Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 229 (2009) 77–95

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Symmetric and Asymmetric Asynchronous Interaction

Rob van Glabbeek[1]

NICTA, Sydney, Australia
University of New South Wales, Sydney, Australia

Ursula Goltz[2] Jens-Wolfhard Schicke[3]

Institute for Programming and Reactive Systems
TU Braunschweig
Braunschweig, Germany

Abstract

We investigate classes of systems based on different interaction patterns with the aim of achieving distributability. As our system model we use Petri nets. In Petri nets, an inherent concept of simultaneity is built in, since when a transition has more than one preplace, it can be crucial that tokens are removed instantaneously. When modelling a system which is intended to be implemented in a distributed way by a Petri net, this built-in concept of synchronous interaction may be problematic. To investigate the problem we assume that removing tokens from places can no longer be considered as instantaneous. We model this by inserting silent (unobservable) transitions between transitions and their preplaces. We investigate three different patterns for modelling this type of asynchronous interaction. *Full asynchrony* assumes that every removal of a token from a place is time consuming. For *symmetric asynchrony*, tokens are only removed slowly in case of backward branched transitions, hence where the concept of simultaneous removal actually occurs. Finally we consider a more intricate pattern by allowing to remove tokens from preplaces of backward branched transitions asynchronously in sequence (*asymmetric asynchrony*).
We investigate the effect of these different transformations of instantaneous interaction into asynchronous interaction patterns by comparing the behaviours of nets before and after insertion of the silent transitions. We exhibit for which classes of Petri nets we obtain equivalent behaviour with respect to failures equivalence. It turns out that the resulting hierarchy of Petri net classes can be described by semi-structural properties. In case of full asynchrony and symmetric asynchrony, we obtain precise characterisations; for asymmetric asynchrony we obtain lower and upper bounds.
We briefly comment on possible applications of our results to Message Sequence Charts.

Keywords: reactive systems, Petri nets, distributed systems, asynchronous interaction, equivalence notions

[1] Email: rvg@cs.stanford.edu
[2] Email: goltz@ips.cs.tu-bs.de
[3] Email: drahflow@gmx.de

doi:10.1016/j.entcs.2009.06.040

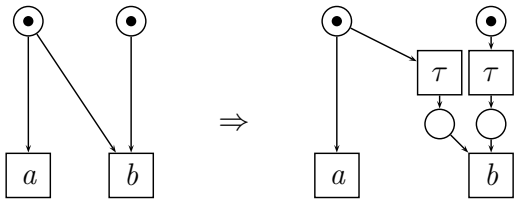


Fig. 1. Transformation to the symmetrically asynchronous implementation

1 Introduction

In this paper, we investigate classes of systems based on different asynchronous interaction patterns with the aim of achieving distributability, i.e. the possibility to execute a system on spatially distributed locations, which do not share a common clock. As our system model we use Petri nets. The main reason for this choice is the detailed way in which a Petri net represents a concurrent system, including the interaction between the components it may consist of. In an interleaving based model of concurrency such as labelled transition systems modulo bisimulation semantics, a system representation as such cannot be said to display synchronous or asynchronous interaction; at best these are properties of composition operators, or communication primitives, defined in terms of such a model. A Petri net on the other hand displays enough detail of a concurrent system to make the presence of synchronous communication discernible. This makes it possible to study asynchronous communication without digressing to the realm of composition operators.

In a Petri net, a transition interacts with its preplaces by consuming tokens. An inherent concept of simultaneity is built in, since when a transition has more than one preplace, it can be crucial that tokens are removed instantaneously, depending on the surrounding structure or—more elaborately—the behaviour of the net.

When modelling a distributed system by a Petri net, this built-in concept of synchronous interaction may become problematic. Assume a transition t on a location l models an activity involving another location l', for example by receiving a message. This can be modelled by a preplace s of t such that s and t are situated in different locations. We assume that taking a token can in this situation not be considered as instantaneous; rather the interaction between s and t takes time. We model this effect by inserting silent (unobservable) transitions between transitions and their preplaces. We call the effect of such a transformation of a net N an *asynchronous implementation* of N.

An example of such an implementation is shown in Figure 1. Note that a can be disabled in the implementation before any visible behaviour has taken place. This difference will cause non-equivalence between the original and the implementation under branching time equivalences.

Our asynchronous implementation allows a token to start its journey from a place to a transition even when not all preplaces of the transition contain a token. This design decision is motivated by the observation that it is fundamentally impossible to check in an asynchronous way whether all preplaces of a transition are marked—it

could be that a token moves back and forth between two such places.

We investigate different interaction patterns for the asynchronous implementation of nets. The simplest pattern (*full asynchrony*) assumes that every removal of a token from a place is time consuming. For the next pattern (*symmetric asynchrony*), tokens are only removed slowly when they are consumed by a backward branched transition, hence where the concept of simultaneous removal actually occurs. Finally we consider a more intricate pattern by allowing to remove tokens from preplaces of backward branched transitions asynchronously in sequence (*asymmetric asynchrony*).

Given a choice of interaction pattern, we call a net N *asynchronous* when there is no essential behavioural difference between N and its asynchronous implementation $I(N)$. In order to formally define this concept, we wish to compare the behaviours of N and $I(N)$ using a semantic equivalence that fully preserves branching time, causality and their interplay, whilst of course abstracting from silent transitions. By choosing the most discriminating equivalence possible, we obtain the smallest possible class of asynchronous nets, thus excluding nets that might be classified as asynchronous merely because a less discriminating equivalence would fail to see the differences between such a net and its asynchronous implementation. To simplify the exposition, here we merely compare the behaviours of N and $I(N)$ up to *failures equivalence* [6]. This interleaving equivalence abstracts from causality and respects branching time only to some degree. However, we conjecture that our results are in fact largely independent of this choice and that more discriminating equivalences, such as the history preserving ST-bisimulation of [21], would yield the same classes of asynchronous nets. Using a linear time equivalence would give rise to larger classes; this possibility is investigated in [19].

Thus we investigate the effect of our three transformations of instantaneous interaction into asynchronous interaction patterns by comparing the behaviours of nets before and after insertion of the silent transitions up to failures equivalence. We show that in the case of full asynchrony, we obtain equivalent behaviour exactly for conflict-free Petri nets. Further we establish that symmetric asynchrony is a valid concept for N-free Petri nets and asymmetric asynchrony for M-free Petri nets, where N and M stand for certain structural properties; the reachability of such structures is crucial. For symmetric asynchrony we obtain a precise characterisation of the class of nets which is asynchronously implementable. For asymmetric asynchrony we obtain lower and upper bounds.

