Study of concurrency in real-time distributed systems

Sandie Balaguer

ENSC-201X-N°YYY

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Présentée par
Sandie BALAGUER

pour obtenir le grade de
Docteur de l'École Normale Supérieure de Cachan

Domaine : INFORMATIQUE

Sujet de la thèse :

Concurrency in Real-Time Distributed Systems

Thèse présentée et soutenue à Cachan le 13 décembre 2012 devant le jury composé de :

Jiří	SRBA	Rapporteur
Walter	VOGLER	Rapporteur
Béatrice	BÉRARD	Examinatrice
Bernard	BERTHOMIEU	Examinateur
Thomas	CHATAIN	Co-encadrant
Stefan	HAAR	Directeur de thèse
Olivier H.	ROUX	Examinateur

Laboratoire Spécification et Vérification
ENS CACHAN/CNRS/UMR 8643
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

Concurrency in Real-Time Distributed Systems

Abstract This thesis is concerned with the modeling and the analysis of distributed real-time systems. In distributed systems, components can evolve independently and communicate with each other. Concurrent actions are performed by different components without influencing each other. The time constraints in distributed real-time systems create complex dependencies between the components and the events that occur. In the previous studies of distributed real-time systems, the distributed aspects are often left aside. This thesis explores these aspects. Our work is based on two formalisms: time Petri nets and networks of timed automata, and is divided into two parts.

In the first part, we highlight the differences between centralized and distributed timed systems. We compare the main formalisms and their extensions, with a novel approach that focuses on the preservation of concurrency. In particular, we show how to translate a time Petri net into a network of timed automata with the same distributed behavior. We then study the problem of shared clocks in networks of timed automata: when one considers the implementation of a model on a multi-core architecture, shared clocks require communications that are not explicitly described. We show how to avoid shared clocks while preserving the distributed behavior, when this is possible.

In the second part, we focus on formalizing the dependencies between events in partial order representations of the executions of Petri nets and time Petri nets. Occurrence nets is one of these partial order representations, and their structure directly provides the causality, conflict and concurrency relations between events. However, we show that, even in the untimed case, some logical dependencies between event occurrences are not directly described by these structural relations. After having formalized these logical dependencies, we solve the following synthesis problem: from a formula that describes a set of runs, build an associated occurrence net. Then we study the logical relations in a simplified timed setting and show that time creates complex dependencies between event occurrences. These dependencies can be used to define a canonical unfolding, for this particular timed setting.

Keywords: distributed real-time systems, concurrency, partial-orders, networks of timed automata, time Petri nets, timed transition systems, shared clocks, implementation on distributed architecture, behavioral equivalence for distributed timed systems, unfoldings, logical characterization of runs, synthesis

La concurrence dans les systèmes temps-réel distribués

Résumé Cette thèse s'intéresse à la modélisation et à l'analyse des systèmes temps-réel distribués. Un système distribué est constitué de plusieurs composants qui évoluent de manière partiellement indépendante. Lorsque des actions exécutables par différents composants sont indépendantes, elles sont dites concurrentes. Dans ce cas, elles peuvent être exécutées dans n'importe quel ordre, sans s'influencer. Dans les systèmes temps-réel distribués, les contraintes de temps créent des dépendances complexes entre les composants et les événements qui ont lieu sur ces composants. Malgré l'omniprésence et l'aspect critique de ces systèmes, beaucoup de leurs propriétés restent encore à étudier. En particulier, la nature distribuée de ces systèmes est souvent laissée de côté. Notre travail s'appuie sur deux formalismes de modélisation : les réseaux de Petri temporels et les réseaux d'automates temporisés, et est divisé en deux parties.

Dans la première partie, nous mettons en évidence les différences entre les systèmes temporisés centralisés et les systèmes temporisés distribués. Nous comparons les formalismes principaux et leurs extensions, avec une approche originale qui considère la concurrence. En particulier, nous montrons comment transformer un réseau de Petri temporel en un réseau d'automates temporisés qui a le même comportement distribué. Nous nous intéressons ensuite aux horloges partagées dans les réseaux d'automates temporisés. Les horloges partagées sont problématiques lorsque l'on envisage d'implanter ces modèles sur des architectures distribuées. Nous montrons comment se passer des horloges partagées, tout en préservant le comportement distribué, lorsque cela est possible.

Dans la seconde partie, nous nous attachons à formaliser les dépendances entre les événements dans les représentations en ordre partiel des exécutions des réseaux de Petri (temporels ou non). Les réseaux d'occurrence sont une de ces représentations, et leur structure donne directement les relations de causalité, conflit et concurrence entre les événements. Cependant, nous montrons que, même dans le cas non temporisé, certaines relations logiques entre les événements ne peuvent pas être directement décrites par ces relations structurelles. Après avoir formalisé les relations logiques en question, nous résolvons le problème de synthèse suivant : étant donnée une formule logique qui décrit un ensemble d'exécutions, construire un réseau d'occurrence associé, quand celui-ci existe. Nous étudions ensuite les relations logiques dans un cadre temporisé simplifié, et montrons que le temps crée des dépendances complexes entre les événements. Ces dépendances peuvent être utilisées pour définir des dépliages canoniques de réseaux de Petri temporels, dans ce cadre simplifié.

Mots-clés : systèmes temps-réel distribués, concurrence, ordres partiels, réseaux d'automates temporisés, réseaux de Petri temporels, systèmes de transitions temporisés, horloges partagées, implantation sur architecture distribuée, équivalence de comportement pour systèmes distribués, dépliages, caractérisation logique d'un ensemble d'exécutions, synthèse

Remerciements

Contents

Introduction

Chapter 1

Introduction

In this chapter, we present real-time distributed systems and some problems related to their formal modeling and analysis.

Context A real-time system is a physical system or a computer program that monitors a physical system, where *time* is a critical factor, in the sense that not only the outputs are important, but also the timing of these outputs. Hence, a real-time program must guarantee response within strict time constraints, otherwise it fails and there can be severe consequences.

Real-time systems appear in a multitude of fields, such as aeronautics and automobile industry (ABS, aircraft control, air traffic control), networking and telecommunication networks (mobile devices, ATMs), energy (monitors in nuclear power station), medicine (ECG/arrhythmia monitor), scientific computing (weather prediction, finance), and rendering in computer graphics.

In addition, we study *distributed* systems, that is systems that are composed of several components, that may be partly independent, or *concurrent*, but that can also interact with one another. A good example of distributed system is a computer network, where desktop computers and servers may behave independently although they can communicate and influence each other. Distributed systems also play an important role in future applications such as smart grids, electrical grids that use distributed energy resources to optimize the production and distribution of electricity.

Formal Methods The importance of time and deadline, and the complex interactions that stem from the communication between components make real-time distributed systems a challenge to comprehend and implement correctly. There are a lot of examples of misbehaviors that evidence this challenge.

Since real-time distributed systems are widespread and often control critical processes whose failure can cause death, injury, or big financial losses, it is crucial that they behave as intended, that is, as described by their specification.

The impossibility of studying most of real-time systems manually leads to the use of formal methods for their verification.

This is the context of this thesis. Our aim is to understand better the formal models for distributed real-time systems, in particular by studying the complex interactions between the components of these systems. We also argue that some formalisms are more adapted than others, especially when we consider implementing the model in a multi-core architecture that allows the parallelization of tasks.

Organization of the Chapter In Section 1.1, we first introduce real-time distributed systems and the need for dedicated formal methods. Then, we present formal methods, in Section 1.2, and some problems that come with the modeling of real-time distributed systems. In particular, we present the state space explosion problem, and the different choices of semantics (discrete or dense time and synchronous or asynchronous components). Then, in Section 1.3 we recall the state of the art. Lastly, we describe our contribution in Section 1.4, and the organization of this thesis in Section 1.5.

1.1 Real-Time Distributed Systems

The number, complexity and criticity of real-time systems have increased substantially over the past few decades. That is why verifying that these systems behave as expected is essential and requires efficient formal methods. The verification is usually done as early as possible, during the conception phase, and before the implementation.

Nowadays, most of physical systems and softwares are distributed, for example to take advantage of multi-core technologies, and here we consider this a key feature. On the one hand, this distribution yields complex dependencies between the components that need to be better formalized, studied and understood. On the other hand, if correctly exploited, this distribution can be used to improve the analysis of the systems, for example by enabling a modular analysis.

Most of physical systems have deadlines and strict time constraints as well. These two aspects, distribution and time constraints, interact because of shared resources and inter-component communications. That is why the relationship between these two factors is complex. For example, two separate components may seem independent although one component has a time constraint that depends on what the other component has previously done.

Hence distributed real-time systems are complex and difficult to comprehend and their failure can have dramatic consequences. Below we list two of the most well-known failures, partly caused by race conditions (for example, pairs of accesses to the same variable by different threads).

The Therac-25 Radiation Therapy Machine was responsible for several patient deaths in the 1980s, when it delivered excessive quantities of X-rays. One of the various engineering issues was a problem of synchronization of the equipment control task with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly. This was missed during testing, since it took some practice before operators were able to work quickly enough to trigger this failure mode [LT93].

The North America Blackout of 2003 was a widespread power outage that affected an estimated 55 million people in Canada and U.S.A.. The blackout was triggered by a local outage that went undetected due to a race condition in the monitoring software. The bug stalled the alarm system for over an hour, so that system operators were unaware of the malfunction. After the alarm system failure, unprocessed events queued up and the primary server failed within 30 minutes. Then all applications (including the stalled alarm system) were automatically transferred to the backup server, which itself failed [For04]. Eventually, the bug was so deeply embedded in the millions of lines of code of the alarm system, it took eight weeks to several experts to find it.

These examples show that the bugs may be very subtle, and very difficult to detect, and speak for the development of tools for simulating and verifying real-time systems during the design phase, and before their implementation and usage. Of course, since physical systems are usually complex and critical, the performances (in terms of time, memory cost or accuracy) of the tools are very important, and therefore the formal methods at the core of these tools have to be as efficient as possible.

Concurrency and Atomicity Writing concurrent programs is much more difficult than writing sequential ones. Nevertheless, concurrent programs are very convenient to parallelize tasks on a multi-core architecture. But the interactions among the tasks, such as the access to a shared memory require special attention. In particular, accesses (read and write operations) to a shared variable must be *atomic*, i.e. indivisible. In programming languages such as Java or C, this is often ensured by locks that prevent the mutual access to a shared variable by several threads. The majority of errors in concurrent programs is due to a violation of atomicity, where a code region is intended to be atomic, but the atomicity is not enforced during execution [LPSZ08].

Principles of Model Checking Model checking [JGP99, BK08] is an automated technique that, given a formal model of a system, and a formal specification of a requirement, checks whether the requirement holds. This approach uses tools called model-checkers. The system is modeled using the model specification language of the model-checker, and the property to be checked is formalized

with the property specification language. Then the model checker checks the validity of the property in the system model, and provides a counterexample if the property is not satisfied.

In practice, model checking has been applied for example to several modules of the NASA's Deep Space-1 spacecraft, and to the verification of the control software for the flood control barrier of the port of Rotterdam. In both cases, some serious design flaws have been identified.

1.2 Formal Methods

Failure of real-time systems can have catastrophic consequences, therefore it is crucial to ensure their correctness. This is why the use of formal methods has drastically increased during the last decades. They are mathematical procedures that consist in modeling the system into a formalism that describes an abstraction of its behaviors, so that it becomes easier to analyze, possibly in an automated way. Most of the time, it is not possible to take into account every detail of the system; in particular, the environment cannot be modeled exactly. Hence, the first step of the modeling is to chose the right level of abstraction that will give an accurate enough description.

Even if the formalisms cannot be infinitely precise, they may describe infinitely many behaviors. Therefore they cannot be explicitly enumerated by a computer, and in order to verify that there is no unintended behavior, a technical trick that is often used is to represent these behaviors by a finite number of classes of equivalent behaviors.

Furthermore, the increasing complexity of systems is often dealt with by the decomposition into several modules. Then each module can be analyzed separately and more easily than the whole system. But it is not easy to cope with the effects of the recomposition of the modules into the original system as they may interact. Also, it is not easy either to verify global properties with a modular analysis. Lastly, the system may also interact with other uncontrollable systems, like the environment. All in all, this makes the comprehension of such systems hard for a human being, and emphasizes the need for formal methods.

1.2.1 Modeling Real-Time Distributed Systems

We focus on formalisms that take time into account, and allow modeling distributed systems. In this thesis, we restrict our focus to networks of timed automata and time Petri nets. These formalisms let us model several components that may communicate, and express constraints on the logical order of events but also on their timing. Both formalisms enable the modeling of *urgency* which is a key feature without which most real-time systems cannot be modeled correctly.

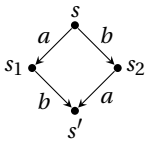
Networks of Timed Automata model each component of a distributed timed system as a timed automaton, and provide a synchronization mechanism between the components. Time is incorporated into the model by means of real valued variables called clocks that can be reset and tested. Networks of timed automata (NTA) are presented in Subsection 2.3.2.

