, possibly with a more elaborate route from $a$ and $c$ back to the marking enabling all three transitions. We did not prove that no fundamentally different problematic structures exists, but we conjecture that this is indeed the case.

4.2.2. *In the $\pi$-calculus.* As already mentioned in the introduction of Section 3 the expressive power of mixed choice in the $\pi$-calculus is already analysed in Palamidessi (2003), Nestmann (2000), Gorla (2010), Peters and Nestmann (2010), Peters and Nestmann (2012), Peters (2012) and Peters *et al.* (2013), but to the best of our knowledge this is the first investigation of encodings from $\pi_{\text{mix}}$ into $\pi_{\text{sep}}$or $\pi_{\text{a}}$ with respect to the preservation of causal independencies.

In Peters *et al.* (2013) and Peters (2012), the same counterexample and a similar proof technique as in Section 3.5 is used to show that no good encoding between $\pi_{\text{mix}}$ and ($\pi_{\text{sep}}$ or) $\pi_{\text{a}}$ preserves distributability. Moreover, it is shown how easily the proofs there can be adapted to show similar separation results between other calculi. Similarly we can adapt the above proofs to show that there is no good and causality preserving encoding from $\pi_{\text{a}}$ into the Join Calculus – using the counterexample that transferred into a Petri net has the shape of a pure **M** of Peters *et al.* (2013) – or how a similar result can be proved for action-guarded variants of CSP.

### 4.3. *Conclusion*

In comparison, although we consider two fundamentally different formalisms of concurrency and apply quite different requirements and notions of a good encoding or implementation, we obtain surprisingly similar results. In both settings we have shown that it is not always possible to implement synchronous interactions within a purely asynchronous setting without the introduction of additional causal dependencies. Hence, this connection between choices, synchronous interactions and causal dependencies is very likely not an artefact of the representation of concurrent systems in either Petri nets or the $\pi$-calculus, but rather a general phenomenon.

## Acknowledgements

We sincerely thank the anonymous referees for their constructive comments. We are particularly grateful for discussions with and technical hints by Daniele Gorla.

## References

Badouel, É., Caillaud, B. and Darondeau, P. (2002) Distributing finite automata through petri net synthesis. *Formal Aspects of Computing* **13** 447–470.

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

Boudol, G. (1992) Asynchrony and the $\pi$-calculus (note). INRIA Research Report RR-1702.

Busi, N. and Gorrieri, R. (1995) A Petri net semantics for $\pi$-calculus. In: *Proceedings of the 6th International Conference on Concurrency Theory*, *Lecture Notes in Computer Science* **962** 145–159.

Carbone, M. and Maffeis, S. (2003) On the expressive power of polyadic synchronisation in $\pi$-calculus. *Nordic Journal of Computing* **10** 1–29.

Degano, P. and Priami, C. (1999) Non-interleaving semantics for mobile processes. *Theoretical Computer Science* **216** (1) 237–270.

Fournet, C. and Gonthier, G. (1996) The reflexive chemical abstract machine and the join-calculus. In: *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 372–385.

van Glabbeek, R. J. (1993) The linear time - branching time spectrum II. In: *Proceedings of the 4th International Conference on Concurrency Theory,* 66–81.

van Glabbeek, R. J. and Goltz, U. (2001) Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica* **37** (4/5) 229–327.

van Glabbeek, R. J., Goltz, U. and Schicke, J.-W. (2008) On synchronous and asynchronous interaction in distributed systems. *Technical Report 2008-04*, TU Braunschweig. Extended abstract in *Proceedings of the 33rd international symposium on Mathematical Foundations of Computer Science*, *Lecture Notes in Computer Science* **5162** 16–35. (Available at http://arxiv.org/abs/0901.0048v1.)

van Glabbeek, R. J., Goltz, U. and Schicke-Uffmann, J.-W. (2012) On distributability of Petri nets (extended abstract). In: *Proceedings of the 15th International Conference on Foundations of Software Science and Computation Structures*, *Lecture Notes in Computer Science* **7213** 331–345.

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

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

Honda, K. (1992) Notes on soundness of a mapping from $\pi$-calculus to $\nu$-calculus. With comments added in October 1993.

Honda, K. and Tokoro, M. (1991) An object calculus for asynchronous communication. *Proceedings of the European Conference on Object-Oriented Programming*, *Lecture Notes in Computer Science* **512** 133–147.

Hopkins, R. P. (1991) Distributable nets. In: Advances in Petri Nets. *Lecture Notes in Computer Science* **524** 161–187.

Milner, R. and Sangiorgi, D. (1992) Barbed bisimulation. In: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* **623** 685–695.

Nestmann, U. (1998) On the expressive power of joint input. In: *Proceedings of the Fifth International Workshop on Expressiveness in Concurrency*, *Electronic Notes in Theoretical Computer Science* **16** 145–152.

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

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

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

Peters, K. (2012) *Translational Expressiveness—Comparing Process Calculi using Encodings*, Ph.D. thesis, TU Berlin. (Available at: http://nbn-resolving.de/urn:nbn:de:kobv:83-opus-37495)

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

Peters, K. and Nestmann, U. (2012) Is it a 'Good' encoding of mixed choice? In: *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures. Lecture Notes in Computer Science* **7213** 210–224.

Peters, K., Nestmann, U. and Goltz, U. (2013) On distributability in process calculi. In: *Proceedings of the 22nd European conference on Programming Languages and Systems. Lecture Notes in Computer Science* 310–329.

Petri, C. A. (1977) Non-sequential processes. *GMD-ISF Report 77.05*, GMD.

Pratt, V. R. (1985) The pomset model of parallel processes: Unifying the temporal and the spatial. In: *Seminar on Concurrency*, Carnegie-Mellon University 180–196.

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

Reisig, W. (1984) Partial order semantics versus interleaving semantics for CSP-like languages and its impact on fairness. In: *Proceedings of the 11th Colloquium on Automata, Languages and Programming* 403–413.

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

Schicke, J.-W. (2008) Studienarbeit: Asynchronous Petri net classes. TU Braunschweig.

Schicke, J.-W. (2009) Diplomarbeit: Synchrony and asynchrony in Petri nets. TU Braunschweig.