In the concluding section, we discuss the use of our results for Message Sequence Charts, as an example how they may be useful for other models than Petri nets. When interpreting basic Message Sequence Chart as Petri nets, the resulting Petri nets lie within the class of conflict-free and hence N-free Petri nets. The more expressive classes give insights in the effect of choices in non-basic MSCs.

This is an extended abstract; for sake of brevity most proofs are omitted. They are contained in the full version of this paper [8], as well as in [19].

The paper is structured as follows. In Section 2 we establish the necessary basic notions. In Section 3 we introduce the fully asynchronous transformation and give

a semi-structural characterisation of the resulting net class. In Section 4 we repeat those steps for the symmetrically asynchronous transformation. Furthermore we describe how the resulting net class relates to the classes of free-choice and extended free choice nets. In Section 5 we introduce the asymmetrically asynchronous transformation. We give semi-structural upper and lower bounds for the resulting net class and relate it to simple and extended simple nets. In the conclusion in Section 6 we compare our findings to similar results in the literature.

2 Basic Notions

We consider here 1-safe net systems, i.e. places never carry more than one token, but a transition can fire even if pre- and postset intersect. To represent unobservable behaviour, which we use to model asynchrony, the set of transitions is partitioned into observable and silent (unobservable) ones.

Definition 2.1
A *net with silent transitions* is a tuple $N = (S, O, U, F, M_0)$ where
- S is a set (of *places*),
- O is a set (of *observable transitions*),
- U is a set (of *silent transitions*),
- $F \subseteq S \times T \cup T \times S$ (the *flow relation*) with $T := O \cup U$ (*transitions*) and
- $M_0 \subseteq S$ (the *initial marking*).

Petri nets are depicted by drawing the places as circles, the transitions as boxes, 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.

Definition 2.2 Let $N = (S, O, U, F, M_0)$ be a net. Let $M_1, M_2 \subseteq S$.
We denote the preset and postset of a net element x by ${}^\bullet x := \{y \mid (y,x) \in F\}$ and $x^\bullet := \{y \mid (x,y) \in F\}$ respectively. A nonempty set of transitions $G \subseteq (O \cup U), G \neq \varnothing$, is called a *step from* M_1 *to* M_2, notation $M_1\ [G\rangle_N\ M_2$, iff

- all transitions contained in G are *enabled*, that is

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

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

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

- in M_2 all tokens have been removed from the *preplaces* of G and new tokens have been inserted at the *postplaces* of G:

$$M_2 = \left(M_1 \setminus \bigcup_{t \in G} {}^\bullet t\right) \cup \bigcup_{t \in G} t^\bullet\ .$$

To simplify statements about possible behaviours of nets, we use some abbreviations.

Definition 2.3 Let $N = (S, O, U, F, M_0)$ be a net with silent transitions.

- $\longrightarrow_N \subseteq \mathcal{P}(S) \times \mathcal{P}(O) \times \mathcal{P}(S)$ is defined by $M_1 \xrightarrow{G}_N M_2 \Leftrightarrow G \subseteq O \wedge M_1[G\rangle_N M_2$
- $\xrightarrow{\tau}_N \subseteq \mathcal{P}(S) \times \mathcal{P}(S)$ is defined by $M_1 \xrightarrow{\tau}_N M_2 \Leftrightarrow \exists t \in U.\ M_1\ [\{t\}\rangle_N\ M_2$
- $\Longrightarrow_N \subseteq \mathcal{P}(S) \times O^* \times \mathcal{P}(S)$ is defined by $M_1 \xRightarrow{t_1 t_2 \cdots t_n}_N M_2 \Leftrightarrow$

$$M_1 \xrightarrow{\tau}{}^*_N \xrightarrow{\{t_1\}}_N \xrightarrow{\tau}{}^*_N \xrightarrow{\{t_2\}}_N \xrightarrow{\tau}{}^*_N \cdots \xrightarrow{\tau}{}^*_N \xrightarrow{\{t_n\}}_N \xrightarrow{\tau}{}^*_N M_2$$

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

We write $M_1 \xrightarrow{G}_N$ for $\exists M_2.\ M_1 \xrightarrow{G}_N M_2$, $M_1 \not\xrightarrow{G}_N$ for $\nexists M_2.\ M_1 \xrightarrow{G}_N M_2$ and similar for the other two relations.

A marking M_1 is said to be *reachable* iff there is a $\sigma \in O^*$ such that $M_0 \xRightarrow{\sigma} M_1$. The set of all reachable markings is denoted by $[M_0\rangle_N$.

We omit the subscript N if clear from context.

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 $t \in O \cup U$ with ${}^\bullet t \subseteq M_1$

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

For such nets, in Definition 2.2 we can just as well consider a transition t to be enabled in M iff ${}^\bullet t \subseteq M$, and two transitions to be independent when ${}^\bullet t \cap {}^\bullet u = \varnothing$. In this paper we furthermore restrict attention to nets for which ${}^\bullet t \neq \emptyset$, and ${}^\bullet t$ and $t^\bullet$ are finite for all $t \in O \cup U$. We also require the initial marking M_0 to be finite. A consequence of these restrictions is that all reachable markings are finite, and it can never happen that infinitely many independent transitions are enabled. Henceforth, we employ the name *τ-nets* for nets with silent transitions obeying the above restrictions, and *plain nets* for τ-nets without silent transitions, i.e. with $U = \emptyset$.

Our nets with silent transitions can be regarded as special *labelled nets*, defined as in Definition 2.1, but without the split of T into O and U, and instead equipped with a *labelling function* $\ell : T \to Act \cup \{\tau\}$, where Act is a set of *visible actions* and $\tau \notin Act$ an invisible one. Nets with silent transitions correspond to labelled nets

in which no two different transitions are labelled by the same visible actions, which can be formalised by taking $\ell(t) = t$ for $t \in O$ and $\ell(t) = \tau$ for $t \in U$.

To describe which nets are "asynchronous", we will compare their behaviour to that of their asynchronous implementations using a suitable equivalence relation. As explained in the introduction, we consider here branching time semantics. Technically, we use failures equivalence, as defined below.

