Habilitation Thesis

# Concurrency in Distributed Real-Time Systems, from Unfoldings to Implementability

Thomas Chatain

Defense on Friday, December 13, 2013, 14:00
at ENS Cachan

before the jury composed of:

- Javier Esparza (reviewer)
- Stefan Haar
- Laure Petrucci (reviewer)
- Jean-François Raskin
- Olivier H. Roux
- Jiri Srba (reviewer)
- François Vernadat

# Contents

# Introduction

In this thesis I present a synthetic view of a large part of my recent research work, organized along a main guideline: concurrency in distributed real-time systems, from unfoldings to implementability. The contributions that I present were obtained since the beginning of Sandie Balaguer's PhD, started in November 2009 and defended in December 2012. Supervising her thesis was a great experience and gave me the opportunity to progress on subjects that I had in mind and wanted to push forward. Also, as the present habilitation thesis aims at demonstrating my ability to supervise students, it makes sense that I focus on the results obtained during Sandie's thesis. I give an overview of other contributions in the conclusion.

Information systems have increased dramatically during the last decades. They have also become more and more complex and difficult to design, maintain and supervise. Formal methods offer a way to deal with the complexity of these systems. They are adapted to a variety of domains like design, verification, model-checking, test and supervision.

But information systems are also more and more often *distributed*, first because of the generalization of information networks, but also because inside a single device, like a computer, the numerous components run concurrently. The problem is that concurrency is known to be a major difficulty for the use of formal methods because it causes a combinatorial explosion of the state space of the systems.

This difficulty comes sometimes with another one due to *time* when it plays an important role in the behavior of the systems, for instance when the execution time is a critical parameter.

These two difficulties, concurrency and real-time, have guided my research works. Sometimes I have tackled one of these two aspects separately, but in many of my works, I have dealt with the problems that arise when one studies systems that are *both concurrent and real-time.*

In the framework of formal methods for the verification of untimed distributed systems, studying concurrency not only helped to understand better the behavior of the systems, but also gave powerful techniques that improve the efficiency of verification tools. I think for instance of the model checking of LTL properties using unfoldings [75, 76]. These techniques rely on a well established theory for concurrency and on reference semantics based on partial orders, like Mazurkiewicz traces [68] or event structures [106].

In the context of real-time distributed systems, one cannot rely on a smooth concurrency theory like in the untimed, asynchronous case. In particular, the nice independency relations used in Mazurkiewicz traces do not work: actions performed on distinct machines may not commute freely because they are ordered by their occurrence time. This explains for instance that little literature exists on partial-order reduction techniques for time Petri nets [107, 115] and for networks of timed automata [21, 103, 98, 105].

## Overview of the Thesis

Chapters 1 and 2 introduce the formalisms of timed automata and networks of timed automata, time Petri nets and unfoldings.

In the following chapters I present a summary of a large part of my recent research work, which is organized along a main guideline. Here is an overview of this guideline.

The largest and most involved contribution of my PhD is certainly the definition of unfoldings for time Petri nets. The main challenge was to marry unfoldings, which are typically tailored for asynchronous systems, with real-time constraints, which force a global ordering of events w.r.t. time. In order to define unfoldings, I had to show that it is possible to define a concurrent operational semantics which relaxes the ordering between events but still respects the original semantics. Nevertheless the time constraints induce complex dependencies between events.

As a matter of fact, the usual binary relations between events in unfoldings – conflict, causality and concurrency – do not suffice to code all the dependencies between events in a distributed real-time context. But I noticed that special dependencies between events arise also when defining unfoldings for other extensions of Petri nets. In particular, a very simple assumption generates logical dependencies which are not represented directly by conflict, causality and concurrency: it suffices to restrict the semantics of (classical, low-level) occurrence nets by considering only maximal runs. Then the occurrence of an event $a$ may imply the occurrence of an event $b$ which is not a causal predecessor of $a$. The *reveals* relation [84] was introduced to capture these cases. It resembles very much some of the dependencies that one can observe with real-time systems, but in a simpler setting. I studied and generalized it. I present some results in Chapter 3.

Another question that interested me is the comparison between real-time concurrent formalisms. Many formalisms exist; two of the most popular are time Petri nets [102] and networks of timed automata [2]. Their expressiveness has been compared and several transformations have been proposed from one formalism to the other, but the behavioral comparisons that are used usually do not take concurrency into account. In Chapter 4, I present new behavioral comparisons based on the distribution of actions.

Finally, I found that the differences between the formalisms are worth being considered from the point of view of implementability. Indeed these formalisms use high-level paradigms such as multi-party rendezvous or shared clocks. Implementing them, or simply translating them to lower-level models, requires introducing new communications e.g. to initiate a rendezvous, or send the value of clocks. Yet the implementation should keep as much as possible of the concurrency specified in the high-level model. This is a research topic that I started investigating quite recently. In Chapter 5, I show the first results and present many perspectives.

# Chapter 1

# Formalisms for Real-Time Sequential Systems

## 1.1 General Assumptions about the Semantics of Time

**Dense Time.** I consider dense time semantics where time progresses continuously. There is another option, not studied in this thesis, which consists in discretizing the time progress. This is perfectly arguable given that digital devices are generally conducted by discrete clock signals; moreover analysis techniques are usually conceptually simpler. On the other hand, dense time is more general, gives much more concise models and allows one to use efficient symbolic analysis techniques like DBMs [69, 110, 93], symbolic state classes or zones.

**Time Divergence.** I consider only models where time *diverges*: in every run that contains infinitely many discrete actions, time diverges to infinity. This assumption is very usual and comes from the intuition that no physical system can execute infinitely many actions in finite time.

## 1.2 Timed Transition Systems

Timed transition systems are used as an abstract low-level formalism to represent the semantics of higher-level formalisms for real-time sequential systems like timed automata and the sequential semantics of formalisms for real-time distributed systems like networks of timed automata or time Petri nets.

**Definition 1.1 (Timed Transition System).** *A* timed transition system *(TTS) is a tuple $(S, s_0, \Sigma, \rightarrow)$ where $S$ is a set of states, $s_0 \in S$ is the initial* *state, $\Sigma$ is a set of actions disjoint from the set $\mathbb{R}_{\geq 0}$ of non-negative reals, and $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is a set of transitions. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$.*

**Runs and Timed Words.** A *run* is a (finite or infinite) path starting from the initial state.

When representing a run, we often forget the information about the intermediate states and delays, and remember only the sequence $(a_1, \theta_1) \dots (a_n, \theta_n)$ of actions with their occurrence dates $\theta_i$ (obtained by summing the delays). This representation is called a *timed word*.

The *timed language* of a TTS is the set of all its runs (represented as timed words). The timed language can be used as a criterion to compare models. Anyway, as soon as branching time properties matter, one needs stronger behavioral comparisons like bisimulation.

**Definition 1.2 (Timed bisimulation).** *Let $T_1 = (S_1, s_1^0, \Sigma, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma, \rightarrow_2)$ be two TTS. We say that $T_1$ and $T_2$ are* timed bisimilar *if there exists a binary relation $\approx$ between $S_1$ and $S_2$, called* timed bisimulation relation*, such that:*

- $s_1^0 \approx s_2^0$,
- *if $s_1 \xrightarrow{a}_1 s_1'$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_2 \xrightarrow{a}_2 s_2'$ such that $s_1' \approx s_2'$; conversely if $s_2 \xrightarrow{a}_2 s_2'$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_1 \xrightarrow{a}_1 s_1'$ such that $s_1' \approx s_2'$.*

## 1.3 Timed Automata

Timed automata [2] are one of the most famous formal models for real-time systems. They have been deeply studied and very mature tools are available, like UPPAAL [94], EPSILON [50] and KRONOS [42].

**Definition 1.3 (Timed automaton [2]).** *A timed automaton (TA) is a tuple $A = (L, \ell_0, C, \Sigma, E, Inv)$ where*

- *$L$ is a set of* locations*,*
- *$\ell_0 \in L$ is the* initial *location,*
- *$C$ is a set of* clocks*,*
- *$\Sigma$ is a set of* actions*,*
- *$E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a set of* edges*,*
- *$Inv : L \longrightarrow \mathcal{B}(C)$ assigns* invariants *to locations.*

*where $\mathcal{B}(C)$ denotes the set of clock constraints over the set of clocks $C$, defined by the grammar $g ::= x \bowtie k \mid g \wedge g$ with $x \in C$, $k \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Invariants are usually restricted to clock constraints of the form $g ::= x \leq k \mid x < k \mid g \wedge g$.*

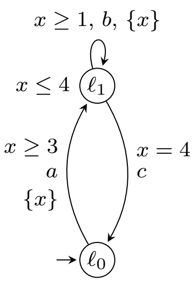


Figure 1.1: A timed automaton (the initial location is indicated by an arrow that is not rooted in any location)

For an edge $(\ell, g, a, r, \ell') \in E$, $\ell$ is called the *source* location, $g$ the *guard*, $a$ the *action*, $r$ the set of clocks to be *reset* and $\ell'$ the *target* location.

Locations are usually pictured as circles, connected by arcs representing the edges.

### Semantics

We use the following notations:

- For each set of clocks $r \subseteq C$, the valuation $v[r]$ is defined by $v[r](x) = 0$ if $x \in r$ and $v[r](x) = v(x)$ otherwise.
- For each $d \in \mathbb{R}_{\geq 0}$, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for each $x \in C$.

The semantics of a TA $(L, \ell_0, C, \Sigma, E, Inv)$ is a TTS whose states are the pairs $(\ell, v)$ where $\ell \in L$ is the current location and $v : C \longrightarrow \mathbb{R}_{\geq 0}$ is a *clock valuation* that satisfies the invariant of location $\ell$ (we write $v \models Inv(\ell)$). The initial state is $(\ell_0, v_0)$, where $v_0$ maps each clock to 0. Two types of transitions are possible:

- Time delay: $\forall d \in \mathbb{R}_{\geq 0}, (\ell, v) \xrightarrow{d} (\ell, v + d)$ iff $\forall d' \in [0, d], v + d' \models Inv(\ell)$.
- Discrete action: $(\ell, v) \xrightarrow{a} (\ell', v')$ iff there exists an edge $(\ell, g, a, r, \ell') \in E$ such that $v \models g$ and $v' = v[r]$ and $v' \models Inv(\ell')$.

### Example

Figure 1.1 shows a timed automaton with one clock $x$. The automaton starts in location $\ell_0$. After waiting at least 3 time units, it can take transition $a$, which resets $x$ to 0. Then it can play $b$ several times, provided it waits 1 to 4 time units before each occurrence. Eventually, after waiting 4 time units, it can play $c$, and then start again with the $a$… So it accepts for instance the timed word $(a, 4.8)(b, 7)(b, 8)(c, 12)(a, 13)(b, 15.3)$.

# Chapter 2

# Formalisms for Real-Time Distributed Systems

## 2.1 About the Semantics of Time in Distributed Systems

The semantics of time in distributed systems has already been debated. One problem is to decide whether it is reasonable to assume that the different components of a distributed system are ruled by the same global absolute time. It has been proposed to localize clocks and some authors [1, 71, 21] have even suggested to use local-time semantics with independently evolving clocks.

Here I stay in the classical setting with perfect clocks evolving at the same speed. This is a key assumption that provides an implicit synchronization. Moreover, the problems that I will develop, about dependencies between events and about implementation of real-time distributed systems, are simply based on the fact that there is a global time progress, which implies that if time has progressed for one component of a distributed system, then the other components must also have observed time progress of a similar duration, no matter if they slightly disagree on its measure.

## 2.2 Product of Timed Transition Systems

A distributed system can usually be viewed as a set of sequential systems, or *components*, which sometimes communicate together. Hence one way to model a real-time distributed system is to model each component separately as a TTS and specify when they communicate. Then communications are actions that are performed simultaneously by several components. Such composition of sequential systems can be given itself a sequential semantics as a *product* of TTS. For simplicity, I present only the product of two TTS. The intersection of their alphabets of actions defines the actions which are synchronizations. Notice

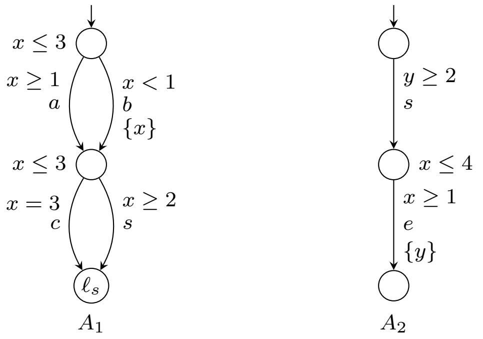


Figure 2.1: A network of two timed automata. The shared action $s$ is a synchronization. Moreover $A_1$ influences $A_2$ via the shared clock $x$.

that time progress is also an implicit synchronization.

**Definition 2.1 (Product of TTS).** *The* product *of two timed transitions systems* $T_1 = (S_1, s_1^0, \Sigma_1, \rightarrow_1)$ *and* $T_2 = (S_2, s_2^0, \Sigma_2, \rightarrow_2)$ *is the TTS* $T_1 \times T_2 \overset{\text{def}}{=} \big(S_1 \times S_2, (s_1^0, s_2^0), \Sigma_1 \cup \Sigma_2, \rightarrow\big)$*, where* $\rightarrow$ *is defined as:*

- $(s_1, s_2) \xrightarrow{a} (s_1', s_2)$ *iff* $s_1 \xrightarrow{a}_1 s_1'$*, for any* $a \in \Sigma_1 \setminus \Sigma_2$*,*
- $(s_1, s_2) \xrightarrow{a} (s_1, s_2')$ *iff* $s_2 \xrightarrow{a}_2 s_2'$*, for any* $a \in \Sigma_2 \setminus \Sigma_1$*,*
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ *iff* $s_1 \xrightarrow{a}_1 s_1'$ *and* $s_2 \xrightarrow{a}_2 s_2'$*, for any* $a \in (\Sigma_1 \cap \Sigma_2) \cup \mathbb{R}$*.*

There exist many other formalisms which allow to model real-time distributed systems directly. I introduce two popular formalisms: safe time Petri nets (TPN) [102] and networks of timed automata (NTA) [2]. These formalisms have different histories but were both designed to model real-time, distributed systems. Moreover they both handle urgency, which is a key feature without which most real-time systems cannot be modeled correctly.

## 2.3 Networks of Timed Automata

A network of timed automata (NTA) is a composition of $n$ timed automata, written $A_1 \parallel \cdots \parallel A_n$, with $A_i = (L_i, \ell_i^0, C_i, \Sigma_i, E_i, Inv_i)$. We denote by $\Sigma = \bigcup_i \Sigma_i$ the set of actions and by $C = \bigcup_i C_i$ the set of clocks.

Shared actions are used to model synchronizations like in products of TTS. But clocks may also be shared, which means that the automata can also influence one another via *shared clocks.*

### Semantics

The semantics of $A_1 \parallel \cdots \parallel A_n$ is a TTS whose states are the pairs $(\vec{\ell}, v)$, where $\vec{\ell} \in L_1 \times \cdots \times L_n$ is the vector of current locations and $v : C \longrightarrow \mathbb{R}_{\geq 0}$ is a valuation of all the clocks that satisfies $\bigwedge_i Inv_i(\ell_i)$. The initial state is $(\vec{\ell_0}, v_0)$ with $\forall x \in C, v_0(x) = 0$.

Time delay is possible iff it is possible for all the automata: $(\vec{\ell}, v) \overset{d}{\to} (\vec{\ell}, v + d) \iff \forall d' \in [0, d], v + d' \models \bigwedge_i Inv_i(\ell_i)$.

For discrete actions, the intersection of the alphabets of actions determines which automata participate: in order to play an action $a \in \Sigma$ from a state $(\vec{\ell}, v)$, all the automata $A_i$ such that $a \in \Sigma_i$ must participate by playing an edge $e_i = (\ell_i, g_i, a, r_i, \ell'_i)$; the others do not participate. This leads to the state $(\vec{\ell'}, v')$ where $\ell'_i$ is the target location of $e_i$ for every automaton $A_i$ which participated, $\ell'_i = \ell_i$ for the others, and $v' = v[\bigcup_{i \text{ s.t. } a\in\Sigma_i} r_i]$.

### Example

Figure 2.1 shows a network of two timed automata, where $s$ is a common action. $A_1$ can play $b$ at time 0.8. This resets the shared clock $x$. After waiting between 2 and 3 time units, for example at time 3.5, the two automata can synchronize and play an $s$. Then $A_2$ must play $e$ while the value of clock $x$ is between 1 and 4, i.e. between 1 and 4 time units after the $b$. This gives, for instance, the run $(b, 0.8)(s, 3.5)(e, 4.1)$.

## 2.4 Time Petri Nets

Time Petri nets are one of the most popular timed extensions of Petri nets. I give directly the definition of time Petri nets; for untimed Petri nets, just forget the delay intervals and their role in the semantics, or set all the delay intervals to $[0, \infty)$.

Time Petri nets are supported by many very mature tools like Romeo [81], TINA and [30].

**Definition 2.2 (Time Petri Net [102]).** *A* time Petri net *(TPN) is a tuple* $(P, T, pre, post, efd, lfd, M_0)$ *where* $P$ *and* $T$ *are sets of* places *and* transitions *respectively, pre and post map each transition* $t \in T$ *to its (nonempty)* preset *denoted* ${}^\bullet t \overset{\text{def}}{=} pre(t) \subseteq P$ *and its (possibly empty)* postset *denoted* $t^\bullet \overset{\text{def}}{=} post(t) \subseteq P$*;* $efd : T \longrightarrow \mathbb{N}$ *and* $lfd : T \longrightarrow \mathbb{N} \cup \{\infty\}$ *associate the* earliest firing delay $efd(t)$ *and* latest firing delay $lfd(t)$ *with each transition* $t$*;* $M_0 \subseteq P$ *is the* initial marking.

A time Petri net is represented as a graph with two types of nodes: places (circles) and transitions (rectangles). The interval $[efd(t), lfd(t)]$ is written near each transition (see Figure 3.9).

