

Petri Net Distributability

Eike Best and Philippe Darondeau

Carl von Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany

INRIA Rennes Bretagne-Atlantique, campus de Beaulieu, F-35042 Rennes Cedex


Paris, September 2011

Purpose of the talk

Highlight some important open problems that have (in our opinion) been slightly neglected by the Petri net / process algebra communities.

Main concern: how can a Petri net (or, for that matter, a concurrent system described by other means) be **distributed**.

Setting the scene 1: nondeterministic choice

$a + b$ (CCS) or a, b (COSY) or  (Petri nets)

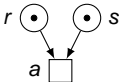
Can a and b be on different locations A and B ?

Information about the *start* of executing a must travel in time 0 from A to B . Otherwise there is some small lapse of time during which a has started to occur and b can start to occur.

Hence our answer is **no**.

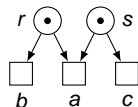
Transitions a and b must be on the **same** location, which is also the location of s .

Setting the scene 2: handshake synchronisation

$a|\hat{a}$ (CCS) or $a||a$ (COSY) or  (Petri nets)

Can r and s be on different locations R and S ?

Not if the context involves nondeterminism, as in



By the previous argument, b , a , c and also r , s must be on the same location.

In general, r , s might be on different locations.

Hence our answer is: **it depends**.

Setting the scene 3: informal problem statement

Given the above constraints,
and given a system with a desired localisation,
is it actually possible to realise the system distributedly?

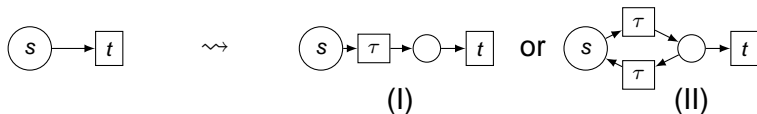
Structure of the talk

- More background
- Transition systems and Petri nets with locations (“colours”)
- Implementing a located transition system by a Petri net
- Towards a systematic approach
- Special cases
- Open problems and outlook.

Using τ -labelled transitions in a Petri net

τ -transitions contribute nothing to the language of a net.

Simulation of a Petri net by a free-choice Petri net:



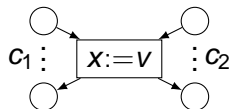
However:

Transformation (I) may introduce new *deadlocks*.

Transformation (II) may introduce *busy waiting*
(also known as *divergence*).

Handshake synchronisation in Hoare's original CSP

$$\underbrace{\dots c_2!v \dots}_{\text{Process } c_1 \text{ (sender)}} \parallel \underbrace{\dots c_1?x \dots}_{\text{Process } c_2 \text{ (receiver)}}$$



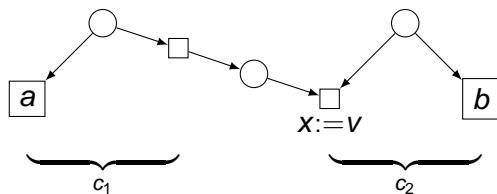
Sending the value v is simultaneous with receiving it in x , having the effect of an assignment $x:=v$.

An obvious Petri net translation creates a two-input / two-output transition labelled by $x:=v$.

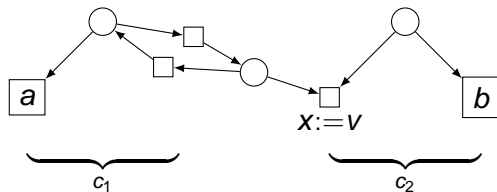
- Can the handshake be implemented if c_1 and c_2 are on different locations?
- Where to put the assignment $x:=v$, on c_1 or on c_2 ?

Trying to distribute the handshake

Consider $\underbrace{(a + c_2!v)}_{\text{Process } c_1} \parallel \underbrace{(c_1?x + b)}_{\text{Process } c_2}$



no good
(deadlock)



no good, either
(divergence)

Distributability in concurrent programming languages

It is apparently impossible, in general, to distribute handshake-based CSP.

The same is true for our Petri-net based language $B(PN)^2$ where bounded buffers of length $n \geq 0$ were introduced.

A buffer of length $n = 0$ corresponds to a handshake synchronisation.