Time Petri Nets In the Petri net model, the components are less visible since it is usually a connected graph. However, we will show how the components can be identified. This model is very convenient to represent resources as tokens that can be consumed or produced by transition firings. Time Petri nets (TPNs) is one of the extensions of Petri nets with time. In this model, transitions are equipped with firing intervals that specify the time interval within which the transition has to fire. Time Petri nets are presented in Subsection 2.3.3.

1.2.2 Concurrency

Concurrency, Parallelism and Interleaving

In distributed systems, concurrency arises when several actions may be performed simultaneously and independently by different processes running in parallel. For instance, actions a and b are concurrent if, from a given state s, they can be executed in any order, leading to the same new state s'. This results in the diamond depicted below.



In networks of timed automata, each automaton corresponds to a process that may behave independently from the others between synchronizations. In the product automaton that models the whole system, this results in a diamond denoting the interleavings, i.e. the possible orderings of concurrent actions.

Likewise, from a state of a Petri net, two transitions may be able to fire concurrently. This means that they consume different tokens. That is why tokens can represent the state of different processes, and explicitly model parallelism.

The State Space Explosion Problem Naive automated methods for the verification of concurrent systems are based on the exploration of the state space of the system, given as a transition system. The main drawback of this approach is the well-know *state space explosion* problem. For instance, a distributed system consisting of n components that can each be in m different states, may have up to m^n reachable states. Therefore, small distributed systems may generate very large transition systems and naive methods may have huge time and space requirements.

That is why, in order to improve the analysis of distributed systems, some approaches use reduction techniques, that take advantage of the fact that only some states and some interleavings are relevant. An overview of techniques for fighting state space explosion is presented in [Pel08].

Conflicts, Fairness, Starvation The problem of resolving conflicts between processes in distributed systems is of practical importance. A conflict between a set of processes must be resolved in favor of some process and against the others. To guarantee fairness, the process selected for favorable treatment should not always be the same, otherwise some unfavored process would be stalled forever, which is called starvation. Fairness is often achieved by additional communications or shared resources.

Concurrency and Communication

In a distributed system, components usually communicate either explicitly by message passing, or implicitly by shared resources. These communications need to be formalized, for they impact the behavior of the whole system.

Explicit Communication via Message Passing There are several communication mechanisms which use message passing. Here we consider a synchronous communication called handshaking: processes interact by performing the same action synchronously. During this interaction, they may also exchange information. However, in the models we consider later, and in particular in networks of timed automata, these communications are only synchronizations on a same action.

Implicit Communication via Shared Variables One popular example of communication with shared variables is the Fischer's mutual exclusion protocol. It is a concurrent programming algorithm for mutual exclusion that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication. It is presented, for example, in [AILS07], and recalled on the right-hand side. Each process has a unique identifier i (different from 0), and can read and write the shared variable id. Since it is a real-time algorithm, it is important to optimize the value $delay$, but this depends on the characteristics of the processes.

```
    repeat
    |   <noncritical section>;
L   |   await id = 0;
    |   id := i;
    |   pause(delay);
    |   if id ≠ i then goto L;
    |   <critical section>;
    |   id := 0;
    forever ;
```

Concurrency and Time

Considering both real-time and distribution necessarily entails solving several difficulties. First, a time semantics has to be chosen, time can be discrete and events occur every fixed amount of time, or dense and events can occur at any real time point. Second, the components can be synchronous (all clocks have the same velocity) or asynchronous (there may be drifts between local clocks).

Discrete versus Dense Time In some cases, it is more relevant to consider that time is discrete, For example, in synchronous hardware circuits, the different components (like adders, multiplexers and inverters) are synchronized by a global digital clock: on each clock tick, they all perform an action. In addition, discrete time semantics is closer to the implementation semantics because an implementation necessarily includes some hardware digital clock.

But a discrete time semantics is inadequate for many distributed systems where the environment stimuli come at any real time point. A dense-time semantics is more natural for physical systems and allows a more intuitive modeling of real-time systems. With this semantics, the concrete sampling rate of the implementation need not be considered in the modeling phase. However, dense-time semantics excludes analysis methods based on explicit enumeration of states and paths, and requires methods based on abstractions such as the region graph construction [AD94].

Both semantics have been studied and have dedicated models, in particular sampled timed automata [AKY10, BLM+11] and discrete time Petri nets [KPSP10, TMBK+11]. In this thesis, we consider only the dense-time semantics, which is most common for networks of timed automata and time Petri nets.

Synchronous versus Asynchronous Components One of the main difficulties is how to render and interpret time when several components are considered. On the one hand, the processors on which the components are executed may have different velocities. This can be modeled by local clocks having different speed in the different components, as in the model of distributed timed automata with independently evolving clocks [ABG+08]. On the other hand, this divergence may be negligible with respect to the other measures, and the components may readjust their clocks when synchronizing. This is commonly assumed in the classical model of networks of timed automata model whose components are synchronous [AD90].

In this thesis, we consider only distributed systems with synchronous components. We observe that, with this assumption, time provides an implicit synchronization between the components. This is discussed in Chapter 5.

1.3 State of the Art

In this section, we present a brief state of the art. Further details are given in the introductory chapters of each part (Chapter 3 for Part I, and Chapter 6 for Part II). We present some formalisms for distributed timed systems, then some notions and techniques for the description and the analysis of their behavior. We also consider the problems related to the implementation of these systems.

1.3.1 Overview of Formalisms

Variants of Networks of Timed Automata

Timed Automata (TA) have been introduced in the 90's by Alur and Dill, and have been extensively studied since then.

There exist several variants and extensions of TA and NTA. TA with silent transitions have been studied in [BDGP98], where it is shown that silent transitions do not change the decidability of the emptiness problem, but strictly increase the expressive power. Most tools also extend TA with integer variables, like UPPAAL [BDL04] and KRONOS [BDM+98]. Updatable TA [BDFP04] is an extension of TA with updates of the form $x :\sim c$ or $x :\sim y + c$, where x and y are clocks, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \neq, \geq, >\}$. In Chapter 3, we will consider only a subclass of updatable TA, where we allow copies of clocks, i.e. resets of the form $x := y$ where x and y are clocks. This class is not more expressive than classical TA (any updatable TA of the class is bisimilar to a classical TA), and the emptiness problem remains PSPACE-complete [BDFP04].

Event-clock automata [AFH99], is a determinizable class of timed automata obtained by restricting the use of clocks. In this restriction, clocks are either *event-recording*: the value of the clock associated with symbol a always equals the time of the last occurrence of a, or *event-predicting*: the value of the clock associated with a always equals the time of the next occurrence of a relative to the current time. This is the kind of TA obtained when a time Petri net is translated into a sequential TA, called marking timed automaton [GRR06].

Other extensions focus more on the distributed aspect of NTA, like distributed TA [ABG+08, DL07] that are NTA where each clock belongs to one TA and can be reset only by this TA. In [LMSP00], the authors introduce timed cooperating automata, an extension of NTA, where the automata view the current state of other automata and the time elapsed since their activation. Lastly, in [LMST03], the authors propose a variant of TA with parallelism, called Concurrent Timed Automata (CTAs), where automata running in parallel can compute only by sensing, at each instant, the same action from the environment. CTAs can be mapped to equivalent TAs by using the Cartesian product. This formalism also considers updates of the form $x := c$ and private or public clocks.

Timed Extensions of Petri Nets

There exist several variants of Petri nets extended with time, we present the most well-known ones. Time Petri nets (TPNs) [Mer74] is the extension we consider and we present in Subsection 2.3.3. In timed arc Petri nets, each token has a clock representing its age, but a non-urgent semantics is assumed: the firing of a transition may be delayed and a transition may be disabled because its input tokens become too old [AN01, dFERA00]. Timed Petri nets [Ram74] associate a firing time to each transition and a transition fires as soon as possible, contrary to time Petri nets, where a transition fires in a time interval. Other variants are presented and compared in [BR08].

Then, several semantics have been proposed for TPNs. In this thesis, we use the original and most common semantics for newly enabled transitions, called *intermediate semantics* [BD91]. We also use the common *strong semantics* for the firing delays of transitions. This allows the modeling of urgency. The different semantics for newly enabled transition are compared in [BCH+05a]. Lastly, a weak firing semantics has also been considered and compared with the strong firing semantics [BR08, RS09].

Other models like timed message sequence charts and time constrained message sequence charts are presented in Subsection 3.1.2. In the latter, we also present some comparisons of formalisms and translations from one formalism into the other.

1.3.2 Describing, Comparing and Analyzing Distributed Timed Behaviors

Some techniques, based on *partial orders*, have been introduced to improve the analysis of distributed systems, and in particular to cope with the *state space explosion problem*. However, there are still few such techniques for distributed *timed* systems.

In distributed systems, some issues related to the *knowledge* and the *view* of a component also arise.

Partial Order Representations

Trace Theory The theory of traces has been developed for the analysis of concurrent systems with static architecture, like safe Petri nets. Mazurkiewicz traces consider an order between two actions only if they are related by a causality relation, this is expressed by the notion of *dependency*. In Subsection 2.1.3, we recall the main notions from [DR95].

Unfoldings The unfolding of a Petri net represents its behaviors in a compact structure. An unfolding prescribes a partial order over the set of events, and some analysis techniques explore the unfolding of the Petri net instead of the

whole state space and all the interleavings. Unfoldings of Petri nets and time Petri nets are reviewed in Subsection 6.1.2.

Partial Order Techniques

There are several reduction techniques for the analysis of distributed systems. Some techniques consider the symmetries in concurrent systems comprised of replicated components, and take into account only one of symmetric states [HBL+03, DM06]. There are also compositional constructions that build the state space in steps. But, the techniques we are the most interested in are partial order reduction techniques that consider only one of equivalent interleavings.

Partial Order Reduction techniques explore the interleaving representation, but exploit information about the concurrency of the system in order to *reduce* the set of global states that need to be explored. For this, given a global state, the techniques compute a subset of the set of transitions leaving it, the *reduced* set, and only explore the transitions of this set. The literature contains different proposals for the computation of reduced sets: *stubborn* sets [Val89, Val92], *ample* sets [Pel96, CGMP99], and *sleep* sets [GW93, WG93, God96]. They are all presented in a survey paper [Val96]. Another partial order reduction technique, called *local first search*, based on a different approach, is presented in [NHZL01].

Lastly, partial order reduction techniques have been applied to timed systems [Min99, BJLY98, LNZ05].

Knowledge Representation

In distributed systems, components can infer information about the state of the other components. If there is no (explicit or implicit) communication in the system, then each component is "blind" and cannot view the rest of the system. However, it is often assumed that each component knows the structure of the other components, and their initial states, therefore, a component can guess that at a given time, another component is in one state among a set of states compatible with its point of view. When there are communications, each component may update its knowledge about the other components.

These problems are common in game theory, in multi player games with partial observation. It is crucial for a player to keep track of the knowledge about the other players during a play of the game. In two player games, *powerset constructions* represent, for any state of player 0, the states of player 1 that player 0 considers possible. In Chapter 5, we use a similar procedure to determine whether a timed automaton can avoid reading the clocks of another timed automaton.

Also, there are some epistemic logics to describe the knowledge of agents

in distributed systems [HFMV95], and they have been extended to real-time [WL04, LPW07, Dim09].

Lastly, in timed games with partial observability, some techniques of partitioning states based on observation are used [BDMP03, DLLN09]. The notion of contextual timed transition system we present in Chapter 5 resembles these partitioning techniques.

1.3.3 Implementation Point of View

When we consider implementing distributed systems on a multi-core architecture, we face several difficulties.

To start with, we want all communications in the model to be explicit, so that we know when the components interacts, and when they have to be independent. In this setting, shared clocks, and more generally shared variables are an issue, and have to be replaced by explicit communications, in order to get a model that is closer to the implementation. That is why some extensions of timed automata consider *distributed timed automata*, networks of timed automata with local clock (clocks can be reset only by their owner, but read by any automaton) [ABG+08, DL07, BJLY98]. Moreover, these works also suggest to use a local-time semantics, where the clocks of the different components may evolve at different velocities.

Then, since computers are digital and the hardware is imprecise, some properties that hold on the model do not hold any more on its implementation [BLM+11]. This leads to define *robustness* of timed automata.

More related works on shared clocks and implementability of timed automata are recalled in Subsection 3.1.2.

1.4 Contribution

In this thesis, we focus on the modeling, rather than on the verification, of distributed real-time systems. But we intend to provide some tools that simplify the verification of such systems, by enabling modular analysis or partial order techniques.

1.4.1 Formalisms for Modeling Distribution and Interaction

We study different formalisms, and present some extensions that we find especially suited for distributed timed systems. In this study, we focus on highlighting implicit communications induced by shared resources (input tokens of synchronizing transitions in time Petri nets, and shared clocks in networks of timed automata) and representing them explicitly while preserving the concurrency, and the individual behavior of each component.

Translation and Preservation of Concurrency We begin by defining a translation from (a subclass of) time Peri nets to networks of timed automata. This translation preserves the concurrency and makes some hidden communications in time Petri nets explicit. We also notice that we have to use an extension of networks of timed automata in order to preserve the distributed behavior of the initial time Petri net. The preservation of concurrency is formalized thanks to the notions of timed traces and distributed timed bisimulation.

These results were published in the proceedings of the conference TIME'10 [BCH10], and in a journal [BCH12a].