### Semantics of Safe Time Petri Nets

I give a semantics with clocks attached to marked places (or tokens). The semantics is more often defined with tokens attached to transitions, but when dealing with partial order semantics, clocks on places are more convenient (see

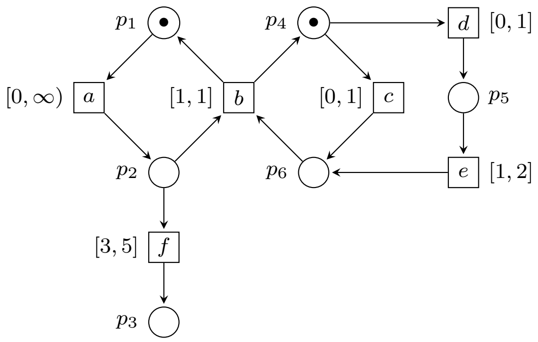


Figure 2.2: A time Petri net

[35] and [56] for a detailed study) I consider only *safe* (or *1-bounded*) TPNs, i.e. TPNs where there is never more than one token in a place. When considering several tokens in places, several semantics are possible (see again [35]) and most verification problems become undecidable.

**State.** A *state* of a time Petri net is given by a triple $(M, dob, \theta)$, where $M \subseteq P$ is the *marking*, $\theta \in \mathbb{R}$ is the current time and $dob : M \longrightarrow \mathbb{R}$ associates a *date of birth* $dob(p) \in \mathbb{R}$ with each token (marked place) $p \in M$.

The *initial state* is $(M_0, dob_0, 0)$ and initially, all the tokens carry the date 0 as date of birth: for all $p \in M_0$, $dob_0(p) \overset{\text{def}}{=} 0$.

Transition $t$ is *enabled* in state $(M, dob, \theta)$ if ${}^\bullet t \subseteq M$. We define its *date of enabling* $doe(t)$ as the date of birth of the youngest token in its input places:

$$doe(t) \overset{\text{def}}{=} \max_{p \in {}^\bullet t} dob(p)\,.$$

**Time delay.** From state $(M, dob, \theta)$, the TPN can wait until time $\theta'$, written $(M, dob, \theta) \xrightarrow{\theta' - \theta} (M, dob, \theta')$, iff

- time progresses: $\theta' \geq \theta$; and
- no enabled transition overtakes its maximum delay:
  $\forall t \in En(M) \quad \theta' \leq doe(t) + lfd(t)$.

**Discrete action.** Transition $t$ can fire from state $(M, dob, \theta)$ if:

- $t$ is enabled: $t \in En(M)$; and
- $t$ has reached its minimum firing delay: $\theta \geq doe(t) + efd(t)$;

Firing transition $t$ from state $(M, dob, \theta)$ leads to state $(M', dob', \theta)$, with $M' \overset{\text{def}}{=} (M \setminus {}^\bullet t) \cup t^\bullet$ and $dob'(p) \overset{\text{def}}{=} dob(p)$ if $p \in M \setminus {}^\bullet t$ and $dob'(p) \overset{\text{def}}{=} \theta'$ if $p \in t^\bullet$ (by assumption the two cases are exclusive). We write $(M, dob, \theta) \xrightarrow{t} (M', dob', \theta)$.

#### Example

For the TPN of Figure 2.2, a possible run is $(a, 0.2)(d, 1)(e, 2.6)(f, 3.4)$: In the initial state, $a$, $c$ and $d$ are enabled. Because the latest firing delay of $c$ and $d$ is 1, a transition must fire before 1, for instance $a$ at time 0.2. This does not disable $c$ and $d$, so another transition must fire before 1, say $d$ at time 1. This enables $e$ which has to fire after waiting before 1 and 2 time units. The marking is now $\{p_2, p_6\}$, so $b$ and $f$ are enabled and only one can consume the token in $p_2$. Because $f$ was enabled at time 0.2 (after the firing of $a$), it can fire at 3.4. Transition $b$, which was enabled at time 2.6 could fire only at time 3.6.

#### Assumption of Safe Untimed Support

I have already mentioned that I consider only safe TPNs. Moreover, I require that even the untimed support is safe, i.e. the TPN remains safe if one replaces all the earliest firing delays by 0 and all the latest firing delays by $\infty$.

## 2.5 Partial Order Semantics

Products of TTS, networks of timed automata and time Petri nets are designed to model distributed systems. Yet their semantics is usually defined in terms of timed transition systems or languages of timed words, which means that the information about concurrency is lost. As an alternative, It is possible to define partial-order semantics for these models.

Partial order semantics were much more developed for Petri nets than for other models of concurrency; an explanation is that a category of Petri nets, called occurrence nets, provides a very natural graphical support for the representation of runs, called processes.

The representation of all runs of a Petri net as an *unfolding* [77] allows one to avoid the state-space explosion due to interleavings when exploring the runs of a Petri net. Unfoldings are infinite in general, but can be represented efficiently by a finite complete prefix [101, 79], for instance to check LTL formulas [75, 76, 89].

Unfoldings were defined for other models of asynchronous systems, like products of transition systems [78], but little literature is available about unfoldings for distributed real-time systems. We can cite [54, 55, 113] for TPNs and [47, 38] for NTA.

### 2.5.1 Partial Order Representation of Runs: Processes

Processes are a way to represent executions of Petri nets so that the actions (called events) are not totally ordered like in firing sequences: only causality orders them.

An execution of a Petri net $\mathcal{N}$ is represented as a labeled Petri net where every transition, called *event* and labeled by a transition $t$ of $\mathcal{N}$, stands for an occurrence of $t$, and every place, called *condition* and labeled by a place $p$ of

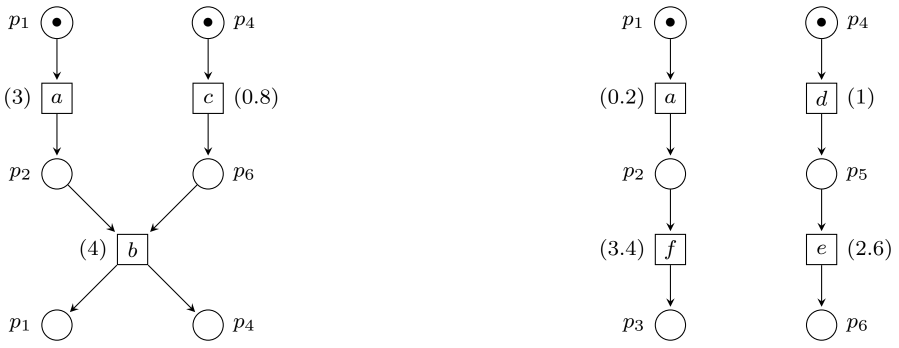


Figure 2.3: Two processes of the time Petri net of Figure 2.2. The occurrence dates are written in parentheses near the events.

$\mathcal{N}$, refers to a token produced by an event in place $p$ or to a token of the initial marking. The arcs represent the creation and consumption of tokens.

Because fresh conditions are created for the tokens created by each event, every condition has either no input arc (if it is an initial condition) or a single input arc, coming from the event that created the token. Symmetrically, each place has no more than one output arc since a token can be consumed by only one event in an execution.

When dealing with time Petri nets, a date is attached to each event. Processes of time Petri nets were studied in [6, 7].

Figure 3.1 shows two processes of the time Petri net of Figure 2.2.

**Causality and Concurrency.** We define the relation $\prec$ on the events as: $e \prec e' \overset{def}{\iff} e^\bullet \cap {}^\bullet e' \neq \emptyset$. The reflexive transitive closure $\prec^*$ of $\to$ is called the *causality* relation. Two events of a process that are not causally related are called *concurrent*. For all event $e$, we denote $\lceil e \rceil$ the *causal past* of $e$, i.e. the set of predecessors of $e$ by $\prec^*$.

### 2.5.2 Branching Processes and Unfoldings

Here I deal with untimed Petri nets only. Defining branching processes and unfoldings for time Petri nets poses other difficulties, which I evoke later.

It is possible to superimpose several processes of a given untimed Petri net in order to represent several runs. The superimposition is done such that the common prefixes of the processes are merged. Hence one gets a labeled Petri net called a *branching process* [74], which remains acyclic, but where conditions can now have several output arcs, representing the choices between the different runs. This is formalized by the notion of *conflict*.

**Conflict.** Two events $e$ and $f$ of a branching process are in *conflict* if there exist two distinct events $e'$ and $f'$ such that $e' \prec^* e$ and $f' \prec^* f$ and ${}^\bullet e' \cap {}^\bullet f' \neq \emptyset$.

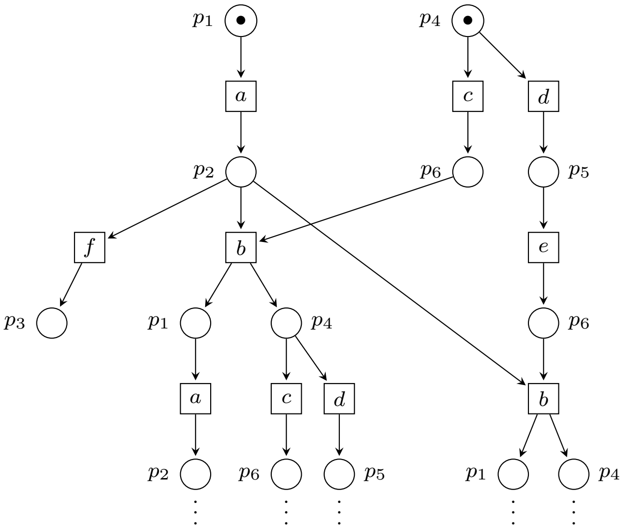


Figure 2.4: A prefix of the unfolding of the untimed support of the TPN of Figure 2.2

Since $e'$ and $f'$ consume the same token, they cannot occur in the same run. This incompatibility propagates to $e$ and $f$ by causality.

**Unfolding.** Superimposing *all* processes of an untimed Petri net $\mathcal{N}$ yields a (usually infinite) branching process, called the *unfolding* of $\mathcal{N}$. Figure 2.4 represents a prefix of the unfolding of the untimed support of the TPN of Figure 2.2.

### 2.5.3 Occurrence Nets

Branching processes, considered simply as Petri nets, form a category that we call *occurrence nets* and can be studied as such. They are characterized as follows.

**Definition 2.3 (Occurrence net).** *An* occurrence net *(ON) is an untimed Petri net* $(B, E, pre, post, B_0)$ *such that:*

- $\forall e \in E \quad \neg(e \prec^+ e)$,
- $\forall b \in B_0 \quad \nexists e \in E \quad b \in e^\bullet$,
- $\forall b \in B \setminus B_0 \quad \exists! e \in E \quad b \in e^\bullet$,
- $\forall e \in E \quad \neg(e \# e)$,
- $\forall e \in E \quad |\lceil e \rceil| < \infty$.

Occurrence nets are closely related to the notion of *event structures* [106].

# Chapter 3

# Dependencies Between Events

*This chapter presents my contributions [11, 12] and [51] on extensions of the reveals relation and facets reduction obtained during and after Sandie Balaguer's PhD, and on my recent contribution [56] about the semantics of time in extended free choice time Petri nets.*

## 3.1 Motivation

### 3.1.1 Unfoldings of Time Petri Nets, and how Complex Dependencies between Events Arise

I worked during my PhD on the definition of unfoldings for time Petri nets [54, 55].

Figure 3.1 shows a prefix of the unfolding of the time Petri net of Figure 2.2. Events are parameterized by their possible occurrence time. Symbolic constraints indicate the timings compatible with the time constraints of the model. As there is only one occurrence of each transition in this prefix (apart from $b$), we use the symbol $\theta_a$ as a variable that represents the firing time of the event corresponding to transition $a$ (and respectively for the other events).

Notice the read arc (line with no arrow head) in input of $f$. Although $f$ consumes only the token produced by $a$, firing $a$ is not as sufficient condition for being able to fire $f$: if $c$ fires too, it enables $b$, which has to fire before $f$, and then disables $f$. On the other hand, the read arc indicates that $f$ is possible in the context where $d$ and $e$ fire.

The symbolic constraint associated with event $f$ is a bit complicated. Its first line simply says that the delay between enabling and firing of $f$ is in the interval $[3, 5]$ (since $f$ is enabled at time $\theta_a$, $\theta_f - \theta_a$ represents the delay). The second line expresses the temporal ordering between $e$ and $f$ imposed by the read arc. The explanation for the third line is the following: when $f$ fires, transition $b$ is enabled; so we have to express that $b$ has not reached its latest

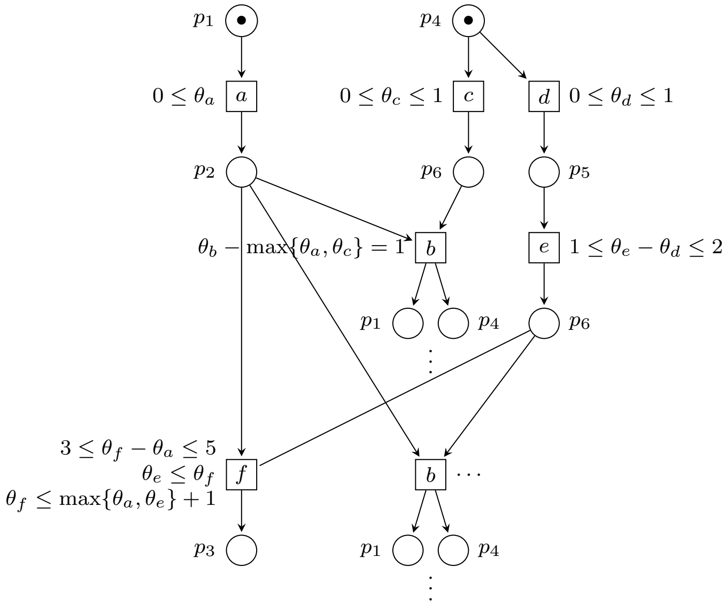


Figure 3.1: A prefix of the unfolding of the time Petri net of Figure 2.2. Notice the read arc (line with no arrow head) in input of the event corresponding to transition $f$.

firing delay when $f$ fires. Remark that this constraint concerns the firing time of $e$, which confirms that $f$ depends on the firing of $e$ and justifies the read arc.

Finally, read arcs are the solution I chose in my PhD thesis. There are other ways to code these dependencies between events; for instance using larger, non local symbolic constraints, like in [113].

### 3.1.2 The Limitations of Conflict, Causality and Concurrency

This work on unfoldings of time Petri nets emphasized the limitations of the structural relations – conflict, causality and concurrency – which are essential to the theory of unfoldings of untimed Petri nets.

In order to study the complex dependencies between events in the real-time setting, I propose to compare it with the untimed case. There, all the dependencies between the events of an occurrence net are described by conflict, causality and concurrency, and these relations have a clear meaning in terms of logical dependencies:

- two events are in *conflict* iff they never occur together in the same run;
- event $e$ is a *causal predecessor* of event $f$ iff every run containing $f$ also contains $e$;

- events $e$ and $f$ are *concurrent* iff there exist runs containing $e$ and $f$, $e$ and not $f$, $f$ and not $e$, (and of course neither $e$ nor $f$: just take the empty run).

In the real-time setting these nice properties do not suffice any more. The structural conflict on the consumption of tokens is not the only reason for logical incompatibility of two events: they can also be incompatible because they require incompatible real-time constraints on the execution. Similarly the occurrence of an event (for example $f$ in Figure 3.1) may imply the occurrence of another event ($e$ in Figure 3.1), even if these two events are not causally related. And many more complex dependencies arise; the read arcs in unfoldings of time Petri nets make some of them explicit, but others can be deduced only from the symbolic constraints on the firing times of the events.

Identifying concurrency in distributed real-time systems requires to study these logical dependencies between events in depth.

Then it is interesting to look carefully at other extensions of the unfolding theory. Even if they are not as dramatic as the real-time setting, other extensions show some limits of the well known structural relations: conflict, causality and concurrency.

**Colored Petri Nets.** One example is the symbolic unfolding of colored Petri nets, studied in [73] and [52]; there events are parameterized by the colors of the tokens they consume and create, and two events, which are not structurally in conflict, may be incompatible simply because they impose constraints on the colors of tokens in their common past which are not jointly satisfiable. These new incompatibilities may even be non-binary, i.e. it happens that three events cannot occur together, while any combination of two of these events is possible.

**Contextual Petri Nets.** Another kind of incompatibility, also not always binary, appears in unfoldings of contextual Petri nets, where read arcs model the reading of a token without consuming it. Unfoldings for contextual Petri nets were studied in [19, 44, 117, 116, 16]. There

- causality is not sufficient: another relation, called *conditional (or weak) causality* has to be introduced to represent the effect of the read arcs;

- the classical definition of conflict is not sufficient either because the conditional causality induces new sources of incompatibility between events. It is interesting to notice that, while the classic conflict is a binary relation, the incompatibilities coming from the conditional causality are not binary any more: in general they involve more than two events.

**Maximal Semantics.** Finally, a very simple assumption generates logical dependencies which are not represented directly by conflict, causality and concurrency: it suffices to restrict the semantics of (classical, low-level) occurrence nets by considering only *maximal* runs.

**Definition 3.1 (Maximal run).** *A run $\omega$ is* maximal *if it is maximal w.r.t. $\subseteq$, i.e. no event remains enabled after $\omega$.*

Consider events $a$ and $c$ in Figure 3.2(a): we have to observe that $a$ is in conflict with $b$ and that any maximal run contains either $b$ or $c$. Therefore, if $a$ occurs in a maximal run, then $b$ does not occur and eventually $c$ necessarily occurs. Yet $c$ and $a$ are concurrent.

Another case is illustrated by events $a$ and $d$ in the same figure: because $a$ is a causal predecessor of $d$, the occurrence of $d$ implies the occurrence of $a$; but in any maximal run, the occurrence of $a$ also implies the occurrence of $d$ because $d$ is the only possible continuation to $a$ and nothing can prevent it. Then $a$ and $d$ are actually made logically equivalent by the maximal progress assumption.

**Maximal Semantics as a Particular Timed Semantics.** An untimed Petri net can actually be viewed as a special kind of time Petri net where transitions can be fired after any delay. This absence of time constraints is usually coded by assigning the firing interval $[0, \infty)$ to every transition. Notice that $\infty$ is excluded from the interval. One can however propose a semantics to intervals including $\infty$: "firing at $\infty$" simply means "staying enabled forever". Hence, a transition with delay interval $[0, \infty)$ can fire after any delay, but *has to fire eventually*, while a transition with delay interval $[0, \infty]$ may fire after any delay or stay enabled forever.