Bounded buffers of length > 0 are distributable and can be implemented by a series of handshakes.

But there is no direct converse implementation.

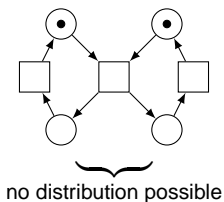
How can a series of handshakes be distributed...

...but not a single handshake?

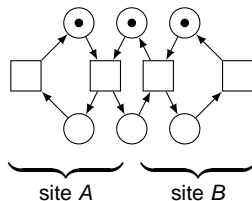
Buffers are deterministic.

In the presence of choices, buffers of length 0 and buffers of length > 0 behave differently.

handshake (0-bounded buffer):



1-bounded buffer:



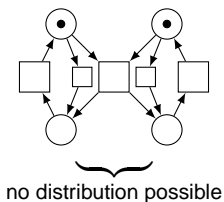
How can a series of handshakes be distributed...

...but not a single handshake?

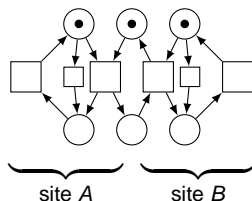
Buffers are deterministic.

In the presence of choices, buffers of length 0 and buffers of length > 0 behave differently.

handshake (0-bounded buffer):



1-bounded buffer:



Distributedness definition: Hopkins

1990-91: Richard Hopkins and I discussed implementing handshakes in a distributed environment without introducing undesirable divergences in the context of $B(PN)^2$.

We identified some situations in which the busy waiting *can* be avoided, by restricting the contexts.

Richard Hopkins wrote a paper which contains a classification of such contexts, along with a definition of *distributability*.

Such a definition was necessary in order to show formally that in certain restrictive circumstances, busy waiting is avoidable.

Distributedness definition: Caillaud

Hopkins' definition was simplified by Benoît Caillaud (2003).

The basic idea is to require for $\overset{s}{\circlearrowleft} \rightarrow \square^t$
that t necessarily resides at the same location as s .

By contrast, in cases such as $\square^t \rightarrow \overset{s}{\circlearrowleft}$,
 s and t may well be on *different* locations.

This corresponds to the observations that

- prolonging arcs from places to transitions introduces either deadlock or divergence,
- prolonging arcs from transitions to places do *not* create any new deadlocks, nor any new divergences.



Arbitration

Note: Informally, **arbitration** means resolving nondeterministic choices (by some device called **arbiter**).

Quotes from Leslie Lamport's 2003 paper on Arbiter-free Synchronization:

“The impossibility of implementing arbitration in a bounded length of time seems to be a fundamental law of nature.

Hence, *what kind of synchronization can be achieved without arbitration* should be a fundamental question in any theory of multiprocess synchronization.

We know of no previous attempt to answer this question.

Not coincidentally, we know of no practical benefits that might come from answering it.

Nevertheless, we consider it to be an interesting question in its own right.”

A possible link between distribution and arbitration

If we could implement arbitration in a bounded length of time, then we ought to be able to distribute the handshake without busy waiting.

Consequently, every class of arbiter-free synchronization primitives should lead to distributable systems.

Two of the classes of systems shown by Lamport to be arbiter-free are also distributable.

Coloured transition systems

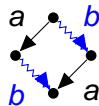
Let *COLOURS* be some set of colours, denoting *locations*.

A *coloured transition system* is a 5-tuple

$TS = (Q, A, \rightarrow, s_0, col)$:

- Q is the set of states,
- A is the set of labels,
- $\rightarrow \subseteq Q \times A \times Q$ is the transition relation,
- s_0 is the initial state,
- col is a colouring function $col: A \rightarrow COLOURS$.

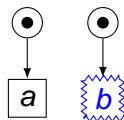
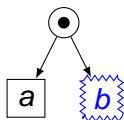
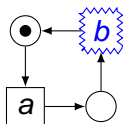
TS is *deterministic* if $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ entail $s_1 = s_2$.



