On Distributability of Petri Nets
(extended abstract)[⋆]

Rob van Glabbeek[1,2], Ursula Goltz[3] and Jens-Wolfhard Schicke-Uffmann[3]

[1] NICTA, Sydney, Australia
[2] School of Computer Sc. and Engineering, University of New South Wales, Sydney, Australia
[3] Institute for Programming and Reactive Systems, TU Braunschweig, Germany

`rvg@cs.stanford.edu` `goltz@ips.cs.tu-bs.de` `drahflow@gmx.de`

Abstract. We formalise a general concept of distributed systems as sequential components interacting asynchronously. We define a corresponding class of Petri nets, called LSGA nets, and precisely characterise those system specifications which can be implemented as LSGA nets up to branching ST-bisimilarity with explicit divergence.

1 Introduction

The aim of this paper is to contribute to a fundamental understanding of the concept of a distributed reactive system and the paradigms of synchronous and asynchronous interaction. We start by giving an intuitive characterisation of the basic features of distributed systems. In particular we assume that distributed systems consist of components that reside on different locations, and that any signal from one component to another takes time to travel. Hence the only interaction mechanism between components is asynchronous communication.

Our aim is to characterise which system specifications may be implemented as distributed systems. In many formalisms for system specification or design, synchronous communication is provided as a basic notion; this happens for example in process algebras. Hence a particular challenge is that it may be necessary to simulate synchronous communication by asynchronous communication.

Trivially, any system specification may be implemented distributedly by locating the whole system on one single component. Hence we need to pose some additional requirements. One option would be to specify locations for system activities and then to ask for implementations satisfying this distribution and still preserving the behaviour of the original specification. This is done in [1]. Here we pursue a different approach. We add another requirement to our notion of a distributed system, namely that its components only allow sequential behaviour. We then ask whether an arbitrary system specification may be implemented as a distributed system consisting of sequential components in an optimal way, that is without restricting the concurrency of the original specification. This is a particular challenge when synchronous communication interacts with concurrency in the specification of the original system. We will give a precise characterisation of the class of distributable systems, which answers in particular under which conditions synchronous communication may be implemented in a distributed setting.

[⋆] This work was partially supported by the DFG (German Research Foundation).

For our investigations we need a model which is expressive enough to represent concurrency. It is also useful to have an explicit representation of the distributed state space of a distributed system, showing in particular the local control states of components. We choose Petri nets, which offer these possibilities and additionally allow finite representations of infinite behaviours. We work within the class of *structural conflict nets* [4] —a proper generalisation of the class of one-safe place/transition systems, where conflict and concurrency are clearly separated.

For comparing the behaviour of systems with their distributed implementation we need a suitable equivalence notion. Since we think of open systems interacting with an environment, and since we do not want to restrict concurrency in applications, we need an equivalence that respects branching time and concurrency to some degree. Our implementations use transitions which are invisible to the environment, and this should be reflected in the equivalence by abstracting from such transitions. However, we do not want implementations to introduce divergence. In the light of these requirements we work with two semantic equivalences. *Step readiness equivalence* is one of the weakest equivalences that captures branching time, concurrency and divergence to some degree; whereas *branching ST-bisimilarity with explicit divergence* fully captures branching time, divergence, and those aspects of concurrency that can be represented by concurrent actions overlapping in time. We obtain the same characterisation for both notions of equivalence, and thus implicitly for all notions in between these extremes.

We model distributed systems consisting of sequential components as an appropriate class of Petri nets, called *LSGA nets*. These are obtained by composing nets with sequential behaviour by means of an asynchronous parallel composition. We show that this class corresponds exactly to a more abstract notion of distributed systems, formalised as *distributed nets* [5].

We then consider distributability of system specifications which are represented as structural conflict nets. A net N is *distributable* if there exists a distributed implementation of N, that is a distributed net which is semantically equivalent to N. In the implementation we allow unobservable transitions, and labellings of transitions, so that single actions of the original system may be implemented by multiple transitions. However, the system specifications for which we search distributed implementations are *plain* nets without these features.

We give a precise characterisation of distributable nets in terms of a semi-structural property. This characterisation provides a formal proof that the interplay between choice and synchronous communication is a key issue for distributability.

2 Basic Notions

We consider here general labelled place/transition nets with arc weights. Arc weights are not necessary for the results of the paper, but are included for the sake of generality.

We will employ the following notations for multisets.

Definition 1. Let X be a set.
– A *multiset* over X is a function $A\colon X \rightarrow \mathbb{N}$, i.e. $A \in \mathbb{N}^X$.
– $x \in X$ is an *element of* a multiset $A \in \mathbb{N}^X$, notation $x \in A$, iff $A(x) > 0$.
– For multisets A and B over X we write $A \leq B$ iff $A(x) \leq B(x)$ for all $x \in X$;

$A + B$ denotes the multiset over X with $(A + B)(x) := A(x) + B(x)$,
$A\backslash B$ denotes the multiset over X with $(A - B)(x) := \max(A(x) - B(x), 0)$, and
for $k \in \mathbb{N}$ the multiset $k \cdot A$ is given by $(k \cdot A)(x) := k \cdot A(x)$.

– The function $\emptyset \colon X \to \mathbb{N}$, given by $\emptyset(x) := 0$ for all $x \in X$, is the *empty* multiset.
– If A is a multiset over X and $Y \subseteq X$ then $A \restriction Y$ denotes the multiset over Y defined by $(A \restriction Y)(x) := A(x)$ for all $x \in Y$.
– The cardinality $|A|$ of a multiset A over X is given by $|A| := \sum_{x \in X} A(x)$.
– A multiset A over X is *finite* iff $\{x \mid x \in A\}$ is finite, i.e., iff $|A| < \infty$.

Two multisets $A : X \to \mathbb{N}$ and $B : Y \to \mathbb{N}$ are *extensionally equivalent* iff $A \restriction (X \backslash Y) = \emptyset$, $B \restriction (Y \backslash X) = \emptyset$, and $A \restriction (X \cap Y) = B \restriction (X \cap Y)$. In this paper we often do not distinguish extensionally equivalent multisets. This enables us, for instance, to use $A + B$ even when A and B have different underlying domains.

A multiset A with $A(x) \in \{0, 1\}$ for all x is identified with the set $\{x \mid A(x) = 1\}$.

Definition 2. Let Act be a set of *visible actions* and $\tau \notin$ Act be an *invisible action*. A (*labelled*) *Petri net* (*over* Act $\dot\cup$ $\{\tau\}$) is a tuple $N = (S, T, F, M_0, \ell)$ where

– S and T are disjoint sets (of *places* and *transitions*),
– $F : (S \times T \cup T \times S) \to \mathbb{N}$ (the *flow relation* including *arc weights*),
– $M_0 : S \to \mathbb{N}$ (the *initial marking*), and
– $\ell : T \to$ Act $\dot\cup$ $\{\tau\}$ (the *labelling function*).