Shared Clocks The previous translation led us to consider shared resources, and models with explicit dependencies. We focus on the problem of shared clocks in networks of timed automata, and solve the problem of deciding, given a network of timed automata $A_1 \parallel A_2$ with shared clocks, whether there exists a network $A'_1 \parallel A'_2$ without shared clocks, and such that the individual behavior of each automaton is preserved. At first, we suggest a formalization of this problem, where we allow a more general synchronization mechanism, and we introduce the notions of contextual timed transition system and of contextual timed bisimulation. Then, we give a criterion that can be checked on a contextual timed transition system, and that allows us to solve our decision problem. Lastly, we show how to build a suitable network $A'_1 \parallel A'_2$ when such network exists.

These results are based on the results published in proceedings of the conference CONCUR'12 [BC12a], whose long version is the research report [BC12b].

1.4.2 Logical Dependencies between Event Occurrences

We then focus on the analysis of logical dependencies between event occurrences in the unfolding of Petri nets and time Petri nets. One major observation is that concurrent events are not always logically independent.

Untimed Setting We start by considering unfoldings of Petri nets. We extend previous results on the binary reveals relation defined in [Haa10]. We then introduce a logic that describes more general dependencies between event occurrences, and solve an associated synthesis problem. Here also, we aim at representing explicitly all logical dependencies.

These results were published in the proceedings of the conference ACSD'11 [BCH11] and in a journal [BCH12b].

Timed Setting Our next step is to study logical dependencies in a simple class of time Petri nets, and show that, even in this case, they are much more complex than in the untimed case. Our objective is to improve the unfolding by representing only minimal dependencies. This work is not published yet.

1.5 Organization of the Document

In Chapter 2, we recall basic notions about formalisms for distributed timed systems. To start with, we present notions for distributed untimed systems: the synchronous product of transition systems, the Petri net model, and some partial order representations. Then, we present notions for sequential timed systems: timed transition systems and timed automata. Finally, we present formalisms for distributed timed systems: synchronous product of timed transition systems, networks of timed automata and time petri nets.

The rest of this manuscript presents our contribution in two parts. In Part I we focus on formalisms for distributed timed systems as opposed to formalisms for centralized timed systems, and study concurrency in these formalisms. In Part II we conduct a study of the dependencies between event occurrences in distributed systems, first in the untimed case, and second, in the timed case.

Concurrency in Timed Models

Chapter 3 is the introductory chapter of Part I. It presents the problem and the related work and motivates the use of distributed semantics for studying distributed systems. This seems obvious, but this is not yet very used for two reasons: most of the time, the distributed semantics of distributed timed systems is not considered, and we still lack some notions for describing it. The chapter ends with some suggestions of such notions.

In Chapter 4, we present a translation from a time Petri net into a network of timed automata that preserve the distributed semantics. We also study the resulting network of timed automata, and show that it reveals some hidden dependencies in the time Petri net.

In Chapter 5, we focus on shared clocks in networks of timed automata. We give a criterion that says whether a given network of two timed automata can avoid using shared clocks, and show how to construct a network without shared clocks and whose automata have the same individual behavior as the initial automata.

Dependencies between Event Occurrences in Distributed Systems: from Untimed to Timed Settings

Chapter 6 is the introduction of Part II. It presents the problem and related work. This part focuses on the unfolding of Petri nets and time Petri nets.

In Chapter 7, we study occurrence nets, that are the structure of the unfolding of a Petri net. They represent the partial order semantics of the Petri net, and describe a set of events on which we can define three structural relations: causality, conflict and concurrency. In this structure, we notice that logical dependencies between event occurrences arise. We formalize these dependencies in a logical framework and solve a synthesis problem.

Lastly, in Chapter 8, we consider a simplified timed setting in which we study the logical dependencies and incompatibilities between events. We then define the notion of minimal enabling past and argue that it could be used in a recursive algorithm that would build valid processes.

The last chapter is a conclusion that summarizes both parts, concludes on the general contribution, and presents some perspectives.

Chapter 2

Formalisms for Distributed Timed Systems

This chapter presents well-known formalisms for real-time distributed systems. In this thesis, we focus on networks of timed automata and time Petri nets. We progressively introduced them by considering first formalisms for *distributed untimed systems*, then formalisms for *sequential timed systems*, and finally formalisms for *distributed and timed systems*

Formalisms for Distributed Systems Transitions systems are mathematical objects that can be used to describe the sequential behavior of an untimed process. Several transitions systems can be synchronized into one transition system. Then, the result of this *synchronous product* represents the sequential behavior of the distributed system formed by the different processes put in parallel. *Petri nets* is a well-known formalism for distributed systems. Some interesting classes of Petri nets enjoy nice algebraic properties that enable, in particular, the decomposition of the Petri net into sequential processes.

Partial order representations describe executions of distributed systems while preserving the information about the concurrent or independent events. Therefore they are much more compact representations than the ones that consider all the possible interleavings. In particular, we present *unfoldings* of Petri nets as a partial order representation of the executions of Petri nets.

Formalisms for Timed Systems A prerequisite for understanding formalisms for timed systems is the notion of *timed transition system* used to describe their *sequential* semantics. One of the most popular formalisms for modeling sequential timed systems is the *timed automata* formalism. Timed automata have been extensively studied since their introduction in the 90's [AD90, AD94], and are now a very well-accepted model for real-time systems. A timed automaton is a finite automaton extended with (dense time) clock variables that can be tested and reset.

Formalisms for Distributed Timed Systems In the last section, we recall some formalisms for both distributed and timed systems. We start by presenting the

synchronous product of timed transition systems, then we present *networks of timed automata*, that are parallel compositions of timed automata. Networks of timed automata are very convenient to model real-time distributed systems, by representing each component as a timed automaton, and providing a synchronization mechanism between the components. Lastly, we present the *time Petri net* model [Mer74] that extends the Petri net model with timing constraints, and enables the modeling of real-time distributed systems as well.

Organization of the Chapter Section 2.1 recalls formalisms for distributed systems without time, and in particular the Petri net model. Section 2.2 presents formalisms for timed systems, and in particular timed automata. Lastly, Section 2.3 presents networks of timed automata and time Petri nets, two formalisms for distributed timed systems.

Notations

$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$	natural numbers, integers, real numbers, and positive real numbers respectively
$[1..n]$	the set $\{k \in \mathbb{N} \mid 1 \leq k \leq n\}$
$[a,b]$	the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$
$[a,\infty)$	the set $\{x \in \mathbb{R} \mid a \leq x\}$
$v_{\mid X}$	the restriction of function v to the set X
$\boldsymbol{tt}$	symbol for true
$\mathbf{M}^{tr}$	transpose of matrix or vector $\mathbf{M}$
$\rightarrow^*$	reflexive transitive closure of binary relation $\rightarrow$

Operations on Multisets A multiset M is a function from a set P to $\mathbb{N}$, that generalizes the notion of set. The union and difference operations are defined as follows. Let M and M' be two multisets over P. Then, for all $p \in P$, $(M \cup M')(p) = M(p) + M'(p)$, and $(M \setminus M')(p) = \max(0, M(p) - M'(p))$.

2.1 Distributed Untimed Systems

Distributed systems are systems composed of several processes that run in parallel. Processes can perform concurrent actions or synchronize with one another. In this section, we focus on *untimed* distributed systems.

The notions of alphabet, word an language are basic notions for the formal analysis of untimed systems. A word describes an execution of an untimed system, and a language describes a set of executions.

Alphabet, Word and Language An *alphabet* Σ is a finite set of *actions*, also called labels. A *word* is a finite or infinite sequence of actions. The set of finite words over Σ is denoted by Σ^*. ε denotes the empty word of Σ^* and is also used to denote a silent action. We use Σ_ε to denote $\Sigma \uplus \{\varepsilon\}$. Lastly, a *language* is a finite or infinite set of words.

2.1.1 Synchronous Product of Transition Systems

A *transition system* is a tuple $(S, s_0, \Sigma, \rightarrow)$, where S is a set of states, $s_0 \in S$ is the initial state, Σ is a finite set of actions, and $\rightarrow \subseteq S \times \Sigma_\varepsilon \times S$ is a set of arcs. When $(s, a, s') \in \rightarrow$, we write $s \xrightarrow{a} s'$, and if $a = \varepsilon$, this transition is called *ε-transition*. In the following, we consider that transition systems may have ε-transitions.

A transition system represents the sequential behavior of an untimed system. Given n transition systems $T_1, \ldots, T_n$ such that each T_i represents the behavior of a system $\mathscr{S}_i$, the behavior of the distributed system composed of $\mathscr{S}_1, \ldots, \mathscr{S}_n$ can be represented by another transition system, called *synchronous product* of $T_1, \ldots, T_n$.

Below we define the synchronous product of two transition systems $T_1 = (S_1, s_1^0, \Sigma_1, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma_2, \rightarrow_2)$. Since this operation is associative, this readily gives the synchronous product of n transition systems.

The *synchronous product* of T_1 and T_2, denoted by $T_1 \otimes T_2$, is the transition system $\big(S_1 \times S_2, (s_1^0, s_2^0), \Sigma_1 \cup \Sigma_2, \rightarrow\big)$, 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 $a \in \Sigma_{1,\varepsilon} \setminus \Sigma_2$,
- $(s_1, s_2) \xrightarrow{a} (s_1, s_2')$ iff $s_2 \xrightarrow{a}_2 s_2'$, for any $a \in \Sigma_{2,\varepsilon} \setminus \Sigma_1$,
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ iff $s_1 \xrightarrow{a}_1 s_1'$ and $s_2 \xrightarrow{a}_2 s_2'$, for any $a \in \Sigma_1 \cap \Sigma_2$.

That is, arcs with the same actions (different from ε) are merged. These arcs are called *synchronizations*, and their label are called *common actions* as opposed to the actions that appear in only one alphabet, that are called *local actions*.

This product gives a sequential semantics to a distributed systems, therefore, when we want to consider the distributed semantics of a distributed system, the synchronous product is not a suitable notion.

We now define Petri nets, a formalism used to model distributed untimed systems. We define their syntax and semantics and recall some of their algebraic properties that we will use to decompose a Petri net into sequential components.

2.1.2 Petri Nets

Petri nets are used for modeling and validation of concurrent systems. They have been used in many fields, like in management of manufacturing systems, communication networks, and hardware design.

Indeed, Petri nets are a graphical formalism for the description of concurrency and synchronization inherent to distributed systems. A Petri net comprises transitions that represent computations or tasks, and places that represent resources or data. Thus, input places of a transition model input data (or required resources) and output places model output data (or produced resources).

Petri nets can also be formally analyzed to find possible problems of the system, for example deadlocks.

Syntax and Semantics

Definition 1 (Petri Net [Pet66, HSSW68])**.** A *Petri net* (PN) is a tuple (P, T, F, M_0) where P and T are two disjoint finite sets, called set of *places* and set of *transitions* respectively, $F \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs* connecting places and transitions, also called *flow relation*, and $M_0 : P \to \mathbb{N}$ is the *initial marking*.

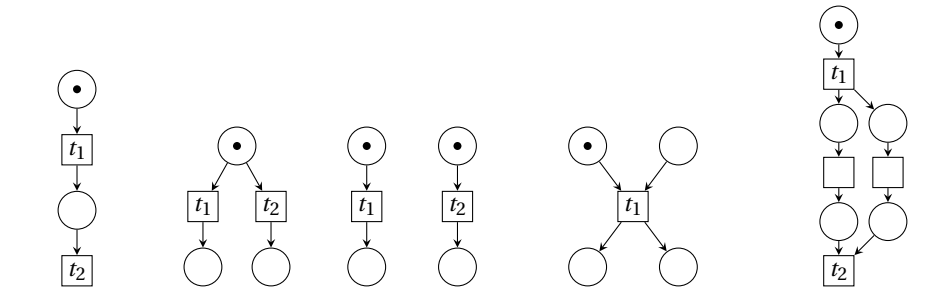
For $x \in P \cup T$, we define the *pre-set* of x as ${}^\bullet x = \{y \mid (y, x) \in F\}$ and the *post-set* of x as $x^\bullet = \{y \mid (x, y) \in F\}$. Given a set $X \subseteq P \cup T$, we define the pre-set and the post-set of X as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$.

Petri nets are sometimes defined with multiplicities on arcs, but we will consider only non multiple arcs, as defined above. This class of Petri nets is called ordinary Petri nets.

Graphical Representation Places are represented as circles, transitions as boxes, and the flow relation is represented by arcs. Marked places contains tokens (black dots). When necessary, we give the names (labels) of transitions inside the boxes, and the names of places beside the circles, see Fig. 2.1, 2.2 and 2.3.

Sequential Semantics A *marking* M of a PN is a multiset of places, i.e. a mapping from P to $\mathbb{N}$. A marking represents a state of the PN. A place p is *marked* if $M(p) > 0$. This is represented by $M(p)$ *tokens* in p. From the initial marking, i.e. the initial position of the tokens, the state of the PN evolves by transition firings that consume and produce tokens.

A transition t is *enabled* in a marking M iff for all $p \in {}^\bullet t$, $M(p) > 0$, what we also note ${}^\bullet t \subseteq M$. The set of transitions enabled in marking M is denoted



(a) Precedence (b) Choice (c) Concurrency (d) Synchronization (e) Fork and join

Fig. 2.1: *Petri net constructs*

by $En(M)$. Firing transition t from marking M consumes the tokens in its input places and produces tokens in its output places.