Definition 2.4 Let $N = (S, O, U, F, M_0)$ be a τ-net, $\sigma \in O^*$ and $X \subseteq O$.
$<\sigma, X>$ is a *failure pair* of N iff

$$\exists M_1.\ M_0 \stackrel{\sigma}{\Longrightarrow} M_1 \wedge M_1 \stackrel{\tau}{\not\longrightarrow} \wedge \forall t \in X.\ M_1 \stackrel{\{t\}}{\not\longrightarrow} .$$

We define $\mathscr{F}(N) := \{ <\sigma, X> \mid <\sigma, X> \text{ is a failure pair of } N\}$.
Two τ-nets N and N' are *failures equivalent*, $N \approx_{\mathscr{F}} N'$, iff $\mathscr{F}(N) = \mathscr{F}(N')$.

A τ-net $N = (S, O, U, F, M_0)$ is called *divergence free* iff there are no infinite chains of markings $M_1 \stackrel{\tau}{\longrightarrow} M_2 \stackrel{\tau}{\longrightarrow} \cdots$ with $M_1 \in [M_0\rangle$.

3 Full Asynchrony

As explained in the introduction, we will examine in this paper different possible assumptions of how asynchronous interaction between transitions and their preplaces takes place. In this section, we start with the simple and intuitive assumption that the removal of any token by a transition takes time. This is implemented by inserting silent transitions between visible ones and their preplaces.

Definition 3.1 Let $N = (S, O, \varnothing, F, M_0)$ be a plain net.
The *fully asynchronous implementation* of N is defined as the net
$FI(N) := (S \cup S^\tau, O, U', F', M_0)$ with

$$\begin{aligned} S^\tau &:= \{s_t \mid t \in O, s \in {}^\bullet t\}\ , \\ U' &:= \{t_s \mid t \in O, s \in {}^\bullet t\} \text{ and} \\ F' &:= (F \cap (O \times S)) \cup \{(s, t_s), (t_s, s_t), (s_t, t) \mid t \in O, s \in {}^\bullet t\}\ . \end{aligned}$$

It is not hard to see that implementations of contact-free nets are contact-free and implementations are always divergence free; in fact an implementation of a plain net is always a divergence free τ-net.

Whereas in a plain net N for any sequence of observable transitions $\sigma \in O^*$ there is at most one marking M with $M_0 \stackrel{\sigma}{\Longrightarrow} M$, in its fully asynchronous implementation $FI(N)$ there can be several such markings. These markings M' differ from M in that some tokens may have wandered off into the added invisible transitions on the incoming arcs of visible ones.

As a consequence, a visible transition t that is enabled in M need not be enabled in M'—we say that in $FI(N)$ t *can be refused after* σ. This may occur for instance for the net N of Figure 2, namely with $\sigma = \varepsilon$ (the empty sequence), M the initial

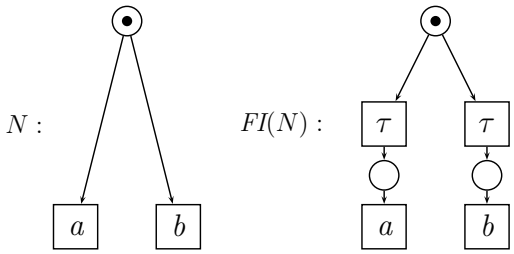


Fig. 2. A net which is not failures equivalent to its fully asynchronous implementation

marking of N, M' the marking of $FI(N)$ obtained by firing the rightmost invisible transition, and $t = a$.

When this happens, we have $<\sigma, \{t\}> \in \mathscr{F}(FI(N)) \setminus \mathscr{F}(N)$, so the nets N and $FI(N)$ are not failures equivalent. If, on the other hand, the wandering off of tokens into τ-transitions never disables a transition that would be enabled otherwise, then there is no essential behavioural difference between N and $FI(N)$, and they are equivalent in any reasonable behavioural equivalence that abstracts from silent transition firings. In that case, N could be called *fully asynchronous*.

Definition 3.2
The class of *fully asynchronous nets respecting branching time equivalence* is defined as $FA(B) := \{N \mid FI(N) \approx_{\mathscr{F}} N\}$.

As for any plain net N we have $\mathscr{F}(N) \subseteq \mathscr{F}(FI(N))$ [8], the class of nets $FA(B)$ can equivalently be defined as $FA(B) := \{N \mid \mathscr{F}(FI(N)) \subseteq \mathscr{F}(N)\}$.

It turns out that there exists a quite structural characterisation of those nets which are failures equivalent to their fully asynchronous implementation.

Definition 3.3
A plain net $N = (S, O, \varnothing, F, M_0)$ *has a partially reachable conflict* iff $\exists t, u \in O.$ $t \neq u \wedge {}^\bullet t \cap {}^\bullet u \neq \varnothing$ and $\exists M \in [M_0\rangle.$ ${}^\bullet t \subseteq M \vee {}^\bullet u \subseteq M$.

The nets N of Figures 2 and 3, for instance, have a partially reachable conflict.

Theorem 3.4 *A plain net N is in $FA(B)$ iff N has no partially reachable conflict.*

Proof. See [19] or [8]. □

4 Symmetric Asynchrony

For investigating the next interaction pattern, we change our notion of asynchronous implementation of a net. We only insert silent transitions wherever a transition has multiple preplaces. These are the situations where the synchronous removal of tokens is really essential.

Definition 4.1 Let $N = (S, O, \varnothing, F, M_0)$ be a net. Let $O^b = \{t \mid t \in O, |{}^\bullet t| > 1\}$. The *symmetrically asynchronous implementation* of N is defined as the net

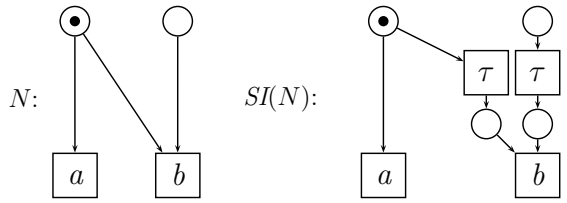


Fig. 3. The transition a can be refused in $SI(N)$ by firing the left τ.

$SI(N) := (S \cup S^\tau, O, U', F', M_0)$ with

$$\begin{aligned} S^\tau &:= \{s_t \mid t \in O^b, s \in {}^\bullet t\}\ , \\ U' &:= \{t_s \mid t \in O^b, s \in {}^\bullet t\} \text{ and} \\ F' &:= F \cap \Big((O \times S) \cup (S \times (O \setminus O^b))\Big) \\ &\quad \cup \{(s, t_s), (t_s, s_t), (s_t, t) \mid t \in O^b, s \in {}^\bullet t\}\ . \end{aligned}$$

An example is shown in Figure 3.

As for the fully asynchronous case, an implementation of a plain net is always a divergence-free τ-net.