Coloured Petri nets

A *coloured Petri net* is a 5-tuple $N = (S, T, F, M_0, col)$:

- S is a set of places,
- T is a set of transitions,
- F is the flow function $F: ((S \times T) \cup (T \times S)) \rightarrow \mathbb{N}$,
- M_0 is the initial marking,
- col is a colouring function $col: T \rightarrow COLOURS$.



From Petri nets to transition systems, and back

- The reachability graph $RG(N)$ of a net N is a deterministic transition system with label set $A = T$, and the colouring function of N is also a colouring function of $RG(N)$.
- *Given a coloured transition system, is there a coloured Petri net implementing it?*
This question is much more difficult to answer.

Distributed Petri nets

Each colour specifies some *location*:

- Same colour, same location.
- Different colours, different locations.

A bad case which we want to avoid is that two differently coloured transitions share a common input place:

Definition (Caillaud):

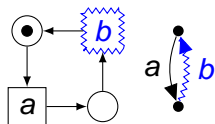
A Petri net whose transitions T are coloured by

$col: T \rightarrow COLOURS$ is called **distributed**

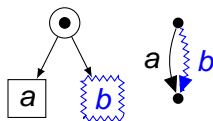
if all t_1, t_2 with $col(t_1) \neq col(t_2)$ satisfy $\bullet t_1 \cap \bullet t_2 = \emptyset$, that is, the sets of pre-places of differently coloured transitions are disjoint.

Three simple examples

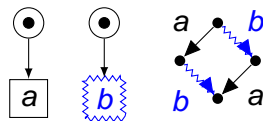
Petri nets with transitions a, b , and their reachability graphs:



distributed



not distributed



distributed

Potential distributability

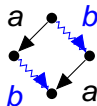
A transition system is called *potentially distributable* if $s_1 \xrightarrow{a} s_2$ and $s_1 \xrightarrow{b} s_3$ and $col(a) \neq col(b)$ imply that there is a state s_4 such that $s_2 \xrightarrow{b} s_4$ and $s_3 \xrightarrow{a} s_4$.



potentially
distributable



not potentially
distributable

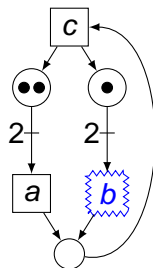
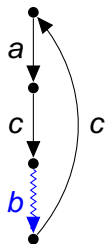
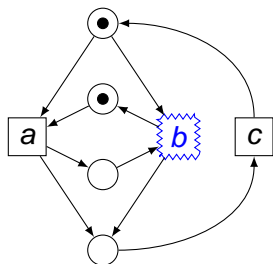


potentially
distributable

For the rest of this talk, all transition systems will be assumed potentially distributable.

It will be discussed whether a potentially distributable transition system can *actually* be implemented distributedly.

A size 4 cycle

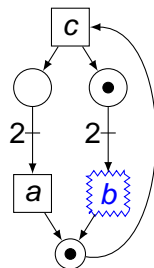
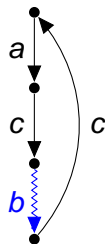
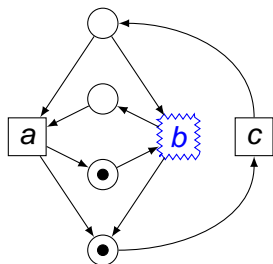


The net on the l.h.s. is not distributed

but its transition system is potentially distributable.

There is some “distributed implementation” of it (r.h.s.).

A size 4 cycle

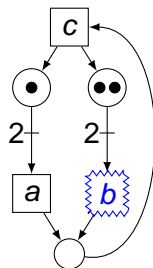
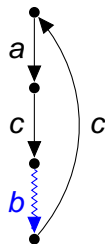
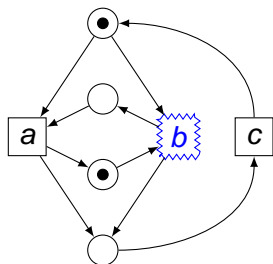


The net on the l.h.s. is not distributed

but its transition system is potentially distributable.

There is some “distributed implementation” of it (r.h.s.).