This allows one to represent untimed Petri nets with general semantics as TPNs with interval $[0, \infty]$ for every transition, and Petri nets with maximal semantics as TPNs with intervals $[0, \infty)$.

Finally, this allows me to view the dependencies that appear between events of Petri nets under maximal semantics, as a kind of dependencies induced by (even very weak) time constraints.

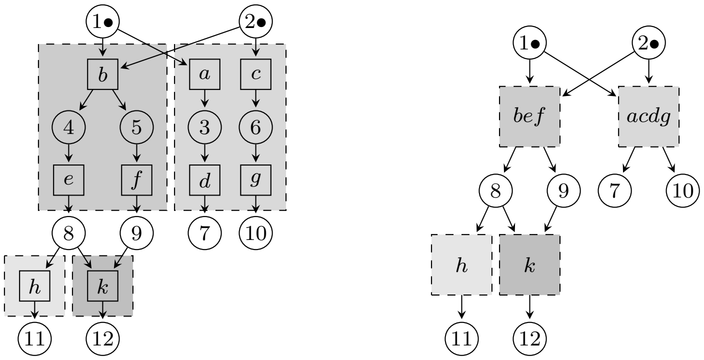
### 3.1.3 Reveals Relation and Facets Abstraction

The *reveals* relation [84, 85, 11, 12] was introduced to capture dependencies of the type "if $e$ occurs, then $f$ has already occurred or will occur eventually" in the sense that any run that contains $e$ also contains $f$. The reveals relation was defined initially for the maximal semantics, which is a very natural setting and still generates rich dependencies.

Here we simply assume that we are in a setting where not all runs are relevant. So we let $\Omega$ denote the subset of runs that we consider, for instance – but not necessarily – maximal runs. Runs are viewed as sets of events, and thus $\Omega$ is a set of sets of events.

**Definition 3.2 (Reveals relation [84]).** *Given a set $\Omega$ of sets of* events *(these sets of events intend to be interpreted as* runs*), we say that event $e$* reveals *event $f$ (in $\Omega$), and write $e \rhd f$, iff $\forall \omega \in \Omega$, $(e \in \omega \Rightarrow f \in \omega)$.*

As announced, for any events $e$ and $f$, $f \prec^* e$ implies $e \rhd f$, but the converse does not hold in general. In Figure 3.2(a), we have for instance $a \rhd c$ and $a \rhd d$ under the maximal semantics, as discussed earlier.



(a) An occurrence net. Squared events are in the same facet.

(b) The corresponding reduced ON.

Figure 3.2: An ON and its reduction through the facet abstraction.

The reveals relation naturally induces an equivalence relation on events.

**Definition 3.3 (Facet [84]).** *Let $\sim$ be the equivalence relation defined as: $\forall e, f \in E, e \sim f \overset{def}{\iff} (e \triangleright f) \wedge (f \triangleright e)$, then a* facet *of an ON is an equivalence class of $\sim$.*

For example, in Figure 3.2(a), and with the maximal semantics, the ON has four facets: $\{a, c, d, g\}$, $\{b, e, f\}$, $\{h\}$ and $\{k\}$. For any facet and for any run, either all events in the facet are in the run or no event in the facet is in the run. Therefore, facets can be seen as atomic events and can even be merged together as shown in Figure 3.2(b). Only conditions at the interface of facets remain.

## 3.2 Generalization of the Reveals Relation

Stefan Haar introduced the binary reveals relation with the application to diagnosis in mind. The idea is: “if I know that $a$ reveals $b$, and I observe $a$, then I can deduce that $b$ has also occurred or predict that it will occur.”

I saw this reveals relation as an interesting way to formalize the dependencies between events which arise, for instance, in unfoldings of TPNs. But I wanted to express more general dependencies (typically non binary conflict), so I showed in that also non-binary dependencies are relevant. We proposed in [11, 12] a representation of these dependencies using propositional logic and we solved the associated synthesis problem: given a formula $\phi$, is there an occurrence net $\mathcal{N}$ such that $\phi$ describes the set of maximal runs of $\mathcal{N}$?

Starting from the idea of the reveals relation and the associated facets abstraction, we also defined a canonical contraction for safe Petri nets [51] that merges transitions that occur together in every maximal run.

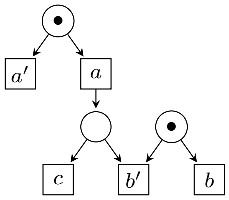


Figure 3.3: An occurrence net where the maximal semantics induces a non-binary dependency: if $a$ and $b$ occur, then $c$ also occurs.

### 3.2.1 Non Binary Dependencies

We have seen that the causality relation does not explain all the dependencies between events of the type "if $a$ occurs in a maximal run, then eventually $b$ also occurs". The reveals relation was introduced to capture all these binary dependencies. But they are still not sufficient to describe more complex logical dependencies between events. Consider the ON of Figure 3.3: causality gives only the dependencies $a \prec c$ and $a \prec b'$. With the reveals relation we get $c \triangleright b$ and $a' \triangleright b$. They express that in any maximal run the occurrence of $c$ implies the occurrence of $b$ and the occurrence of $a'$ implies the occurrence of $b$. But is it true that any set of events that satisfies these constraints, is a maximal run? The answer is no: for instance $\{a, b\}$ satisfies these constraints, but is not a valid maximal run, since $c$ is enabled and does not occur. Actually, all the maximal runs of this ON satisfy the following constraint: if $a$ and $b$ occur, then $c$ also occurs.

### 3.2.2 Representation of General Dependencies

**ERL: A Logic for Occurrence Nets**

In [11, 12], we introduced a logic, called *ERL* for *Event Reveal Logic*, that describes the properties of the runs of an ON by giving relations between event occurrences. Events are used as boolean variables: $e$ stands for the presence of event $e$ in a run.

*ERL formulas* are defined using the logical connectives $\wedge$ and $\neg$, the constants ***true*** and ***false***, and a set $E$ of event names, which play the role of variables. Well formed ERL formulas are defined inductively by the following grammar.

$$\phi ::= \boldsymbol{true} \mid \boldsymbol{false} \mid e \mid \neg\phi \mid \phi \wedge \phi, \text{ where } e \in E$$

The semantics is given for a set of events $\omega \subseteq E$ and an ERL formula $\phi$. We write $\omega \models \phi$ when $\omega$ satisfies $\phi$, defined as follows:

- for any event $e \in E$, $\omega \models e$ iff $e \in \omega$,
- the logical connectives $\neg$ and $\wedge$ have the usual semantics.

From $\neg$ and $\wedge$ we derive the other logical connectives $\vee$, $\rightarrow$ etc. as usual.

Since we are interested in properties of sets of runs, we look at the satisfaction of ERL formulas by sets of sets of events: for any ERL formula $\phi$ and for any set of sets of events $\Omega$,

$$\Omega \models \phi \overset{def}{\iff} \forall \omega \in \Omega \quad \omega \models \phi \,.$$

We define the set $\llbracket \phi \rrbracket$ as $\llbracket \phi \rrbracket \overset{def}{=} \{\omega \subseteq E \mid \omega \models \phi\}$.

**Examples.** If $\Omega$ denotes the set of maximal runs of the Petri net of Figure 3.3, we can now express complex dependencies like

- $\Omega \models (a \wedge b) \rightarrow c$,
- $\Omega \models a \rightarrow (c \vee b')$,
- $\Omega \models b' \rightarrow (a \wedge \neg c \wedge \neg b)$,
- $\Omega \not\models c \vee b'$ because the maximal run $\{a', b\} \in \Omega$ does not satisfy this formula.

### Extended Reveals Relation

Any well-formed formula can be brought into *conjunctive normal form*:

$$\bigwedge_{i \in I} (b_{i,1} \vee \cdots \vee b_{i,n_i} \vee \neg a_{i,1} \vee \cdots \vee \neg a_{i,m_i})$$

which can also be written as

$$\bigwedge_{i \in I} ( \bigwedge_{a \in A_i} a \rightarrow \bigvee_{b \in B_i} b)$$

with $A_i = \{a_{i,1}, \ldots, a_{i,m_i}\}$ and $B_i = \{b_{i,1}, \ldots, b_{i,n_i}\}$.

Hence we focus on formulas of the form $\bigwedge_{a \in A} a \rightarrow \bigvee_{b \in B} b$, where $A$ and $B$ are two sets of events. This leads us to define the *extended reveals relation*.

**Definition 3.4 (Extended reveals relation).** *Let $\Omega \subseteq 2^E$ be a set of runs, and $A, B$ two sets of events. We define the* extended reveals *relation (in $\Omega$) $A \rightarrowtriangle B$ as*

$$\begin{aligned} A \rightarrowtriangle B \quad &\overset{def}{\iff} \quad \Omega \models \bigwedge_{a \in A} a \rightarrow \bigvee_{b \in B} b, \quad \textit{or equivalently,} \\ &\overset{def}{\iff} \quad \forall \omega \in \Omega \quad A \subseteq \omega \Rightarrow B \cap \omega \neq \emptyset \end{aligned}$$

Notice that the binary reveals relations $a \triangleright b$ correspond to the extended reveals relations between singletons $\{a\} \rightarrowtriangle \{b\}$.

**Examples.** If $\Omega$ denotes the set of maximal runs of the Petri net of Figure 3.3, we have for example

- $\{a, b\} \rightarrowtriangle \{c\}$ (every maximal run that contains $a$ and $b$ also contains $c$),
- $\{a'\} \rightarrowtriangle \{b, b'\}$ (every run which contains $a'$, contains either $b$ or $b'$),
- $\{a, a'\} \rightarrowtriangle \emptyset$ (no run contains $a$ and $a'$),
- $\emptyset \rightarrowtriangle \{a, b\}$ (every run contains either $a$ or $b$).

**Properties of the Extended Reveals Relation for General and Maximal Semantics.** Remember that the extended reveals relation is interpreted over a set $\Omega$ of runs which satisfy certain properties: for instance maximal runs, runs compatible with some timed constraints...

When one chooses for $\Omega$ the set of all runs (general semantics) or the set of maximal runs (maximal semantics), the extended reveals relation has the following strong properties. First, every set $A$ of incompatible events (i.e. such that $A \rightarrowtriangle \emptyset$) contains two incompatible events (i.e. events $a, b \in A$ such that $\{a, b\} \rightarrowtriangle \emptyset$. Second, the only cause for incompatibility between events is structural conflict, i.e. $\{a, b\} \rightarrowtriangle \emptyset$ iff $a \# b$.

These two properties are simple corollaries of the definition of runs. However, one should consider them as important properties of the general and maximal semantics and notice that they would not hold, for instance, with a timed semantics nor for contextual occurrence nets [19, 44, 117, 116] used for unfoldings of nets with read arcs, where weak causality may cause non binary conflicts. Non binary conflicts have also arisen from symbolic unfoldings of colored Petri nets [73, 52, 59].

### Coding the Set of Runs of an ON as an ERL Formula

For every finite ON $\mathcal{N}$, one can build an ERL formula $\Phi^{\mathcal{N}}_{max}$ such that $[\![\Phi^{\mathcal{N}}_{max}]\!]$ is the set of maximal runs of $\mathcal{N}$. By definition, a set of events of an ON is a general run iff it is conflict-free and closed under causality. We add the maximality constraint: "for any event $a$, if $a$ is enabled, then $a$ or an event in direct conflict with $a$ has to fire". For a given finite ON $\mathcal{N}$, we get the following formula $\Phi^{\mathcal{N}}_{max}$:

$$\begin{aligned}
\Phi^{\mathcal{N}}_{max} \quad = \quad & \bigwedge_{a,b \in E, a \prec b} (b \rightarrow a) && \text{(causal closure)} \\
\wedge \quad & \bigwedge_{a,b \in E, {}^\bullet a \cap {}^\bullet b \neq \emptyset} (\neg a \vee \neg b) && \text{(conflict-freeness)} \\
\wedge \quad & \bigwedge_{a \in E} \Big( \big( \underbrace{\bigwedge_{b \in E, b \prec a} b}_{a \text{ enabled}} \big) \rightarrow \big( a \vee \bigvee_{c \in E, {}^\bullet c \cap {}^\bullet a \neq \emptyset} c \big) \Big) && \text{(progress assumption)}
\end{aligned}$$

The size of the formula is quadratic in the number of events in $\mathcal{N}$.

Without the maximality constraint, this formula is similar to the "configuration constraint" used in [90].

For the example of Figure 3.3, one gets the formula:

$$\begin{array}{lll}
 & (a \rightarrow c) \wedge (b' \rightarrow a) & \text{(causality)} \\
\wedge & (\neg a' \vee \neg a) \wedge (\neg b' \vee \neg b) \wedge (\neg b' \vee \neg c) & \text{(conflicts)} \\
\wedge & (a \vee a') & \text{(progress assumption for } a \text{ and } a'\text{)} \\
\wedge & (a \rightarrow (c \vee b')) & \text{(progress assumption for } c\text{)} \\
\wedge & (a \rightarrow (b' \vee c \vee b)) & \text{(progress assumption for } b'\text{)} \\
\wedge & (b \vee b') & \text{(progress assumption for } b\text{)}
\end{array}$$

## 3.3 From ERL Formulas to Occurrence Nets: a Synthesis Procedure

A natural question arises from our formulation of dependencies between events: given an ERL formula $\phi$, is there an ON where the dependencies between events (under the maximal semantics) are the ones described by $\phi$? In other words, is there an ON $\mathcal{N}$ such that the set of maximal runs of $\mathcal{N}$ is $\llbracket\phi\rrbracket$?

This synthesis procedure allows us to understand what shape the dependencies between events can take.

We focus on the synthesis of reduced ONs, i.e. we assume that the formula does not imply that a variable is true or that two variables are equivalent. In other words, no formula of the type $(\phi \rightarrow e)$ or $(\phi \rightarrow (e \leftrightarrow f)$ (with $e, f$ variables of $\phi$) is a tautology.

The synthesis procedure is in two steps:

1. we first construct a net CN($\phi$) from the formula,
2. then we check if the set of maximal runs of CN($\phi$) is $\llbracket\phi\rrbracket$. This can be done by checking if $(\Phi_{max}^{\mathrm{CN}(\phi)} \leftrightarrow \phi)$ is a tautology.

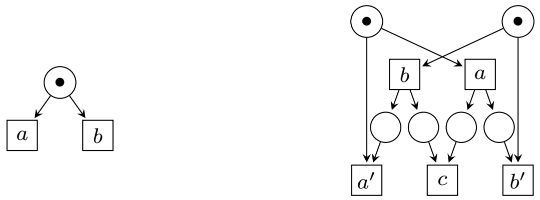
The construction is correct in the sense that if the formula $\phi$ describes the set of maximal runs of an occurrence net $\mathcal{N}$, then CN($\phi$) is a reduced occurrence net and its set of maximal runs is $\llbracket\phi\rrbracket$.

Conversely, if the constructed net CN($\phi$) is not a reduced occurrence net or if its set of runs is not $\llbracket\phi\rrbracket$, then our synthesis problem has no solution.

The construction works also for the general semantics. Only in the end we compare $\llbracket\phi\rrbracket$ to the set of general runs of CN($\phi$) instead of the set of maximal runs.

### Importance of Binary Constraints

Remember that ERL formulas and extended reveals relations were introduced to capture non binary dependencies between events. Therefore it is noticeable and quite surprising that binary constraints play a central role in our synthesis procedure: the construction of the net CN($\phi$) relies only on them. Non binary



(a) CN($\phi_1$)

(b) CN($\phi_2$)

Figure 3.4: Two examples of the construction CN($\phi$)

constraints do matter, but they only come into play at the end of the procedure, when the executions of the constructed net are compared with the semantics of the formula.

This paradox is due to the fact that our construction is designed for the setting of the general and maximal semantics, and for these semantics, the following property holds: if two nets $\mathcal{N}_1$ and $\mathcal{N}_2$ over the same set of event names have the same binary constraints, then they have the same runs, and consequently they have also the same non binary constraints. Note that this is again an important property of the maximal and general semantics. It would not hold for other selections of runs (timed semantics for instance).

### Construction of CN($\phi$)

Let $\phi$ be an ERL formula over a set $E$ of variables representing the occurrence of an event. The net CN($\phi$) is defined as follows. Its events are the variables $E$. Then each binary constraint of the form $e \rightarrow f$ or $\neg(e \wedge f)$ which is a logical consequence of $\phi$ is represented by a condition connected to the events concerned by the constraint:

- for each constraint of the form $e \rightarrow f$, a condition $b$ is created and connected to $e$ and $f$ such that ${}^\bullet b = \{f\}$ and $b^\bullet = \{e\}$,
- for each constraint of the form $\neg(e \wedge f)$, an initial condition $b$ is created and connected to $e$ and $f$ such that $b^\bullet = \{e, f\}$.

The constructed net CN($\phi$) is at most quadratic in the number of events (or variables) in $\phi$.

Actually, in order to get a smaller and more readable synthesized net, it is possible to represent only *minimal* binary constraints, i.e. binary constraints which cannot be deduced from others. This corresponds to keeping only immediate causalities and conflicts in the constructed ON.

### Examples

1. Consider the formula $\phi_1 \equiv \neg(a \wedge b)$. The only binary constraint is the formula itself. The net CN($\phi_1$) synthesized from this constraint is given

in Figure 3.4(a). It is a reduced ON but the empty run, which satisfies $\phi_1$, is not a maximal run of CN($\phi_1$). One concludes that there is no reduced ON $\mathcal{N}$ whose set of maximal runs is $[\![\phi_1]\!]$.

Actually, $\phi_1$ describes the set of *general* runs of CN($\phi_1$). For the set of maximal runs, the progress constraint $a \vee b$ is missing from $\phi_1$.