Again, the only difference in behaviour between the original net and its implementation is that observable transitions can potentially be refused in the implementation, as in Figure 3. This yields a concept of a *symmetrically asynchronous* net.

Definition 4.2
The class of *symmetrically asynchronous nets respecting branching time equivalence* is defined as $SA(B) := \{N \mid SI(N) \approx_{\mathscr{F}} N\}$.

Again we have $\mathscr{F}(N) \subseteq \mathscr{F}(SI(N))$ for any plain net N [8]. We now show that plain nets can be implemented symmetrically asynchronously with respect to failure equivalence exactly when they do not contain reachable structures of the form shown in Figure 3.

Definition 4.3
A plain net $N = (S, O, \varnothing, F, M_0)$ *has a partially reachable* N iff $\exists t, u \in O.\ t \neq u \wedge {}^\bullet t \cap {}^\bullet u \neq \varnothing \wedge |{}^\bullet u| > 1 \wedge \exists M \in [M_0\rangle_N.\ {}^\bullet t \subseteq M \vee {}^\bullet u \subseteq M$.

Theorem 4.4 *A plain net N is in $SA(B)$ iff N has no partially reachable* N.

Proof. See [19] or [8]. □

The following proposition shows that the current class of nets strictly extends the one from the previous section.

Proposition 4.5 $FA(B) \subsetneq SA(B)$.

Proof. A net without partially reachable conflict surely has no partially reachable N. The inequality follows from the example in Figure 2. □

It turns out that our class of nets $SA(B)$ is strongly related to the following established net classes [2,3].

Definition 4.6 Let $N = (S, O, \varnothing, F, M_0)$ be a plain net.
(i) N is *free choice*, $N \in FC$, iff $\forall p, q \in S.\ p \neq q \wedge p^\bullet \cap q^\bullet \neq \varnothing \Rightarrow |p^\bullet| = |q^\bullet| = 1$.
(ii) N is *extended free choice*, $N \in EFC$, iff $\forall p, q \in S.\ p^\bullet \cap q^\bullet \neq \varnothing \Rightarrow p^\bullet = q^\bullet$.
(iii) N is *behaviourally free choice*, $N \in BFC$, iff $\forall u, v \in O.\ {}^\bullet u \cap {}^\bullet v \neq \varnothing \Rightarrow$ $(\forall M_1 \in [M_0\rangle.\ {}^\bullet u \subseteq M_1 \Leftrightarrow {}^\bullet v \subseteq M_1)$.

The above definition of a free choice net is in terms of places, but the notion can equivalently be defined in terms of transitions:

$$N \in FC \ \text{ iff } \ \forall t, u \in T.\ t \neq u \wedge {}^\bullet t \cap {}^\bullet u \neq \varnothing \Rightarrow |{}^\bullet t| = |{}^\bullet u| = 1.$$

Both conditions are equivalent to the requirement that N must be N-free, where N is defined as in Definition 4.3 but without the reachability clause. Also the notion of an extended free choice net can equivalently be defined in terms of transitions:

$$N \in EFC \ \text{ iff } \ \forall t, u \in T.\ {}^\bullet t \cap {}^\bullet u \neq \varnothing \Rightarrow {}^\bullet t = {}^\bullet u.$$

This condition says that N may not contain what we call a *pure* N: places p, q and transitions t, u such that $p \in {}^\bullet t \cap {}^\bullet u$, $q \in {}^\bullet u$ and $q \notin {}^\bullet t$.

In [3] it has been established that $FC \subsetneq EFC \subsetneq BFC$. In fact, the inclusions follow directly from the definitions, and Figure 4 displays counterexamples to strictness.

The class of free choice nets is strictly smaller than the class of symmetrically asynchronous nets respecting branching time equivalence, which in turn is strictly smaller than the class of behavioural free choice nets. The class of extended free choice nets and the class of symmetrically asynchronous nets respecting branching time equivalence are incomparable.

Proposition 4.7 *$FC \subsetneq SA(B) \subsetneq BFC$, $EFC \not\subseteq SA(B)$ and $SA(B) \not\subseteq EFC$.*

Proof. The first inclusion follows because a partially reachable N is surely an N, and also the second inclusion follows directly from the definitions. The four inequalities follow from the examples in Figure 4. The first net is unmarked and thus trivially in $SA(B)$. The second ones symmetrically asynchronous implementation has the additional failure $<\varepsilon, \{a, b\}>$ and hence this net is not in $SA(B)$. □

In Figure 5 the relations between our semantically defined net class $SA(B)$, the structurally defined classes FC, EFC, and the more behaviourally defined class BFC are summarised. These relations may be interpreted as follows.

Starting at the top of the diagram, free choice nets are characterised structurally, enforcing that for every place, a token therein can choose freely (i.e. without inquiring about the existence of tokens in any other places) which outgoing arc to take. This property makes it possible to implement the system asynchronously. In particular, the component which holds the information represented by a token can choose

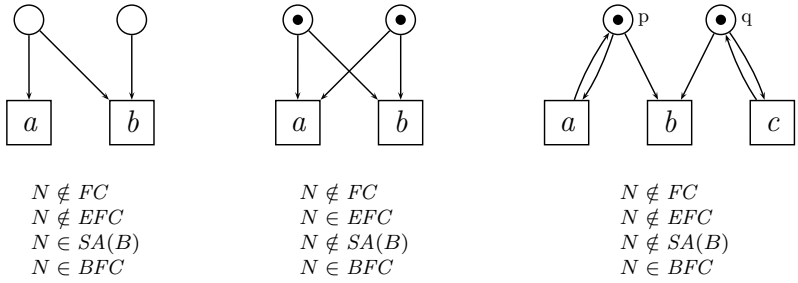


Fig. 4. Differences between various classes of free-choice-like nets

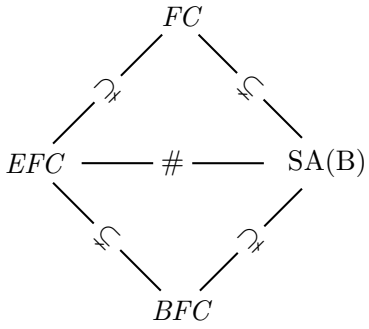


Fig. 5. Overview of free-choice-like net classes

arbitrarily when and into which of multiple asynchronous output channels to forward said information, without further knowledge about the rest of the system. As this decision is solely in the discretion of the sending component and not based upon any knowledge of the rest of the system, no synchronisation with other components is necessary.

The difference between $SA(B)$ and FC is that in $SA(B)$ the quantification over the places is dropped, making the requirement more straightforward: Every token can choose freely which outgoing arc to follow. Thus, $SA(B)$ allows for non-free-choice structures as long as these never receive any tokens.

