Synchrony vs. Causality in Asynchronous Petri Nets[*]

Jens-Wolfhard Schicke
Institute for Programming and Reactive Systems, TU Braunschweig, Germany
drahflow@gmx.de

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

Ursula Goltz
Institute for Programming and Reactive Systems, TU Braunschweig, Germany
goltz@ips.cs.tu-bs.de

Given a synchronous system, we study the question whether the behaviour of that system can be exhibited by a (non-trivially) distributed and hence asynchronous implementation. In this paper we show, by counterexample, that synchronous systems cannot in general be implemented in an asynchronous fashion without either introducing an infinite implementation or changing the causal structure of the system behaviour.

keywords: asynchrony, distributed systems, causal semantics, Petri nets

1 Introduction

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) asynchronous implementation executed on multiple processing units in parallel. We consider the question under which circumstances such an approach is applicable, or equivalently, what restrictions must be placed on the synchronous design in order that it may be simulated asynchronously.

We formalise this problem by means of Petri nets (Section 2), a semi-structural requirement (Section 3) on Petri nets to enforce asynchrony in the implementation, and an equivalence relation (Section 4) on possible Petri net behaviours to decide whether a candidate implementation is indeed faithful to the synchronous specification.

Countless equivalence relations for system behaviour have already been proposed. When comparing the strictness of these equivalences, as done in [2] or [3], and exploring the resulting lattice, one finds multiple "dimensions" of features along which such an equivalence may be more or less discriminating. The most 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 to 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 τ-transitions, a suitable equivalence must abstract from them, as long as they do not allow a divergence.

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

B. Luttik and F. D. Valencia (Eds.): 18th International Workshop on
Expressiveness in Concurrency (EXPRESS 2011)
EPTCS 64, 2011, pp. 119–131, doi:10.4204/EPTCS.64.9

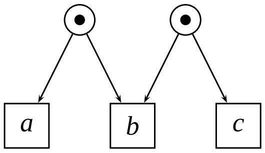

Figure 1: A fully reached, pure **M**, the problematic structure from [4]

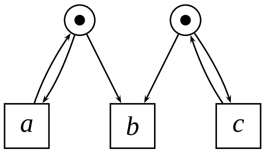

Figure 2: A repeated pure **M**. A finite, 1-safe, undistributable net used as a running counterexample.

[4] 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 characterisation 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 [4] a synchronous net is distributable only if it contains no fully reachable pure **M**. The other direction needed for exactness has not been published yet, as the only as of yet existing proofs utilises an infinite implementation.

Using the strictly weaker completed step trace equivalence, [10] 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.

In this paper we show that this drawback holds for any sensible encoding of synchronous interactions, i.e., it is a general phenomenon of encoding synchrony. 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 [4] along a different, namely the causal, dimension of the spectrum of behavioural equivalences.

We introduce basic Petri net concepts in Section 2, then turn to recounting the definition of distributability in Section 3. Afterwards we introduce completed pomset trace equivalence in Section 4, justify it by means of illustrative examples, and use it in Section 5 to prove the impossibility of implementing general Petri nets while respecting causality. Finally Section 6 concludes.

2 Basic Notions

Most material in this section has been taken verbatim or with minimal adaptation from [4] or [10]. 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 1. 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.

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 utilise).

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. Let $N = (S, T, F, M_0, \ell)$ be a net. Let $M_1, M_2 \subseteq S \times \mathcal{P}(\mathrm{Act})$.
$G \subseteq T, G \neq \varnothing$, 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.

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

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

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

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

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

Applying 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. We will however 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 $P \cap Q = \varnothing$.

To simplify the following argumentation we use some abbreviations. $\xrightarrow{\mu}_N$ denotes a labelled step on a single transition labelled μ. $\overset{a}{\Longrightarrow}_N$ denotes a step on a surrounded by arbitrary τ-steps, i.e., $\Longrightarrow_N$ abstracts from τ-steps.

Definition 3. Let $N = (S,T,F,M_0,\ell)$ be a labelled net.