Formally, the behavior of a Petri net is defined as a transition relation on its markings. We write $M \xrightarrow{t} M'$ when firing t from marking M leads to marking M', defined as follows.

Firing of a transition: $M \xrightarrow{t} M'$ iff $t \in En(M) \wedge M' = (M \setminus {}^\bullet t) \cup t^\bullet$

Reachable Markings We note $M \rightarrow M'$ when a marking M' is reachable from a marking M in one step, i.e. if $M \xrightarrow{t} M'$ for some transition t. A marking M' is reachable from a marking M if $M \rightarrow^* M'$. A marking M is *reachable* if it is reachable from the initial marking M_0.

Firing Sequences A *firing sequence* or word generated by a Petri net is a sequence of transitions $\sigma = t_1 t_2 \cdots t_n$ such that $M_0 \xrightarrow{t_1} M_1 \cdots M_{n-1} \xrightarrow{t_n} M_n$. This is also denoted by $M_0 \xrightarrow{\sigma} M_n$.

Petri Nets Constructs The typical structural and behavioral characteristics exhibited by distributed systems, such as precedence, concurrency, choice, and synchronization, can be modeled by Petri nets. For example, consider transitions t_1 and t_2 in Fig. 2.1.

Precedence In Fig. 2.1(a), t_2 cannot fire if t_1 has not. We say that t_1 precedes t_2 or that t_1 is a *cause* of t_2.

Choice In Fig. 2.1(b), t_1 and t_2 compete for the same token, we say that they are in *conflict*. Both are enabled but firing one of them disables the other, therefore only one of them will fire at a given time. This let us model *mutual exclusion*, an important feature in systems with shared resources (see also Fig. 2.3), and non-deterministic choice.

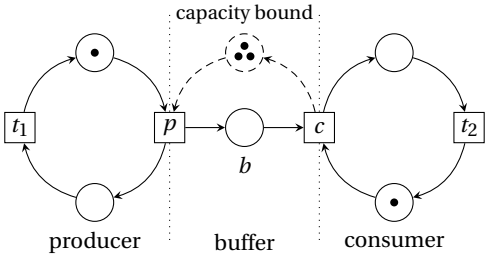


Fig. 2.2: *Producer-consumer example*

Concurrency In Fig. 2.1(c), t_1 and t_2 are *concurrent*, firing one of them has no influence on the other, and they can be fired in any order.

Synchronization In Fig. 2.1(d), t_1 models a task waiting for resources. Here, t_1 will be enabled only when each of its input places is marked. This results in a *synchronization* of threads: t_1 is performed by both threads before they split again.

Fork and Join In Fig. 2.1(e), t_1 creates one child thread that runs in parallel with the primary thread, and t_2 waits for the child to finish and proceeds the execution of the primary thread. This fork-join feature is a common model of concurrency.

But the relations can be more complex and need to be considered in terms of *firings* of transitions (later called events) rather than in terms of transitions. This will be described further in Subsection 2.1.3. For now, we just give some insights on classical examples depicted in Fig. 2.2 and 2.3.

Classical Examples We present two classical examples that illustrate concurrency and conflict in Petri nets.

Producer-Consumer Consider Fig. 2.2 without the dashed items. The semantics of the PN exhibits unordered firings of transitions. In particular, transitions t_1 and t_2 seem concurrent, both pct_1t_2 and pct_2t_1 are firing sequences that lead to the same marking. But because of the shared buffer in the middle, the second firing of t_2 must follow the first firing of t_1. Therefore, t_1 and t_2 are not concurrent although some of their firings are.

This example also shows that the number of tokens in a reachable marking can be unbounded. Here, still without the dashed items, the producer can loop forever, performing pt_1, each time producing a token in place b. If these tokens are not consumed, then the number of tokens in place b is infinite. There are some mechanisms to bound the number of tokens in a place, for example by adding a complementary place, as denoted by the dashed items in Fig. 2.2.

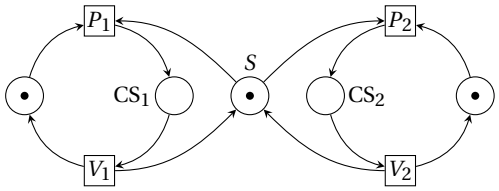


Fig. 2.3: *Two processes that share a semaphore*

For controlling access by multiple processes to a common resource, such as a shared buffer, one solution is to use semaphores.

Semaphore A semaphore [Dij65] is an object that encapsulates one integer variable (counter), and three atomic methods: $init(s)$ (initializes the counter with the number of available resources), $P(s)$ (waits for the counter to be not null and decrements it) and $V(s)$ (increments the counter).

Figure 2.3 is a classical example of how to model semaphores. In this example, two processes share a common resource that must not be accessed simultaneously. This mutual exclusion is modeled by a semaphore, i.e. a lock that prevents the two critical sections (i.e. the sections that access the shared resource) to overlap. Here, transition P_1 and P_2, that model the entrance in the critical sections, share a common input place, S, that models the lock (one token in S means the lock is free). Therefore, the first firings of P_1 and P_2 are in conflict, and if one of them fires, the other has to wait for the lock to be released to be enabled again.

Known Results Below we summarize some known results of interest. An overview of these results can be found in [EN94, Esp96].

Boundedness and k-Boundedness A Petri net is *bounded* if its set of reachable markings is finite. Boundedness was proved decidable in [KM69]. The complexity of the algorithm was later improved by [Rac78], [Lip76] and [RY86].

Another related problem is the k-boundedness problem. A Petri net is *k-bounded* iff, for any reachable marking M, for any place p, $M(p) \leq k$. The k-boundedness problem was proved to be PSPACE-complete in [JLL77].

The Reachability Problem for Petri nets consists in deciding, given a Petri net N and a marking M, if M is reachable. This problem was shown to be decidable in [May81] and later on, the proof was simplified in [Kos82]. The complexity of the reachability problem has been open for many years. An exponential space lower bound is known [Lip76], but no tight complexity bounds are known. However some solutions exist for different classes of

Petri nets. In particular, reachability is PSPACE-complete for 1-bounded Petri nets [CEP95].

Liveness and Deadlock-Freedom A Petri net is *live* if for every reachable marking M and every transition t, there exists a marking M' reachable from M that enables t. The liveness problem is recursively equivalent to the reachability problem, and thus decidable [Hac72]. The computational complexity of the liveness problem is open, but as for the reachability problem, there exist some solutions for different classes. In particular, the liveness problem is PSPACE-complete for 1-bounded Petri nets [CEP95], co-NP-complete for free-choice (see below) nets [JLL77], and polynomial for bounded free-choice Petri nets [ES92].

A Petri net is *deadlock-free* if every reachable marking enables some transition. Deadlock-freedom is reducible in polynomial time to the reachability problem [CEP95]. Deadlock-freedom is PSPACE-complete for 1-bounded Petri nets, and NP-complete for 1-bounded free-choice (see below) Petri nets [CEP95].

Some Interesting Classes of Petri Nets Since in physical systems, resources are limited, and tokens represent resources, it is reasonable to assume that PNs that model real systems are bounded. In particular, 1-bounded PNs are a well-studied class.

In the sequel, we shall mainly consider *1-bounded* Petri nets, also called *safe* Petri nets. Since with these PNs, there is never more than one token in each place, we will consider markings as sets of places. Safe Petri nets enjoy many interesting properties, in particular the complexity bounds of the problems defined above are known [CEP95], and they have a connection to Mazurkiewicz trace theory [DR95] that enables the use of efficient partial order verification methods.

Some structural classes of PNs have also been particularly studied. Below, we give three of them. These classes relate only to the structure (P, T, F) of a Petri net, that is called a net.

Free-Choice Nets A net is a *free-choice* net iff, for every two transitions t_1 and t_2, ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ implies ${}^\bullet t_1 = {}^\bullet t_2$ (see Fig. 2.4). This condition is sometimes called "extended free-choice", because there is a stronger condition for free-choice nets (that we will not consider). The free-choice property ensures that for any pair of conflicting transitions, every marking enables one of them iff it enables the other. Liveness and boundedness of free-choice PNs can be characterized by structural properties, i.e. without investigating all reachable markings [DE95].

Well-Formed Nets A net (P, T, F) is *well-formed* iff there is a marking M_0 such that (P, T, F, M_0) is live and bounded (see Fig. 2.4). Well-formed nets have

also been extensively studied, for example in [DE95]. One of the results is that well-formed nets are strongly connected [Bes86].

S-Nets A net (P,T,F) is an *S-net* if $\forall t \in T, |{}^\bullet t| = |t^\bullet| = 1$. Thus, an S-net can be seen as an automaton (places are locations and transitions are edges). In an S-net, there cannot be concurrency, but there can be conflict.

Algebraic Properties of Petri Nets

In this part, we focus on the structure (P,T,F) of a Petri net, without considering the initial marking, this structure is called a *net*. Such a net can be represented by a matrix called *incidence matrix*. This matrix is a mathematical tool for identifying some interesting properties of the flow relation F in terms of conservation of tokens. In particular, *S-invariants* can be computed from the incidence matrix, and some interesting associated subnets can be identified.

The Incidence Matrix of a net $N = (P,T,F)$ is a matrix $\mathbf{N} : (P \times T) \to \{-1,0,1\}$ that represents the flow relation, that is the dynamics of the net, as defined below.

Definition 2 (Incidence Matrix)**.** Let N be the net (P,T,F). The incidence matrix $\mathbf{N} : (P \times T) \to \{-1,0,1\}$ of N is defined by

$$\mathbf{N}(p,t) = \begin{cases} -1 & \text{if } (p,t) \in F \text{ and } (t,p) \notin F \\ 1 & \text{if } (p,t) \notin F \text{ and } (t,p) \in F \\ 0 & \text{otherwise.} \end{cases}$$

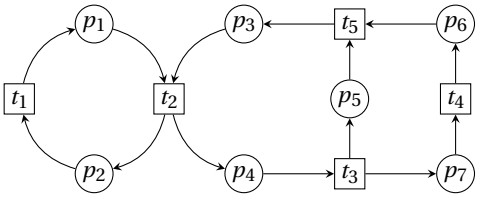
That is, the entry $\mathbf{N}(p,t)$ is 1 if the firing of t produces one token in p, -1 if the firing of t consumes one token in p, and 0 otherwise. Therefore, $\mathbf{N}(p,t)$ corresponds to the change of the marking of place p caused by the firing of transition t. Hence, if t is fired from marking M, the new marking is $M' = M + \mathbf{t}$, where $\mathbf{t}$ is the column vector of $\mathbf{N}$ associated with t, and markings are taken as column vectors. More generally, if $M \xrightarrow{\sigma} M'$ for some firing sequence σ, then $M' = M + \mathbf{N} \cdot \vec{\sigma}$. This equation is called *marking equation*. $\vec{\sigma}$ is the Parikh vector of σ, i.e. a vector of natural numbers with index set T, where $\vec{\sigma}(t)$ is the number of occurrences of t in σ. For example, the Parikh vector of $t_2\,t_1\,t_3\,t_4\,t_5\,t_2\,t_1$ is $(2\ 2\ 1\ 1\ 1)^{tr}$, while the Parikh vector of t_1 is $(1\ 0\ 0\ 0\ 0)^{tr}$.

An incidence matrix is given in Fig. 2.4, together with the associated net.

Let us now define S-components and then give their algebraic characterization as S-invariants.

S-Components can be regarded as processes of a net, in the sense that they are concurrency-free subnets, and they result from a decomposition of the net (for some classes of nets). We first need to define what is a P-closed subnet of a net.

A net (P',T',F') is a *subnet* of a net $N = (P,T,F)$ if $P' \subseteq P$, $T' \subseteq T$ and $F' = F \cap \big((P' \times T') \cup (T' \times P')\big)$. That is, only arcs connecting a place and a transition that are in the subnet are kept.



	t_1	t_2	t_3	t_4	t_5
p_1	1	−1	0	0	0
p_2	−1	1	0	0	0
p_3	0	−1	0	0	1
p_4	0	1	−1	0	0
p_5	0	0	1	0	−1
p_6	0	0	0	1	−1
p_7	0	0	1	−1	0

Fig. 2.4: *A net and its incidence matrix*

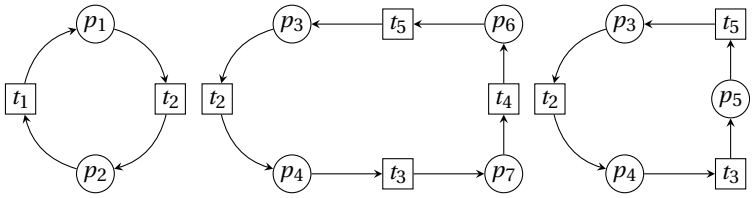


Fig. 2.5: *The S-components of the well-formed free-choice net of Fig. 2.4*

We say that the subnet (P', T', F') of N is *P-closed* if $T' = {}^\bullet P' \cup P'^\bullet$. That is, any transition connected to a place which is in the subnet is also in the subnet. The subnet of N *generated* by a set of places P' is the P-closed subnet (P', T', F') of N.