A size 4 cycle

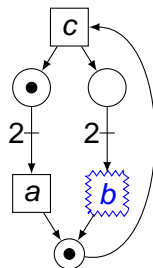
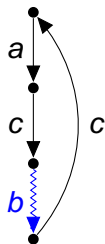
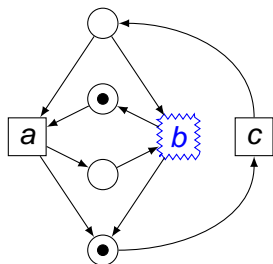


The net on the l.h.s. is not distributed

but its transition system is potentially distributable.

There is some “distributed implementation” of it (r.h.s.).

A size 4 cycle

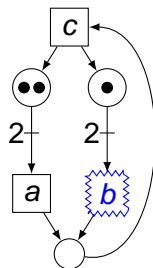
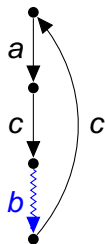
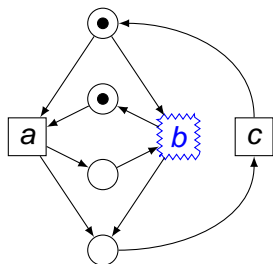


The net on the l.h.s. is not distributed

but its transition system is potentially distributable.

There is some “distributed implementation” of it (r.h.s.).

A size 4 cycle



The net on the l.h.s. is not distributed

but its transition system is potentially distributable.

There is some “distributed implementation” of it (r.h.s.).

Distributed implementation

A coloured transition system TS is **distributable** if there is some transition system TS' satisfying

$$\boxed{\textit{related}(TS, TS')}$$

such that TS' is the reachability graph of a distributed Petri net.

A coloured Petri net is distributable if its reachability graph is distributable.

How can “*related*” be interpreted meaningfully in this context?

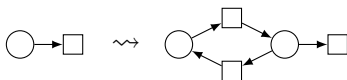
Three answers: **Identity**; **Renaming**; **Unfolding**.

One non-answer: **Refinement**.

Refinement: two techniques

Can every coloured Petri net easily be distributed using τ -transitions? Two techniques for doing so:

- Refine every place-transition arc by a busy wait loop involving two τ -transitions:

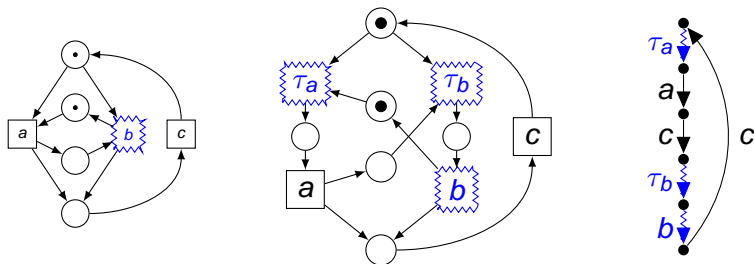


But: Introduces divergence and/or a τ -transition may reside on a location which differs from that of the transition whose input arc it replaces. Unrealistic and counterintuitive.

- Refine every transition t by a sequence $(\tau; t)$.

Creates some freedom for colouring the newly introduced τ -transitions. We might use a unique colour for all of them.

Refinement: an example for the second technique

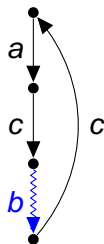


The net is now distributed. However, this construction is also rather artificial. It amounts essentially to shifting choice resolutions onto only one fixed unique location, which is counter to what is meant by “distributing” a net.

Thus, we **discard** refinement and the use of τ -transitions.

Identity

Consider again the cycle of length 4:



It is the reachability graph RG of a non-distributed Petri net.

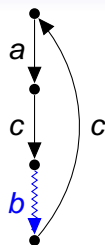
But it is also the reachability graph RG' of a distributed Petri net.

Hence it is distributable if

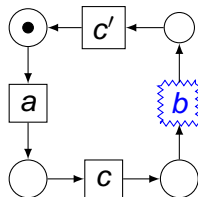
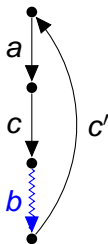
$$TS \text{ and } TS' \text{ are isomorphic} \Rightarrow \text{related}(TS, TS')$$

Renaming

Consider once more the cycle of length 4:
 Note that the two transitions labelled c
 can be considered as distinct, as they are
not mutually involved in concurrent diamonds.