2. For the second example, we take the formula

$$\begin{array}{rl} \phi_2 \equiv & (a \rightarrow c) \wedge (b' \rightarrow a) \wedge (\neg a' \vee \neg a) \wedge (\neg b' \vee \neg b) \wedge (\neg b' \vee \neg c) \\ \wedge & (a \vee a') \wedge (a \rightarrow (c \vee b')) \wedge (a \rightarrow (b' \vee c \vee b)) \wedge (b \vee b') \end{array}$$

that codes the maximal runs of the ON of Figure 3.3 (see page 24). The minimal binary constraints implied by $\phi_2$ are $c \rightarrow a$, $c \rightarrow b$, $a' \rightarrow b$, $b' \rightarrow a$, $\neg(a \wedge a')$ and $\neg(b \wedge b')$. The net obtained by the synthesis from these constraints is represented in Figure 3.4(b). By construction, it has the same maximal runs $\{a, b, c\}$, $\{a, b'\}$ and $\{a', b\}$ than the original ON.

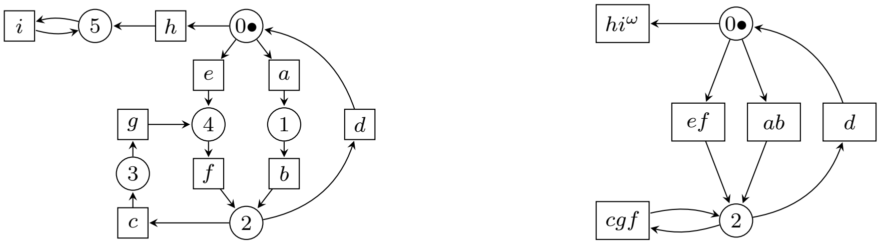
## 3.4 Tight Nets as a Canonical Form for Reduced ONs

The previous example illustrates that several ONs with very different shape can have the same maximal runs (a run being understood as a set of event names). Our synthesis procedure provides a canonical representative for each class of ONs. An interest of this class is that all the binary dependencies between events are directly represented in the net as causalities and conflicts. In particular, in the constructed nets, any reveals relation $a \rhd b$ is explicitly represented by causal arcs $b \prec^* a$. I discuss applications of this canonical form at the end of this chapter.

## 3.5 A Canonical Contraction for Safe Petri Nets

One interest of the facets abstraction is to provide a contracted (or reduced) version of a Petri net which preserves its maximal semantics. We compare with a simple operation on sequential systems which has a similar effect: Consider the sequential system shown in Figure 3.5(a). It is given here as a Petri net for convenience, but easily translated into an equivalent finite automaton of six states, eight transitions and initial state 0. When in state 0, the system can perform either $a$, $e$, or $h$. Whatever the choice of the first transition, however, in each case the *second* choice is imposed: after $a$ no other transition than $b$ is possible, after $e$ only $f$, and after $h$ only $i$.

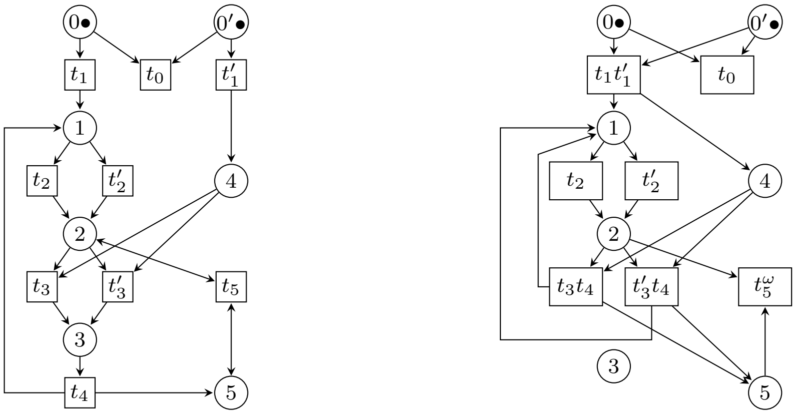
In Figure 3.5(b) each of the new transitions is labeled with the transition chain that it represents. Note that the infinite word $hi^\omega$ is obtained via a single macro-transition without post-place, since the word has no last transition. Of course, not all *temporal* properties of the system are preserved, since not all *finite* words survive the contraction: *abcg* is a word produced by a run in Figure 3.5(a),



(a) (A Petri net representation of) an automaton

(b) Its contraction

Figure 3.5: Contracting automata by removing non-branching states (here 1, 3, 4 and 5)



(a) A safe Petri net

(b) Its canonical contraction. (Place 3 is not used any more and could be forgotten.)

Figure 3.6: Overview of the canonical contraction of a safe Petri net

but not in Figure 3.5(b) which has no intermediate word between $(ab)$ and $(ab)(cgf)$. However, one sees quickly that the *maximal* words – which coincide with the *infinite* words – of the original system of Figure 3.5(a) are in bijection with the infinite words of the contracted system in Figure 3.5(b).

We propose now to combine the ideas shown, on the one hand, in the automata contraction such as in the example of Figure 3.5, and on the other hand of the facet contraction in the context of occurrence nets.

### 3.5.1 Macro-Transitions

For this we identify *macro-transitions* in safe Petri nets that allow contraction with preservation of *maximal* semantics, and thus to give a contracted normal form for any given Petri net. If the definition is applied to occurrence nets,

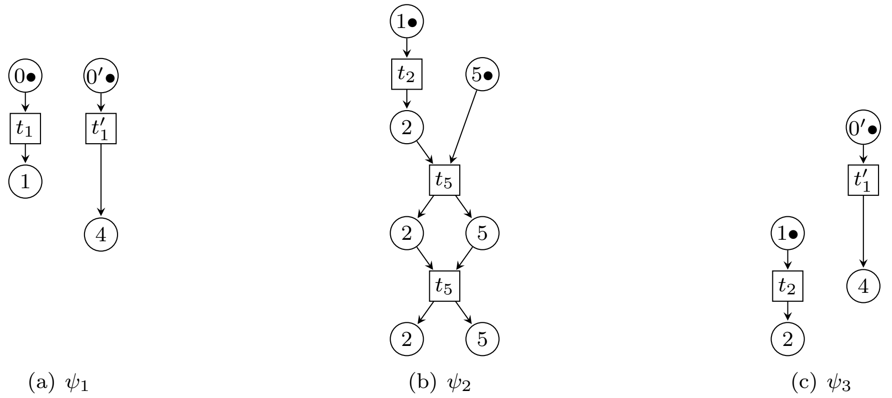


Figure 3.7: Two examples ($\psi_1$ and $\psi_2$) and a counter-example ($\psi_3$) of macro-transitions of the Petri net of Figure 3.6(a)

we obtain exactly the facets according to [84, 85, 11, 12]. At the same time, the reduced net has never more, and generally much fewer, transitions than the original net.

Technically, a *macro-transition* $\psi$ of a safe Petri net $\mathcal{N}$ is defined as a piece of partial-order behavior of $\mathcal{N}$ which satisfies the property that from any reachable marking $M$ which enables $\psi$, any maximal run that starts with a nonempty prefix of $\psi$, contains the *entire* $\psi$.

As a particular case, any single transition constitutes a macro-transition.

More interesting is the case of facets: any facet of an occurrence net $O$ is a macro-transition of $O$.

**Examples.** Figure 3.7 shows two examples and one counter-example of macro-transitions of the Petri net of Figure 3.6(a).

- In $\psi_1$ we have two events: an occurrence of $t_1$ and one of $t'_1$. The initial conditions of $\psi_1$ are mapped to places 0 and 0′ of $\mathcal{N}$. The only reachable marking of $\mathcal{N}$ which contains $\{0, 0'\}$ is $\{0, 0'\}$ itself; in $\{0, 0'\}$, if one of the two transitions fires, the other one will necessarily fire in any maximal run.

- Consider now $\psi_2$: again the only reachable marking of $\mathcal{N}$ which contains $\{1, 5\}$ is $\{1, 5\}$ itself. From it, if $t_2$ fires, it is necessarily followed by an infinite sequence of firings of $t_5$. $\psi_2$ is exactly a prefix of it.

- Concerning $\psi_3$, it is exactly a prefix of every maximal run from $\{1, 0'\}$ starting by an occurrence of $t_2$, but not of every run starting by an occurrence of $t'_1$ (because $t'_2$ can fire instead of $t_2$).

**$\Psi$-contracted net**

Given a set $\Psi$ of macro-transitions of a Petri net $\mathcal{N}$, we construct the $\Psi$-*contracted net* $\mathcal{N}_{/\Psi}$ by replacing the transitions of $\mathcal{N}$ by new transitions which summarize the macro-transitions.

In a branching process of the contracted net $\mathcal{N}_{/\Psi}$, every event represents an occurrence of a macro-transition. Expanding every event to the content of the corresponding macro-transition yields a branching process of $\mathcal{N}$.

### 3.5.2 Canonical Contraction

Notice that in general not every marking reachable in $\mathcal{N}$ is reachable in $\mathcal{N}_{/\Psi}$. This is actually what allows us to skip some intermediate markings and give a more compact representation of the behavior of the net.

In this sense we can say that a complete contracted net $\mathcal{N}_{/\Psi}$ is more compact than another $\mathcal{N}_{/\Psi'}$ if all markings reachable in $\mathcal{N}_{/\Psi}$ are also reachable in $\mathcal{N}_{/\Psi'}$. We define a canonical contracted net $\overline{\mathcal{N}}$ which is optimal w.r.t. this criterion.

The result is a unique contracted safe Petri net with no more macro-transitions than transitions in the original net. The construction provides a *canonical* version for any given safe Petri net, whose maximal behavior offers a condensed view of the maximal behavior of the original net.

Computing the contraction (with finite representations of the macro-transitions) is in general costly (computing the reveals relation on the unfolding of a finite Petri net is PSPACE-complete [85]), but in practice many syntactic sufficient conditions can be used to identify macro-transitions. Hence our contraction appears as an optimal, canonical contraction, to which other contractions based on macro-transitions can be compared.

## 3.6 Conclusion on Reduction and Contraction

As announced, the contraction procedure coincides with the facets reduction when it is applied to an ON.

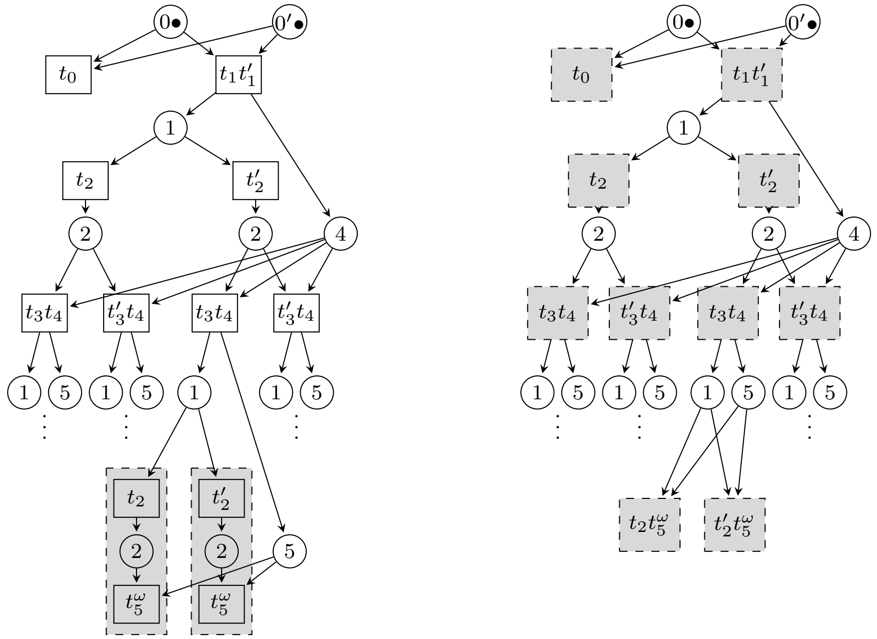
As illustrated in Figure 3.8, the operation of reduction does not entirely commute with unfolding. That is, in general, the unfolding $U(\overline{\mathcal{N}})$ of reduced Petri net $\overline{\mathcal{N}}$ is coarser, as an occurrence net, than the reduction $\overline{U(\mathcal{N})}$ of the original net $N$'s unfolding. In the example of Figure 3.8, the facets labeled $t_2t_5^\omega$ and $t_2't_5^\omega$ in $\overline{U(\mathcal{N})}$ are both split into two events of $U(\overline{\mathcal{N}})$.

However, one retrieves the reduction of $U(\mathcal{N})$ by applying again the reduction to the unfolding $U(\overline{\mathcal{N}})$ of the contracted net.

### 3.6.1 Applications and Related Work

**Contraction of Nets**

It is known that structural transformations can facilitate verification of some system properties, as witnessed by e.g. Berthelot [28], Desel and Merceron [66],



(a) The unfolding of the contracted Petri net of Figure 3.6(b). Remark that the unfolding is not reduced: the last occurrence of $t_2$ and the following $t_5^\omega$ are in the same facet (similarly for $t'_2$ and the following $t_5^\omega$).

(b) Its reduction (or contraction) is isomorphic to the reduction of the unfolding of the Petri net of Figure 3.6(a).

$$\begin{tikzcd}
\mathcal{N} \arrow[r, "\text{contraction}"] \arrow[dd, "\text{unfolding}"'] & \overline{\mathcal{N}} \arrow[d, "\text{unfolding}"] & \\
 & U(\overline{\mathcal{N}}) \arrow[dr, "\text{contraction}"] & \\
U(\mathcal{N}) \arrow[rr, "\text{contraction (or reduction)}"'] & & \overline{U(\mathcal{N})}
\end{tikzcd}$$

Figure 3.8: Unfolding and contraction.

and other works. Our contraction technique preserves properties that depend only on the language of the *maximal* runs of the system, such as liveness properties. It has also been used in [86] for fault diagnosis.

By computing offline the canonical version, verification procedures for any property that depends only on the maximal run behavior can be run on the smaller contracted net instead. This could be exploited in predicting (in the sense e.g. of failure prognosis, see [91]) events that inevitably will occur.

Best and Randell [33] also considered atomicity of subnets in occurrence graphs, focusing on non-interference in the temporal behavior and identifying atomic and hence contractable blocks of behavior. The structures obtained can be embedded into non-branching occurrence nets, allowing the approach to be compared with ours. However, while the construction of facets appears geometrically similar, the approach of [84, 85, 11, 12] focuses on the question of *logical occurrence* regardless of the order in which events occur.

### Energy Consumption in Asynchronous Circuits

We have started to discuss with colleagues in Newcastle upon Tyne about using our canonical contraction technique for estimating the energy consumption in asynchronous circuits [96]. There, identifying blocks of partial-order behavior like our macro-transitions, improves the existing techniques by avoiding state explosion.

### Synthesis of Nets

The synthesis problem for Petri nets has been widely studied. It consists in answering whether, given a behavior, there exists a Petri net with this behavior. The behavior can be specified as a transition system [27, 67, 8, 46] or a language. In [62], the behavior is bounded by two regular languages. Most of the time, the synthesis procedure is based on the notion of region [72, 9].

Synthesis for finite set of labeled partial orders are considered in [26, 25].

Note that our synthesis techniques deals only with *finite* occurrence nets. Naturally, one would hope to obtain synthesis procedures for occurrence nets of arbitrary size, imposing only regularity properties; the set of events would then be structured by an adequate equivalence relation of finite index. However, the technical difficulties posed by this general endeavor have not been resolved.

## 3.7 Back to Time Petri Nets

I come back now to my initial concern about dependencies between events in time Petri nets. I know that they induce more general dependencies than untimed Petri nets with or without the maximal progress assumption. In particular binary dependencies do not play the same central role. Anyway, some questions remain. For instance, given an ERL formula, I can use our synthesis procedure to determine whether it describes the dependencies between events in an untimed Petri net (with or without the maximal progress assumption), but I do not know yet a procedure to decide the same for TPNs. In other words, I do not know precisely how complex the dependencies in unfoldings of TPNs can be.

Also, I have not explored how to compute the reveals relation on a timed occurrence net. This could find applications, for instance in diagnosis or prediction, like in the untimed case.

Finally, it would be worth exploring how the idea of 'tightening' a net can be used in a definition of unfoldings for TPNs. Indeed, tightening forces an

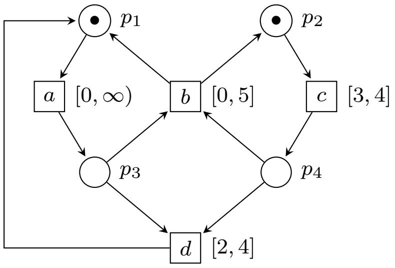


Figure 3.9: An extended free choice safe time Petri net.

ordering between events, like read arcs in unfoldings of TPN. When putting an event $e$ in the unfolding of a TPN, the symbolic constraint attached to it may very well concern the date of some events that will for sure occur (revealed by $e$).

### 3.7.1 The Case of Extended Free Choice (T)PNs

An *extended free choice* (time) Petri net [31, 65] is a (time) Petri net where every two transitions $t$ and $t'$ in conflict, have exactly the same preset:

$$^\bullet t \cap {}^\bullet t' \neq \emptyset \implies {}^\bullet t = {}^\bullet t' \ .$$

Extended free choice Petri nets have been extensively studied and have many interesting properties. When defining unfoldings of TPNs during my PhD, I identified this class a simple case where the complex dependencies between events that I evoked at the beginning of this chapter do not occur.

Then I have expected that extended free choice Petri nets would behave well also from the point of view of the reveals relations, but for the moment I did not find any satisfactory result in this direction.

On the other hand, the good properties of extended free choice TPNs were confirmed by the following nice result.

### 3.7.2 Back in Time Petri Nets

The time progress assumption is at the core of the semantics of real-time formalisms. It simply says that the delay between two consecutive actions must be non-negative. Zero delays, although not always realistic, are usually allowed and often used as a convenient modeling feature to decompose an action in two logical steps; but negative delays are rejected.

However, regarding concurrent transitions, this can be disputable. Indeed, concurrent transitions can be executed in parallel on remote computers. From this point of view, their relative execution time does not matter so much as long as they are really independent and no communication between the remote devices establishes a causal dependency between them.