Definition 3 (S-Component)**.** A P-closed subnet (P', T', F') of a net $N = (P, T, F)$, with $P' \neq \emptyset$, is an *S-component* iff it is a strongly connected S-net.

An elementary property of S-components is the conservation of the number of tokens. That is, for any reachable marking M, if (P', T', F') is an S-component, then $|M \cap P'| = |M_0 \cap P'|$.

A net is *S-coverable* iff for any node (i.e. any place or transition) there exists an S-component which contains this node. The node is said to be covered by the S-component. Therefore, if we assume that any transition is connected to a place, since S-components are P-closed, a net is S-coverable iff any place is covered. That is why S-components will be identified to subsets of places, or mappings $P \to \{0, 1\}$.

S-Invariants characterize, for example, the conservation of the number of tokens in a subnet. They are defined as follows.

Definition 4 (S-invariant [Lau75])**.** An *S-invariant* of a net N is an integer-valued solution of the equation $X \cdot \mathbf{N} = \mathbf{0}$.

From the definition of incidence matrix it follows that a mapping $I : P \to \mathbb{Z}$ is an S-invariant iff for every transition t holds $\sum_{p \in {}^\bullet t} I(p) = \sum_{p \in t^\bullet} I(p)$.

Also, it follows from the marking equation that for any reachable marking M, and any S-invariant I, $I \cdot M = I \cdot M_0$. Therefore, the weighted sum of tokens is preserved.

An S-invariant I of a net is called *semi-positive* if $I : P \to \mathbb{N}$ and $I \neq \vec{0}$. The *support* of a semi-positive S-invariant I, denoted by $\langle I \rangle$, is the set of places p satisfying $I(p) > 0$. Every semi-positive S-invariant I satisfies ${}^\bullet\langle I \rangle = \langle I \rangle^\bullet$. A semi-positive S-invariant I is *minimal* if no semi-positive S-invariant J satisfies $\langle J \rangle \subsetneq \langle I \rangle$.

We are now able to state the link between S-components and S-invariants: S-components, as mappings $P \to \{0,1\}$, are minimal S-invariants [DE95, Proposition 5.7]. Therefore, the computation of S-invariants with values in $\{0,1\}$ gives the S-components of the net (S-invariants that generate strongly connected S-nets directly correspond to S-components).

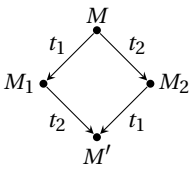
Lastly, one important theorem related to S-components is given below.

Theorem 5 (S-coverability Theorem [Hac72])**.** *Well-formed free-choice nets are covered by S-components.*

This theorem says that well-formed free-choice nets can be decomposed into S-components, as shown in Fig. 2.5. In Chapter 4, we will use this result to decompose a PN in components that correspond to the processes of the original physical system.

2.1.3 Partial Order Representations

Two transition firings are *concurrent* if they can be performed from the same marking M in any order, leading to the same marking M'. This results in the diamond depicted below. Representing and considering all interleavings has



some drawbacks: n concurrent firings would yield $n!$ possible executions and 2^n reachable states. However, more compact representations take advantage of this independency, consider that all these interleavings are equivalent, and use partial orders to describe equivalent executions.

Unfoldings of Petri Nets

Occurrence nets can be used to give the semantics of Petri nets by representing their executions (called processes). The unfolding of a Petri net is an occurrence net accompanied by a morphism relating the unfolding back to the original net. Here, we define unfolding of 1-bounded Petri nets, as it was introduced in [NPW81].

An occurrence net is an acyclic (possibly infinite) net, that comprises conditions (related to places of a PN) and events (related to transitions of a PN), with some characteristics that ensure in particular that each event is firable. We first need to introduce formally the causality, conflict and concurrency relations.

Causality, Conflict and Concurrency Given a net (P,T,F), we denote by $\leq$ the *causality* relation defined as: for any transitions s and t, $s \leq t$ iff $s\ F^*\ t$, and by $<$ the corresponding strict relation. For any transition t, the set $\lceil t \rceil = \{s \mid s \leq t\}$ is the *causal past* or *prime configuration* of t.

Two distinct transitions s and t are in *direct conflict*, denoted by $s \#_d t$, iff ${}^\bullet s \cap {}^\bullet t \neq \emptyset$. Two transitions s and t are in *conflict*, denoted by $s \# t$, iff $\exists s' \in \lceil s \rceil, t' \in \lceil t \rceil : s' \#_d t'$. The *conflict set* of t is defined as $\#[t] = \{s \mid s \# t\}$.

Lastly, two transitions s and t are *concurrent*, denoted by $s\ co\ t$, iff $\neg(s \# t) \wedge \neg(s \leq t) \wedge \neg(t \leq s)$.

Formally, an occurrence net is defined as follows.

Definition 6 (Occurrence Net)**.** An *occurrence net* (ON) is a net (B,E,F) where elements of B and E are called *conditions* and *events*, respectively, and such that:

1. $\forall e \in E, \neg(e \# e)$ (no self-conflict),
2. $\forall e \in E, \neg(e < e)$ ($\leq$ is a partial order, and such net is called *acyclic*),
3. $\forall e \in E, |\lceil e \rceil| < \infty$ (such net is called *finitary*),
4. $\forall b \in B, |{}^\bullet b| = 1$ (no backward branching),
5. $\bot \in E$ is the only $\leq$-minimal node (event $\bot$ creates the initial conditions).

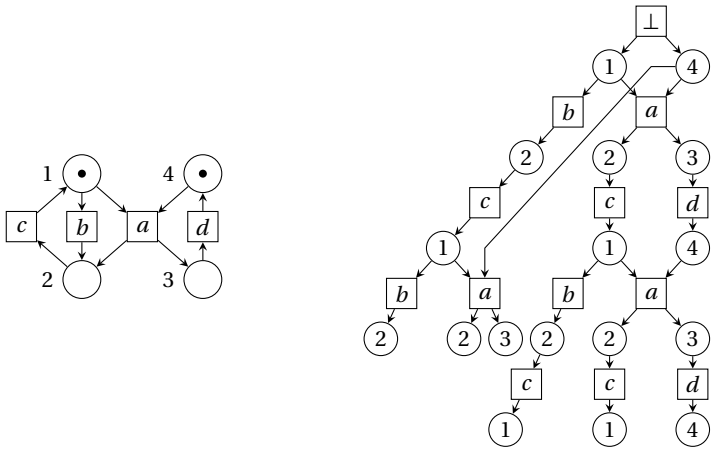
Figure 2.6(b) gives an example of ON. An ON can also be given as a tuple $(B, E \setminus \{\bot\}, F, \mathbf{c}_0)$, where $\mathbf{c}_0 = \bot^\bullet$ is the set of minimal conditions.

Unfoldings A *net homomorphism* from a net $N = (P,T,F)$ to a net $N' = (P',T',F')$ is a map $\pi : P \cup T \to P' \cup T'$ such that $\pi(P) \subseteq P'$, $\pi(T) \subseteq T'$, and for all $t \in T$, $\pi_{|{}^\bullet t}$, the restriction of π to ${}^\bullet t$, is a bijection between ${}^\bullet t$ and ${}^\bullet \pi(t)$, and $\pi_{|t^\bullet}$ is a bijection between $t^\bullet$ and $\pi(t)^\bullet$.

Let $N = (P,T,F,M_0)$ be a PN. A *branching process* of N is a pair (N',π), where $N' = (P',T',F',\mathbf{c}_0)$ is an ON and π is a homomorphism from (P',T',F') to (P,T,F), such that:

1. $\pi_{|\mathbf{c}_0}$ is a bijection between $\mathbf{c}_0$ and M_0,
2. $\forall t,t' \in T', \left({}^\bullet t = {}^\bullet t' \wedge \pi(t) = \pi(t')\right) \Rightarrow t = t'$

For $\Pi_1 = (N_1,\pi_1)$ and $\Pi_2 = (N_2,\pi_2)$ two branching processes, Π_1 is a *prefix* of Π_2, written $\Pi_1 \sqsubseteq \Pi_2$, if there exists an injective homomorphism h from N_1



(a) Petri net (b) Occurrence net

Fig. 2.6: *A Petri net and a prefix of its unfolding*

into a prefix of N_2, such that h induces a bijection between $\mathbf{c}_0^1$ and $\mathbf{c}_0^2$ and the composition $\pi_2 \circ h$ coincides with π_1.

By [NPW81, Theorem 23], there exists a unique (up to an isomorphism) $\sqsubseteq$-maximal branching process, called the *unfolding* of N. By abuse of language, we will also call unfolding of N the ON obtained by the unfolding. Although the unfolding may be infinite, it is possible to construct a finite complete prefix of the unfolding, such that each marking reachable in the original net corresponds to some concurrent set of places in such a prefix [McM92].

For now on, we denote ON as tuples (B, E, F) where $\bot \in E$. This allows simpler definitions. We now define configurations and cuts that can be regarded as runs and states of an ON.

Configuration: A *configuration* is a conflict-free and causally-closed set of events. That is, $\omega \subseteq E$ is a configuration iff $\forall e \in \omega$, $(\#\lceil e \rceil \cap \omega = \emptyset) \wedge (\lceil e \rceil \subseteq \omega)$.

Cut: A *cut* of a configuration ω is defined as $Cut(\omega) = \omega^\bullet \setminus {}^\bullet\omega$. i.e. $Cut(\omega)$ is the set of pairwise concurrent conditions that remain marked after the firing of ω. Also, $\pi(Cut(\omega))$ corresponds to a reachable marking of the initial PN [Eng91].

Below we summarize some applications, extensions and tools. A more complete description can be found in [EH08].

Some Applications The unfolding technique can be used as an approach for the analysis and verification of concurrent systems. It has been applied to the analysis of distributed algorithms, communication protocols, hardware and software systems, etc. In particular, it has been used in the analysis and synthesis of asynchronous logic circuits modeled as signal transition graphs

(STG) [KKY04, KKY06, KMY08]. Lastly, unfolding-based techniques are also developed for monitoring and diagnosis of discrete event systems [BFHJ03, CJ04, FBHJ05].

Some Extensions Unfoldings were first defined for 1-bounded PNs, but they were extended to bounded PNs [ERV02], and also to unbounded PNs [AIN00]. They were also defined for some extensions of PNs, as PNs with read arcs [VSY98, RSB11], high-level PNs [SK04], and time Petri nets [AL00, CJ06, FS02]. Lastly, unfoldings of NTA have also been proposed [BHR06, CCJ06]. Various tools exist for unfolding PNs and their extensions.

Other Partial Order Representations

Mazurkiewicz Traces The theory of traces has been developed for the analysis of concurrent systems with static architecture, like safe Petri nets. Mazurkiewicz traces consider an order between two actions only if they are related by a causality relation, this is expressed by the notion of *dependency*. Below we recall the main notions defined in [DR95].

A *dependency* relation D is defined over an alphabet Σ. D is finite, symmetric and reflexive. Then relation $I_D = (\Sigma \times \Sigma) \setminus D$ is the *independency* induced by D. Clearly I_D is symmetric and irreflexive. For example, if $D = \{a,b\}^2 \cup \{a,c\}^2$, then $I_D = \{(b,c),(c,b)\}$. The *trace equivalence* for D, $\equiv_D$, uses the fact that independent letters can be commuted: if $(a,b) \in I_D$, then $wabw' \equiv_D wbaw'$. A *trace* is an equivalence class of words, i.e a trace represents a single concurrent behavior, and the equivalent words are the different linearizations of the trace.

Lastly, a trace over (Σ, D) is also a labeled partial order $(E, \leq, \lambda)$ such that

- for all $e, f \in E$, $\left(e \not\leq f \wedge f \not\leq e\right) \implies (\lambda(e), \lambda(f)) \in I_D$ (unordered events correspond to independent actions),
- for all $e, f \in E$, $e \lessdot f \implies (\lambda(e), \lambda(f)) \in D$, where $\lessdot$ is the transitive reduction of $\leq$,
- for all $e \in E$, the set $\{f \mid f \leq e\}$ is finite (each event has a finite past).

Message Sequence Charts MSC [Rec11] is a scenario-based visual language for describing the interactions between asynchronous processes called instances. Graphically, the instances are represented by vertical lines, and the communications by arrows from the sending instance line to the receiving instance line. Thus, semantically, an MSC prescribes the *partial order* in which the communications occur: events (message sendings and receivings) are totally ordered along each instance line, and a message sending precedes its receiving. MSCs can be combined in high-level MSCs to describe more elaborate specifications.

There are also timed extensions of MSCs, for example timed MSCs [ZKH02], and time-constrained MSCs [ABG07].

2.2 Sequential Timed Systems

In this section, we introduce fundamental theoretical notions for describing and comparing the behavior of (models of) sequential timed systems. Such behavior can be described by *timed transition systems* which can then be used to compare the behavior of two models, possibly in two different formalisms. This comparison is generally done by means of *timed bisimulations*. In particular, when translating a model from a formalism into another formalism, timed bisimulations are used to prove that the translation is correct, i.e. that the new model has the same behavior as the original one. *Timed automata* is one of the most well-know formalism for sequential timed systems.