We might as well
 actually distinguish them
 by renaming one of them
 to c' :



Hence it is distributable if

TS' is an arc-label renamed version of $TS \Rightarrow related(TS, TS')$

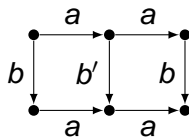
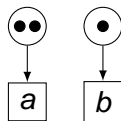
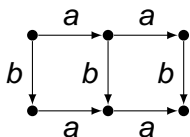
Implementing a transition system, (i) and (ii)

So far, we have identified two ways of implementing a TS :

- (i) *Direct realisation*: Find a coloured net with reachability graph TS . Caillaud's tool `synet` is an expert in doing this. The relation $related(TS, TS')$ describes the **identity** relation (TS and TS' are isomorphic).
- (ii) *Renaming*: Find a coloured transition system TS' by renaming some transitions of TS suitably, leaving the set of states unchanged. The relation $related(TS, TS')$ describes the **renaming** relation (some transitions are renamed in TS' w.r.t. TS).

An aside: Renaming is not innocuous

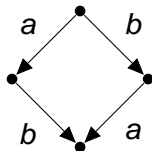
We may, for instance, create a transition system which is not the reachability graph of a Petri net out of one which is:



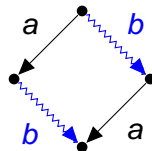
Not a
reachability graph
of any Petri net

Renaming local, nondistributed diamonds

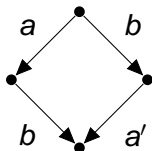
Nondistributed:



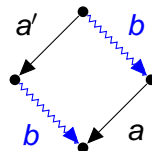
Distributed:



In the second case, a (say, in Argentina) may *not* influence b (say, in Belgium) locally. With this intuition, renaming is only allowed for local diamonds:

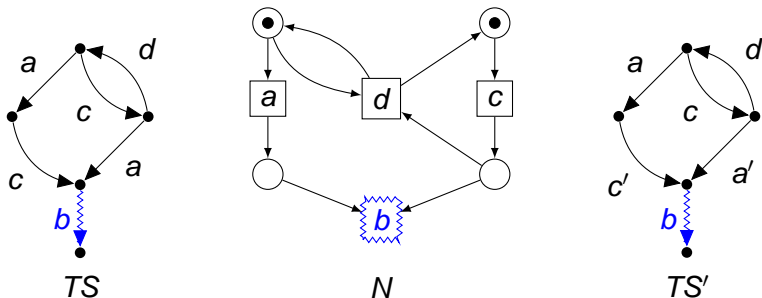


Allowed!



Forbidden!

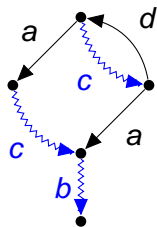
An example where Identity fails but Renaming works



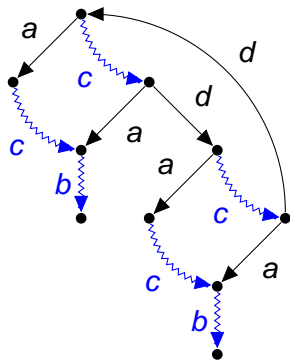
- TS is the reachability graph of N .
- N is not distributed.
- There is **no** distributed Petri net with reachability graph TS .
- TS' is a renamed version of TS .
- A distributed Petri net with reachability graph TS' exists.

An example where Renaming does not work

TS:



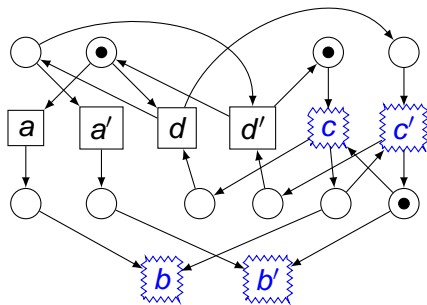
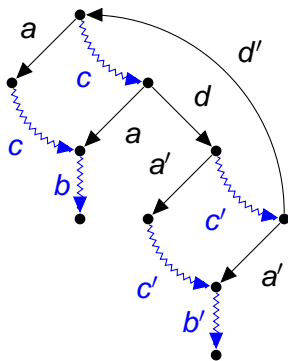
No renamed version
of *TS* is distributable



However, we may
“unfold” *TS* in this way

The unfolded transition system is still not distributable.
However, an additional renaming (giving the second copy
different names) works.