Petri nets are depicted by drawing the places as circles and the transitions as boxes, containing their label. Identities of places and transitions are displayed next to the net element. When $F(x, y) > 0$ for $x, y \in S \cup T$ there is an arrow (*arc*) from x to y, labelled with the *arc weight* $F(x, y)$. Weights 1 are elided. When a Petri net represents a concurrent system, a global state of this system is given as a *marking*, a multiset M of places, depicted by placing $M(s)$ dots (*tokens*) in each place s. The initial state is M_0.

To compress the graphical notation, we also allow universal quantifiers of the form $\forall x.\phi(x)$ to appear in the drawing (cf. Fig. 3). A quantifier replaces occurrences of x in element identities with all concrete values for which $\phi(x)$ holds, possibly creating a set of elements instead of the depicted single one. An arc of which only one end is replicated by a given quantifier results in a fan of arcs, one for each replicated element. If both ends of an arc are affected by the same quantifier, an arc is created between pairs of elements corresponding to the same x, but not between elements created due to differing values of x.

The behaviour of a Petri net is defined by the possible moves between markings M and M', which take place when a finite multiset G of transitions *fires*. In that case, each occurrence of a transition t in G consumes $F(s, t)$ tokens from each place s. Naturally, this can happen only if M makes all these tokens available in the first place. Next, each t produces $F(t, s)$ tokens in each s. Definition 4 formalises this notion of behaviour.

Definition 3. Let $N = (S, T, F, M_0, \ell)$ be a Petri net and $x \in S \cup T$.
The multisets ${}^\bullet x,\ x^\bullet : S \cup T \to \mathbb{N}$ are given by ${}^\bullet x(y) = F(y, x)$ and $x^\bullet(y) = F(x, y)$ for all $y \in S \cup T$. If $x \in T$, the elements of ${}^\bullet x$ and $x^\bullet$ are called *pre-* and *postplaces* of x, respectively. These functions extend to multisets $X : S \cup T \to \mathbb{N}$ as usual, by ${}^\bullet X := \sum_{x \in S \cup T} X(x) \cdot {}^\bullet x$ and $X^\bullet := \sum_{x \in S \cup T} X(x) \cdot x^\bullet$.

Definition 4. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $G \in \mathbb{N}^T$, G non-empty and finite, and $M, M' \in \mathbb{N}^S$. G is a *step* from M to M', written $M\ [G\rangle_N\ M'$, iff ${}^\bullet G \subseteq M$ (G is *enabled*) and $M' = (M \setminus {}^\bullet G) + G^\bullet$.

Note that steps are (finite) multisets, thus allowing self-concurrency, i.e. the same transition can occur multiple times in a single step.

In our nets transitions are labelled with *actions* drawn from a set $\mathrm{Act} \overset{\bullet}{\cup} \{\tau\}$. A transition t can be thought of as the occurrence of the action $\ell(t)$. If $\ell(t) \in \mathrm{Act}$, this occurrence can be observed and influenced by the environment, but if $\ell(t) = \tau$, it cannot and t is an *internal* or *silent* transition. Transitions whose occurrences cannot be distinguished by the environment carry the same label. In particular, since the environment cannot observe the occurrence of internal transitions at all, they are all labelled τ.

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

Definition 5. Let $N = (S, T, F, M_0, \ell)$ be a Petri net.
We write $M_1 \xrightarrow{\alpha}_N M_2$, for $\alpha \in \mathrm{Act} \overset{\bullet}{\cup} \{\tau\}$, when $\exists\, t \in T.\ \alpha = \ell(t) \wedge M_1\ [\{t\}\rangle_N\ M_2$.
Furthermore, for $a_1 a_2 \cdots a_n \in \mathrm{Act}^*$ we write $M_1 \xRightarrow{a_1 a_2 \cdots a_n}_N M_2$ when

$$M_1 \Longrightarrow_N \xrightarrow{a_1}_N \Longrightarrow_N \xrightarrow{a_2}_N \Longrightarrow_N \cdots \Longrightarrow_N \xrightarrow{a_n}_N \Longrightarrow_N M_2$$

where $\Longrightarrow_N$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}_N$.
For $\alpha \in \mathrm{Act} \overset{\bullet}{\cup} \{\tau\}$, we write $M_1 \xrightarrow{(\alpha)}_N M_2$ for $M_1 \xrightarrow{\alpha}_N M_2 \vee (\alpha = \tau \wedge M_1 = M_2)$, meaning that in case $\alpha = \tau$ performing a τ-transition is optional. We write $M_1 \xrightarrow{\alpha}_N$ for $\exists M_2. M_1 \xrightarrow{\alpha}_N M_2$, and $M_1 \not\xrightarrow{\alpha}_N$ for $\nexists M_2. M_1 \xrightarrow{\alpha}_N M_2$. Likewise $M_1[G\rangle_N$ abbreviates $\exists M_2. M_1[G\rangle_N M_2$. We omit the subscript N if clear from context.

Definition 6. Let $N = (S, T, F, M_0, \ell)$ be a Petri net.

– A marking $M \in \mathbb{N}^S$ is said to be *reachable in* N iff there is a $\sigma \in \mathrm{Act}^*$ such that $M_0 \xRightarrow{\sigma}_N M$. The set of all *reachable markings of* N is denoted by $[M_0\rangle_N$.
– N is *one-safe* iff $M \in [M_0\rangle_N \Rightarrow \forall x \in S.\, M(x) \leq 1$.
– The *concurrency relation* $\smile\ \subseteq T^2$ is given by $t \smile u \Leftrightarrow \exists M \in [M_0\rangle.\, M[\{t, u\}\rangle$.
– N is a *structural conflict net* iff for all $t, u \in T$ with $t \smile u$ we have ${}^\bullet t \cap {}^\bullet u = \emptyset$.

We use the term *plain nets* for Petri nets where ℓ is injective and no transition has the label τ, i.e. essentially unlabelled nets.

This paper first of all aims at studying finite Petri nets: nets with finitely many places and transitions. However, our work also applies to infinite nets with the properties that ${}^\bullet t \neq \emptyset$ for all transitions $t \in T$, and any reachable marking (a) is finite, and (b) enables only finitely many transitions. Henceforth, we call such nets *finitary*. Finitariness can be ensured by requiring $|M_0| < \infty \wedge \forall t \in T.\, {}^\bullet t \neq \emptyset \wedge \forall x \in S \cup T.\, |x^\bullet| < \infty$.

We use the following variant of readiness semantics [11] to compare behaviour.

Definition 7. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $\sigma \in \mathrm{Act}^*$ and $X \subseteq \mathbb{N}^{\mathrm{Act}}$.
$\langle \sigma, X \rangle$ is a *step ready pair* of N iff

$$\exists M. M_0 \xRightarrow{\sigma} M \wedge M \not\xrightarrow{\tau} \wedge X = \{\ell(G) \mid M[G\rangle\}.$$

Here we extend the labelling function ℓ to finite multisets of transitions elementwise.
We write $\mathscr{R}(N)$ for the set of all step ready pairs of N.
Two Petri nets N_1 and N_2 are *step readiness equivalent*, $N_1 \approx_{\mathscr{R}} N_2$, iff $\mathscr{R}(N_1) = \mathscr{R}(N_2)$.