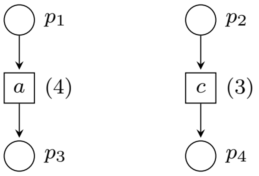


Figure 3.10: A process of the time Petri net of Figure 3.9. The dates of the events are in brackets. This process has two linearizations: $(c, 3)(a, 4)$ and $(a, 4)(c, 3)$. Only the first one is accepted by the classical semantics, but both are accepted by the relaxed semantics.

In [56], we explore the original idea of simply dropping the time progress assumption between concurrent transitions in safe TPNs. Of course this generates new timed words and in general it changes dramatically the semantics of the model. But in the case of extended free choice TPNs, the relaxed semantics remains related to the classical semantics in the sense that it produces the same partial-order executions.

More precisely, every process of an extended free choice TPN $\mathcal{N}$ under the relaxed semantics, is a prefix of a process of $\mathcal{N}$ under the classical semantics. Concerning timed words, this implies that every timed word accepted by $\mathcal{N}$ under the relaxed semantics is a prefix of a permutation of a timed word accepted by $\mathcal{N}$ under the classical clocks-on-tokens semantics.

As an example, consider the TPN of Figure 3.9. The timed word $(a, 4)(c, 3)$ is accepted by the relaxed semantics: after firing $a$ at time 4, $p_2$ and $p_3$ are marked, with $dob(p_2) = 0$ and $dob(p_3) = 4$, and the current time is 4. After that, time may go back to 3, and $c$ can fire. The process shown on Figure 3.10, which represents this execution, is also the process corresponding to $(c, 3)(a, 4)$, which is accepted under the classical semantics.

This result establishes a nice relation between partial-order semantics and time progress assumption and confirms the strong properties of extended free choice safe time Petri nets when partial order semantics is considered.

Finally, the algorithms for the analysis of timed models construct and solve systems of linear constraints on temporal parameters (like occurrence time, value of clock. . . ) By relaxing the constraints about time progress, our relaxed semantics would generate fewer inequalities and fewer symbolic states. And it preserves properties like fireability of a transition or reachability of a place. We plan to experiment the construction of the symbolic state graph generated by our semantics and evaluate how much it improves the analysis algorithms.

# Chapter 4

# Behavioral Comparisons Between Real-Time Distributed Systems

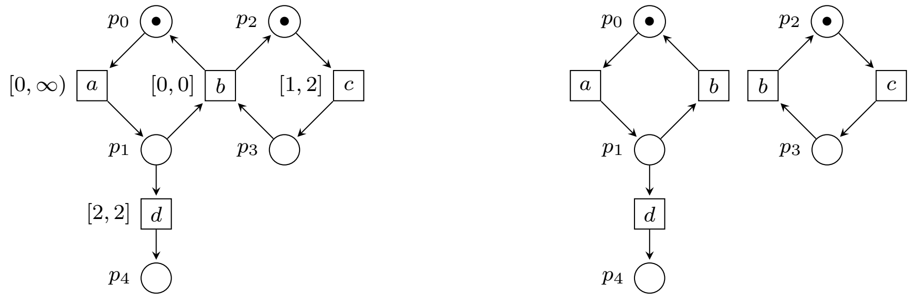
*This chapter presents behavioral comparisons taken from my contributions on translation from TPN to NTA [14, 15] and on shared clocks in NTA [13], both obtained during Sandie Balaguer's PhD.*

Behavioral comparisons are a classical tool for the study of formal models. In a setting like distributed real-time systems, an additional difficulty is to deal with the variety of models, here at least time Petri nets (TPNs) and networks of timed automata (NTA). Hence, behavioral comparisons must be able to deal with heterogeneous formalisms or rely on a common semantics, like timed transitions systems (TTS).

Then behavioral comparisons allow one to compare the expressiveness of the formalisms, to switch from a formalism to another, and to reuse techniques and tools developed initially for one particular formalism. Studying a complex system generally requires the use of multiple techniques and tools. Consequently the system must be translated from one formalism to another. The difficulty is to show that the different representations are equivalent.

Many transformations have been proposed, for instance between TPNs and NTA; we observe the following.

- The transformations mainly rely on natural structural equivalences between the basic elements of the formalisms. For instance, the location of an automaton corresponds to a place of a Petri net, a transition of a Petri net corresponds to a tuple of synchronized transitions of an NTA, and the delay interval associated with a transition of a Petri net becomes a pair (guard, invariant) in a timed automaton.



(a) A time Petri net

(b) A decomposition of its untimed support into S-subnets

Figure 4.1: Decomposition in components

- Beyond these natural equivalences, limitations for more general models are not clear.

Indeed, the natural transformations tend to preserve concurrency. But when the transformations become less immediate, one uses tricks that unfortunately destroy concurrency.

Therefore it is not surprising that the first works about formal comparisons of the expressiveness of these models [24, 41, 49, 112, 23] do not consider preservation of concurrency. In [49], a structural transformation from TPN to NTA is defined. This transformation builds one timed automaton per transition of the TPN and preserves weak timed bisimilarity. In the other direction, [22] shows that there exist timed automata that are not weakly timed bisimilar to any TPN. [29] shows that equipping TPNs with priorities reduces the gap with timed automata.

Because I work on several aspects of distributed real-time formalisms, I am interested in comparisons between these formalisms. For instance, I am interested in comparing the dependencies that appear in TPNs with those that appear in NTA, and unfoldings of TPNs with unfoldings of NTA.

More recently, I have tackled problems related to the implementability of distributed real-time systems (I describe this in the next chapter). Here also I need to study transformations from high-level models to lower-level ones, which are closer to what physical devices can execute. Then I need tools to formalize how these transformations preserve the specified behavior.

## 4.1 Limitations of Behavioral Comparisons for Sequential Systems

Consider the TPN of Figure 4.1(a). Although this is not explicitly specified in the model, it is very reasonable to think that it models a distributed system

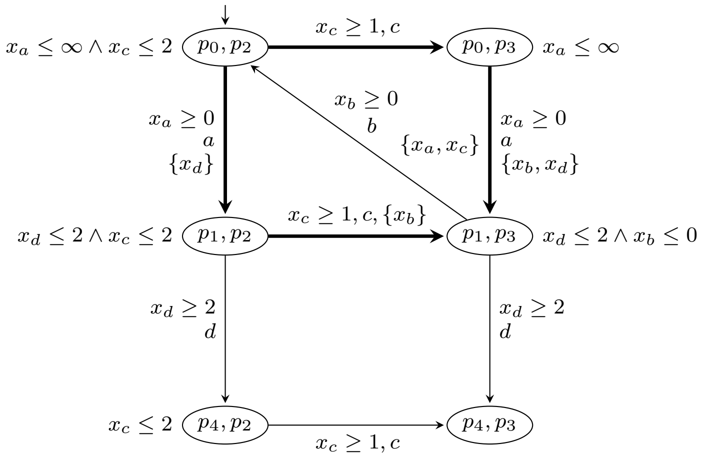


Figure 4.2: The semantics of the TPN of Figure 4.1(a) as a timed automaton

made of two components, shown on Figure 4.1(b), which communicate via the common transition $b$.

It can however be translated to a timed automaton called marking TA and introduced in [82]. The result is shown in Figure 4.2. The marking TA of a TPN $(P, T, \mathit{pre}, \mathit{post}, M_0, \mathit{efd}, \mathit{lfd})$ is the TA $(L, \ell_0, C, \Sigma, E, \mathit{Inv})$ where the set of actions $\Sigma$ is $T$, the set of locations is the set of reachable markings ($L \subseteq 2^P$), with the initial location $\ell_0 = M_0$. One clock $x_t \in C$ is associated with each transition $t$ (they could alternatively be associated with places). For every marking $M$ and every transition $t$ enabled in $M$, we draw an edge from $M$ to $M \setminus {}^\bullet t) \cup t^\bullet$ with the guard $x_t \geq \mathit{efd}(t)$. This edge resets the clocks of all the transitions newly enabled by the firing of $t$. Finally, every location $M$ has an invariant $\bigwedge_{{}^\bullet t \subseteq M} x_t \leq \mathit{lfd}(t)$, which ensures that no enabled transition overtakes its latest firing delay.

The marking TA of a TPN is strongly timed bisimilar to the TPN. Yet we note that concurrency is not explicit in this TA, as it naturally mimics the sequential semantics of the TPN, even though we can observe a diamond (bold edges) that shows the possible interleavings between actions $a$ and $c$.

Here I focus on the preservation of the semantics w.r.t. distribution. Since both TPNs and NTA were designed to model distributed systems, I consider that not only their sequential behavior as timed transition systems is relevant, but also their distributed behavior.

## 4.2 Concurrent Bisimulations

The limitations of behavioral comparisons for distributed systems have already been identified, and more powerful comparisons were proposed. The notion of *concurrent bisimulation* was developed in the Petri net community [32, 114], essentially in the 90'. The idea is to rewrite the definition of bisimulation and focus on the simulation of *partial order runs* rather than single actions. Therefore a system that performs $a$ and $b$ concurrently can be distinguished from a sequential system that can perform $ab$ and $ba$. This allows one to distinguish, for example, the behavior of a Petri net with concurrent actions, from the sequential behavior represented by its marking graph.

These works consider only fully asynchronous untimed systems. I started to think of generalizing these notions in order to capture behavioral comparisons between real-time distributed systems. I visited Lucia Pomello (who is one of the main authors of concurrent bisimulations) and Luca Bernardinello in Milano in February 2013 and I proposed to look at networks of timed automata.

It was natural to assume that, when one compares two NTA, one first assume that the two NTA have the same number of automata and try to match the automata of one NTA with the automata of the other, such that their alphabets of actions $\Sigma_i$ correspond. We discovered that, in this setting, concurrent bisimulation

- can easily be extended to the real-time setting,
- but coincides exactly with usual sequential bisimulation between the two systems.

This limits, to my eyes, the interest of concurrent bisimulations, but also confirms the importance of identifying components and comparing models w.r.t. their decompositions into components.

## 4.3 Identification of Components

Dealing with the distributed behavior of models implies that, if a model represents a system that involves several components, then the model should be structured so that it is easy to identify each component.

In order to formalize behavioral comparisons in the context of real-time distributed models, we take into account the distribution of actions over a set of components, each component having its own alphabet of actions.

As NTA are built as compositions of timed automata, it is natural to identify each automaton as one component. This gives an immediate decomposition of the model into components.

### 4.3.1 S-subnets as Components for Petri Nets

Identifying components in TPNs is not as immediate as in an NTA. Nevertheless, in practice, when a system is modeled as a TPN, the designer knows its physical structure and builds the TPN as a composition of components that model the subsystems. Anyway, if a TPN is given without its decomposition, these components can be identified.

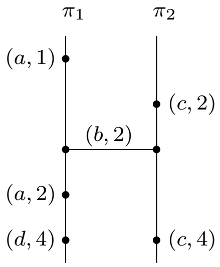


Figure 4.3: A timed trace representing a run of the TPN of Figure 4.1(a), with two components ($\pi_1$ and $\pi_2$) and actions $a$, $b$, $c$ and $d$. Actions $a$ and $d$ are local to component $\pi_1$, $c$ is local to $\pi_2$ and $b$ is a synchronization of the two components.

Given a TPN $\mathcal{N} = (P, T, pre, post, efd, lfd, M_0)$, we proposed in [14, 15] to focus on its untimed support $(P, T, pre, post, M_0)$ and to decompose it in *S-subnets*. Each S-subnet $\mathcal{N}' = (P', T', pre', post', M'_0)$ is induced by a subset of places $P' \subseteq P$ and all the connected transitions and has the property that for every transition $t \in T'$, $|pre'(t)| = |post'(t)| = 1$; moreover exactly one place must be marked in the initial marking, defined as $M'_0 \overset{\text{def}}{=} M_0 \cap P'$. Hence every S-subnet can be viewed as an automaton where the active location is the marked place. Therefore we consider these S-subnets as the appropriate notion of *components* for TPN.

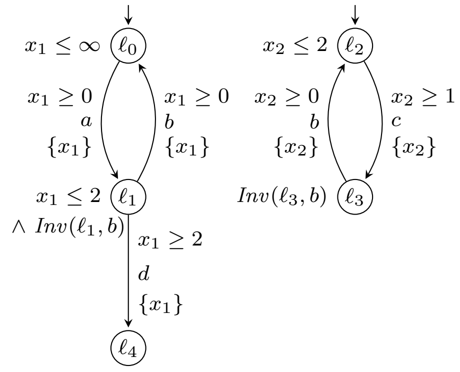
The question is to decompose a given Petri net into a set of S-subnets which cover the net. This is not always possible. Decomposition algorithms (for similar decompositions) are described in [65, 87, 61] and rely on the notion of S-invariants [95] defined using the incidence matrix of the nets. The number of places in the decomposition is never more than $|P|^2$ and the number of transitions never more than $|T| \times |P|$.

Figure 4.1(a) illustrates the decomposition of a Petri net in S-subnets.

## 4.4 Behavioral Comparisons Based on Distribution of Actions

Once distributed real-time formalisms come with a distribution of actions over identified components, it is possible to incorporate this information into the definition of new behavioral comparisons.

We defined in [14, 15] a notion of *timed trace* as a partial order representation of executions of our models for real-time distributed systems. Timed traces represent executions of either NTA or TPN on which components have been identified. Each action is associated with a set of components that always perform it together and simultaneously, therefore it may be local or shared (synchronizations).



$$Inv(\ell_1, b) \equiv \neg\ell_3 \vee x_1 \leq 0 \vee x_2 \leq 0$$
$$Inv(\ell_3, b) \equiv \neg\ell_1 \vee x_1 \leq 0 \vee x_2 \leq 0$$

Figure 4.4: An NTA which is distributed timed (strongly) bisimilar to the TPN of Figure 4.1(a). Such NTA can be computed automatically from the TPN by a procedure described in next chapter.

Every timed word over the alphabet of actions $\Sigma$ induces a corresponding timed trace.

Figure 4.3 gives a representation of a timed trace. Each component is represented by a vertical line, and each event is represented by a dot or dots connected by a horizontal line, depending on whether it occurs on one component or on several components. Each event is also labeled by a pair indicating its action label and its occurrence date.

We can now define behavioral comparisons for real-time distributed models.

**Definition 4.1.** *Two models are* distributed timed language equivalent *if they have the same distribution of actions and accept the same timed language (and then the same timed traces).*

Of course one can also combine the distribution of actions with other behavioral equivalences like bisimulation.

**Definition 4.2.** *Two models are* distributed timed bisimilar *if they have the same distribution of actions and are timed bisimilar.*

These definitions allow us to distinguish for instance between the TPN $\mathcal{N}$ of Figure 4.1(a) (given with its decomposition into components), and the timed automaton obtained from its marking graph, depicted in Figure 4.2. On the other hand, $\mathcal{N}$ remains equivalent to the NTA of Figure 4.4, which has two components and the same distribution of actions than $\mathcal{N}$.

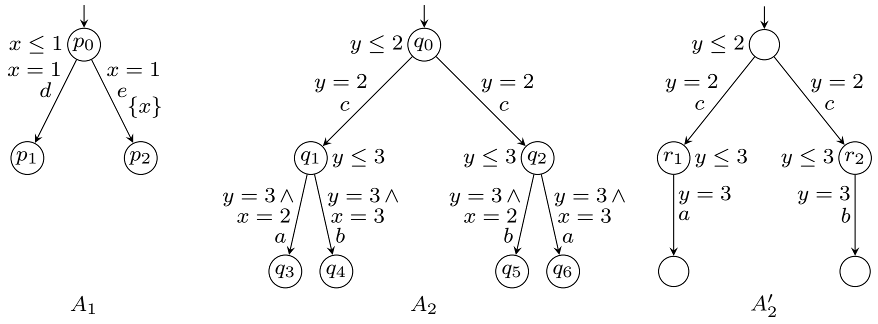


Figure 4.5: $A_1 \parallel A_2$ and $A_1 \parallel A'_2$ are distributed timed bisimilar but $A_2$ and $A'_2$ do not have the same behavior in the context of $A_1$.

## 4.5 Contextual Transition System and Contextual Bisimulation

In some cases, even distributed timed bisimulation is not sufficient. Consider the two NTA $A_1 \parallel A_2$ and $A_1 \parallel A'_2$ represented in Figure 4.5. Note that clock $x$, which is reset by $A_1$, is also read by $A_2$. The situation is asymmetric and we consider that $x$ is actually owned by $A_1$ and only read by $A_2$. The two NTA $A_1 \parallel A_2$ and $A_1 \parallel A'_2$ have the same distribution of actions and are timed bisimilar. Yet, $A_2$ and $A'_2$ do not have the same behavior in this NTA.

Intuitively $A_2$ needs to read $x$ when in $q_1$ or in $q_2$ at time 3, because the value of $x$ determines whether it will perform $a$ or $b$. Notice by the way that this reading is really mandatory because, depending on the choice done by $A_1$ at time 1, the value of $x$ at time 3 is either 2 or 3.

On the other hand, $A'_2$ has no choice at time 3.

As a matter of fact, when $A_2$ takes an edge labeled by $c$, his choice determines the action that it will play at time 3, but it has to read the remote $x$ to know the result. On the other hand, when $A'_2$ takes an edge labeled by $c$, it chooses alone the action that it will play at time 3.

Anyway $A_1 \parallel A'_2$ is bisimilar to $A_1 \parallel A_2$. Here the bisimulation relates $(p_1, q_1)$ to $(p_1, r_1)$ and $(p_2, q_1)$ to $(p_2, r_2)$. Hence it indicates which $c$ one of the systems must take at time 2 to simulate the other, but *this choice is done according to the state of the whole system.* This is not suitable if we take the point of view of a component that does not have a global view of the system.

Because of the distributed nature of the system, a component cannot observe the moves and the state of the other and must choose its local actions according to its partial knowledge of the state of the system.

What we see here is that, if we focus on the point of view of $A_2$ and $A'_2$, these two automata do not have the same behavior. Therefore we need a behavioral comparison that distinguishes $A_2$ from $A'_2$.