An example where Renaming combined with Unfolding works



A distributable transition system (l.h.s.)
and its distributed Petri net (r.h.s.).

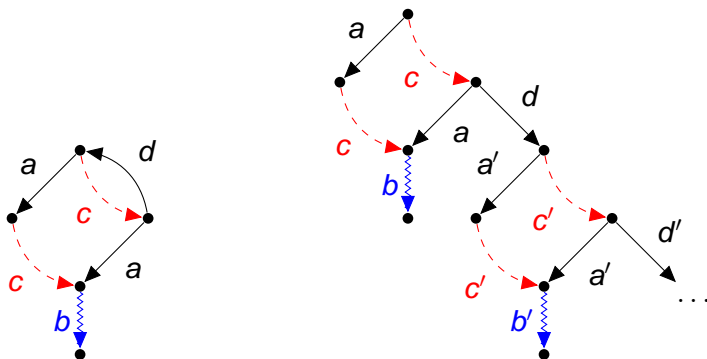
Implementing a TS, (iii)

A third method (in addition to (i) direct realisation, and (ii) renaming) for implementing a coloured transition system TS was identified:

(iii) *Unfolding*: Find a coloured transition system TS' by unfolding (and perhaps also renaming) a given transition system TS suitably, and find a Petri net implementing TS' . Here, $related(TS, TS')$ means that TS' is an **unfolding** (perhaps with renaming) of TS .

We have found no other ways of distributing a transition system systematically.

An example where none of the three methods work



This example, using three colors, cannot even be resolved if one attempts to unfold infinitely often (as on the right-hand side) and if the distributed Petri net is allowed to be unbounded.

Categorising the methods (i)–(iii) using patterns

The three methods (identity, renaming, unfolding) will be categorised and evaluated.

Patterns that may be obstructive for distributability will be identified.

The previous examples will be recapitulated in the light of these patterns.

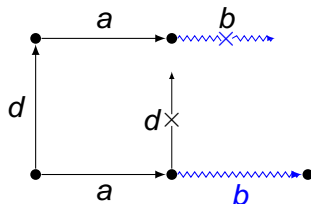
Pattern 1: Dangerous nondistributed half-persistency

Transitions a, d are forming three quarters of a diamond.

Nondistributed: a, d are from the same colour.

Half-persistency:

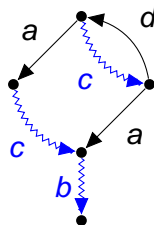
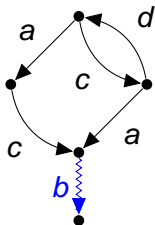
a persists over d , but d does not persist over a .



From the tip, some transition b of a different colour is not enabled, while b is enabled from one of the sides.

This *dangerous*, because it signifies *delayed* (by a) choice between d and b , which is likely to create an input place of d and b (making it non-distributed).

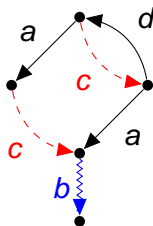
Examples of nondistributed half-persistenceency



Dangerous half-persistenceency occurs on the left-hand side.
It can be mended by renaming one of the *a* transitions.

The pattern also occurs on the right-hand side.
It can be mended there only after an unfolding step.

Example of bad cycle condition



This *TS* satisfies the bad cycle condition.

Hence there can be **no** distributed Petri net implementing it.

Summary so far

- It is difficult to give precise conditions for a transition system to correspond to a distributed Petri net.
- In case there is dangerous nondistributed (half-)persistency, one may hope to get rid of it by renaming and/or unfolding.
- If the bad cycle condition is satisfied, there is no hope of getting a distributed implementation at all.