2.2.1 Timed Transitions Systems

Describing Timed Behaviors

An execution of a real-time system can be described by a timed word, that is a sequence of actions with time stampings. A timed language describes a set of executions as a set of timed words.

Timed transition systems are used to describe the behavior (all possible executions) of a model of a real-time system, regardless of the formalism in which it is given. They are therefore used to describe the semantics of the formalisms for timed systems, i.e. the meaning of each syntactical element of the formalism.

A timed transition system comprises a set of states and a set of transitions between states. Each transition denotes either a time delay or an action. Thus, a timed transition system prescribes a set of admissible timed words. However, we will see that a timed transition system provides more information than a timed language.

Most frequently, owing to the dense nature of time, timed transitions systems are infinite and even uncountable. That is why they are used as theoretical entities to describe the executions of a system, but they are not used directly to perform an analysis on these executions.

Timed Words and Timed Languages: A Description of Timed Executions A *timed word* (resp. *timed ε-word*) w over an alphabet Σ is a finite or infinite sequence $w = (a_0, t_0)(a_1, t_1)\dots(a_n, t_n)\dots$ such that for each $i \geq 0$, $a_i \in \Sigma$ (resp. $a_i \in \Sigma_\varepsilon$), $d_i \in \mathbb{R}_{\geq 0}$ and $t_{i+1} \geq t_i$ (the t_i's are absolute dates). Any timed ε-word is associated with a timed word obtained by removing the pairs (a_i, t_i) where $a_i = \varepsilon$. The untimed ε-word of timed ε-word $w = (a_0, t_0)(a_1, t_1)\dots(a_n, t_n)\dots$ is $\lambda(w) = a_0 a_1 \dots a_n \dots$ and the duration of w is $\delta(w) = \sup_i t_i$.

Lastly, a *timed language* over Σ is a set of timed words over Σ.

Timed Transition Systems: A Description of Timed Behaviors Below, we define timed transition systems that are a fundamental notion for the description

and the comparison of sequential timed behaviors.

Definition 7 (Timed Transition System [AD94])**.** 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,
- Σ is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$,
- $\rightarrow \subseteq S \times (\Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}) \times S$ is a set of arcs.

For any $a \in \Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}$, we write $s \xrightarrow{a} s'$ if $(s, a, s') \in \rightarrow$, and $s \xrightarrow{a}$ if for some s', $s \xrightarrow{a} s'$, in this last case, we say that s *enables* a.

A *path* of a timed transition system is a possibly infinite sequence of transitions $\rho = s \xrightarrow{d_0} s'_0 \xrightarrow{a_0} \cdots s_n \xrightarrow{d_n} s'_n \xrightarrow{a_n} \cdots$, where, for all i, $d_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \Sigma_\varepsilon$. A path is *initial* if it starts in s_0. Thus, an initial path describes one execution of the system. A path is *maximal* if it is infinite or ends in a state without outgoing arcs. In a path, delays and actions alternate, and a finite path may end with a delay. The duration of a path is the sum of the delays that occur along the path.

Generated Timed Language A path $\rho = s \xrightarrow{d_0} s'_0 \xrightarrow{a_0} \cdots s_n \xrightarrow{d_n} s'_n \xrightarrow{a_n} s'_n \cdots$ generates a timed ε-word $w = (a_0, t_0)(a_1, t_1)\ldots(a_n, t_n)\ldots$ where, for all i, $t_i = \sum_{k=0}^{i} d_k$. We write $s \xrightarrow{\rho} s'$ if ρ is a path from s to s'. We also write $s \xrightarrow{w} s'$ if there is a path from s to s' that generates the timed ε-word w. If w contains pairs (a_i, t_i) such that $a_i = \varepsilon$, these pairs have to be removed to obtain a generated timed word.

For describing the set of timed words *accepted* by the TTS, called the *generated* timed language, only initial paths are considered, because any execution starts in the initial state. Then *accepted* words are the words generated by maximal initial paths.

Lastly, for a given TTS T, the timed language generated by T is denoted by $\mathscr{L}(T)$.

Properties of Timed Transition Systems Since a TTS is supposed to represent (an abstraction of) the behavior of a physical (real) timed system, it has to satisfy some conditions. In particular, the transition relation verifies the following properties.

Time determinism: if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ for some $d \in \mathbb{R}_{\geq 0}$, then $s' = s''$;

0-delay: if $s \xrightarrow{0} s'$, then $s = s'$;

Additivity: if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ for some $d, d' \in \mathbb{R}_{\geq 0}$, then $s \xrightarrow{d+d'} s''$;

Continuity: if $s \xrightarrow{d} s'$ for some $d \in \mathbb{R}_{\geq 0}$, then for any $d' \in [0, d]$, there exists s'' such that $s \xrightarrow{d'} s''$ and $s'' \xrightarrow{d-d'} s'$.

For example, the time-determinism property states that, if a delay d is performed from a state s, and no action is performed meanwhile, then the new state depends only on s.

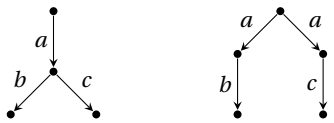
Zeno Behaviors Another property we want to ensure for the realism of the modeling, is the exclusion of Zeno behaviors. An infinite timed word whose duration is finite is called *Zeno*. Such a word describes an infinite behavior that takes a finite amount of time. Since this kind of behavior is not realistic, this needs to be excluded. Therefore a valid model does not allow such behaviors, i.e. its associated TTS (as it will be defined in Subsections 2.2.2 and 2.3.3) must not generate Zeno words.

Comparing Timed Behaviors

There exist several notions to compare timed systems. The choice of the notion to use depends on the desired level of comparison because some notions are coarser than others.

Timed Language Equivalence First, two TTS T_1 and T_2 are *timed language equivalent* if $\mathscr{L}(T_1) = \mathscr{L}(T_2)$, that is they accept the same timed words. This notion is interesting if we focus only on the sequences of actions that can occur. However, it does not capture the structure of the TTS. In particular, the possible choices at each state are not considered.

To illustrate this, we give a classical example, on untimed transition systems. The two transition systems below are language equivalent since they both accept the words ab and ac. Here, observe that the first transition system allows both b and c after a, whereas the second one allows only b or c depending on which a action has been performed, therefore the choice on performing b or c is not done at the same level. With only language equivalence, this difference is not caught, and we are not able to distinguish the two transition systems. This distinction is enabled by the notions of (timed) simulation and bisimulation.



For example, we can consider the strong timed simulation: state s' simulates state s when any transition from s to some state q can be simulated from s' by a transition with the same label (the same action or the same amount of time) that reaches a state q' that also simulates state q. When the transposed relation is also a simulation, the relation is a bisimulation. Bisimulation is accepted as

the finest equivalence relation among the many equivalence relations for concurrent systems [vG93]. It was extended to timed systems in [Yi90].

Timed Bisimulations Strong and weak timed bisimulations are formally defined below. We use the two TTS $T_1 = (S_1, s_1^0, \Sigma, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma, \rightarrow_2)$.

Definition 8 (Strong Timed Bisimulation)**.** Let $\approx$ be a binary relation over $S_1 \times S_2$. We write $s_1 \approx s_2$ for $(s_1, s_2) \in \approx$. $\approx$ is a *strong timed bisimulation* relation between T_1 and T_2 if $s_1^0 \approx s_2^0$ and $s_1 \approx s_2$ implies that, for any $a \in \Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}$,

- if $s_1 \xrightarrow{a}_1 s_1'$, then, for some s_2', $s_2 \xrightarrow{a}_2 s_2'$ and $s_1' \approx s_2'$;
- and conversely, if $s_2 \xrightarrow{a}_2 s_2'$, then, for some s_1', $s_1 \xrightarrow{a}_1 s_1'$ and $s_1' \approx s_2'$.

Let $\Rightarrow_i$ (for $i \in \{1,2\}$) be the transition relation defined as:

- $s \overset{\varepsilon}{\Rightarrow}_i s'$ if $s(\overset{\varepsilon}{\rightarrow}_i)^* s'$,
- $\forall a \in \Sigma$, $s \overset{a}{\Rightarrow}_i s'$ if $s(\overset{\varepsilon}{\rightarrow}_i)^* \overset{a}{\rightarrow}_i (\overset{\varepsilon}{\rightarrow}_i)^* s'$,
- $\forall d \in \mathbb{R}_{\geq 0}$, $s \overset{d}{\Rightarrow}_i s'$ if $s(\overset{\varepsilon}{\rightarrow}_i)^* \overset{d_0}{\Rightarrow}_i (\overset{\varepsilon}{\rightarrow}_i)^* \cdots \overset{d_n}{\Rightarrow}_i (\overset{\varepsilon}{\rightarrow}_i)^* s'$, where $\sum_{k=0}^{n} d_k = d$.

Definition 9 (Weak Timed Bisimulation)**.** A binary relation, $\approx$ over $S_1 \times S_2$ is a *weak timed bisimulation* relation between T_1 and T_2 if $s_1^0 \approx s_2^0$ and $s_1 \approx s_2$ implies that, for any $a \in \Sigma \cup \mathbb{R}_{\geq 0}$,

- if $s_1 \xrightarrow{a}_1 s_1'$, then, for some s_2', $s_2 \overset{a}{\Rightarrow}_2 s_2'$ and $s_1' \approx s_2'$;
- and conversely, if $s_2 \xrightarrow{a}_2 s_2'$, then, for some s_1', $s_1 \overset{a}{\Rightarrow}_1 s_1'$ and $s_1' \approx s_2'$.

We write $T_1 \approx T_2$ (resp. $T_1 \sim T_2$) when there is a strong (resp. weak) timed bisimulation between T_1 and T_2.

Relations Between Behavioral Equivalences It is possible to compare timed systems with respect to several behavioral equivalences. Timed language equivalence only considers the accepted words and completely ignores branching, although most theories consider the latter to be meaningful.

Timed bisimulations however consider the branching structure of the TTS and capture subtle differences. For that reason, they are generally accepted as the finest behavioral equivalence.

The three equivalences we presented can be ordered from the finest to the coarsest as follows. Two TTS that are strongly timed bisimilar are also weakly timed bisimilar, and two TTS that are weakly timed bisimilar are also timed language equivalent. Obviously, if the TTS have no ε-transitions, strong and weak timed bisimulations are just the same, because a weak bisimulation is just a bisimulation that abstracts ε-transitions. Lastly, as we saw on a simple example, language equivalence does not imply bisimulation.

An Operational Semantics for Models of Timed Systems

The aim of using operational semantics is to understand precisely the meaning of a model and to reason about it. For this, an operational semantics gives a formal description of the behavior of a model, and goes beyond the specificities of its formalism. It allows proving properties regardless of the formalism, and verifying that the model is working as expected.

Timed transition systems give the operational semantics of formalisms for timed systems. In Subsections 2.2.2 and 2.3.3, we present two formalisms: timed automata and time Petri nets respectively, and specify their semantics by timed transition systems.

Finally, for real models, the associated TTS are infinite. That is why TTS cannot be used directly to analyze and verify a model. For verification purpose, some abstractions are needed. In particular, we will present the region automaton abstraction at the end of Subsection 2.2.2.

2.2.2 Timed Automata

Timed automata are an extension of finite automata with a finite set of real-valued variables, called clocks, that evolve synchronously with time and enable measuring delays [AD90].

Clocks can be reset on transitions, therefore, at any instant, the value of a clock equals the time elapsed since the last time it was reset. Transitions are guarded by clock constraints that compare the current values of the clocks with time constants. That is, a transition is enabled only if its associated timing constraint is satisfied by the current values of the clocks.

Lastly, some definitions also assume invariants in the model [HNSY94]. Invariants are clock constraints assigned to locations that have to be satisfied while the location is active. Hence, invariants restrict time elapsing and ensure progress.

Syntax and Semantics

We describe first the syntax of clock constraints (guards and invariants), then the general syntax of a timed automaton. After that, we explain how clock constraints are evaluated, and give the semantics of a timed automaton as an associated TTS.

Clock Constraints Clocks are positive real-valued variables, that increase at the same rate. The set $\mathscr{B}(X)$ of clock constraints over the set of clocks X is defined by the following grammar.

$$g ::= x \bowtie k \mid g \wedge g \mid \textit{tt}, \text{ where } x \in X,\ k \in \mathbb{N} \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}$$

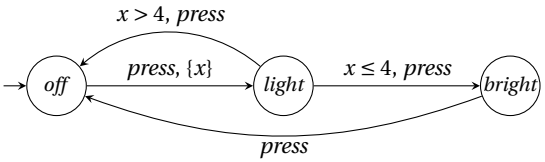


Fig. 2.7: *A timed automaton*

Invariants are clock constraints of the following form.

$$i ::= x \leq k \mid x < k \mid i \wedge i \mid \mathbf{tt}, \text{ where } x \in X,\ k \in \mathbb{N}$$

Clock constraints are used to impose enabling conditions on actions - in this case, they are called guards - or to restrict time delays - in this case, they are called invariants.

Definition 10 (Timed Automaton [AD94])**.** A *timed automaton* (TA) is a tuple $A = (L, \ell_0, X, \Sigma, E, Inv)$ where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial* location,
- X is a finite set of *clocks*,
- Σ is a finite set of *actions*,
- $E \subseteq L \times \mathscr{B}(X) \times \Sigma_\varepsilon \times 2^X \times L$ is a set of directed *edges*,
- and $Inv : L \to \mathscr{B}(X)$ assigns *invariants* to locations.