ST-bisimilarity was proposed in [7] as a non-interleaved version of bisimilarity that respects causality to the extent that it can be expressed in terms of the possibility of durational actions to overlap in time. It was extended to a setting with internal actions in [15], based on the notion of *weak bisimilarity* of [10]. Here we apply the same idea, but based on *branching bisimilarity* [8], which unlike weak bisimilarity fully respects the branching structure of related systems.

An *ST-marking* of a net $N = (S, T, F, M_0, \ell)$ is a pair $(M, U) \in \mathbb{N}^S \times T^*$ of a normal marking, together with a sequence of transitions *currently firing*. The *initial* ST-marking is $\mathfrak{M}_\circ := (M_0, \varepsilon)$. The elements of $\text{Act}^\pm := \{a^+,\ a^{-n} \mid a \in \text{Act},\ n > 0\}$ are called *visible action phases*, and $Act^\pm_\tau := \text{Act}^\pm \stackrel{\bullet}{\cup} \{\tau\}$. For $U \in T^*$, we write $t \in^{(n)} U$ if t is the n^{th} element of U. Furthermore U^{-n} denotes U after removal of the n^{th} transition.

Definition 8. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, labelled over $\text{Act} \stackrel{\bullet}{\cup} \{\tau\}$.
The *ST-transition relations* $\xrightarrow{\eta}$ for $\eta \in \text{Act}^\pm_\tau$ between ST-markings are given by
$(M, U) \xrightarrow{a^+} (M', U')$ iff $\exists t \in T.\ \ell(t) = a \wedge M[\{t\}\rangle \wedge M' = M - {}^\bullet t \wedge U' = Ut$.
$(M, U) \xrightarrow{a^{-n}} (M', U')$ iff $\exists t \in^{(n)} U.\ \ell(t) = a \wedge U' = U^{-n} \wedge M' = M + t^\bullet$.
$(M, U) \xrightarrow{\tau} (M', U')$ iff $M \xrightarrow{\tau} M' \wedge U' = U$.

Now *branching ST-bisimilarity* is *branching bisimilarity* [8], applied to the labelled transition system made up of ST-markings of nets and the ST-transitions between them.

Definition 9. Two Petri nets N_1 and N_2 are *branching ST-bisimilar* iff there exists a relation $\mathcal{R}$ between the ST-markings of N_1 and N_2 such that, for all $\eta \in \text{Act}^\pm_\tau$:

1. $\mathfrak{M}_{\circ 1} \mathcal{R} \mathfrak{M}_{\circ 2}$;
2. if $\mathfrak{M}_1 \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}_1 \xrightarrow{\eta} \mathfrak{M}'_1$ then $\exists \mathfrak{M}^\dagger_2, \mathfrak{M}'_2$ such that
$\mathfrak{M}_2 \Longrightarrow \mathfrak{M}^\dagger_2 \xrightarrow{(\eta)} \mathfrak{M}'_2,\ \mathfrak{M}_1 \mathcal{R} \mathfrak{M}^\dagger_2$ and $\mathfrak{M}'_1 \mathcal{R} \mathfrak{M}'_2$;
3. if $\mathfrak{M}_1 \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}_2 \xrightarrow{\eta} \mathfrak{M}'_2$ then $\exists \mathfrak{M}^\dagger_1, \mathfrak{M}'_1$ such that
$\mathfrak{M}_1 \Longrightarrow \mathfrak{M}^\dagger_1 \xrightarrow{(\eta)} \mathfrak{M}'_1,\ \mathfrak{M}^\dagger_1 \mathcal{R} \mathfrak{M}_2$ and $\mathfrak{M}'_1 \mathcal{R} \mathfrak{M}'_2$.

If a system has the potential to engage in an infinite sequence of internal actions, one speaks of *divergence*. Branching bisimilarity *with explicit divergence* [8], is a variant of branching bisimilarity that fully respects the diverging behaviour of related systems. Since here we only compare systems of which one admits no divergence at all, the definition simplifies to the requirement that the other system may not diverge either. We write $N_1 \approx^\Delta_{bSTb} N_2$ iff N_1 and N_2 are branching ST-bisimilar with explicit divergence.

3 Distributed Systems

In this section, we stipulate what we understand by a distributed system, and subsequently formalise a model of distributed systems in terms of Petri nets.

- A distributed system consists of components residing on different locations.
- Components work concurrently.
- Interactions between components are only possible by explicit communications.
- Communication between components is time consuming and asynchronous.

Asynchronous communication is the only interaction mechanism in a distributed system for exchanging signals or information.

– The sending of a message happens always strictly before its receipt (there is a causal relation between sending and receiving a message).
– A sending component sends without regarding the state of the receiver; in particular there is no need to synchronise with a receiving component. After sending the sender continues its behaviour independently of receipt of the message.

As explained in the introduction, we will add another requirement to our notion of a distributed system, namely that its components only allow sequential behaviour.

Formally, we model distributed systems as nets consisting of component nets with sequential behaviour and interfaces in terms of input and output places.

Definition 10. Let $N = (S, T, F, M_0, \ell)$ be a Petri net, $I, O \subseteq S$, $I \cap O = \emptyset$ and $O^\bullet = \emptyset$.
1. (N, I, O) is a *component with interface* (I, O).
2. (N, I, O) is a *sequential* component with interface (I, O) iff $\exists Q \subseteq S \backslash (I \cup O)$ with $\forall t \in T. |{}^\bullet t \upharpoonright Q| = 1 \wedge |t^\bullet \upharpoonright Q| = 1$ and $|M_0 \upharpoonright Q| = 1$.

An input place $i \in I$ of a component $\mathcal{C}$ can be regarded as a mailbox of $\mathcal{C}$ for a specific type of messages. An output place $o \in O$, on the other hand, is an address outside $\mathcal{C}$ to which $\mathcal{C}$ can send messages. Moving a token into o is like posting a letter. The condition $o^\bullet = \emptyset$ says that a message, once posted, cannot be retrieved by the component.

A set of places like Q above is called an S*-invariant*. The requirements guarantee that the number of tokens in these places remains constant, in this case 1. It follows that no two transitions can ever fire concurrently (in one step). Conversely, whenever a net is sequential, in the sense that no two transitions can fire in one step, it is easily converted into a behaviourally equivalent net with the required S-invariant, namely by adding a single marked place with a self-loop to all transitions. This modification preserves virtually all semantic equivalences on Petri nets from the literature, including $\approx^{\Delta}_{bSTb}$.

Next we define an operator for combining components with asynchronous communication by fusing input and output places.