This also explains why BFC includes $SA(B)$. Since $SA(B)$ guarantees that all transitions of a problematic structure are never enabled, transitions in such structures are never enabled while others are disabled.

The incomparability between the left and the right side of the diagram stems from the conceptual allowance of slight transformations of the net before evaluating whether it is free choice or not. Extended free choice nets and behavioural free choice nets were proposed as nets that are easily seen to be behaviourally equivalent to free choice nets, and hence share some of their desirable properties: in [2,3] constructions can be found to turn any extended free choice net into an equivalent free choice net, and any behavioural free choice net into an extended free choice net. [1] Applied on the last two nets in Figure 4 these constructions yield:

For the second net of Figure 4, a τ-transition is introduced, which collects both

Fig. 6. Transformed nets from Figure 4

tokens and then marks a single postplace from which the two original transitions are enabled. Hence the choice between the two transitions is centralised in the newly introduced place and thus free again. In the definition of our symmetrically asynchronous implementation *SI*, we do not allow any insertion of such "helping" τ-transitions, as it seems unclear to us how much computing power should be allowed in possibly larger networks of such transitions. This becomes especially problematic if these networks somehow track part of the global status of the net inside themselves and thus make quite informed decisions about what outgoing transition to enable.

5 Asymmetric Asynchrony

As seen in the previous section, the class of symmetrically asynchronous nets is quite small. It precludes the implementation of many real-world behaviours, like waiting for one of multiple inputs to become readable, a Petri net representation of which will always include non free-choice structures.

Therefore we propose a less strict definition of asynchrony such that actions may depend synchronously on a single predetermined condition. In a hardware implementation the places which earlier could always forward a token into some silent transitions must now wait until they receive an explicit token removal signal from their posttransitions.

To this end we introduce a static priority over the preplaces of each transition. Every transition first removes the token from the most prioritised preplace and then continues along decreasing priority. To formalise this behaviour in a Petri net we insert a silent transition for each incoming arc of every transition. These silent transitions are forced to execute in sequence by newly introduced buffer places between them. In the final position of this chain, the original visible transition is executed. An example of this transformation is given in Figure 7.

Definition 5.1 Let $N = (S, O, \varnothing, F, M_0)$ be a plain net.
Let $g \subseteq (S \times O) \times (S \times O)$ be a relation on $F \cap (S \times O)$ such that for each $t \in O$ $g \cap ({}^\bullet t \times \{t\})$ is a total order on ${}^\bullet t \times \{t\}$. Let $\leq_g^t$ be the total order on ${}^\bullet t$ given by $p \leq_g^t s$ iff $((p,t),(s,t)) \in g$.

[1] In [2,3] the nature of the equivalence between the original and transformed net is not precisely specified. However, it can be argued that whereas the transformation from *EFC*-nets to *FC*-nets preserves branching time as well as causality, the transformation from *BFC*-nets to *EFC*-nets preserves branching time only: the third net of Figure 4 is interleaving bisimulation equivalent with its *EFC*-counterpart in Figure 6, but whereas the original net can perform the transitions a and c concurrently (in one step), the transformed net cannot.

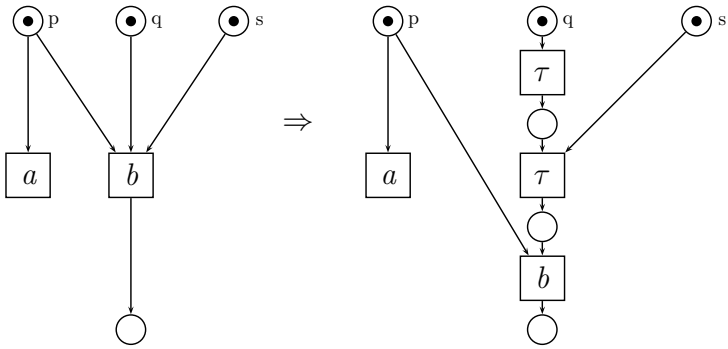


Fig. 7. Transformation to asymmetric asynchrony; g such that $p <^b_g s <^b_g q$.

We write $\min^t_g$ for the $\leq^t_g$-minimal element of ${}^\bullet t$ and $(s-1)^t_g$ for the next place in ${}^\bullet t$ that is $\leq^t_g$-smaller than s.
We define a set of silent transitions as $X := \{t_s \mid t \in O, s \in {}^\bullet t\}$.
Let $h : X \to X \cup O$ be the function

$$h(t_s) = \begin{cases} t & \text{iff } s = \min^t_g \\ t_s & \text{otherwise} \end{cases}$$

The *asymmetrically asynchronous implementation with respect to* g of N is defined as the net $AI_g := (S \cup S^\tau, O, U', F', M_0)$ with

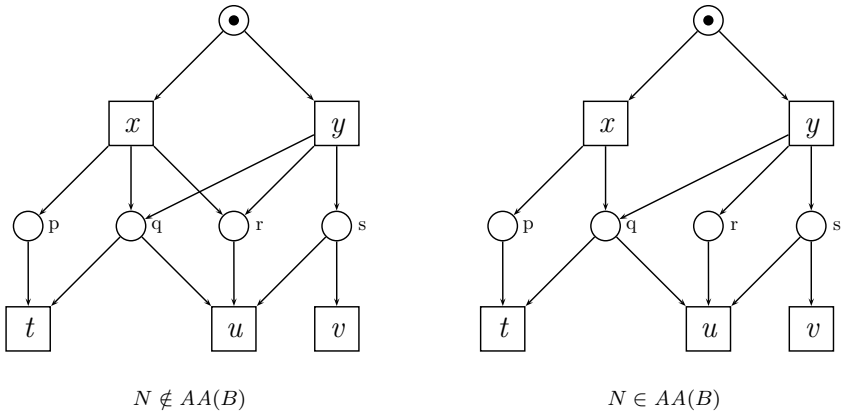
$$\begin{aligned} S^\tau &:= \{s_t \mid t \in O,\ s \in {}^\bullet t,\ s \neq \min^t_g\}\ , \\ U' &:= h(X) \setminus O = \{t_s \mid t \in O,\ s \in {}^\bullet t,\ s \neq \min^t_g\} \text{ and} \\ F' &:= F \cap (O \times S) \\ &\quad \cup \{(s, h(t_s)) \mid t \in O,\ s \in {}^\bullet t\} \\ &\quad \cup \{(t_s, s_t) \mid t \in O,\ s \in {}^\bullet t,\ s \neq \min^t_g\} \\ &\quad \cup \{(s_t, h(t_p)) \mid t \in O,\ s \in {}^\bullet t,\ s \neq \min^t_g,\ p = (s-1)^t_g\}\ . \end{aligned}$$

As before, we are interested in the relationship between nets and their possible implementations. The definition of asymmetric asynchrony however allows different implementations for the same net.