We extend the labelling function ℓ 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
$$M_1 \overset{a_1 a_2 \cdots a_n}{\Longrightarrow}_N M_2 \Leftrightarrow M_1 \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_1\}}_N \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_2\}}_N \xrightarrow{\tau}{}^*_N \cdots \xrightarrow{\tau}{}^*_N \xrightarrow{\{a_n\}}_N \xrightarrow{\tau}{}^*_N M_2$$

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 $M_1[G\rangle_N$ 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 $M_0 \times \{\emptyset\} \overset{\sigma}{\Longrightarrow}_N M_1$. 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 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 ${}^\bullet t \cap {}^\bullet u = \varnothing$.

3 Distributed Nets

After having introduced Petri nets in general, we still need to find a notion of such a net being distributed before being able to answer the question of distributed implementability. A straightforward approach is to 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 will regard locations as sequential execution 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

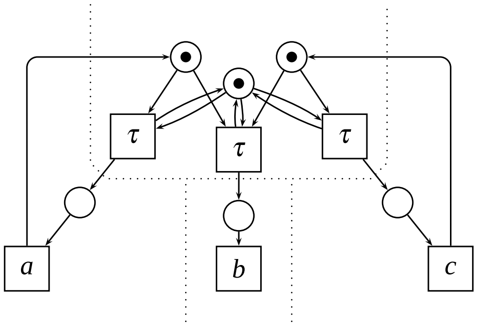

Figure 3: A centralised implementation of Figure 2, location borders dotted.

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 [4], whence the following description and definition of the present version have been derived from. 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.

We model the association of locations to the places and transitions in a net $N = (S,T,F,M_0,\ell)$ as a function $D: S \cup T \to \text{Loc}$, 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 4. 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 straightforward to give a semi-structural[1] characterisation of this class of nets:

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

4 Completed Pomset Trace

We now motivate the equivalence relation used for the rest of the paper by means of highlighting some possible shortcomings of implementations one would intuitively like to avoid.

When trying to implement a synchronous Petri net by a distributed one, one of the easiest approaches is central serialisation 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

[1] mainly structural, but with a reachability side-condition

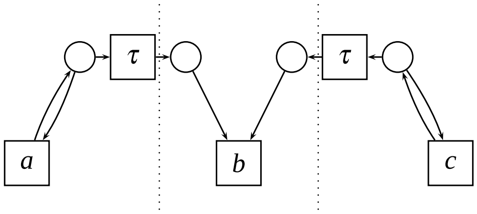

Figure 4: A locally deadlocking implementation of Figure 2, location borders dotted.

the implementation, to the repeated pure **M**. Nonetheless, this approach is intuitively not scalable, 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 trough 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 τ-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 [10], called completed pomset trace equivalence. It extends the pomset trace equivalence of [8] as to detect local deadlocks, which can be regarded as unjust executions in the sense of [9].

Pomset trace equivalence is obtained by unrolling a Petri net into a process as defined by [7]. 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 $\mathscr{N}$ and their correspondence to the original places recorded in a mapping π. Then, whenever in N a transition t is fired, this is replayed in $\mathscr{N}$ by a new transition connected to places corresponding by π to the original preplaces of t and which are not yet connected to any other post-transition. A new place of $\mathscr{N}$ is created for every token produced by t. Again all correspondences are recorded in π. Every place of $\mathscr{N}$ 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 5. Let $N = (S,T,F,M_0,\ell)$ be a labelled net.
The *end* of the net is defined as $N^\circ := \{s \in S \mid s^\bullet = \varnothing\}$.

Definition 6.