Definition 11. Let $\mathfrak{K}$ be an index set.
Let $((S_k, T_k, F_k, M_{0k}, \ell_k), I_k, O_k)$ with $k \in \mathfrak{K}$ be components with interface such that $(S_k \cup T_k) \cap (S_l \cup T_l) = (I_k \cup O_k) \cap (I_l \cup O_l)$ for all $k, l \in \mathfrak{K}$ with $k \neq l$ (components are disjoint except for interface places) and moreover $I_k \cap I_l = \emptyset$ for all $k, l \in \mathfrak{K}$ with $k \neq l$ (mailboxes cannot be shared; the recipient of a message is always unique).
Then the *asynchronous parallel composition* of these components is defined by

$$\Big\|_{i \in \mathfrak{K}} ((S_k, T_k, F_k, M_{0k}, \ell_k), I_k, O_k) = ((S, T, F, M_0, \ell), I, O)$$

with $S = \bigcup_{k \in \mathfrak{K}} S_k,\ T = \bigcup_{k \in \mathfrak{K}} T_k,\ F = \bigcup_{k \in \mathfrak{K}} F_k,\ M_0 = \sum_{k \in \mathfrak{K}} M_{0k},\ \ell = \bigcup_{k \in \mathfrak{K}} \ell_k$ (componentwise union of all nets), $I = \bigcup_{k \in \mathfrak{K}} I_k$ (we accept additional inputs from outside), and $O = \bigcup_{k \in \mathfrak{K}} O_k \setminus \bigcup_{k \in \mathfrak{K}} I_k$ (once fused with an input, $o \in O_I$ is no longer an output).

Observation 1. $\|$ is associative.

This follows directly from the associativity of the (multi)set union operator. □
We are now ready to define the class of nets representing systems of asynchronously communicating sequential components.

Definition 12. A Petri net N is an *LSGA net* (a *locally sequential globally asynchronous net*) iff there exists an index set $\mathfrak{K}$ and sequential components with interface $\mathcal{C}_k,\ k \in \mathfrak{K}$, such that $(N, I, O) = \|_{k \in \mathfrak{K}} \mathcal{C}_k$ for some I and O.

Up to $\approx^{\Delta}_{bSTb}$—or any reasonable equivalence preserving causality and branching time but abstracting from internal activity—the same class of LSGA systems would have been obtained if we had imposed, in Def. 10, that I, O and Q form a partition of S and that ${}^\bullet I = \emptyset$. However, it is essential that our definition allows multiple transitions of a component to read from the same input place.

In the remainder of this section we give a more abstract characterisation of Petri nets representing distributed systems, namely as *distributed* Petri nets, which we introduced in [5]. This will be useful in Section 4, where we investigate distributability using this more semantic characterisation. We show below that the concrete characterisation of distributed systems as LSGA nets and this abstract characterisation agree.

Following [1], to arrive at a class of nets representing distributed systems, we associate *localities* to the elements of a net $N = (S, T, F, M_0, \ell)$. We model this by 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 $D(x) = D(y)$.

Following [5], we impose a fundamental restriction on distributions, namely that when two transitions can occur in one step, they cannot be co-located. This reflects our assumption that at a given location actions can only occur sequentially.

In [5] we observed that Petri nets incorporate a notion of synchronous interaction, in that a transition can fire only by synchronously taking the tokens from all of its preplaces. In general the behaviour of a net would change radically if a transition would take its input tokens one by one—in particular deadlocks may be introduced. Therefore we insist that in a distributed Petri net, a transition and all its input places reside on the same location. There is no reason to require the same for the output places of a transition, for the behaviour of a net would not change significantly if transitions were to deposit their output tokens one by one [5].

This leads to the following definition of a distributed Petri net.

Definition 13 ([5]). A Petri net $N = (S, T, F, M_0, \ell)$ is *distributed* iff there exists a distribution D such that

(1) $\forall s \in S,\ t \in T.\ s \in {}^\bullet t \Rightarrow t \equiv_D s$,

(2) $\forall t, u \in T.\ t \smile u \Rightarrow t \not\equiv_D u$.

N is *essentially distributed* if (2) is weakened to $\forall t, u \in T.\ t \smile u \wedge \ell(t) \neq \tau \Rightarrow t \not\equiv_D u$.

A typical example of a net which is not distributed is shown in Fig. 1 on Page 9. Transitions t and v are concurrently executable and hence should be placed on different locations. However, both have preplaces in common with u which would enforce putting all three transitions on the same location. In fact, distributed nets can be characterised in the following semi-structural way.

Observation 2. *A Petri 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 $i = 1, \ldots, n$.* □

It turns out that the classes of LSGA nets and distributable nets essentially coincide. Moreover, up to $\approx^{\Delta}_{bSTb}$ these classes also coincide with the more liberal notion of essentially distributed nets, permitting concurrency of internal transitions at the same location. We will make use of that in proving our main theorem.

Theorem 1. *Any LSGA net is distributed, and for any essentially distributed net N there is an LSGA net N' with $N' \approx^{\Delta}_{bSTb} N$.*

Proof. In the full version of this paper [6]. □

Observation 3. *Every distributed Petri net is a structural conflict net.* □

Corollary 1. *Every LSGA net is a structural conflict net.* □

4 Distributable Systems

We now consider Petri nets as specifications of concurrent systems and ask the question which of those specifications can be implemented as distributed systems. This question can be formalised as

Which Petri nets are semantically equivalent to distributed nets?

Of course the answer depends on the choice of a suitable semantic equivalence. Here we will answer this question using the two equivalences introduced in Section 2. We will give a precise characterisation of those nets for which we can find semantically equivalent distributed nets. For the negative part of this characterisation, stating that certain nets are not distributable, we will use step readiness equivalence, which is one of the simplest and least discriminating equivalences imaginable that abstracts from internal actions, but preserves branching time, concurrency and divergence to some small degree. As explained in [5], giving up on any of these latter three properties would make any Petri net distributable, but in a rather trivial and unsatisfactory way. For the positive part, namely that all other nets are indeed distributable, we will use the most discriminating equivalence for which our implementation works, namely branching ST-bisimilarity with explicit divergence, which is finer than step readiness equivalence. Hence we will obtain the strongest possible results for both directions and it turns out that the concept of distributability is fairly robust w.r.t. the choice of a suitable equivalence: any equivalence notion between step readiness equivalence and branching ST-bisimilarity with explicit divergence will yield the same characterisation.

Definition 14. A Petri net N is *distributable* up to an equivalence $\approx$ iff there exists a distributed net N' with $N' \approx N$.

Formally we give our characterisation of distributability by classifying which finitary plain structural conflict nets can be implemented as distributed nets, and hence as LSGA nets. In such implementations, we use invisible transitions. We study the concept "distributable" for plain nets only, but in order to get the largest class possible we allow non-plain implementations, where a given transition may be split into multiple transitions carrying the same label.