We define a net to be *asymmetrically asynchronous* if any of the possible implementations simulates the net sufficiently.

Definition 5.2
The class of *asymmetrically asynchronous nets respecting branching time equivalence* is defined as $AA(B) := \{N \mid \exists g.\ AI_g(N) \approx_{\mathscr{F}} N\}$.

As before, we have $\mathscr{F}(N) \subseteq \mathscr{F}(AI_g(N))$ for any plain net N and any priority relation g [8]. Additionally we would like to obtain a semi-structural characterisation of $AA(B)$ in the spirit of Theorems 3.4 and 4.4. Unfortunately we didn't succeed in this, but we obtained structural upper and lower bounds for this net class.



$N \notin AA(B)$ $N \in AA(B)$

Fig. 8. Nets which have a left and right border reachable M, but no left and right reachable M

Definition 5.3
A net $N = (S, O, \varnothing, F, M_0)$ *has a left and right reachable* M iff $\exists t, u, v \in O\ \exists p \in {}^\bullet t \cap {}^\bullet u\ \exists q \in {}^\bullet u \cap {}^\bullet v.\ t \neq u \wedge u \neq v \wedge p \neq q \wedge \exists M_1, M_2 \in [M_0\rangle.\ {}^\bullet t \cup {}^\bullet u \subseteq M_1 \wedge {}^\bullet v \cup {}^\bullet u \subseteq M_2$.
A net $N = (S, O, \varnothing, F, M_0)$ *has a left and right border reachable* M iff $\exists t, u, v \in O\ \exists p \in {}^\bullet t \cap {}^\bullet u\ \exists q \in {}^\bullet u \cap {}^\bullet v.\ t \neq u \wedge u \neq v \wedge p \neq q \wedge \exists M_1, M_2 \in [M_0\rangle.\ {}^\bullet t \subseteq M_1 \wedge {}^\bullet v \subseteq M_2$.

Theorem 5.4 *A plain net N in $AA(B)$ has no left and right reachable* M.
A plain net N which has no left and right border reachable M *is in $AA(B)$.*

Proof. See [19] or [8]. □

Figure 8 shows two nets, each with a left and right border reachable M but no left and right reachable M, that thus fall in the grey area between our structural upper and lower bounds for the class $AA(B)$. In this case the first net falls outside $AA(B)$, whereas the second net falls inside. The crucial difference between these two examples is the information available to u about the execution of y.

There exists an implementation for the right net, namely by u taking the tokens from r, q and s in that order. The first token (from r) conveys the information that y was executed, and thus t is not enabled. Collecting the last token (from s) could fail, due to v removing it earlier. Even so, removing the tokens from r and q did not disable any transition that could fire in the original net. In the left net such an implementation will not work.

The following proposition says that our class of symmetrically asynchronous nets strictly extends the corresponding class of asymmetrically asynchronous nets.

Proposition 5.5 $SA(B) \subsetneq AA(B)$.

Proof. A net which has no partially reachable N also has no left or right border reachable M. The inequality follows from the example in Figure 3. □

As before, our class $AA(B)$ is related to some known net classes [3].

Definition 5.6 Let $N = (S, O, \varnothing, F, M_0)$ be a plain net.

(i) N is *simple*, $N \in SPL$, iff $\forall p, q \in S.\ p \neq q \wedge p^\bullet \cap q^\bullet \neq \varnothing \Rightarrow |p^\bullet| = 1 \vee |q^\bullet| = 1$.
(ii) N is *extended simple*, $N \in ESPL$, iff $\forall p, q \in S.\ p^\bullet \cap q^\bullet \neq \varnothing \Rightarrow p^\bullet \subseteq q^\bullet \vee q^\bullet \subseteq p^\bullet$.

Extended simple nets appear in [2] under the name *asymmetric choice systems*. Note that simple is equivalent to M-free, where M is as in Definition 5.3 but without the reachability clauses. Clearly, we have $FC \subsetneq SPL \subsetneq ESPL$ and $EFC \subsetneq ESPL$, whereas $EFC \not\subseteq SPL$ and $SPL \not\subseteq EFC$: the inclusions follow immediately from the definitions, and the first two nets of Figure 4 provide counterexamples to the inequalities.

The class of asymmetrically asynchronous nets respecting branching time equivalence strictly extends the class of simple nets, whereas it is incomparable with the class of extended simple nets.

Proposition 5.7 *$SPL \subsetneq AA(B)$, $AA(B) \not\subseteq ESPL$ and $ESPL \not\subseteq AA(B)$.*

Proof. The inclusion is straightforward, and the inequalities follow from the counterexamples in Figure 4 (the second one) and Figure 9. The missing tokens in the latter example are intended. As no action is possible there will not be any additional implementation failures. □

The relations between the classes *SPL*, *ESPL* and $AA(B)$ are summarised in Figure 10. Similarly to what we did in Section 4, we now try to translate Figure 10 into an intuitive description.

The basic intuition behind *SPL* is that for every transition there is only one preplace where conflict can possibly occur. Whereas in *SPL* that possibility is determined by the static net structure, in $AA(B)$ reachability is also considered.

Similar to the difference between FC and EFC there exists a difference between *ESPL* and *SPL* which originates from the fact that *ESPL* allows small transformations to a net before testing whether it lies in *SPL*. Again our class $AA(B)$ does not allow such "helping" transformations.

6 Conclusion and Related Work

We have investigated the effect of different types of asynchronous interaction, using Petri nets as our system model. We propose three different interaction patterns: fully asynchronous, symmetrically asynchronous and asymmetrically asynchronous. An asynchronous implementation of a net is then obtained by inserting silent (unobservable) transitions according to the respective pattern. The pattern for asymmetric asynchrony is parametric in the sense that the actual asynchronous imple-

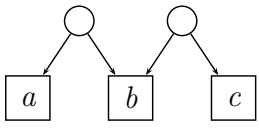


Fig. 9. $N \in AA(B)$, $N \notin ESPL$

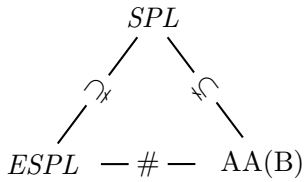


Fig. 10. Overview of asymmetric-choice-like net classes

mentation of a net depends on a chosen priority function on the input places of a transition. For each of these cases, we investigated for which types of nets the asynchronous implementation of a net changes its behaviour with respect to failures equivalence (in the case of asymmetric asynchrony, the 'best' priority function may be used). It turns out that we obtain a hierarchy of Petri net classes, where each class contains those nets which do not change their behaviour when transformed into the asynchronous version according to one of the interaction patterns. This is not surprising because later constructions allow a more fine-grained control over the interactions than earlier ones.