If $(\ell, g, a, r, \ell') \in E$, we also write $\ell \xrightarrow{g,a,r} \ell'$. For such an edge, ℓ is called the *source* location, g is the *guard*, a the *action* or label, r the set of clocks to be *reset* and ℓ' the *target* location.

Graphical Representation and Example Fig. 2.7 shows an example TA, taken from [AILS07]. Graphically, locations are represented by circles containing their name when useful, and edges by arrows labeled by the associated guard, action and reset. Invariants are written beside their associated location (see Fig. 2.9), and the initial location has an incoming arrow that is not rooted in any location.

This example models a light with two brightness levels, that can be adjusted by a switch. The lamp is initially off. When the switch is first pressed, the lamp is lighten at the lowest level. Then, if the switch is pressed again within 4 time units, the light becomes bright: the edge from *light* to *false* is not enabled because its guard is false, and the edge from *light* to *bright* is enabled. Otherwise, if the switch is pressed after 4 time units, the light is switched off. Lastly, if

the switch is pressed while the lamp is in the bright state, then the lamp is also switched off.

The operational semantics of a TA is given by an associated TTS. A state of the TA at a given time is defined by the current active location and the current values of the clocks. For instance, $\big(\mathit{light}, [x = 2]\big)$ is a state of the TA of Fig. 2.7. Below, we first present clock valuations, and then give the TTS associated with a TA.

Clock Valuations The values of the clocks of X at a given time are given by a function $v : X \to \mathbb{R}_{\geq 0}$, called *clock valuation* over X. The satisfaction of a clock constraint γ by a clock valuation v, denoted by $v \models \gamma$, is defined inductively as follows.

$$\begin{array}{lll} v \models x \bowtie k & \iff & v(x) \bowtie k \\ v \models \gamma_1 \wedge \gamma_2 & \iff & v \models \gamma_1 \text{ and } v \models \gamma_1 \\ v \models \mathit{tt} & \iff & \mathit{tt} \end{array}$$

For each set of clocks $r \subseteq X$, 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 $x \in X$.

Timed Transition System Generated by a Timed Automaton We denote by (ℓ, v) a *state* of a TA, where $\ell \in L$ is the current location and v is a clock valuation that maps each clock to its current value. The pair (ℓ, v) is a legal state for the timed automaton only if $v \models \mathit{Inv}(\ell)$. The initial state is $s_0 = (\ell_0, v_0)$, where ℓ_0 is the initial location, and v_0 maps each clock to 0.

Let $A = (L, \ell_0, X, \Sigma, E, \mathit{Inv})$ be a TA. We define TTS(A), the timed transition system generated by A as TTS(A) = $(S, s_0, \Sigma, \to)$, where

- $S = \{(\ell, v) \in L \times (X \to \mathbb{R}_{\geq 0}) \mid v \models \mathit{Inv}(\ell)\}$,
- $\to \in S \times (\Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}) \times S$ is defined by

  Time delay step: $(\ell, v) \xrightarrow{d} (\ell, v + d)$ for some $d \in \mathbb{R}_{\geq 0}$, iff $v + d \models \mathit{Inv}(\ell)$,

  Action step: $(\ell, v) \xrightarrow{a} (\ell', v')$ iff $\ell \xrightarrow{g,a,r} \ell'$, $v \models g$, $v' = v[r]$ and $v' \models \mathit{Inv}(\ell')$.

A *run* of a TA A is an initial path in TTS(A) where time delay steps and action steps alternate. A timed word is *accepted by* A if it is accepted by TTS(A). The timed language *generated by* A, $\mathscr{L}(A)$, is the timed language generated by TTS(A): $\mathscr{L}(A) = \mathscr{L}(\text{TTS}(A))$.

Known Results Below, we briefly recall some important results. When no precision is given, we assume that the TA have no ε-transitions.

We first recall undecidable problems, that limit the algorithmic analysis of TA. In fact, these undecidability results all follow from the undecidability of the

universal problem. Then we present two decidable problems that are fundamental for the verification of TA. Lastly, we give results on the class of TA with ε-transitions and on the class of TA with clock copies (a subclass of updatable TA [BDFP00a]).

The Universality Problem is to decide whether the language of a given automaton over Σ comprises all the timed words over Σ. This problem is undecidable for TA with 2 clocks or more [AD94] (proof by reduction from the recurring problem of a two-counter machine), and decidable for single clock TA [OW04] (because timed language inclusion also is). However, this problem is undecidable for single clock TA with ε-transitions [LW08].

Timed Language Inclusion and Equivalence are also undecidable, by reduction from the universality problem [AD94].

Complementability and Determinizability The universality problem is also reducible to the complementability and determinizability problems, that are therefore undecidable [Tri04]. It has also been shown that, even if a witness is not asked, the decision problems are still undecidable [Fin06].

Complementation is used for capturing the negation of the specification of an automaton. Determinization is important for implementability in particular.

The Emptiness Problem is to decide whether the language accepted by a timed automaton is empty. The set of states is infinite and thus, the classical methods for finite-state systems cannot be applied. Nevertheless, this problem was shown decidable and PSPACE-complete with the construction of a finite abstraction [AD94] (see region automaton below).

The Reachability Problem is to decide whether a given location ℓ is reachable, i.e. if there exists an execution that reaches a state where the current location is ℓ. This problem, that is equivalent to the emptiness problem, is fundamental in verification.

Strong Timed (bi)simulation between timed automata is decidable and EXPTIME-complete [LS00].

ε-Transitions do not change the decidability of the emptiness problem, but strictly increase the expressive power [BDGP98] (contrary to the untimed case). Moreover, it has been shown that the problem of deciding whether, for a given TA with ε-transitions, there exists a TA without ε-transition that accepts the same timed language is undecidable [BHR09]. This is however decidable for some subclasses, for example if ε-transitions do not reset clocks [BDGP98].

Updatable Timed Automata are an extension of TA introduced in [BDFP00a, BDFP00b]. In Chapter 3, we will consider only a subclass of updatable TA, where we allow copies of clocks, i.e. resets of the form $x := y$ where x and y are clocks. This class is not more expressive than classical TA (any updatable TA of the class is bisimilar to a classical TA), and the emptiness problem remains PSPACE-complete [BDFP04]. Clock copies are also available in the development snapshot of UPPAAL (since version 4.1.8 of February 29, 2012).

Region Automaton

Because a state of a timed automaton contains a clock valuation and clocks may have any value in $\mathbb{R}_{\geq 0}$, even a simple timed automaton may generate an uncountable set of reachable states. In order to be able to perform an algorithmic analysis of a timed automaton, a finite abstraction of the state space is needed. The idea beyond the *region abstraction* [AD94] is that clock valuations can be partitioned into finitely many equivalence classes, such that two valuations from the same equivalence class will induce "equivalent behaviors".

Notations First, for any $d \in \mathbb{R}_{\geq 0}$, we define the following numbers: $\lfloor d \rfloor$ is the integral part of d and $frac(d)$ is the fractional part of d. Then, for a given TA A, and any clock x of A, we define $c_x \in \mathbb{N}$ as the largest constant against which x is compared in the guards and invariants of A. For example, for the TA of Fig. 2.7, c_x is 4.

Definition 11. We say that two clock valuations $v, v' \in (X \to \mathbb{R}_{\geq 0})$ are *equivalent*, and we write $v \equiv v'$ iff

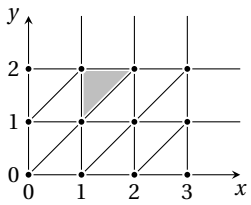
- for each $x \in X$, either $\lfloor v(x) \rfloor = \lfloor v(x') \rfloor$, or both $v(x) > c_x$ and $v'(x) > c_x$,
- for every $x \in X$ such that $v(x) \leq c_x$, $frac(v(x)) = 0$ iff $frac(v'(x)) = 0$,
- for every $x, y \in X$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$, $frac(v(x)) \leq frac(v(y))$ iff $frac(v'(x)) \leq frac(v'(y))$.

$[v]$ denotes the equivalence class of v with respect to $\equiv$.

Definition 12 (Region)**.** An $\equiv$-equivalence class $[v]$ represented by some clock valuation v is called a *region*.

For example, consider a timed automata with two clocks x and y with $c_x = 3$ and $c_y = 2$. The regions are shown in Fig. 2.8. The gray triangle represents the region described by $1 < x < y < 2$.

The number of regions is bounded by $n! \cdot 2^n \cdot \prod_{x \in X}(2c_x + 2)$, where n is the number of clocks. Therefore there are finitely many regions, but their number is



- 12 corner points,
- 30 open line segments,
- 18 open regions.

Fig. 2.8: *Regions for $X = \{x, y\}$, $c_x = 3$ and $c_y = 2$*

exponential in the number of clocks and in the constants that appear in the clock constraints. Moreover, the equivalence relation satisfies the following property.

$$v \equiv v' \implies \begin{cases} \text{for any guard or invariant } \gamma \text{ of } A, v \models \gamma \iff v' \models \gamma \\ \forall d \in \mathbb{R}_{\geq 0}, \exists d' \in \mathbb{R}_{\geq 0} : v + d \equiv v' + d' \end{cases}$$

A region r' is a *time-successor* of a region r iff for each $v \in r$, there exists a positive $d \in \mathbb{R}_{\geq 0}$ such that $v + d \in r'$.

Definition 13 (Region Automaton)**.** Let $A = (L, \ell_0, X, \Sigma, E, Inv)$ be a timed automaton. The region automaton of A, denoted by $R(A)$, is the transition system $R(A) = (S, s_0, \Sigma, \Rightarrow)$, where

- $S = \{(\ell, [v]) \mid \ell \in L, v : X \to \mathbb{R}_{\geq 0}\}$ is the set of *symbolic* states,
- $s_0 = (\ell_0, [v_0])$ where v_0 maps each clock to 0, and
- $\Rightarrow \subseteq S \times \Sigma_\varepsilon \times S$ is defined as follows: $(\ell, [v]) \stackrel{a}{\Rightarrow} (\ell', [v'])$ iff for some v'' such that $[v'']$ is a time-successor of $[v]$, $(\ell, v'') \stackrel{a}{\to} (\ell', v')$.

An example of region automaton is given in Fig. 5.8.

Theorem 14 ([AD94])**.** *Let A be a timed automaton. A and $R(A)$ are untimed bisimilar.*

In particular, this means that $\mathscr{L}(R(A)) = \mathscr{L}_u(A)$, where $\mathscr{L}_u(A) = \{\sigma \in \Sigma^* \mid \exists w \in \mathscr{L}(A) : \sigma = \lambda(w)\}$.

2.3 Distributed Timed Systems

We now consider the formalisms for both distributed and timed systems. In this section, their semantics is given as sequential notions, such as timed transition systems, product automaton or marking timed automaton. This is what is usually done, but we will later argue that, when we focus on concurrency, these notions are not suitable. Indeed, with a sequential description, it is as if there were only one component that was performing local actions.

2.3.1 Synchronous Product of Timed Transition Systems

As for transition systems, we can define the synchronous product of two TTS. For this definition, we use two TTS $T_1 = (S_1, s_1^0, \Sigma_1, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma_2, \rightarrow_2)$.

The *synchronous product* of T_1 and T_2, denoted by $T_1 \otimes T_2$, is the TTS $\big(S_1 \times S_2, (s_1^0, s_2^0), \Sigma_1 \cup \Sigma_2, \rightarrow\big)$, 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 $a \in \Sigma_{1,\varepsilon} \setminus \Sigma_2$,
- $(s_1, s_2) \xrightarrow{a} (s_1, s_2')$ iff $s_2 \xrightarrow{a}_2 s_2'$, for any $a \in \Sigma_{2,\varepsilon} \setminus \Sigma_1$,
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ iff $s_1 \xrightarrow{a}_1 s_1'$ and $s_2 \xrightarrow{a}_2 s_2'$, for any $a \in (\Sigma_1 \cap \Sigma_2) \cup \mathbb{R}_{\geq 0}$.

That is, arcs with the same actions (different from ε) are merged. These arcs are called *synchronizations*, and their label are called *common actions* as opposed to the actions that appear in only one alphabet, that are called *local actions*.

2.3.2 Networks of Timed Automata

The formalism of timed automata enables the modeling of only one component. For modeling real systems that are often distributed, it is convenient to be able to compose several timed automata.

Networks of timed automata [AD90, AD94] is a formalism that enables the modeling of distributed real-timed systems as a collection of TA. These TA run in parallel but may also synchronize with each other. Here we use a synchronization mechanism based on common actions: actions that appear in several alphabets are performed synchronously by all the automata whose alphabet contains the action, whereas actions that appear in only one alphabet are performed locally by the associated automaton.

Networks of timed automata have been extensively studied, and successfully used for specification and verification of real-time systems [CY92, HNSY94]. Some dedicated tools like KRONOS [BDM+98] and UPPAAL [BDL04] have also been developed.

Syntax and Semantics

Definition 15 (Network of Timed Automata)**.** A *network of timed automata* (NTA) is a parallel composition of n timed automata $(A_1 \parallel \cdots \parallel A_n)$, with $A_i = (L_i, \ell_i^0, X_i, \Sigma_i, E_i, Inv_i)$.