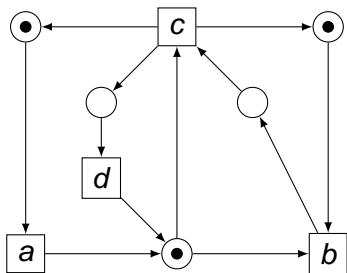
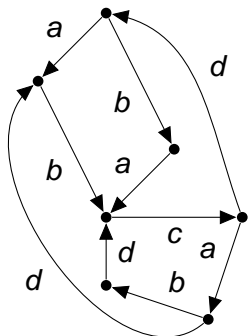
It is unknown whether the absence of dangerous nondistributed (half-)persistency and of bad cycles already guarantees distributed implementability.

Therefore: we examine classes of transition systems which do not exhibit either of these patterns, in order to find out whether there are any other detrimental situations.

A class of transition systems

Consider reachability graphs of Petri nets satisfying:

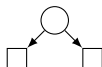
- (a) They are finite and strongly connected.
- (b) They are persistent (i.e., if a state enables a and b , and if $a \neq b$, then also ab and ba are enabled).
- (c) All simple cycles in the reachability graph have the same Parikh (i.e.: transition count) vector.



Two classes of Petri nets

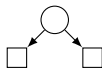
A Petri net is called **output-nonbranching** (or *ON*, for short) if every place has at most one output transition.

No

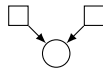


A Petri net is called **marked graph** if every place has at most one input and at most output transition.

No



and no



Marked graphs are well understood since 1969 (Genrich, Commoner, Pnueli et al.).

Both classes correspond to system classes shown by Lamport to be arbiter-free.

A conjecture

Claim:

Every transition system satisfying (a)–(c) is the reachability graph of some ON Petri net.

Corollary:

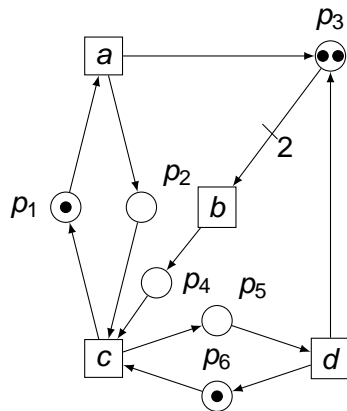
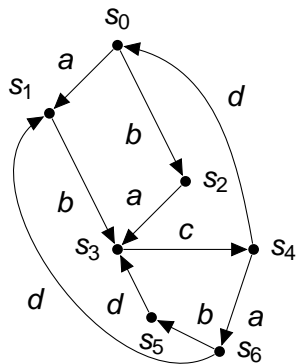
Transition systems satisfying (a)–(c) are distributable.

There are some Petri net classes which have been studied in the literature and which satisfy Properties (a)–(c):

- live marked graphs
- plain, bounded, persistent, and reversible nets which are k -marked with $k \geq 2$.

The class delineated by (a)-(c) is larger than either.

ON Petri net of a reachability graph satisfying (a)–(c)



Note: Standard region theory will **not** produce this Petri net.

Open problems

- Prove the conjecture.
- Collect necessary conditions for distributability.
- Collect sufficient conditions for distributability.
- Solve the distributability problem.
I.e.: find a *necessary and sufficient* condition for distributability.
- Forge links to arbitration-free synchronisation primitives.
- Find practical applications by deciding which nets can be distributed and which cannot.

References

- Éric Badouel, Benoît Caillaud, Philippe Darondeau: Distributing Finite Automata through Petri Net Synthesis. Journal on Formal Aspects of Computing (2002).
- Éric Badouel, Philippe Darondeau: Theory of regions. Lectures on Petri Nets I: Basic Models, LNCS Vol. 1491 (1999).
- Eike Best, Philippe Darondeau: Separability in Persistent Petri Nets. In J. Lilius and W. Penczek, editors, Petri Nets 2010, LNCS Vol. 6128 (2010).
- Eike Best, Richard P. Hopkins: $B(PN)^2$ - a Basic Petri Net Programming Notation. PARLE (1993).
- Benoît Caillaud: Try out his tool at <http://www.irisa.fr/s4/tools/synet/>.
- Richard P. Hopkins: Distributable Nets. In Advances of Petri Nets, LNCS Vol. 524 (1991).
- Leslie Lamport: Arbiter-Free Synchronization. Distributed Computing 16 (2003).