We did not consider connections from transitions to their postplaces as relevant to determine asynchrony and distributability. This is because we only discussed contact-free nets, where no synchronisation by postplaces is necessary. In the spirit of Definition 3.1 we could insert τ-transitions on any or all arcs from transitions to their postplaces, and the resulting net would always be equivalent to the original.

Although we compare the behaviour of a net and its asynchronous implementations in terms of failures equivalence, we believe that the very same classes of nets are obtained when using any other reasonable behavioural equivalence that respects branching time to some degree and abstracts from silent transitions—no matter if this is an interleaving equivalence, or one that respects causality. We would get larger classes of nets, for example for the case of full asynchrony including the net of Figure 2, if we merely required a net N and its implementation to be equivalent under a suitably chosen linear time equivalence. This option is investigated in [19].

The central results of the paper give semi-structural characterisations of our semantically defined classes of nets. Moreover, we relate these classes to well-known and well-understood structurally defined classes of nets, like free choice nets, extended free choice nets and simple nets.

To illustrate the potential interpretation of our results in other models of distributed systems, we give an example.

Message sequence charts (MSCs), also contained in UML 2.0 under the name sequence diagrams, are a model for specifying interactions between components (*instances*) of a system. A simple kind are *basic message sequence charts* (BMSCs) as defined in [13], where choices are not allowed. A Petri net semantics of BMCSs with asynchronous communication and a unique sending and receiving event for each message will yield Petri nets with unbranched places (see for instance [10]). Hence in this case the resulting Petri nets are conflict-free and therefore fully asynchronously

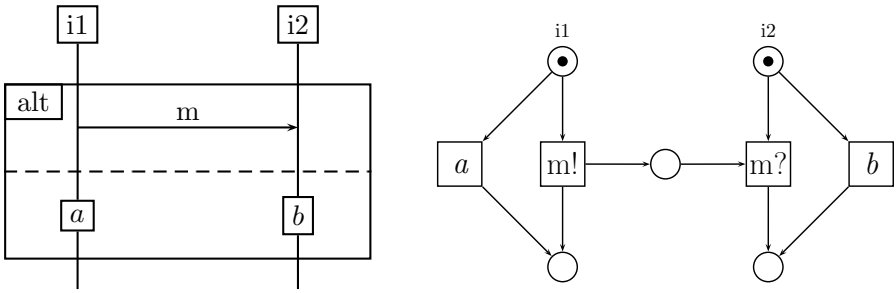


Fig. 11. An MSC and a potential implementation as Petri net, which has an N.

implementable according to Theorem 3.4.

However in extended versions of MSCs, e.g. in UML 2.0 or in live sequence charts (LSCs, see [11]), inline expressions allow to describe choices between possible behaviours in MSCs. Consider for example the MSC given in Figure 11 and a naive Petri net representation. The instances i1 and i2 can either communicate or execute their local actions. Obviously, this requires some mechanism in order to make sure that the choice is performed in a coherent way (see e.g. [7] for a discussion of this type of problem). In the Petri net representation, we find a reachable N, hence with Theorem 4.4 the net does not belong to the class $SA(B)$ of symmetrically asynchronously implementable nets. However, the net is M-free, and thus does belong to the class $AA(B)$ of asymmetrically asynchronously implementable nets. By giving priority to the collection of the message token (choosing the appropriate function g in our notion of implementation), it can be assured that instance i2 does not make the wrong choice and gets stuck (however it is still not clear whether the message will actually be consumed).

The obvious question is whether the naive Petri net interpretation we have given is conform with the intended semantics of the *alt*-construct (according to the informal UML semantics the alternatives always have to be executed completely; in LSCs it is specified explicitly whether messages are assured to arrive). However, on basis of a maybe more elaborate Petri nets semantics, it could be discussed what types of MSCs can be used to describe physically distributed systems, in particular which type of construct for choices is reasonable in this case.

Another model of reactive systems where we can transfer our results to are process algebras. When giving Petri net semantics to process algebras, it is an interesting question to investigate which classes of nets in our classification are obtained for certain types of operators or restricted languages, and to compare the results with results on language hierarchies (as summarised below).

We now give an overview on related work. A more extensive discussion is contained in [19]. We start by commenting on related work in Petri net theory.

The structural net classes we compare our constructions to were all taken from [3], where Eike Best and Mike Shields introduce various transformations between free choice nets, simple nets and extended variants thereof. They use “essential equivalence” to compare the behaviour of different nets, which they only give in-

formally. This equivalence is insensitive to divergence, which is also relied upon in their transformations. As observed in Footnote 1, it also does not preserve concurrency. They continue to show conditions under which liveness can be guaranteed for some of the classes.

In [1], Wil van der Aalst, Ekkart Kindler and Jörg Desel introduce two extensions to extended simple nets, by allowing self-loops to ignore the discipline imposed by the *ESPL*-requirement. This however assumes a kind of "atomicity" of self-loops, which we did not allow in this paper. In particular we do not implicitly assume that a transition will not change the state of a place it is connected to by a self-loop, since in case of deadlock, the temporary removal of a token from such a place might not be temporary indeed.

In [18] Wolfgang Reisig introduces a class of systems which communicate using buffers and where the relative speeds of different components are guaranteed to be irrelevant. The resulting nets are simple nets. He then proceeds introducing a decision procedure for the problem whether a marking exists which makes the complete system live.

The most similar work to our approach we have found is [12], where Richard Hopkins introduces the concept of *distributable* Petri Nets. These are defined in terms of *locality functions*, which assign to every transition t a set of possible machines or locations $L(t)$ on which t may be executed, subject to the restriction that a set of transitions with a common preplace must share a common machine. A plain net N is distributable iff for every locality function L that can be imposed on it, it has a "distributed implementation", a τ-net N' with the same set of visible transitions, in which each transition is assigned a specific location, subject to three restrictions:

- the location of a visible transition t is chosen from $L(t)$,
- transitions with a common preplace must have the same location
- and there exists a weak bisimulation between N and N', such that all τ-transitions involved in simulating a transition t from N reside on one of the locations $L(t)$.