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

- $\mathscr{N} = (\mathscr{S},\mathscr{T},\mathscr{F},\mathscr{M}_0,\mathscr{L})$ is a net, satisfying
  - $\forall s \in \mathscr{S}. |{}^\bullet s| \leq 1 \geq |s^\bullet| \wedge s \in \mathscr{M}_0 \Leftrightarrow {}^\bullet s = \emptyset$

  - $\mathcal{F}$ is acyclic, i.e. $\forall x \in \mathcal{S} \cup \mathcal{T}.(x,x) \notin \mathcal{F}^+$,
    where $\mathcal{F}^+$ is the transitive closure of $\{(t,u) \mid \mathcal{F}(t,u) > 0\}$,
  - and $\{t \mid (t,u) \in \mathcal{F}^+\}$ is finite for all $u \in \mathcal{T}$.
- $\pi : \mathcal{S} \cup \mathcal{T} \to S \cup T$ is a function with $\pi(\mathcal{S}) \subseteq S$ and $\pi(\mathcal{T}) \subseteq T$, satisfying
  - $s \in M_0 \Leftrightarrow |\pi^{-1}(s) \cap \mathcal{M}_0| = 1$ for all $s \in S$,
  - π is injective on $\mathcal{M}_0$,
  - $\forall t \in \mathcal{T}, s \in S. F(s, \pi(t)) = |\pi^{-1}(s) \cap {}^\bullet t| \wedge F(\pi(t), s) = |\pi^{-1}(s) \cap t^\bullet|$, and
  - $\forall t \in \mathcal{T}. \ell(t) = \ell(\pi(t))$.[2]

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

$\mathcal{P}$ is *maximal* iff $\pi(\mathcal{N}^\circ) \not\longrightarrow_N$. The set of all maximal processes of a net N is denoted by $MP(N)$.
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 [9], i.e. a transition which remained enabled infinitely long must ultimately fire.

To abstract from the τ-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 7.
A *labelled partial order* is a structure $(V, T, \leq, l)$ where

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

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

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

Definition 8. Let $o = (V, T, \leq, l)$ 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 9. Let $\mathcal{P} = ((\mathcal{S}, \mathcal{T}, \mathcal{F}, \mathcal{M}_0, \ell), \pi)$ be a process.
Let $\mathcal{O} := \{t \in \mathcal{T} \mid \ell(t) \neq \tau\}$, i.e. the visible transitions of the process. The *visible pomset* of $\mathcal{P}$ is the pomset $VP(\mathcal{P}) := [(\mathcal{O}, \mathrm{Act}, \mathcal{F}^* \cap \mathcal{O} \times \mathcal{O}, \ell \cap (\mathcal{O} \times \mathrm{Act}))]$ where $\mathcal{F}^*$ is the transitive and reflexive closure of the flow relation $\mathcal{F}$.

$\mathrm{MVP}(N) := \{VP(\mathcal{P}) \mid \mathcal{P} \in MP(N)\}$ is the set of pomsets of all maximal processes of N.
Using this notion we can now define completed pomset trace equivalence.

Definition 10.
Two nets N and N' are *completed pomset trace equivalent*, $N \simeq_{CPT} N'$, iff $\mathrm{MVP}(N) = \mathrm{MVP}(N')$.

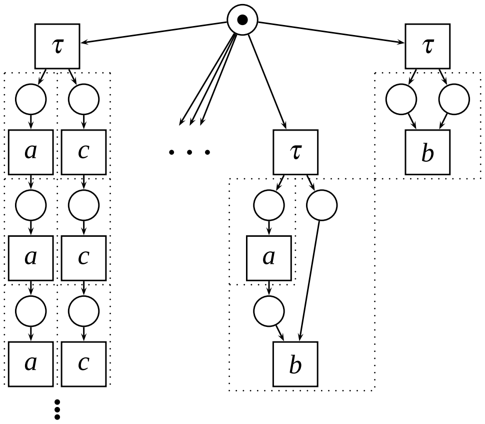

Figure 5: An infinite implementation of Figure 2, constructed by taking every maximal process and initially choosing one, location borders dotted.

5 Impossibility

As completed pomset trace equivalence is a very linear-time equivalence, it disregards the decision structure of a system and 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 fully satisfactory. 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 τ-transition before an arbitrary transition of a net, as described below, does not significantly influence the properties of that net.