It is well known that sometimes a global protocol is necessary to implement synchronous interaction present in system specifications. In particular, this may be needed for deciding choices in a coherent way, when these choices require agreement of multiple components. The simple net in Fig. 1 shows a typical situation of this kind. Independent decisions of the two choices might lead to a deadlock. As remarked in [5], for this particular net there exists no satisfactory distributed implementation that fully respects the reactive behaviour of the original system. Indeed such M-structures, representing interference between concurrency and choice, turn out to play a crucial rôle for characterising distributability.

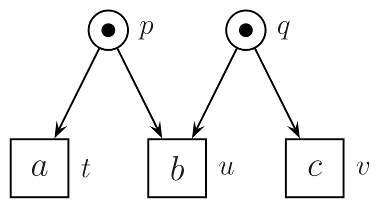

Fig. 1. A fully marked M.

Definition 15. Let $N = (S, T, F, M_0, \ell)$ be a Petri net. N has a *fully reachable pure* M iff $\exists t, u, v \in T. {}^\bullet t \cap {}^\bullet u \neq \emptyset \wedge {}^\bullet u \cap {}^\bullet v \neq \emptyset \wedge {}^\bullet t \cap {}^\bullet v = \emptyset \wedge \exists M \in [M_0\rangle. {}^\bullet t \cup {}^\bullet u \cup {}^\bullet v \subseteq M$.

Note that Definition 15 implies that $t \neq u$, $u \neq v$ and $t \neq v$.

We now give an upper bound on the class of distributable nets by adopting a result from [5].

Theorem 2. *Let N be a plain structural conflict Petri net. If N has a fully reachable pure M, then N is not distributable up to step readiness equivalence.*

Proof. In [5] this theorem was obtained for plain one-safe nets.[4] The proof applies verbatim to plain structural conflict nets as well. □

Since $\approx^{\Delta}_{bSTb}$ is finer than $\approx_{\mathscr{R}}$, this result holds also for distributability up to $\approx^{\Delta}_{bSTb}$ (and any equivalence between $\approx_{\mathscr{R}}$ and $\approx^{\Delta}_{bSTb}$).

In the following, we establish that this upper bound is tight, and hence a finitary plain structural conflict net is distributable iff it has no fully reachable pure M. For this, it is helpful to first introduce macros in Petri nets for reversibility of transitions.

4.1 Petri nets with reversible transitions

A *Petri net with reversible transitions* generalises the notion of a Petri net; its semantics is given by a translation to an ordinary Petri net, thereby interpreting the reversible transitions as syntactic sugar for certain net fragments. It is defined as a tuple $(S, T, \Omega, \imath, F, M_0, \ell)$ with S a set of places, T a set of (reversible) transitions, labelled by $\ell : T \rightarrow \text{Act} \dot\cup \{\tau\}$, Ω a set of *undo interfaces* with the relation $\imath \subseteq \Omega \times T$ linking interfaces to transitions, $M_0 \in \mathbb{N}^S$ an initial marking, and

$$F\colon (S \times T \times \{\mathit{in},\ \mathit{early},\ \mathit{late},\ \mathit{out},\ \mathit{far}\} \rightarrow \mathbb{N})$$

the flow relation. For $t \in T$ and $\mathit{type} \in \{\mathit{in},\ \mathit{early},\ \mathit{late},\ \mathit{out},\ \mathit{far}\}$, the multiset of places $t^{\mathit{type}} \in \mathbb{N}^S$ is given by $t^{\mathit{type}}(s) = F(s, t, \mathit{type})$. When $s \in t^{\mathit{type}}$ for $\mathit{type} \in \{\mathit{in},\ \mathit{early},\ \mathit{late}\}$, the place s is called a *preplace* of t of type type; when $s \in t^{\mathit{type}}$ for $\mathit{type} \in \{\mathit{out},\ \mathit{far}\}$, s is called a *postplace* of t of type type. For each undo interface $\omega \in \Omega$ and transition t with $\imath(\omega, t)$ there must be places $\mathsf{undo}_\omega(t)$, $\mathsf{reset}_\omega(t)$ and $\mathsf{ack}_\omega(t)$ in S. A transition with a nonempty set of interfaces is called *reversible*; the other (*standard*) transitions may have pre- and postplaces of types in and out only—for these transitions $t^{\mathit{in}} = {}^\bullet t$ and $t^{\mathit{out}} = t^\bullet$. In case $\Omega = \emptyset$, the net is just a normal Petri net.

[4] In [5] the theorem was claimed and proven only for plain nets with a fully reachable *visible* pure M; however, for plain nets the requirement of visibility is irrelevant.

A global state of a Petri net with reversible transitions is given by a marking $M \in \mathbb{N}^S$, together with the state of each reversible transition "currently in progress". Each transition in the net can fire as usual. A reversible transition can moreover take back (some of) its output tokens, and be *undone* and *reset*. When a transition t fires, it consumes $\sum_{type\in\{in,\ early,\ late\}} F(s,t,type)$ tokens from each of its preplaces s and produces $\sum_{type\in\{out,\ far\}} F(s,t,type)$ tokens in each of its postplaces s. A reversible transition t that has fired can start its reversal by consuming a token from $\mathsf{undo}_\omega(t)$ for one of its interfaces ω. Subsequently, it can take back one by one a token from its postplaces of type *far*. After it has retrieved all its output of type *far*, the transition is undone, thereby returning $F(s,t,early)$ tokens in each of its preplaces s of type *early*. Afterwards, by consuming a token from $\mathsf{reset}_\omega(t)$, for the same interface ω that started the undo-process, the transition terminates its chain of activities by returning $F(s,t,late)$ tokens in each of its *late* preplaces s. At that occasion it also produces a token in $\mathsf{ack}_\omega(t)$. Alternatively, two tokens in $\mathsf{undo}_\omega(t)$ and $\mathsf{reset}_\omega(t)$ can annihilate each other without involving the transition t; this also produces a token in $\mathsf{ack}_\omega(t)$. The latter mechanism comes in action when trying to undo a transition that has not yet fired.

Fig. 2 shows the translation of a reversible transition t with $\ell(t) = a$ into a ordinary net fragment. The arc weights on the green (or grey) arcs are inherited from the untranslated net; the other arcs have weight 1.