The last clause enforces both a behavioural correspondence between N and N' and a structural one (through the requirement on locations). Thus, as in our work, the implementation is a τ-net that is required to be behaviourally equivalent to the original net. However, whereas we enforce particular implementations of an original net, Hopkins allows implementations which are quite elaborate and make informed decisions based upon global knowledge of the net. Consequently, his class of distributable nets is larger than our asynchronous net classes. As Hopkins notes, due to his use of interleaving semantics, his distributed implementations do not always display the same concurrent behaviour as the original nets, namely they add concurrency in some cases. This does not happen in our asynchronous implementations.

Another branch of related work is in the context of distributed algorithms. In [5] Luc Bougé considers the problem of implementing symmetric leader election in the sublanguages of CSP obtained by either allowing all guards, only input guards or no communication guards at all in guarded choice. He finds that the possibility

of implementing it depends heavily on the structure of the communication graphs, while truly symmetric schemes are only possible in CSP with input and output guards.

Quite a number of papers consider the question of synchronous versus asynchronous interaction in the realm of process algebras and the π-calculus. In [4] Frank de Boer and Catuscia Palamidessi consider various dialects of CSP with differing degrees of asynchrony. In particular, they consider CSP without output guards and CSP without any communication based guards. They also consider explicitly asynchronous variants of CSP where output actions cannot block, i.e. asynchronous sending is assumed. Similar work is done for the π-calculus in [17] by Catuscia Palamidessi, in [16] by Uwe Nestmann and in [9] by Dianele Gorla. A rich hierarchy of asynchronous π-calculi has been mapped out in these papers. Again mixed-choice, i.e. the ability to combine input and output guards in a single choice, plays a central role in the implementation of truly synchronous behaviour. It would be interesting to explore the possible connections between these languages and our net classes.

In [20], Peter Selinger considers labelled transition systems whose visible actions are partitioned into input and output actions. He defines asynchronous implementations of such a system by composing it with in- and output queues, and then characterises the systems that are behaviourally equivalent to their asynchronous implementations. The main difference with our approach is that we focus on asynchrony within a system, whereas Selinger focusses on the asynchronous nature of the communications of a system with the outside world.

Finally, there are approaches on hardware design where asynchronous interaction is an intriguing feature due to performance issues. For this, see the papers [14] and [15] by Leslie Lamport. In [15] he considers arbitration in hardware and outlines various arbitration-free "wait/signal" registers. He notes that nondeterminism is thought to require arbitration, but no proof is known. He concludes that only marked graphs can be implemented using these registers. Lamport then introduces "Or-Waiting", i.e. waiting for any of two signals, but has no model available to characterise the resulting processes. The used communication primitives bear a striking similarity to our symmetrically asynchronous nets.

References

[1] W.M.P. van der Aalst, E. Kindler & J. Desel (1998): *Beyond asymmetric choice: A note on some extensions. Petri Net Newsletter* 55, pp. 3–13.

[2] E. Best (1987): *Structure theory of Petri nets: The free choice hiatus.* In W. Brauer, W. Reisig & G. Rozenberg, editors: *Advances in Petri Nets 1986*, LNCS 254, Springer, pp. 168–206.

[3] E. Best & M.W. Shields (1983): *Some equivalence results for free choice nets and simple nets and on the periodicity of live free choice nets.* In G. Ausiello & M. Protasi, editors: Proceedings 8th Colloquium on *Trees in Algebra and Programming (CAAP '83)*, LNCS 159, Springer, pp. 141–154.

[4] F.S. de Boer & C. Palamidessi (1991): *Embedding as a tool for language comparison: On the CSP hierarchy.* In J.C.M. Baeten & J.F. Groote, editors: Proceedings 2nd International Conference on *Concurrency Theory* (CONCUR '91), Amsterdam, The Netherlands, LNCS 527, Springer, pp. 127–141.

[5] L. Bougé (1988): *On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. Acta Informatica* 25(2), pp. 179–201.

[6] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes.* *Journal of the ACM* 31(3), pp. 560–599.

[7] T. Gehrke, U. Goltz & H. Wehrheim (1999): *Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen.* In E. Schnieder, editor: Proceedings 6th Symposium on *Development and Operation of Complex Automation Systems*.

[8] R.J. van Glabbeek, U. Goltz & J.-W. Schicke (2008): *Symmetric and asymmetric asynchronous interaction.* Technical Report 2008-03, TU Braunschweig.

[9] D. Gorla (2006): *On the relative expressive power of asynchronous communication primitives.* In L. Aceto & A. Ingólfsdóttir, editors: Proceedings 9th International Conference on *Foundations of Software Science and Computation Structures* (FoSSaCS '06), LNCS 3921, Springer, pp. 47–62.

[10] P. Graubmann, E. Rudolph & J. Grabowski (1993): *Towards a petri net based semantics definition for message sequence charts.* In Proceedings 6th *SDL Forum* (SDL '93).

[11] D. Harel & R. Marelly (2003): *Come, Let's Play.* Springer.

[12] R.P. Hopkins (1991): *Distributable nets.* In *Advances in Petri Nets 1991*, LNCS 524, Springer, pp. 161–187.

[13] International Telecommunication Union (1996): *Message sequence chart.* Standard ITU-T Z.120.

[14] L. Lamport (1978): *Time, clocks, and the ordering of events in a distributed system.* *Communications of the ACM* 21(7), pp. 558–565.

[15] L. Lamport (2003): *Arbitration-free synchronization.* *Distributed Computing* 16(2-3), pp. 219–237.

[16] U. Nestmann (2000): *What is a 'good' encoding of guarded choice?* *Information and Computation* 156, pp. 287–319.

[17] C. Palamidessi (1997): *Comparing the expressive power of the synchronous and the asynchronous pi-calculus.* In Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on *Principles of Programming Languages (POPL '97)*, ACM Press, pp. 256–265.

[18] W. Reisig (1982): *Deterministic buffer synchronization of sequential processes.* *Acta Informatica* 18, pp. 115–134.

[19] J.-W. Schicke (2008): *Studienarbeit: Asynchronous Petri net classes.*

[20] P. Selinger (1997): *First-order axioms for asynchrony.* In Proceedings 8th International Conference on *Concurrency Theory* (CONCUR '97), Warsaw, Poland, LNCS 1243, Springer, pp. 376–390.

[21] W. Vogler (1993): *Bisimulation and action refinement.* *Theoretical Computer Science* 114(1), pp. 173–200.