If the systems were defined as products of two automata, it would be possible to impose bisimilarities $A_1 \approx A'_1$ and $A_2 \approx A'_2$. But this is not appropriate here for two reasons:

- this would be unnecessarily strong: assume for simplicity that, like in the previous example $A_1 = A'_1$; composing $A_2$ or $A'_2$ with $A_1$ restricts their behavior, and there is no problem if they differ on a part of their behavior that is anyway prevented by $A_1$;
- moreover the semantics of a component in isolation is not well defined when there are shared clocks. There would be a problem also with TPNs, since the delay intervals concern all the components that participate in a transition: they do not restrict the behavior of a single component taken in isolation.

What we need is that $A_1$ *in the context of* $A_2$ behaves like $A'_1$ *in the context of* $A'_2$. We formalize this idea by the notion of *contextual timed transition system* (contextual TTS).

### 4.5.1 Contextual Timed Transition System

For simplicity, we stay in the framework of networks of two components.

We defined in [13] the *contextual TTS of* $A_2$ *in the context of* $A_1$, denoted $TTS_{A_1}(A_2)$. It is a TTS with transitions labeled either by positive reals (for time passage) or by actions of $A_2$. Each state of $TTS_{A_1}(A_2)$ is a pair $(S_1, s_2)$ where $s_2$ is a state of $A_2$ and $S_1$ is a nonempty *set* of states of $A_1$. Such a state can be seen as the knowledge of $A_2$ at a time of the execution: it knows in which precise state $s_2$ it is currently, but it does not know precisely the state of $A_1$; therefore the set $S_1$ stores all the possible states for $A_1$ according to what $A_2$ has observed.

To illustrate contextual TTS, consider $A_1$ and $A_2$ of Figure 4.5. The initial state is $\big(\{(p_0, x = 0)\}, (q_0, y = 0)\big)$. From this contextual state, $A_2$ waits 2 time units and we reach the contextual state $\big(\{(p_1, x = 2), (p_2, x = 1)\}, (q_0, y = 2)\big)$. Indeed, during this delay, $A_1$ has to perform either $e$ (which resets $x$), or $d$. Now, from this contextual state, we can take an edge labeled by $c$, and reach $\big(\{(p_1, x = 2), (p_2, x = 1)\}, (q_1, y = 2)\big)$. After a delay of one time unit, we get $\big(\{(p_1, x = 3), (p_2, x = 2)\}, (q_1, y = 3)\big)$. Lastly, $a$ can be taken, because it is enabled by $((p_2, x = 2), (q_1, y = 3))$ in the NTA, and the reached contextual state is $\big(\{(p_2, x = 2)\}, (q_3, y = 3)\big)$.

This notion of contextual TTS resembles the powerset construction used in game theory to capture the knowledge of an agent about another agent [109]. Also in timed games under partial observability [36, 64], similar techniques are used to partition states based on the observation. Finally, reasoning about knowledge of agents is the aim of epistemic logics [80], which have been extended to real-time in [118, 70].

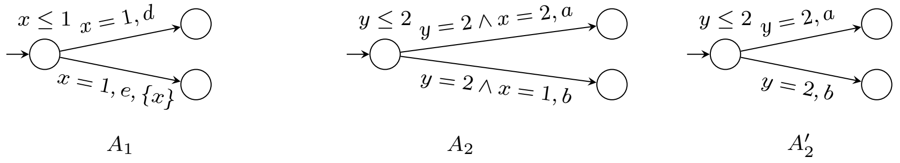


Figure 4.6: $TTS_{A_1}(A_2)$ and $TTS_{A_1}(A'_2)$ are bisimilar, and yet $A_1 \parallel A_2$ is not bisimilar to $A'_1 \parallel A'_2$.

### Contextual Bisimulation Alone is not Enough

One could now expect that the two bisimilarities $TTS_{A_1}(A_2) \approx TTS_{A'_1}(A'_2)$ and $TTS_{A_2}(A_1) \approx TTS_{A'_2}(A'_1)$, when they hold, imply global bisimilarity. This is wrong in general.

In the example of Figure 4.6, we have $TTS_{A_1}(A_2) \approx TTS_{A_1}(A'_2)$ (here also $A'_1 = A_1$), and yet $A_1 \parallel A_2$ is not bisimilar to $A'_1 \parallel A'_2$.

To summarize, we have seen with the example of Figure 4.5 that global bisimulation does not imply contextual bisimulations, and we see now that the converse is not true either. Nevertheless, an interesting case arises.

### Unrestricted Contextual TTS

We say that there is no restriction in $TTS_{A_1}(A_2)$ if whenever a local step is possible from a reachable contextual state, then it is possible from all the states $(s_1, s_2)$ that are grouped into this contextual state. In the example of Figure 4.6, there is a restriction in $TTS_{A_1}(A_2)$ because we have seen that $a$ is possible only from $((p_2, x = 2), (q_1, y = 3))$, but not from all the states merged in $\big(\{(p_1, x = 3), (p_2, x = 2)\}, (q_1, y = 3)\big)$.

The class of models which yield unrestricted contextual TTS is interesting because, in the case of NTA, it generalizes the class of NTA without shared clocks: *NTA without shared clocks have no restriction, but shared clocks do not always induce restriction, see next chapter.*

Moreover, when there is no restriction, contextual bisimulations correspond to global bisimulation.

**Lemma 4.1 (Unrestricted Contextual TTS Preserve Bisimulation).** *Let $A_1 \parallel A_2$ and $A'_1 \parallel A'_2$ represent two systems with the same distribution of actions. If there is no restriction neither in $TTS_{A_1}(A_2)$ nor in $TTS_{A'_1}(A'_2)$, then*

$$A_1 \parallel A_2 \approx A'_1 \parallel A'_2 \iff \begin{cases} TTS_{A_1}(A_2) \approx TTS_{A'_1}(A'_2) \wedge \\ TTS_{A_2}(A_1) \approx TTS_{A'_2}(A'_1)\ . \end{cases}$$

# Chapter 5

# Implementability of Real-Time Distributed Systems

> *This chapter (as well as the previous one) is based on my contributions [14, 15] and [13] obtained during Sandie Balaguer's PhD. I view them as first contributions about implementability of real-time distributed systems, which is the topic I would like to develop in priority in the coming years. Therefore, I conclude this chapter with several perspectives on the subject.*

Formal models for real-time systems, like timed automata [2] and time Petri nets [102], have been extensively studied and have proved their interest for the verification of real-time systems. On the other hand, the question of using these models as specifications for designing real-time systems raises some difficulties. One of those comes from the fact that the real-time constraints introduce some artifacts and because of them some syntactically correct models have a formal semantics that is clearly unrealistic. One famous situation is the case of Zeno executions, where the formal semantics allows the system to do infinitely many actions in finite time. But there are other problems, and some of them are related to the distributed nature of the system. These are the ones I tackle in priority.

One approach to implementability problems is to formalize either syntactical or behavioral requirements about what should be considered as a reasonable model, and reject other models. Another approach is to adapt the formal semantics such that only realistic behaviors are considered.

These techniques are preliminaries for dealing with the problem of implementability of models. Indeed implementing a model may be possible at the cost of some transformation, which make it suitable for the target device. By the way these transformations may be of interest for the designer who can now use high-level features in a model of a system or protocol, and rely on the transformation to make it implementable. After defining the transformations one has to formalize how they preserve the specified behavior, or how far they are from it.

In the following sections, I present two transformations. Their correctness is expressed w.r.t. the behavioral comparisons defined in the previous chapter.

## 5.1 A Translation from Safe TPN to NTA which Preserves Distribution

Several translations have been proposed between timed extensions of Petri nets and NTA. First, recall that every safe TPN can be translated to a strongly timed bisimilar TA (see Figure 4.2).

My motivation for looking at transformations from TPN to NTA is to question implementability of TPNs on distributed architectures. To this respect, I consider NTA closer to implementation than TPN because components are well identified in NTA and also because clocks are assigned to components, which is a useful information if one is to implement a system. However, clocks may be shared; then implementation is more difficult, but at least the presence of shared clocks is a clear signal for an implementation difficulty. Next section is dedicated to this problem.

Preserving the distribution of actions is also interesting for other reasons: first, a transformation is much more readable if it preserves the components and yields a model that is closer to the real system; second, preserving the components avoids combinatorial explosion of the size of the model and makes it possible to use modular analysis based on the components or partial order techniques, which are crucial when one analyzes large distributed systems.

Here I show how a TPN which is decomposable into components (see Section 4.3.1) can be translated to an NTA which respects the decomposition. How to decompose the TPN is not the purpose here: I assume that the decomposition is known and focus on the problem of coding the time constraints of the TPN using the syntax of NTA.

The result is an NTA which is distributed timed bisimilar (see Definition 4.2, page 39) to the original TPN.

Some transformations in the literature have considered preserving some properties of distribution. In [45], the authors propose a translation from bounded timed-arc Petri nets (another variant of Petri nets extended with time) to NTA, based on the decomposition of the net in sequential components that communicate through handshake synchronizations (in the UPPAAL style). This translation is at the origin of the development of tool TAPAAL [63] for verification of timed-arc Petri nets. In [111], another timed extension of Petri nets with intervals on arcs is considered. In order to guarantee compositional properties, their Petri nets are translated to timed automata enriched with an ad-hoc

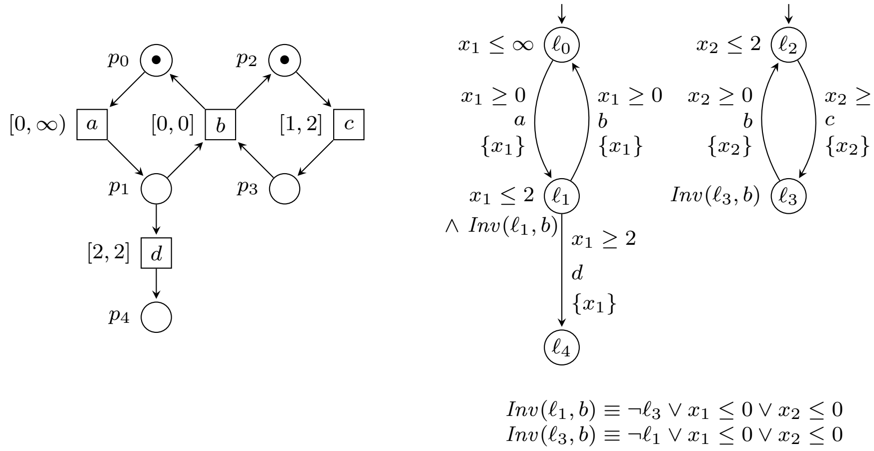


Figure 5.1: A TPN (left) and its translation as an NTA (right). The transformation is based on the decomposition of the TPN into components (as S-subnets) illustrated in Figure 4.1.

mechanism of deadlines, which hides the communications between components that would be necessary to implement it.

### The Translation Procedure

Our translation is based on the decomposition of the TPN $\mathcal{N}$ in S-subnets, which we view as components. This translation procedure involves the following steps.

1. Each S-subnet is translated into an automaton preserving its structure (places become locations and transitions become edges). Each edge is labeled with the name of the corresponding transition.

2. A single clock $x_i$ per automaton suffices for the time constraints. This clock is reset on each edge, so that the value of $x_i$ gives the time elapsed in the current location of the $i^{\text{th}}$ automaton.

3. Each transition $t$ of $\mathcal{N}$ is represented by an edge on each automaton corresponding to a component participating in $t$ (a single one if $t$ is local to a component, more if $t$ represents a synchronization). On each of these edges, a guard $x_i \geq \mathit{efd}(t)$ codes the earliest firing delay in the NTA. The synchronization will be possible only when $x_i \geq \mathit{efd}(t)$ for every component participating in $t$, i.e. $\min_i x_i \geq \mathit{efd}(t)$ which corresponds to the semantics of the earliest firing delay in TPNs.

4. It remains to code the latest firing delays. Their role in TPNs is to make transitions urgent. Therefore they will be coded on the invariants of the

automata. More precisely, the invariant attached to the location corresponding to a place $p$ of the TPN will ensure that we cannot overtake the latest firing delay of the enabled transitions which consume a token in $p$.

For a transition $t$ of $\mathcal{N}$ which is local to one component, it suffices to add an invariant $x_i \leq \mathit{lfd}(t)$ on the source location of the edge corresponding to $t$.

Otherwise, if $t$ represents a synchronization, it has to fire if it is enabled and its latest firing delay is reached. On the example of Figure 5.1, we can stay in $(\ell_1, \ell_3)$ as long as $\min(v(x_1), v(x_2)) \leq 0$ (because $\min(v(x_1), v(x_2))$ is the elapsed time since $b$ was enabled and $\mathit{lfd}(b) = 0$). Thus, we add $\mathit{Inv}(\ell_1, b) \equiv \neg\ell_3 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ and $\mathit{Inv}(\ell_3, b) \equiv \neg\ell_1 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ in the invariants of $\ell_1$ and $\ell_3$ (actually only one of them would be sufficient).

**Know thy Neighbor!**

Notice that invariants use a non standard syntax: first, they use disjunction, which is not usual in TA; second, they read the current location of the other automata (the name of a location $\ell$ of automaton $A_i$ in an invariant is interpreted as "automaton $A_i$ is currently in location $\ell$"; this syntax is supported in UPPAAL [94]); third, they read the clocks of the other automata (but do not reset them).

Note however that this extended syntax does not change the expressiveness of NTA w.r.t. sequential semantics. But we show in [14, 15] that the use of this extended syntax cannot be avoided in general, i.e. the automata have to read information about the state of the others. This creates a dependency between the automata, which is not as strong as in the case of a synchronization on a common action, since it is asymmetric: only one automaton reads. Yet, we are interested in identifying the cases where the automata do not need to read information about the state of their neighbors, which we regard as a good decompositional property. Conversely, a shared clock that cannot be avoided can be interpreted as a warning concerning the implementation of the model: it means that a distributed implementation will need more communications between components than those explicitly specified in the model. Applying our translation to a TPN makes this information explicit (via shared clocks), when it was hidden in the original TPN.

## 5.2 Avoiding Shared Clocks in Networks of Timed Automata

Quite often in the literature, the automata of an NTA are allowed to share clocks, which provides a special way of making the behavior of one automaton depend on what the others do. Actually shared clocks are relatively well accepted and can be a convenient feature for modeling systems. Moreover, since NTA are almost always given a sequential semantics, shared clocks can be handled very

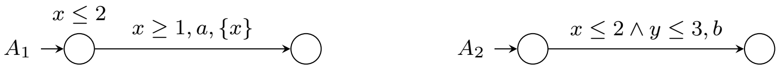


Figure 5.2: $A_2$ could avoid reading clock $x$ which belongs to $A_1$.

easily even by tools: once the NTA is transformed into a single timed automaton by the classical product construction, the notion of distribution is lost and the notion of shared clock itself becomes meaningless.

Nevertheless, implementing a model with shared clocks in a distributed architecture is not straightforward since reading clocks a priori requires communications which are not explicitly described in the model.

We are interested in detecting the cases where it is possible to avoid sharing clocks, so that the model can be implemented using no other synchronization than those explicitly described by common actions. Or, in terms of NTA, we identify NTA which syntactically use shared clocks, but whose semantics can be achieved by another NTA without shared clocks.

We restrict our study to the case of a network of two TA, $A_1 \parallel A_2$, such that $A_1$ does not read the clocks reset by $A_2$, and $A_2$ may read the clocks reset by $A_1$. We want to know whether $A_2$ really needs to read these clocks, or if another NTA $A'_1 \parallel A'_2$ could achieve the same behavior as $A_1 \parallel A_2$ without using shared clocks.

Before formalizing the problem, we study two examples informally.

### A First Example

Consider the example of Figure 5.2, made of two TA, supposed to describe two separate components. Remark that $A_2$ reads clock $x$ which is reset by $A_1$. But a simple analysis shows that this reading could be avoided: because of the condition on its clock $y$, $A_2$ can only take transition $b$ before time 3; but $x$ cannot reach value 2 before time 3, since it must be reset between time 1 and 2. Thus, forgetting the condition on $x$ in $A_2$ would not change the behavior of the system.

### Transmitting Information During Synchronizations

Consider now the example of Figure 5.3. Here also $A_2$ reads clock $x$ which is reset by $A_1$, and here also this reading could be avoided. But here, simply removing the constraints on $x$ in $A_2$ would change the behavior. We need or a more sophisticated construction. The idea is that $A_1$ can transmit the value of $x$ when synchronizing, and afterwards, any reading of $x$ in $A_2$ can be replaced by the reading of a new clock $x'$ dedicated to storing the value of $x$ which is copied on the synchronization. Therefore $A_2$ can be replaced by $A'_2$ pictured in Figure 5.3. This preserves the behavior of the NTA, including the behavior of $A_2$ in the context of $A_1$.

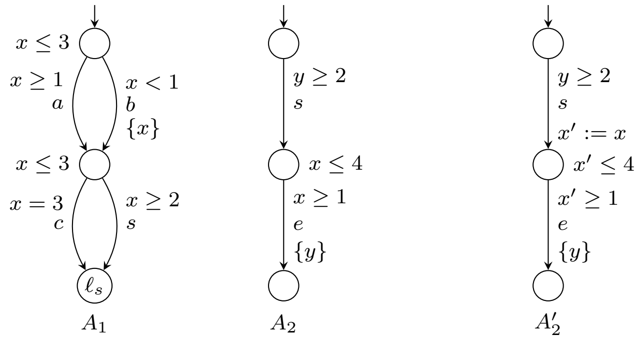


Figure 5.3: $A_2$ reads $x$ which belongs to $A_1$ and $A'_2$ does not.

We claim that we cannot avoid reading $x$ without this copy of clock. Indeed, after the synchronization, the maximal delay in the current location depends on the exact value of $x$, and even if we find a mechanism to allow $A'_2$ to move to different locations according to the value of $x$ at synchronization time, infinitely many locations would be required (for example, if $s$ occurs at time 2, $x$ may have any value in $(1, 2]$).

#### Coding Transmission of Information

In order to model the transmission of information during synchronizations, we allow $A'_1$ and $A'_2$ to use a larger synchronization alphabet than $A_1$ and $A_2$. This allows $A'_1$ to transmit discrete information like its current location, to $A'_2$. The correspondence between the new synchronization alphabet and the original one will be done by a mapping $\psi$.