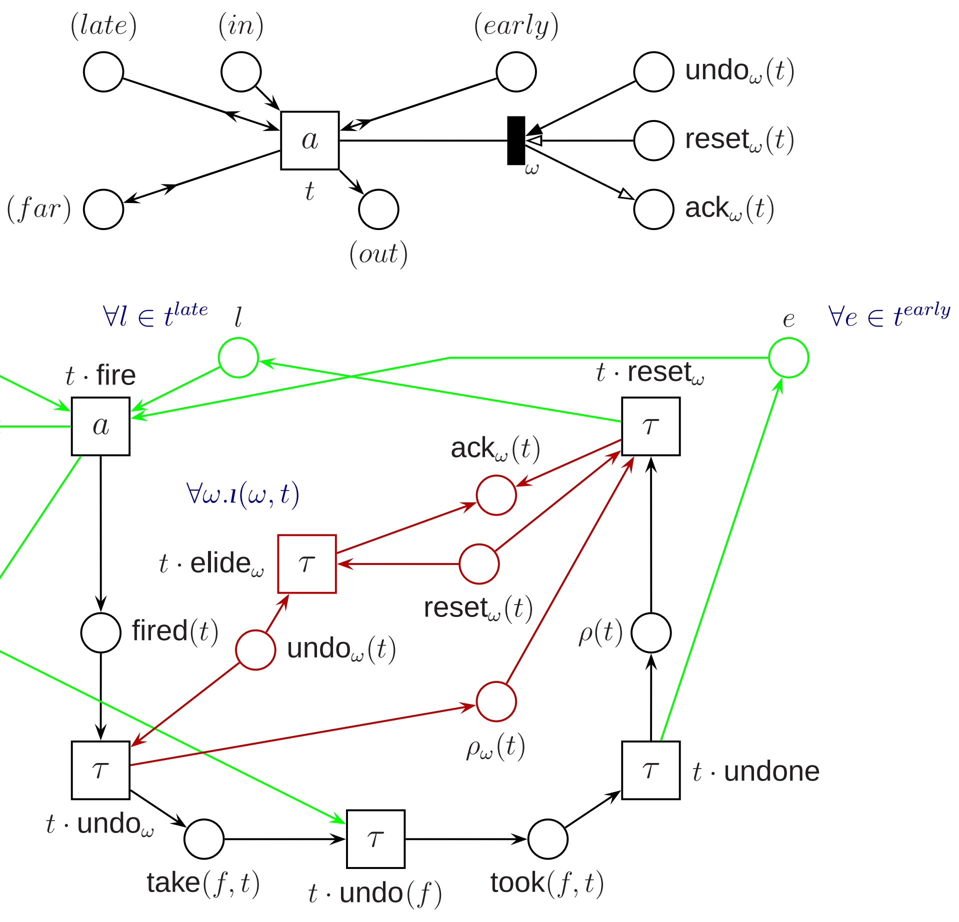

Fig. 2. A reversible transition and its macro expansion.

4.2 The conflict replicating implementation

Now we establish that a finitary plain structural conflict net that has no fully reachable pure M is distributable. We do this by proposing the *conflict replicating implementation* of any such net, and show that this implementation is always (a) essentially distributed, and (b) equivalent to the original net. In order to get the strongest possible result, for (b) we use branching ST-bisimilarity with explicit divergence.

To define the conflict replicating implementation of a net $N = (S, T, F, M_0, \ell)$ we fix an arbitrary well-ordering $<$ on its transitions. We let b, c, h, i, j, k, l range over these ordered transitions, and write

– $i \# j$ iff $i \neq j \wedge {}^\bullet i \cap {}^\bullet j \neq \emptyset$ (transitions i and j are *in conflict*), and $i \overset{\#}{=} j$ iff $i \# j \vee i = j$,
– $i <^\# j$ iff $i < j \wedge i \# j$, and $i \leq^\# j$ iff $i <^\# j \vee i = j$.

Fig. 3 shows the conflict replicating implementation N. It is presented as a Petri net $\mathcal{I}(N) = (S', T', F', \Omega, \imath, M'_0, \ell')$ with reversible transitions. The set Ω of undo interfaces (not drawn) is T, and for $i \in \Omega$ we have $\imath(i, t)$ iff $t \in \Omega_i$, where the sets of transitions $\Omega_i \in \mathbb{N}^{T'}$ are specified in Fig. 3. The implementation $\mathcal{I}(N)$ inherits the places of N (i.e. $S' \supseteq S$), and we postulate that $M'_0 \upharpoonright S = M_0$. Given this, Fig. 3 is not merely an illustration of $\mathcal{I}(N)$—it provides a complete and accurate description of it, thereby defining the conflict replicating implementation of any net. In interpreting this figure it is important to realise that net elements are completely determined by their name (identity), and exist only once, even if they show up multiple times in the figure. For instance, the place $\pi_{h\#j}$ with $h{=}2$ and $j{=}5$ (when using natural numbers for the transitions in T) is the same as the place $\pi_{j\#l}$ with $j{=}2$ and $l{=}5$; it is a standard preplace of $\mathsf{execute}^i_2$ (for all $i \leq^\# 2$), a standard postplace of $\mathsf{fetched}^i_2$, as well as a late preplace of $\mathsf{transfer}^2_5$.

The rôle of the transitions $\mathsf{distribute}_p$ for $p \in S$ is to distribute a token in p to copies p_j of p in the localities of all transitions $j \in T$ with $p \in {}^\bullet j$. In case j is enabled in N, the transition $\mathsf{initialise}_j$ will become enabled in $\mathcal{I}(N)$. These transitions put tokens in the places pre^j_k, which are preconditions for all transitions $\mathsf{execute}^j_k$, which model the execution of j at the location of k. When two conflicting transitions h and j are both enabled in N, the first steps $\mathsf{initialise}_h$ and $\mathsf{initialise}_j$ towards their execution in $\mathcal{I}(N)$ can happen in parallel. To prevent them from executing both, $\mathsf{execute}^j_j$ (of j at its own location) is only possible after $\mathsf{transfer}^h_j$, which disables $\mathsf{execute}^h_h$.

The main idea behind the conflict replicating implementation is that a transition $h \in T$ is primarily executed by a sequential component of its own, but when a conflicting transition j gets enabled, the sequential component implementing j may "steal" the possibility to execute h from the home component of h, and keep the options to do h and j open until one of them occurs. To prevent h and j from stealing each other's initiative, which would result in deadlock, a global asymmetry is built in by ordering the transitions. Transition j can steal the initiative from h only when $h < j$.

In case j is also in conflict with a transition l, with $j < l$, the initiative to perform j may subsequently be stolen by l. In that case either h and l are in conflict too—then l takes responsibility for the execution of h as well—or h and l are concurrent—in that case h will not be enabled, due to the absence of fully reachable pure Ms in N. The absence of fully reachable pure Ms also guarantees that it cannot happen that two concurrent transitions j and k both steal the initiative from an enabled transition h.