Lemma 1. *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

- $S' = S \cup \{s_t\}$,
- $T' = T \cup \{\tau_t\}$,
- $F' = (F \setminus (S \times {}^\bullet t)) \cup \{(s,\tau_t) \mid s \in {}^\bullet t\} \cup \{(\tau_t,s_t),(s_t,t)\}$, *and*
- $\ell'(x) = \begin{cases} \tau & \textit{if } x = \tau_t \\ \ell(x) & \textit{otherwise} \end{cases}$

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 $(\mathscr{N},\pi)$ of N, a process of N' can be constructed by refining in $\mathscr{N}$ 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

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

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 $D'(x) := \begin{cases} D(t) & \text{if } x = s_t \lor x = \tau_t \\ D(x) & \text{otherwise} \end{cases}$. 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 τ_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 an execution, the dependencies of all tokens consumed and produced by a transition firing in such a cycle are equal.

Lemma 2. *Let $N = (S,T,F,M_0,\ell)$ be a finite, 1-safe net. Let $t_s, t_{s+1}, \dots, t_{e-1}, t_e \in T$ 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 $s \leq i \leq e$ such that t_i consumed an L-independent token from one of its preplaces (for some $L \subsetneq \text{Act}$), 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 4.

Theorem 5.1.
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 1, a new net $N = (S,T,F,M_0,\ell)$ is derived. By Lemma 1 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 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 $G_1, G_2, \dots G_n$ 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 twice. Of all those dependency markings which occur twice, we take the one occurring last while firing $(ac)^m b$ and call it M_{base}. Let $G_s, G_{s+1}, \dots, G_{e-1}, G_e$ 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 Lemma 2 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 $G_s \cap T_{\{a\}}, G_{s+1} \cap T_{\{a\}}, \dots, G_e \cap T_{\{a\}}$ (skipping empty steps) repeatedly, $M_{base} \overset{a^m}{\Longrightarrow}$. By firing $G_s \cap T_{\{c\}}, G_{s+1} \cap T_{\{c\}}, \dots, G_e \cap T_{\{c\}}$ (skipping empty steps) repeatedly, $M_{base} \overset{c^m}{\Longrightarrow}$.

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

Assume a reachable marking M'' of N with $M'' \overset{a^m}{\Longrightarrow}$. If $M'' \overset{c^m}{\not\Longrightarrow}$ 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 rôles of a and c reversed, hence $M'' \overset{a^m}{\Longrightarrow}$ iff $M'' \overset{c^m}{\Longrightarrow}$.

We start from M_{base} and start to fire the steps G_s, $G_{s+1}, \dots, G_n$ 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 $M \overset{a^m}{\Longrightarrow}$ but also $M \overset{c^m}{\Longrightarrow}$ and not only $M' \overset{a^m}{\not\Longrightarrow}$ but also $M' \overset{c^m}{\not\Longrightarrow}$, as both M and M' are reachable markings.

t_b is not itself labelled b, as the refined net has a τ-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 $G_i \cap T_{\{a\}}$, $s \leq i \leq e$ 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.[3] 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.

[3] 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'.

This paper only considered 1-safe nets as possible implementations. We conjecture however, that the proof of Theorem 5.1 can be extended to non-safe nets 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.

6 Conclusion

A review of existing literature in the related area can be found in [4], nonetheless we wish to refer the reader also to [5], 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 earlier is [1], 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 serialising all actions on a single location. In contrast to the present paper and similar to [5], the authors start with a user-supplied map from events to locations, and answer the concrete problem of whether that specific distribution is realisable or not instead of requiring the maximal possible parallelism to be realised.

Comparing the proof of Theorem 5.1 with the proof in [4] we observe that the counterexample in both proofs is based on two conflicts overlapping by a transition, i.e., on what is therein called a fully reachable pure **M**. In the synchronous 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 simultaneously 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 in [10], be enforced by the encoding, leading to additional causal dependencies.

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 implementation in general and those which do not. In [4] we showed 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 of [10] 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.

That the observed effects are not peculiarities of the Petri net model of systems but a reality of asynchronous systems in general is underlined by the existence of an companion paper [6], giving a result similar to the one achieved here in the setting of the asynchronous π-calculus.