But we saw that $A'_1$ also needs to transmit the exact value of its clocks. For this we allow an automaton to copy its neighbor's clocks into local clocks during synchronizations. This is denoted as updates of the form $x' := x$ in $A'_2$ (see Figure 5.3). This is a special case of updatable timed automata as defined in [37].

### 5.2.1 Formalization of the Problem

In order to formalize the problem, we use the behavioral comparisons introduced in Chapter 4.

Given a NTA $A_1 \parallel A_2$ such that $A_1$ does not read the clocks of $A_2$, we say that $A_2$ *does not need to read the clocks of* $A_1$ iff there exists an NTA $A'_1 \parallel A'_2$ without shared clocks (but with clock copies during synchronizations), such that:

1. $\psi(A'_1 \parallel A'_2)$ is distributed timed weakly bisimilar to $A_1 \parallel A_2$,
2. $\psi(TTS_{A'_2}(A'_1))$ is weakly timed bisimilar to $TTS_{A_2}(A_1)$, and

3. $\psi(TTS_{A'_1}(A'_2))$ is weakly timed bisimilar to $TTS_{A_1}(A_2)$,

with $\psi$ a mapping that relates the synchronization alphabet of $A'_1 \parallel A'_2$ with the one of $A_1 \parallel A_2$ (this is required to code the transmission of information during synchronizations as we explained earlier).

Then we give a criterion to decide whether shared clocks are necessary.

**Theorem 5.1.** *If there is no restriction in $TTS_{A_1}(A_2)$, then $A_2$ does not need to read the clocks of $A_1$. When $A_2$ is deterministic, this condition becomes necessary.*

This criterion confirms the interest of unrestricted contextual TTS mentioned at the end of Chapter 4.

In next subsection, we show how to effectively construct an NTA without shared clocks when it exists.

### 5.2.2 Constructing a Network of Timed Automata without Shared Clocks

As announced, our $A'_1$ is obtained from $A_1$ by replacing the label $a$ on every synchronization edge of $A_1$ by $(a, \ell_1)$, where $\ell_1$ is the output location of the edge. This allows $A'_1$ to transmit its location after each synchronization.

Then, the idea is to build $A'_2$ as a product $A_{1,2} \times A_{2,mod}$ ($\times$ denotes the usual synchronous product of TA [2]), where $A_{2,mod}$ plays the role of $A_2$ and $A_{1,2}$ acts as a local copy of $A'_1$, from which $A_{2,mod}$ reads clocks instead of reading those of $A'_1$. Then the system behaves as follows.

- Each time $A'_1$ synchronizes with $A'_2$, $A'_2$ updates $A_{1,2}$ to the actual state of $A'_1$. These updates are represented by the dashed arcs in Figure 5.4.
- Between two synchronizations, $A_{1,2}$ evolves, simulating a run of $A'_1$ that is compatible with what $A'_2$ knows about $A'_1$. Of course the simulated run may differ from the actual run of $A_1$. Then two situations are possible.
  - If the clocks of $A_{1,2}$ always give the same truth value to the guards and invariants of $A_{2,mod}$ than the actual value of the clocks of $A'_1$, then our construction behaves like $A_1 \parallel A_2$.
  - Otherwise the shared clocks cannot be avoided.

  Therefore, we equip $A'_2$ with an error location, ☹, and edges that lead to it if there is a contradiction between the values of the clocks of $A'_1$ and the values of the clocks of $A_{1,2}$. The guards of these edges are the only cases where $A'_2$ reads clocks of $A'_1$. Hence, if ☹ is not reachable, they can be removed so that $A'_2$ does not read the clocks of $A'_1$.

Figure 5.4 shows $A_{1,2}$ and $A_{2,mod}$ for the example of Figure 5.3.

The first property of this construction is that ☹ is reachable in $A'_1 \parallel A'_2$ iff there is a restriction in $TTS_{A_1}(A_2)$.

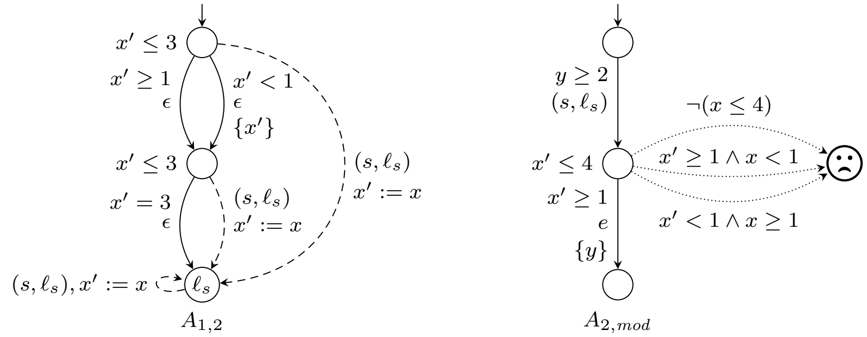


Figure 5.4: $A_{1,2}$ and $A_{2,mod}$ for the example of Figure 5.3. We represent by dotted arcs the edges leading to the error state, and by dashed arcs those used during synchronizations to reset $A_{1,2}$ to the actual state of $A_1$. The transitions of $A_{1,2}$ that simulate internal actions of $A_1$ are labeled by $\epsilon$ since, in the final system, they are internal to the component $A'_2 = A_{1,2} \times A_{2,mod}$ and do not participate in the observable behavior of the system.

In this case, we announced that it is possible to build an NTA without shared clocks which behaves like $A_1 \parallel A_2$. In the "good" cases, the construction $A'_1 \parallel A'_2$, with the unreachable error location ☹ removed, is suitable. "Good" means that $A_1$ has no urgent synchronization, i.e. for every location, a local action can always be taken immediately before the invariant expires.

In the other cases, when $A_1$ has urgent synchronizations, our construction allows one to check the absence of restriction in $TTS_{A_1}(A_2)$, but it does not give directly a suitable $A'_2$.

### Dealing with Urgent Synchronizations

The problem if we use our construction when $A_1$ has urgent synchronizations is the following. Remind that $A_{1,2}$ simulates a possible run of $A'_1$ while $A'_1$ plays its actual run. There is no reason why the two runs should coincide. Thus it may happen that the run simulated by $A_{1,2}$ reaches a state where the invariant expires and only a synchronization is possible. Then $A'_2$ is expecting a synchronization with $A'_1$, but it is possible that the actual $A'_1$ has not reached a state that enables this synchronization. Intuitively, $A'_2$ should then realize that the simulated run cannot be the actual one and try another run compatible with the absence of synchronization.

In fact, between two synchronizations, $A_{1,2}$, the local copy of $A_1$, can be constructed to simulate only one fixed run of $A_1$, instead of being able to simulate all its runs. If this run is well chosen, then the situation described above never happens, and we can use a construction similar to the one above, on which we can prove that if ☹ is not reachable, then any run of $A_1$ is compatible with the fixed run of $A_{1,2}$, and $A_2$ can avoid reading the clocks of $A_1$.

Therefore, the idea is to force $A_{1,2}$ to simulate one of the runs of $A_1$ (from the state reached after the last synchronization) that has maximal duration before it synchronizes again with $A_{2,mod}$ (or never synchronizes again if possible). There may not be any such run if some time constraints are strict inequalities, but the idea can be adapted even to this case.

### 5.2.3 Conclusion

We have shown that in a distributed framework, when locality of actions and synchronizations matter, transforming an NTA with shared clocks into an NTA without shared clocks is nontrivial an not always possible. The fact that the transformation is possible can be characterized using the notion of contextual TTS which represents the knowledge of one automaton about the other. Checking if shared clocks can be avoided is PSPACE-complete.

A first point to notice is that, contrary to what happens when one considers the sequential semantics, NTA with shared clocks are strictly more expressive if we take distribution into account. This somehow justifies why shared clocks were introduced: they are actually more than syntactic sugar.

Another interesting point that I want to recall here, is the use of transmitting information during synchronizations. It is noticeable that infinitely precise information is required in general. This advocates the interest of updatable (N)TA [37] used in an appropriate way, and more generally gives a flavor of a class of NTA closer to implementation.

Concerning transmission of information between components in a real-time distributed system, a real-time epistemic logics is introduced in [97] to reason about what a TA knows about the state of the others.

## 5.3 Perspectives

As I announced, implementability of real-time distributed systems is the main topic I would like to develop in the coming years. My first two contributions on the subject, obtained during Sandie Balaguer's PhD, open several natural questions.

#### Interest for Design of Distributed Real-Time Systems

One reason maybe why shared clocks are relatively well accepted in the community is that they can be a very convenient feature for modeling systems. Imagine for instance several agents performing together a distributed task according to a predefined schedule. In a typical implementation the schedule would be sent to the agents at the beginning and every agent would store its own copy of the schedule. But for a (simplified) model of the system, it is much easier to have one timed automaton modeling a single copy of the schedule and every agent referring to it via shared clocks.

In system design, our technique for dealing with shared clocks could help a designer to use shared clocks in an abstract specification, and build automatically an implementable distributed model without shared clocks. For the example of several agents performing together a distributed task according to a predefined schedule, this would generate the mechanism for creating the local copies of the schedule.

It would also be worth investigating applications in multi-core programming.

### Shared Clocks in General Architectures

Contextual TTS developed for the simple case of networks of two automata with only one reading the clocks of the other. Now we have the necessary background to tackle more general architectures.

The first step will be to generalize our result to the symmetrical case where $A_1$ also reads clocks from $A_2$. Then of course we can tackle general NTA with more than two automata. But transmission of information in general architectures is much more complicated than between two automata. It leads to situations where the components have incomparable knowledge, which is known as a source of undecidability.

### Transmission of Information

Another line of research is to focus on transmission of information. The goal would be to minimize the information transmitted during synchronizations, and see for example where the limits of finite information lay.

Also in practice a component cannot even get the infinitely precise time values to transmit since its clocks have bounded precision. Then the question is to find the minimum precision, if it exists, that is required to implement a given distributed specification.

Finally, it is also necessary to deal with the precision of communication delays.

### Measuring Concurrency

When shared clocks cannot be avoided, one can however discuss how to minimize them, or how to implement the model on a distributed architecture and how to handle shared clocks with as few communications as possible.

For this, one needs to find the good notions for measuring "how much concurrent" an implementation is, i.e. do components run independently, or on the contrary, do they often synchronize.

### Robustness Issues

Even when infinitely precise information is required to achieve the exact semantics of the NTA, it is worth studying how this semantics can be approximated using information of bounded precision.

This relates to the problem of robustness of models with dense real-time constraints: their semantics introduce some continuity aspects in finite state models like automata or bounded Petri nets, which have typically discrete behavior otherwise. Therefore one can expect that small changes in the constants that serve as time bounds, will result in small changes in the behavior. But many situations occur when even arbitrarily small changes in some constant induces a qualitative change in the behavior. For instance enlarging the firing delay of a transition may simply delay it, but may also cause another transition to fire instead.

This situation requires some care when the model is supposed to be implemented, because it may not be possible to ensure an exact value for a parameter that represents for example the propagation time of an electrical signal throw an electronic component.

Also when one has to check a safety property of a model that is supposed to represent a real system, it may be reasonable to check that the safety result is robust to small errors in the measures, that is the model is still safe if one introduces small enough variations in the time constants.

In the context of sequential timed systems, robustness issues were studied for instance in [39, 40].

Here again my intention is to study robustness issues related to the *distribution* of the system. One problem is that clocks from physically different components cannot be assumed perfectly synchronous and some drifts can be observed. In [1] the authors consider distributed timed automata with independently evolving clocks to study this problem.

### Models Adapted to Implementation

Dealing with implementability of models necessarily means wondering which models or which features in the models are realistic w.r.t. a physical implementation.

For instance my work on shared clocks advocates the interest of updatable NTA [37] used in an appropriate way, and more generally gives a flavor of a class of NTA closer to implementation.

It could be compared for instance with another variant, Timed Cooperating Automata, proposed in [92], where the edges can be guarded by time constraints of the form $q = \tau$ (location $q$ is enabled for $\tau$ time units), $q[\tau]$ (location $q$ is enabled for at least $\tau$ time units), $q\{\tau\}$ (location $q$ is disabled for at most $\tau$ time units) or boolean combinations of these terms.

Concerning synchronizations on common actions, it is again a high-level feature that can be debated from the point of view of implementability. More realistic paradigms were proposed like input-enabled [99].

### Limiting Synchronizations in Networks of (Timed) Automata

The idea here is to consider a high-level model where some communications between components are explicitly specified as part of the desired behavior, while others are just used in the model to constrain the executions of the automata but do not need to be actually implemented. Some of them may be redundant, for instance a synchronization whose purpose is to transfer information from a component to another is useless if the information was already sent earlier. Since communications take time and have a cost, it is desirable to avoid those that are useless.

The question is to find an implementation of the model which satisfies the specifications and minimizes the synchronizations. The problem of avoiding shared clocks could be seen as a particular case where the synchronizations to avoid are the clock readings (which a priori occur at any time, continuously in the semantics of NTA).

### Implementation of Rendezvous

Here we focus on the problem of implementation on distributed architectures. Consider for instance the formalism of networks of timed automata: it naturally extends timed automata to a distributed setting where each automaton models a sequential component, and the communications are described as synchronizations on common actions, also called rendezvous. But actually implementing rendezvous on a distributed architecture is a nontrivial problem even without real-time constraints, since in general the components that initiate a rendezvous cannot be sure that the other participants are ready to synchronize. Bagrodia proposed an algorithm to solve this problem in [10].

Adapting this algorithm to a real-time setting is a real challenge since communications required to establish the rendezvous take some time and could delay the desired synchronization, which may completely change the behavior of the system.

In a discrete time setting, the problem of distributed implementation of high-level component-based models was addressed for instance in [34] and [108]. To my knowledge the implementation of distributed models with dense time is still an open problem and I expect that the techniques to deal with dense time will be very different from those used in discrete time, as it is usually the case with this kind of formal models.

The challenge here is to design algorithms to implement formal models of (dense) real-time distributed systems on distributed architectures. By implementation I do not necessarily mean final implementation to hardware, but rather transformation to lower-level real-time models that do not use the high-level rendezvous mechanism.

# Conclusion

## Summary of Contributions

### The Contributions Presented in this Thesis

Chapter 3 is based on my contributions [11, 12] and [51] on extensions of the reveals relation and facets reduction and on my recent contribution [56] about the time progress assumption in extended free choice time Petri nets.

Chapters 4 and 5 deal with implementability of high-level real-time distributed models by defining transformations to lower-level models closer to implementation. These transformations, as well as the behavioral comparisons needed to validate them, are taken from my contributions [14, 15] and [13].

### Older Contributions on Unfoldings

My PhD thesis, prepared under supervision of Claude Jard and defended in Rennes in 2006, deals with unfoldings of high-level Petri nets and their applications to diagnosis in distributed systems. I also obtained other contributions on unfoldings after my PhD. Here is an overview.

**Symbolic Unfoldings of Safe Time Petri Nets.** This is the largest and most involved contribution of my PhD. The difficulty in unfoldings of real-time systems is due to the ordering of events induced by time, which breaks the nice properties of concurrency in asynchronous systems.

I showed that these difficulties come essentially from the combination of the structural pattern called *confusion*, which makes choices non local, with *strong* time constraints, which introduce *urgency* in the semantics. Models without confusion or with weak time constraints can be treated easily.

Then I gave the first definition of symbolic unfoldings of TPNs. My approach is based on the definition of a concurrent operational semantics, to reduce the implicit synchronization due to time progress. I showed the existence of finite complete prefixes for symbolic unfoldings of safe TPNs in [55].

**Unfoldings of NTA.** We proposed a symbolic unfolding technique for NTA in [47].

**Symbolic Unfoldings of Colored Petri Nets.** In [52], we define symbolic unfoldings for colored Petri nets. Symbolic unfoldings allow us to group family of executions which share the same structure but differ only by the colors of the tokens. In [59], we propose a category theoretic formalization of these symbolic unfoldings in order to get factorization properties.

**Unfoldings of Dynamic Systems.** I dealt also with very expressive extensions of Petri nets, adapted to modeling systems whose structure evolves during time. In [53], these dynamic aspects are modeled by adding and removing transitions during the execution. The difficulty when unfolding such model is to know what transitions are in the net after a given execution. The originality of our approach is to represent transitions as particular tokens so that they appear in the marking and can be added and removed dynamically.

In [17, 18], we propose to model dynamic systems as graph grammars, and give a category theoretic definition of unfoldings for this model.

**Application of Unfoldings to Diagnosis in Distributed Systems.** Because of the number of components that take part in complex networks, failures cannot be avoided and the system has to be designed so that they have as little impact as possible on the activity of the whole system. For this purpose many components are now designed so that they emit alarms when some particular conditions are met.

Nevertheless, inferring the causes of the failures is a challenging problem. One reason for this is the large number of alarms that are emitted. Consequently the supervisor has to select the ones that report an actual defect. Moreover some errors result from complex scenarios that involve several components. Then, the causes of a failure can only be inferred from a set of alarms, by reconstructing a part of the history of the system.

In our approach, the explanations are given as processes of a Petri net model of the supervised system. The interest of using unfoldings for supervision is not only to avoid the state space explosion problem, but also to highlight the causal dependencies between the events that are involved in the explanations. An interesting point is that this original application of unfoldings does not require the existence of finite complete prefixes, since the size of the explanations is bounded by the observation. This allows one to use very expressive high-level extensions of Petri nets: colored Petri nets in [52], time Petri nets in [54], dynamic nets in [53].

**Well-Foundedness of Adequate Orders.** Petri net unfolding prefixes are an important technique for formal verification and synthesis of concurrent systems. In [57] we show that the requirement that the *adequate order* used for truncating a Petri net unfolding must be well-founded is superfluous in many important cases, i.e. it logically follows from other requirements. This result concerns the very core of the unfolding theory.

## Alternating Simulation Between Timed Games

In [58] we focus on property-preserving preorders between timed game automata and their application to control of partially observable systems. Following the example of timed simulation between timed automata, we define timed alternating simulation as a preorder between timed game automata, which preserves controllability. We define a method to reduce the timed alternating simulation problem to a safety game. We show how timed alternating simulation can be used to control efficiently a partially observable system.