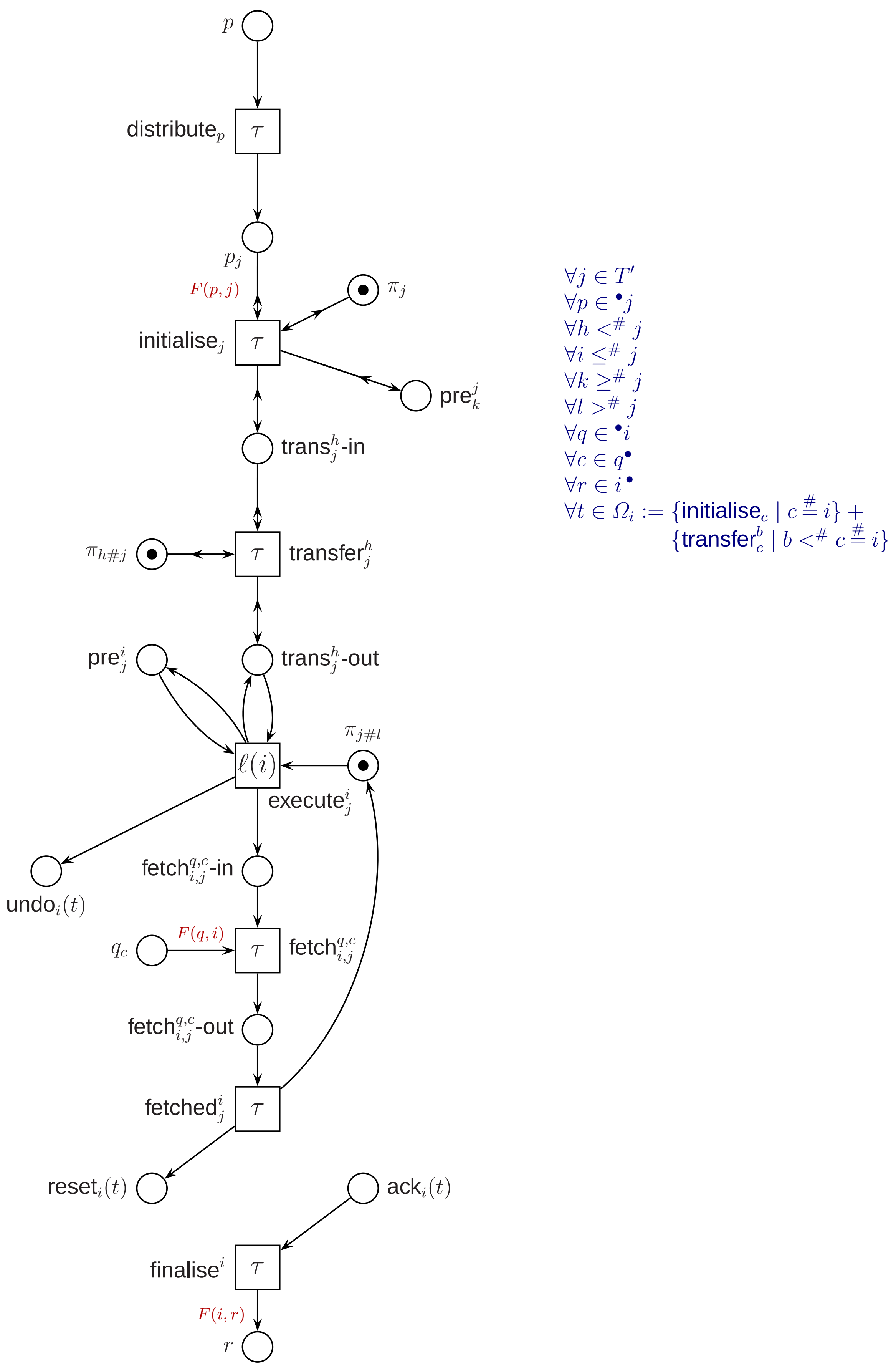

Fig. 3. The conflict replicating implementation

After the firing of $\mathsf{execute}^i_j$ all tokens that were left behind in the process of carefully orchestrating this firing will have to be cleaned up, to prepare the net for the next activity in the same neighbourhood. This is the reason for the reversibility of the transitions preparing the firing of $\mathsf{execute}^i_j$. Hence there is an undo interface for each transition $i \in T'$, cleaning up the mess made in preparation of firing $\mathsf{execute}^i_j$ for some j. Ω_i is the multiset of all transitions t that could possibly have contributed to this. For each of them its interface i is activated, by $\mathsf{execute}^i_j$ depositing a token in $\mathsf{undo}_i(t)$. When all preparatory transitions that have fired are undone, tokens appear in the places p_c for all $p \in {}^\bullet i$ and $c \in p^\bullet$. These are collected by $\mathsf{fetch}^{p,c}_{i,j}$, after which all $t \in \Omega_i$ get a reset signal. Those that have fired and were undone are reset, and those that never fired perform $\mathsf{elide}_i(t)$. In either case a token appears in $\mathsf{ack}_i(t)$. These are collected by $\mathsf{finalise}^i$, which finishes the execution of i by depositing tokens in its postplaces.

Proposition 1. *$\mathcal{I}(N)$ is essentially distributed for every Petri net N.*

Proof sketch. We take the *canonical* distribution D of N, in which $\equiv_D$ is the equivalence relation on places and transitions *generated* by Condition (1) of Definition 13. We need to show that D satisfies the weakened Condition 2. Any location that harbours an external transition $\mathsf{execute}^i_j$ for some $i \leq j \in T'$, also harbours $\mathsf{initialise}_j{\cdot}\mathsf{undo}(\mathsf{pre}^i_j)$, $\mathsf{transfer}^h_j{\cdot}\mathsf{undo}(\mathsf{trans}^h_j\text{-}\mathsf{out})$ for all $h <^\# j$, $\mathsf{execute}^i_j$ for all $i \leq^\# j$, and, for all $l >^\# j$, $\mathsf{transfer}^j_l{\cdot}\mathsf{fire}$ and $\mathsf{initialise}_l{\cdot}\mathsf{undo}(\mathsf{trans}^j_l\text{-}\mathsf{in})$. In [6] we show that none of these transitions can happen concurrently with $\mathsf{execute}^i_j$. ■

Theorem 3. *Let N be a finitary plain structural conflict net without a fully reachable pure M. Then N is distributable up to $\approx^\Delta_{bSTb}$.*

Proof. In the full version of this paper [6]. There we show that $\mathcal{I}(N) \approx^\Delta_{bSTb} N$. Hence $\mathcal{I}(N)$ is a essentially distributed implementation of N. Now apply Theorem 1. □

Given the complexity of our construction, no techniques known to us were adequate for performing this proof. We therefore had to develop an entirely new method for rigorously proving the equivalence of two Petri nets up to $\approx^\Delta_{bSTb}$, one of which known to be plain. This method is presented in [6].

Corollary 2. *Let N be a finitary plain structural conflict net. Then N is distributable iff it has no fully reachable pure M.* □

5 Conclusion

In this paper, we have given a precise characterisation of distributable Petri nets in terms of a semi-structural property. Moreover, we have shown that our notion of distributability corresponds to an intuitive notion of a distributed system by establishing that any distributable net may be implemented as a network of asynchronously communicating components.