A closer look on the proof in [6] reveals that this proof depends on counterexamples that are so called symmetric networks including mixed choices in a similar way as our result depends on counterexamples including a pure **M**. A symmetric network – for instance $R = \overline{a} + b + b.\checkmark \mid \overline{b} + a + a.\checkmark$ in the second part of the proof – consists of some parallel processes that differ only due to some permutation of names. In combination with mixed choice, i.e., a choice between input as well as output capabilities, symmetric networks result in conflicting steps on different links. Hence in both cases the counterexamples refer to some situation in the synchronous setting in which there are two distinct but conflicting steps. To solve this conflict two simultaneous activities are necessary – in case of Petri nets two tokens are removed simultaneously and in case of the π-calculus two sums are reduced simultaneously in one step. In the asynchronous setting this simultaneous solution has to be serialised by some kind of lock. It blocks the enabling of the asynchronous implementations of source steps, such that no two implementations of conflicting source steps are enabled concurrently. In both formalisms, Petri nets and the π-calculus, it is this temporally blocking of the implementation of source steps, necessary to avoid deadlock or divergence in case of conflicting source steps, that leads to additional causal dependencies.

Apart from this apparent similarity however, much of the relation between the two results remains mysterious to us. To begin with, the requirements imposed on Petri net implementations and π-calculus implementations take wildly different forms. Additionally, in contrast to the π-calculus result, the present paper connected implementation and original by means of behaviour only without any reference to the system structure. The π-calculus result on the other hand had no need to give special attention to infinite implementations. Finally, we also have no explanation for why the difference in expressive power (the π-calculus is turing-complete) should not make a difference for results such as this. We hope to answer some of these questions in future work.

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.

References

[1] Éric Badouel, Benoît Caillaud & Philippe Darondeau (2002): *Distributing Finite Automata Through Petri Net Synthesis*. *Formal Aspects of Computing* 13, pp. 447–470, doi:10.1007/s001650200022.

[2] Rob J. van Glabbeek (1993): *The Linear Time - Branching Time Spectrum II*. In: *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, Springer, London, UK, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[3] Rob J. van Glabbeek & Ursula Goltz (2001): *Refinement of actions and equivalence notions for concurrent systems*. *Acta Informatica* 37(4/5), pp. 229–327, doi:10.1007/s002360000041.

[4] Rob J. van Glabbeek, Ursula Goltz & Jens-Wolfhard Schicke (2008): *On Synchronous and Asynchronous Interaction in Distributed Systems*. Technical Report 2008-04, TU Braunschweig. Available at `http://arxiv.org/abs/0901.0048v1`. Extended abstract in Proceedings 33rd *International Symposium on Mathematical Foundations of Computer Science* (MFCS 2008), Toruń, Poland, August 2008 (E. Ochmański & J. Tyszkiewicz, eds.), LNCS 5162, Springer, 2008, pp. 16-35.

[5] Richard P. Hopkins (1991): *Distributable nets*. In: *Advances in Petri Nets 1991*, LNCS 524, Springer, pp. 161–187, doi:10.1007/BFb0019974.

[6] Kirstin Peters, Jens-Wolfhard Schicke & Uwe Nestmann (2011): *Synchrony vs Causality in the Asynchronous Pi-Calculus*. To appear in the Proceedings of EXPRESS'11.

[7] Carl Adam Petri (1977): *Non-sequential Processes*. GMD-ISF Report 77.05, GMD.

[8] Vaughan R. Pratt (1985): *The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial*. In: *Seminar on Concurrency, Carnegie-Mellon University*, Springer, London, UK, pp. 180–196, doi:10.1007/3-540-15670-4_9.

[9] Wolfgang Reisig (1984): *Partial Order Semantics versus Interleaving Semantics for CSP-like Languages and its Impact on Fairness*. In: *Proc. of the 11th Colloquium on Automata, Languages and Programming*, Springer, London, UK, pp. 403–413, doi:10.1007/3-540-13345-3_37.

[10] Jens-Wolfhard Schicke (2009): *Diplomarbeit: Synchrony and Asynchrony in Petri Nets*. TU Braunschweig.