A second article [43] presents a direct method to solve the problem without constructing explicitly the timed game to be solved. This method was implemented in the tool Uppaal Tiga.

## Time Parameter Synthesis for Design of Distributed Automation Architectures

We consider the design process of a distributed automation architecture, where some time constants are not completely fixed and can still be adjusted by the designer. Therefore the system is modeled as a parametric NTA where some time constants are replaced by symbolic parameters. We assume that a proper reference valuation of the parameters is known. In [3, 4], we describe a procedure for deriving constraints on the parametric timings in order to ensure that, for each value of parameters satisfying these constraints, the behavior of the instantiated NTA is time-abstract equivalent to its proper behavior with the reference valuation.

## Synthesis of Distributed Asynchronous Systems

We study the synthesis problem in an asynchronous distributed setting: a finite set of processes interact locally with an uncontrollable environment and communicate with each other by sending signals, i.e. actions that are immediately received by the target process. The synthesis problem is to come up with a local strategy for each process such that the resulting behaviors of the system meet a given specification. In [60], we consider *external* specifications over *partial orders*. External means that specifications only relate input and output actions from and to the environment and not signals exchanged by processes. We also require some closure properties of the specification, which we regard as natural and reasonable. We present this new setting for studying the distributed synthesis problem, and give decidability results for the subclass of networks where the communication architecture is a strongly connected graph.

# Summary of Perspectives

### Implementability of Real-Time Distributed Systems

This is my main perspective for the coming years. I presented it in detail at the end of Chapter 5.

Papers [14, 15] and [13], described in Chapters 4 and 5 are my first contributions on this subject, and they open a large spectrum of new questions. Beyond the natural questions about generalization of our techniques to more general models, the perspectives involve problems related to transmission of information, approximation of models and robustness issues, always keeping in mind that distribution of actions must be preserved and that communications between components must be limited.

For many aspects, it is still a fresh research topic with very little literature available.

### Applications of the Reveals Relation and of the Net Contraction Techniques

I presented in Section 3.6.1 several applications of the reveals relation and of the net contraction techniques. They range from synthesis of nets to verification, diagnosis, prediction and estimation of energy consumption in asynchronous circuits.

### Games and Controller Synthesis for Real-Time Distributed Systems

This is, I think, a very challenging topic which I mention as perspective for the long term. It combines the difficulties of distributed concurrent systems with partial observation, incomplete knowledge, real-time constraints, games and strategies, collaborative or not...

Many situations where multiple agents interact together, can be modeled elegantly using the metaphor of a game. For example, interactions between an open system and its environment are often represented as a two players game. Game theory allows one to solve problems coming from various domains such as modeling, simulation, controller synthesis, verification, testing... Representing the problem as a game helps in understanding its algorithmic nature and in expressing its solutions under the form of strategies.

But the notion of game must be adapted when one deals with real-time systems [100, 5, 88]. Some algorithms were already proposed [48] and implemented in the tool Uppaal Tiga [20].

On the other hand, synthesis of distributed controllers is also a problem that has been studied only recently [83, 104, 60] and where the interest of partial order semantics appeared clearly. A conclusion of our contribution [60] is that the use of realistic and natural models and specifications is crucial there. This relates it to the problems of implementability that I addressed earlier.

Finally, a very sensitive aspect in synthesis of distributed controllers is the role of fairness conditions. Indeed, in an asynchronous distributed setting, two agents may want to play at the same time. And it is meaningless to try to synthesize a controller if its environment is always going to prevent it from playing. The aim of fairness conditions is to ensure that the actions of each player are taken into account sufficiently often.

Several classical definitions exist for fairness constraints and their differences are so subtle that it is difficult to know which one is adapted in a given context. Instead, in a real-time context, the comparison of the execution speed of the different agents can be quantified and this gives very natural notions of fairness constraints.

# Bibliography

[1] S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *CONCUR'08*, volume 5201 of *LNCS*, pages 82–97, Toronto, Canada, 2008. Springer.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] É. André, Th. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. In *Proceedings of the 2nd Workshop on Reachability Problems in Computational Models (RP'08)*, volume 223 of *Electronic Notes in Theoretical Computer Science*, pages 29–46. Elsevier Science Publishers, 2008.

[4] É. André, Th. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, Oct. 2009.

[5] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symp. on System Structure & Control*, pages 469–474. Elsevier Science, 1998.

[6] T. Aura and J. Lilius. Time processes for time Petri nets. In *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 136–155. Springer, 1997.

[7] T. Aura and J. Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1-2):409–447, 2000.

[8] E. Badouel, B. Caillaud, and P. Darondeau. Distributing finite automata through Petri net synthesis. *Journal on Formal Aspects of Computing*, 13:447–470, 2002.

[9] E. Badouel and P. Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 529–586. Springer Berlin / Heidelberg, 1998.

[10] R. Bagrodia. A distributed algorithm to implement n-party rendevouz. In *FSTTCS*, volume 287 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 1987.

[11] S. Balaguer, T. Chatain, and S. Haar. Building tight occurrence nets from reveals relations. In B. Caillaud and J. Carmona, editors, *Proceedings of the 11th International Conference on Application of Concurrency to System Design (ACSD'11)*, pages 44–53, Newcastle upon Tyne, UK, June 2011. IEEE Computer Society Press.

[12] S. Balaguer, T. Chatain, and S. Haar. Building occurrence nets from reveals relations. *Fundamenta Informaticae*, 123(3):245–272, 2013.

[13] S. Balaguer and Th. Chatain. Avoiding shared clocks in networks of timed automata. In M. Koutny and I. Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 100–114, Newcastle, UK, Sept. 2012. Springer.

[14] S. Balaguer, Th. Chatain, and S. Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. In *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME'10)*, pages 77–84, Paris, France, 2010. IEEE Computer Society Press.

[15] S. Balaguer, Th. Chatain, and S. Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. *Formal Methods in System Design*, 40(3):330–355, 2012.

[16] P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, and S. Schwoon. Efficient unfolding of contextual Petri nets. *Theor. Comput. Sci.*, 449:2–22, 2012.

[17] P. Baldan, Th. Chatain, S. Haar, and B. König. Unfolding-based diagnosis of systems with an evolving topology. In *CONCUR'2008*, volume 5201 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2008.

[18] P. Baldan, Th. Chatain, S. Haar, and B. König. Unfolding-based diagnosis of systems with an evolving topology. *Information and Computation*, 208(10):1169–1192, Oct. 2010.

[19] P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1–49, 2001.

[20] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.

[21] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998.

[22] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. When are timed automata weakly timed bisimilar to time Petri nets? *Theoretical Computer Science*, 403(2-3):202–220, 2008.

[23] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. The expressive power of time Petri nets. *Theor. Comput. Sci.*, 474:1–20, 2013.

[24] B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of the expressiveness of timed automata and time Petri nets. In *FORMATS'05*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2005.

[25] R. Bergenthum. Faster verification of partially ordered runs in Petri nets using compact tokenflows. In *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 330–348. Springer, 2013.

[26] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri nets from finite partial languages. *Fundam. Inform.*, 88(4):437–468, 2008.

[27] L. Bernardinello. Synthesis of net systems. In *ICATPN*, volume 691 of *LNCS*, pages 89–105. Springer, 1993.

[28] G. Berthelot. Checking properties of nets using transformation. In *Applications and Theory in Petri Nets*, volume 222 of *LNCS*, pages 19–40. Springer, 1985.

[29] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2006.

[30] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[31] E. Best. Structure theory of Petri nets: the free choice hiatus. In *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part I*, pages 168–205, London, UK, 1987. Springer-Verlag.

[32] E. Best, R. R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in Petri nets. *Acta Inf.*, 28(3):231–264, 1991.

[33] E. Best and B. Randell. A formal model of atomicity in asynchronous systems. *Acta Informatica*, 16(1):93–124, 1981.

[34] B. Bonakdarpour, M. Bozga, and J. Quilbeuf. Automated distributed implementation of component-based models with priorities. In *EMSOFT*, pages 59–68. ACM, 2011.

[35] H. Boucheneb, D. Lime, and O. H. Roux. On multi-enabledness in time Petri nets. In *ICATPN*, volume 7927 of *Lecture Notes in Computer Science*. Springer, 2013.

[36] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In W. A. Hunt, Jr and F. Somenzi, editors, *CAV 2003*, volume 2725 of *LNCS*, pages 180–192. Springer, Heidelberg, 2003.

[37] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[38] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed unfoldings for networks of timed automata. In *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306, Beijing, ROC, 2006. Springer.

[39] P. Bouyer, K. G. Larsen, N. Markey, O. Sankur, and C. Thrane. Timed automata can always be made implementable. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2011.

[40] P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[41] M. Boyer and O. H. Roux. On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3):225–249, 2008.

[42] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *International Conference on Computer Aided Verification (CAV)*, volume 1427 of *LNCS*, pages 546–550, 1998.

[43] P. Bulychev, Th. Chatain, A. David, and K. G. Larsen. Checking simulation relation between timed game automata. In *Proceedings of the 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009.

[44] N. Busi and G. M. Pinna. Non sequential semantics for contextual P/T nets. In *Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 113–132. Springer, 1996.

[45] J. Byg, K. Joergensen, and J. Srba. An efficient translation of timed-arc Petri nets to networks of timed automata. In *International Conference on Formal Engineering Methods*, volume 5885 of *LNCS*, pages 698–716. Springer-Verlag, 2009.

[46] J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. In *ICATPN*, volume 5062 of *LNCS*, pages 92–111. Springer-Verlag, 2008.

[47] F. Cassez, T. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 307–321, Beijing, ROC, 2006. Springer.

[48] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[49] F. Cassez and O. Roux. Structural translation from time Petri nets to timed automata. *Journal of Systems and Software*, 2006.

[50] K. Cerans, J. C. Godskesen, and K. G. Larsen. Timed modal specification - theory and tools. In *CAV*, volume 697 of *LNCS*, pages 253–267. Springer, 1993.

[51] T. Chatain and S. Haar. A canonical contraction for safe Petri nets. In *Proceedings of the International Workshop on Petri Nets and Software Engineering*, volume 989, pages 19–33. CEUR-WS, 2013.

[52] T. Chatain and C. Jard. Symbolic diagnosis of partially observable concurrent systems. In *FORTE*, volume 3235 of *LNCS*, pages 326–342, 2004.

[53] T. Chatain and C. Jard. Models for the supervision of web services orchestration with dynamic changes. In *Advanced Industrial Conference on Telecommunication / Service Assurance with Partial and Intermittent Resources (AICT/SAPIR 2005)*, 2005.

[54] T. Chatain and C. Jard. Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In *FORMATS*, volume 3829 of *LNCS*, pages 196–210, 2005.

[55] T. Chatain and C. Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In *ICATPN*, volume 4024 of *LNCS*, pages 125–145, june 2006.

[56] T. Chatain and C. Jard. Back in time Petri nets. In V. Braberman and L. Fribourg, editors, *Proceedings of the 11th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'13)*, volume 8053 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2013.

[57] T. Chatain and V. Khomenko. On the well-foundedness of adequate orders used for construction of complete unfolding prefixes. *Information Processing Letters*, 104(4):129–136, 2007.

[58] Th. Chatain, A. David, and K. G. Larsen. Playing games with timed games. In *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, 2009.

[59] Th. Chatain and É. Fabre. Factorization properties of symbolic unfoldings of colored Petri nets. In *Proceedings of the 31st International Conference on Applications and Theory of Petri Nets (ICATPN'10)*, volume 6128 of *Lecture Notes in Computer Science*, pages 165–184, Braga, Portugal, 2010. Springer.

[60] Th. Chatain, P. Gastin, and N. Sznajder. Natural specifications yield decidability for distributed synthesis of asynchronous systems. In *Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, volume 5404 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2009.

[61] J. M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, pages 79–112, London, UK, 1991. Springer-Verlag.

[62] P. Darondeau. Deriving unbounded Petri nets from formal languages. In *CONCUR*, volume 1466 of *LNCS*, pages 533–548. Springer, 1998.

[63] A. David, L. Jacobsen, M. Jacobsen, K. Y. Jørgensen, M. H. Møller, and J. Srba. Tapaal 2.0: Integrated development environment for timed-arc Petri nets. In *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 492–497. Springer, 2012.

[64] A. David, K. G. Larsen, S. Li, and B. Nielsen. Timed testing under partial observability. In *ICST*, pages 61–70. IEEE Computer Society, 2009.

[65] J. Desel and J. Esparza. *Free Choice Petri nets.* Cambridge University Press, 1995.

[66] J. Desel and A. Merceron. Vicinity respecting homomorphisms for abstracting system requirements. In *Proc. Int. Workshop on Abstractions for Petri Nets and Other Models of Concurrency (APNOC)*, 2009.

[67] J. Desel and W. Reisig. The synthesis problem of Petri nets. *Acta Inf.*, 33:297–315, 1996.

[68] V. Diekert. *The Book of Traces.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.

[69] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[70] C. Dima. Positive and negative results on the decidability of the model-checking problem for an epistemic extension of timed ctl. In *TIME*, pages 29–36. IEEE Computer Society, 2009.

[71] C. Dima and R. Lanotte. Distributed time-asynchronous automata. In *ICTAC*, pages 185–200. Springer-Verlag, 2007.

[72] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. parts I and II. *Acta Inf.*, 27(4):315–368, 1989.

[73] H. Ehrig, K. Hoffmann, J. Padberg, P. Baldan, and R. Heckel. High-level net processes. In *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*, pages 191–219. Springer, 2002.

[74] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[75] J. Esparza and K. Heljanko. A new unfolding approach to LTL model checking. In *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 475–486. Springer, 2000.

[76] J. Esparza and K. Heljanko. Implementing LTL model checking with net unfoldings. In *SPIN*, volume 2057 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2001.

[77] J. Esparza and K. Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. 2008.

[78] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 1999.

[79] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[80] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[81] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time Petri nets. In *International Conference on Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.

[82] G. Gardey, O. H. Roux, and O. F. Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming*, 6(3):301–320, 2006.

[83] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games and distributed control for asynchronous systems. In *LATIN*, volume 2976 of *Lecture Notes in Computer Science*, pages 455–465. Springer, 2004.

[84] S. Haar. Types of asynchronous diagnosability and the *reveals*-relation in occurrence nets. *IEEE Transactions on Automatic Control*, 55(10):2310–2320, 2010.

[85] S. Haar, C. Kern, and S. Schwoon. Computing the reveals relation in occurrence nets. In *Proceedings of GandALF'11*, volume 54 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–44, 2011.

[86] S. Haar, C. Rodríguez, and S. Schwoon. Reveal your faults: It's only fair! In *Proceedings of the 13th International Conference on Application of Concurrency to System Design (ACSD'13)*, pages 127–136. IEEE Computer Society Press, 2013.

[87] M. Hack. Analysis of production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, USA, 1972.

[88] T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1-2):369–392, 1999.

[89] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, School of Computing Science, University of Newcastle upon Tyne, 2003.

[90] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state encoding conflicts in STG unfoldings using SAT. *Fundam. Inf.*, 62(2):221–241, 2004.

[91] R. Kumar and S. Takai. Decentralized prognosis of failures in discrete event systems. *IEEE Transactions on Automatic Control*, 55(1):48–59, 2010.

[92] R. Lanotte, A. Maggiolo-Schettini, and A. Peron. Timed cooperating automata. *Fundamenta Informaticae*, 43:153–173, August 2000.

[93] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *FCT*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995.

[94] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.

[95] K. Lautenbach. Liveness in Petri nets. Technical report, Gesellschaft fÃ$\frac{1}{4}$r Mathematik und Datenverarbeitung, Bonn, Germany, July 1975.

[96] L. Lloyd, A. V. Yakovlev, E. Pastor, and A. Koelmans. Estimations of power consumption in asynchronous logic as derived from graph based circuit representations. In *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'98)*, pages 367–376. Dept. of Information Technology, Technical University of Denmark, 1998.

[97] A. Lomuscio, W. Penczek, and B. Wozna. Bounded model checking for knowledge and real time. *Artif. Intell.*, 171(16-17):1011–1038, 2007.

[98] D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.

[99] N. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[100] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242, 1995.

[101] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[102] P. M. Merlin and D. J. Farber. Recoverability of communication protocols – implications of a theorical study. *IEEE Transactions on Communications*, 24, 1976.

[103] M. Minea. Partial order reduction for model checking of timed automata. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.

[104] S. Mohalik and I. Walukiewicz. Distributed games. In *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2003.

[105] P. Niebert and H. Qu. Adding invariants to event zone automata. In *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2006.

[106] M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

[107] W. Penczek and A. Pólrola. Abstractions and partial order reductions for checking branching properties of time Petri nets. In *ICATPN*, volume 2075 of *LNCS*, pages 323–342, 2001.

[108] D. Potop-Butucaru, R. de Simone, Y. Sorel, and J.-P. Talpin. Clock-driven distributed real-time implementation of endochronous synchronous programs. In *EMSOFT*, pages 147–156. ACM, 2009.

[109] J. Reif. The complexity of two-player games of incomplete information. *Jour. Computer and Systems Sciences*, 29:274–301, 1984.

[110] T. G. Rokicki. *Representing and Modeling Digital Circuits.* PhD thesis, Standford University, 1993.

[111] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 347–359, London, UK, 1996. Springer-Verlag.

[112] J. Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In *FORMATS*, volume 5215 of *LNCS*, pages 15–32. Springer, 2008.

[113] L.-M. Traonouez, B. Grabiec, C. Jard, D. Lime, and O. H. Roux. Symbolic unfolding of parametric stopwatch Petri nets. In *ATVA*, volume 6252 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2010.

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

[115] I. Virbitskaite and E. Pokozy. A partial order method for the verification of time Petri nets. In *FCT*, volume 1684 of *Lecture Notes in Computer Science*, pages 547–558. Springer, 1999.

[116] W. Vogler. Partial order semantics and read arcs. *Theoretical Computer Science*, 286(1):33–63, 2002.

[117] J. Winkowski. Processes of contextual nets and their characteristics. *Fundamenta Informaticae*, 36(1), 1998.

[118] B. Wozna and A. Lomuscio. A logic for knowledge, correctness, and real time. In *CLIMA*, volume 3487 of *LNCS*, pages 1–15. Springer, 2004.