In order to formalise what qualifies as a valid implementation, we needed a suitable equivalence relation. We have chosen step readiness equivalence for showing the impossibility part of our characterisation, since it is one of the simplest and least discriminating semantic equivalences imaginable that abstracts from internal actions but preserves branching time, concurrency and divergence to some small degree. For the positive part, stating that all other nets are implementable, we have introduced a combination of several well known rather discriminating equivalences, namely a divergence sensitive version of branching bisimulation adapted to ST-semantics. Hence our characterisation is rather robust against the chosen equivalence; it holds in fact for all equivalences between these two notions. However, ST-equivalence (and our version of it) preserves the causal structure between action occurrences only as far as it can be expressed in terms of the possibility of durational actions to overlap in time. Hence a natural question is whether we could have chosen an even stronger causality sensitive equivalence for our implementability result, respecting e.g. pomset equivalence or history preserving bisimulation. Our conflict replicating implementation does not fully preserve the causal behaviour of nets; we are convinced we have chosen the strongest possible equivalence for which our implementation works. It is an open problem to find a class of nets that can be implemented distributedly while preserving divergence, branching time and causality in full. Another line of research is to investigate which Petri nets can be implemented as distributed nets when relaxing the requirement of preserving the branching structure. If we allow linear time correct implementations (using a step trace equivalence), we conjecture that all Petri nets become distributable. However, also in this case it is problematic, in fact even impossible in our setting, to preserve the causal structure, as has been shown in [14]. A similar impossibility result has been obtained in the world of the π-calculus in [12].

The interplay between choice and synchronous communication has already been investigated in quite a number of approaches in different frameworks. We refer to [5] for a rather comprehensive overview and concentrate here on recent and closely related work.

The idea of modelling asynchronously communicating sequential components by sequential Petri nets interacting though buffer places has already been considered in [13]. There Wolfgang Reisig introduces a class of systems, represented as Petri nets, where the relative speeds of different components are guaranteed to be irrelevant. His class is a strict subset of our LSGA nets, requiring additionally, amongst others, that all choices in sequential components are free, i.e. do not depend upon the existence of buffer tokens, and that places are output buffers of only one component. Another quite similar approach was taken in [3], where transition labels are classified as being either input or output. There, asynchrony is introduced by adding new buffer places during net composition. This framework does not allow multiple senders for a single receiver.

Other notions of distributed and distributable Petri nets are proposed in [9,1,2]. In these works, given a distribution of the transitions of a net, the net is distributable iff it can be implemented by a net that is distributed w.r.t. that distribution. The requirement that concurrent transitions may not be co-located is absent; given the fixed distribution, there is no need for such a requirement. These papers differ from each other, and from ours, in what counts as a valid implementation. A comparison of our criterion with that of Hopkins [9] is provided in [5].

In [5] we have obtained a characterisation similar to Corollary 2, but for a much more restricted notion of distributed implementation (*plain distributability*), disallowing nontrivial transition labellings in distributed implementations. We also proved that fully reachable pure Ms are not implementable in a distributed way, even when using transition labels (Theorem 2). However, we were not able to show that this upper bound on the class of distributable systems was tight. Our current work implies the validity of Conjecture 1 of [5]. While in [5] we considered only one-safe place/transition systems, the present paper employs a more general class of place/transition systems, namely structural conflict nets. This enables us to give a concrete characterisation of distributed nets as systems of sequential components interacting via non-safe buffer places.

References

1. E. Badouel, B. Caillaud & P. Darondeau (2002): *Distributing Finite Automata Through Petri Net Synthesis*. *Formal Aspects of Computing* 13(6), pp. 447–470.
2. E. Best & Ph. Darondeau (2011): *Petri Net Distributability*. In: *Proceedings Ershov Informatics Conference (PSI'11), Novosibirsk, Russia*, LNCS, Springer. To appear.
3. D. El Hog-Benzina, S. Haddad & R. Hennicker (2010): *Process Refinement and Asynchronous Composition with Modalities*. In N. Sidorova & A. Serebrenik, editors: Proceedings of the 2nd International Workshop on *Abstractions for Petri Nets and Other Models of Concurrency* (APNOC'10), Braga, Portugal. Available at `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/EHH-apnoc10.pdf`.
4. R.J. van Glabbeek, U. Goltz & J.-W. Schicke (2011): *Abstract Processes of Place/Transition Systems*. *Information Processing Letters* 111(13), pp. 626 – 633, doi:`10.1016/j.ipl.2011.03.013`.
5. R.J. van Glabbeek, U. Goltz & J.-W. Schicke (2008): *On Synchronous and Asynchronous Interaction in Distributed Systems*. In E. Ochmański & J. Tyszkiewicz, editors: *Mathematical Foundations of Computer Science 2008*, LNCS 5162, Springer, pp. 16–35, doi:`10.1007/978-3-540-85238-4_2`. Full version available as Technical Report 2008-03, TU-Braunschweig; `http://arxiv.org/abs/0901.0048`.
6. R.J. van Glabbeek, Goltz U & J.-W. Schicke-Uffmann (2011): *On Distributability of Petri Nets*. Technical Report 2011-10, TU Braunschweig. Available at `http://theory.stanford.edu/~rvg/abstracts.html#95`. Full version of this paper, to appear.
7. R.J. van Glabbeek & F.W. Vaandrager (1987): *Petri net models for algebraic theories of concurrency (extended abstract)*. In: Proc. *PARLE '87*, LNCS 259, Springer, pp. 224–242.
8. R.J. van Glabbeek & W.P. Weijland (1996): *Branching Time and Abstraction in Bisimulation Semantics*. *Journal of the ACM* 43(3), pp. 555–600, doi:`10.1145/233551.233556`.
9. R.P. Hopkins (1991): *Distributable nets*. In: *Advances in Petri Nets 1991*, LNCS 524, Springer, pp. 161–187, doi:`10.1007/BFb0019974`.
10. R. Milner (1989): *Communication and Concurrency*. Prentice Hall, Englewood Cliffs.
11. E.-R. Olderog & C.A.R. Hoare (1986): *Specification-oriented semantics for communicating processes*. *Acta Informatica* 23, pp. 9–66, doi:`10.1007/BF00268075`.
12. K. Peters, J.-W. Schicke & U. Nestmann (2011): *Synchrony vs Causality in the Asynchronous Pi-Calculus*. In B. Luttik & F. Valencia, editors: Proceedings 18th International Workshop on *Expressiveness in Concurrency,* Aachen, Germany, 5th September 2011, *Electronic Proceedings in Theoretical Computer Science* 64, pp. 89–103, doi:`10.4204/EPTCS.64.7`.
13. W. Reisig (1982): *Deterministic Buffer Synchronization of Sequential Processes*. *Acta Informatica* 18, pp. 115–134, doi:`10.1007/BF00264434`.
14. J.-W. Schicke, K. Peters & U. Goltz (2011): *Synchrony vs. Causality in Asynchronous Petri Nets*. In B. Luttik & F. Valencia, editors: Proceedings 18th International Workshop on *Expressiveness in Concurrency,* Aachen, Germany, 5th September 2011, *Electronic Proceedings in Theoretical Computer Science* 64, pp. 119–131, doi:`10.4204/EPTCS.64.9`.
15. W. Vogler (1993): *Bisimulation and Action Refinement*. *Theor. Comput. Sci.* 114(1), pp. 173–200, doi:`10.1016/0304-3975(93)90157-O`.