Figure 2.9 shows an example of NTA, taken from [AILS07]. The lamp presented before is now put in parallel with a user that can press the switch. The *press* action is now a common action and has to be performed simultaneously by the two automata. In this example, at some point before 10 time units, the user will press the button. The *press* action is then performed by both automata that move to states *light* and U' respectively, and reset their clock. Then the user

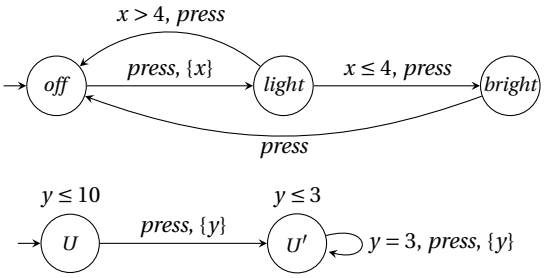


Fig. 2.9: *A network of timed automata*

keeps pressing the switch every 3 time units, causing the lamp to visit states *bright*, then *off* then *light*, then *bright*, and so on.

We denote by $X = \bigcup_{i\in[1..n]} X_i$ the set of clocks and by $\Sigma = \bigcup_{i\in[1..n]} \Sigma_i$ the set of actions. Clocks and actions may be shared.

Synchronizations The set of synchronizations *Sync* is defined as the set of $(e_1,\ldots,e_n) \in (E_1 \cup \{\bullet\}) \times \cdots \times (E_n \cup \{\bullet\}) \setminus \{(\bullet,\ldots,\bullet)\}$ such that

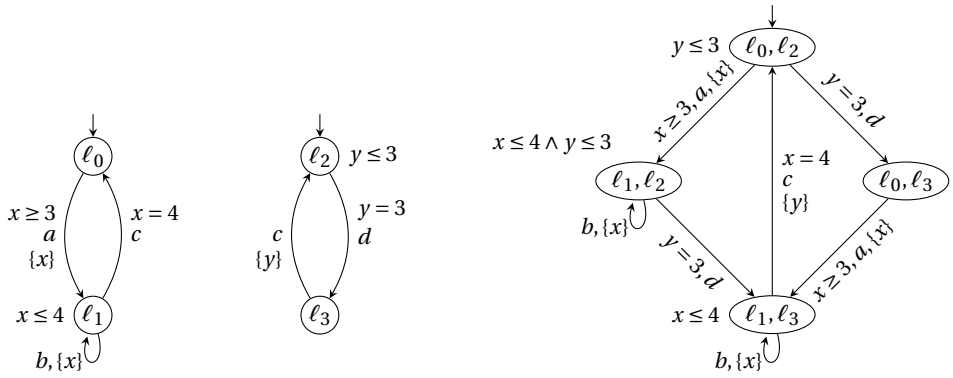
- either the same label $a \neq \varepsilon$ is attached to all the edges $e_i \neq \bullet$, and for all i such that $e_i = \bullet$, $a \notin \Sigma_i$,
- or only one component e_i is different from $\bullet$, and label ε is attached to this edge.

That is, we use the symbol $\bullet$ when an automaton is not involved in a step of the global system, and silent actions are performed by only one automaton. Lastly, for any $s = (e_1,\ldots,e_n) \in Sync$, $I_s = \{i \in [1..n] \mid e_i \neq \bullet\}$ denotes the indices of the automata that are concerned by the synchronization.

Sequential Semantics as a Timed Transition System We denote by $(\vec{\ell}, v)$ a state of an NTA, where $\vec{\ell} \in L_1 \times \cdots \times L_n$ is the vector of current locations and v is a clock valuation over X. We note ℓ_i the location associated with the i^{th} automaton in $\vec{\ell}$. The semantics of the NTA $(A_1 \parallel \cdots \parallel A_n)$ can be described as the timed transition system $\mathrm{TTS}(A_1 \parallel \cdots \parallel A_n) = (S, s_0, \Sigma, \rightarrow)$ where

- $S = \{(\vec{\ell}, v) \in (L_1 \times \cdots \times L_n) \times (X \to \mathbb{R}_{\geq 0}) \mid v \models \bigwedge_{i\in[1..n]} Inv_i(\ell_i)\}$,
- $s_0 = (\vec{\ell}_0, v_0)$ with $\vec{\ell}_0 = (\ell_1^0,\ldots,\ell_n^0)$, and $\forall x \in X, v_0(x) = 0$,
- $\rightarrow \in S \times (\Sigma_\varepsilon \cup \mathbb{R}_{\geq 0}) \times S$ is defined by

  Time delay step: $(\vec{\ell}, v) \xrightarrow{d} (\vec{\ell}, v+d)$ for some $d \in \mathbb{R}_{\geq 0}$, iff
  $v + d \models \bigwedge_{i\in[1..n]} Inv(\ell_i)$,



(a) Network of timed automata (b) Product automaton

Fig. 2.10: *Two timed automata and their product automaton*

Action step: $(\vec{\ell}, v) \xrightarrow{a} (\vec{\ell}', v')$ for some $a \in \Sigma_\varepsilon$, iff
$\exists s = (e_1, \ldots, e_n) \in \mathit{Sync}$ s.t. $\forall i \in [1..n]$, if $e_i = \bullet$, then $\ell'_i = \ell_i$,
otherwise $e_i = (\ell_i, g_i, a, r_i, \ell'_i)$,
$v \models \bigwedge_{i \in I_s} g_i$, $v' = v[\bigcup_{i \in I_s} r_i]$, and $v' \models \bigwedge_{i \in [1..n]} \mathit{Inv}_i(\ell'_i)$.

Sequential Semantics as a Product Automaton The semantics of a network of timed automata can also be given as a big timed automaton called *product automaton*. Below, we define the product of n timed automata.

Definition 16 (Product Automaton)**.** The *product automaton* of timed automata $A_1, \ldots, A_n$, denoted by $A_1 \otimes \cdots \otimes A_n$, is the timed automaton $(L, \ell_0, X, \Sigma, E, \mathit{Inv})$, where

- $L \subseteq L_1 \times \cdots \times L_n$ is the smallest set that contains ℓ_0 and is closed under E,
- $\vec{\ell}_0 = (\ell^0_1, \ldots, \ell^0_n)$,
- E is defined by: $\vec{\ell} \xrightarrow{g,a,r} \vec{\ell}'$ iff
  $\exists s = (e_1, \ldots, e_n) \in \mathit{Sync}$ s.t. $\forall i \in [1..n]$, if $e_i = \bullet$, then $\ell'_i = \ell_i$,
  otherwise $e_i = (\ell_i, g_i, a, r_i, \ell'_i)$,
  $g = \bigwedge_{i \in I_s} g_i$, and $r = \bigcup_{i \in I_s} r_i$,
- $\forall \vec{\ell} \in L, \mathit{Inv}(\vec{\ell}) = \bigwedge_{i \in [1..n]} \mathit{Inv}_i(\ell_i)$.

Thus the set of edges is obtained by coupling the edges of the individual automata having the same label (different from ε).

For example, Fig. 2.10(b) shows the product automaton of the two timed automata of Fig. 2.10(a).

Tools and Case Studies

NTA are very useful for modeling distributed real-time systems. Despite the uncountable state space, the reachability problem is decidable via the construction of a finite abstraction, called region automaton. This fundamental result enables the use of formal methods for the verification of NTA. However, in order to reduce the state space and be able to analyze large systems, some reduction methods are used.

Several tools have been implemented for the verification of NTA. Among them, UPPAAL [BDL04] uses NTA with handshake (binary) synchronizations, augmented with integer variables. KRONOS [BDM+98] uses NTA with binary or n-ary rendez-vous, also augmented with integer variables. Lastly, CMC [LL98] (for "Compositional Model Checking") focuses on avoiding the state explosion problem and implements a compositional method.

These tools have been successfully used in various industrial case studies, for example [BGK+96, DY00, RSV11] for UPPAAL and [MY96] for KRONOS.

2.3.3 Time Petri Nets

The Petri net model lacks the notion of time that is a critical factor in many real systems. In order to capture timing aspects, several timed extensions have been proposed. Time Petri nets [Mer74] is one of them. In this formalism, a firing interval $[efd(t), lfd(t)]$ is associated with each transition t. If δ denotes the moment when transition t has been enabled, then t *has* to fire within the interval $[\delta + efd(t), \delta + lfd(t)]$, unless it is disabled by the firing of another transition.

There are other variants of Petri nets extended with time. In timed arc Petri nets, each token has a clock representing its age, but a non-urgent semantics is assumed: the firing of a transition may be delayed and a transition may be disabled because its input tokens become too old [AN01, dFERA00]. Timed Petri nets [Ram74] associate a firing time to each transition and a transition fires as soon as possible, contrary to time Petri nets, where a transition fires in a time interval. Other variants are presented and compared in [BR08].

Syntax and Semantics

Definition 17 (Time Petri Net [Mer74])**.** A *time Petri net* (TPN) is a tuple (P, T, F, M_0, efd, lfd) where (P, T, F, M_0) is a Petri net and $efd : T \to \mathbb{R}_{\geq 0}$ and $lfd : T \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ associate an *earliest firing delay* $efd(t)$ and a *latest firing delay* $lfd(t)$ with each transition t.

Several semantics have been proposed for TPNs. Here we use the original and most common one, called *intermediate semantics* [BD91]. The different semantics are compared in [BCH+05a].

Newly Enabled Transitions We use the intermediate semantics: t' is *newly enabled* by the firing of t from marking M if it is not enabled by $M \backslash {}^\bullet t$ (intermediate marking) and it is enabled by $M' = (M \backslash {}^\bullet t) \cup t^\bullet$ (reached marking). Formally, we define the predicate $\uparrow enabled(t', M, t)$ as follows:

$$\uparrow enabled(t', M, t) \iff ({}^\bullet t' \subseteq M') \wedge \left({}^\bullet t' \not\subseteq (M \backslash {}^\bullet t)\right)$$

Strong Semantics For the firing delays of a transition, we use the *strong semantics*: t can fire if it is enabled and $\nu(t) \geq efd(t)$, and t *has to fire* before $\nu(t)$ overtakes $lfd(t)$.

The strong semantics, that enables to model urgency, is the most common one. However, a weak semantics has also been considered [BR08, RS09].

With these rules, we are able to define the semantics of a TPN as a TTS.

Sequential Semantics as a Timed Transition System A state of a TPN is given by (M, ν) where M is a marking and $\nu : T \to \mathbb{R}_{\geq 0}$ is a valuation such that each value $\nu(t)$ is the elapsed time since the last time transition t was enabled. ν_0 is the initial valuation with $\forall t \in T, \nu_0(t) = 0$.

The timed transition system generated by the TPN $\mathcal{N}$, is defined by $\text{TTS}(\mathcal{N}) = (S, s_0, T, \to)$ such that:

- $S = \{(M, \nu) \in \mathbb{N}^P \times (T \to \mathbb{R}_{\geq 0}) \mid \forall t \in T, {}^\bullet t \subseteq M \Rightarrow \nu(t) \leq lfd(t)\}$,
- $s_0 = (M_0, \nu_0)$ is the initial state,
- $\to \in S \times (T \cup \mathbb{R}_{\geq 0}) \times S$ is defined by:
  - discrete transition: $\forall t \in T, (M, \nu) \xrightarrow{t} (M', \nu')$ iff
    $$\begin{cases} ({}^\bullet t \subseteq M) \wedge efd(t) \leq \nu(t) \leq lfd(t)) \\ M' = M - {}^\bullet t + t^\bullet \\ \forall t' \in T, \nu'(t') = \begin{cases} 0 \text{ if } \uparrow enabled(t', M, t), \\ \nu(t') \text{ otherwise.} \end{cases} \end{cases}$$
  - continuous transition: $\forall d \in \mathbb{R}_{\geq 0}, (M, \nu) \xrightarrow{d} (M, \nu')$ iff $(\nu' = \nu + d) \wedge (\forall t \in T, {}^\bullet t \subseteq M \Rightarrow \nu'(t) \leq lfd(t))$

Example Figure 2.11 shows an example of TPN. Initially, transitions a and c are enabled, but only a is firable. If a 1 time unit delay is performed, then a and c are firable. Assume c fires at time 1, and a fires at time 2, then b has to fire immediately after a, because of its firing interval $[0, 0]$, and d is disabled before being firable. However, if a fires at time 0, and no other transition fires before time 2, then c and d reach their latest firing delay, therefore they have to fire immediately, unless they are disabled (b can fire immediately after c, and disable d).

Also observe that in the case when c and d both fire at time 2, the two firings are not ordered (neither causally, nor chronologically).

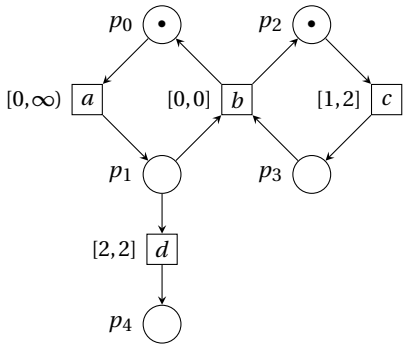


Fig. 2.11: *A time Petri net*

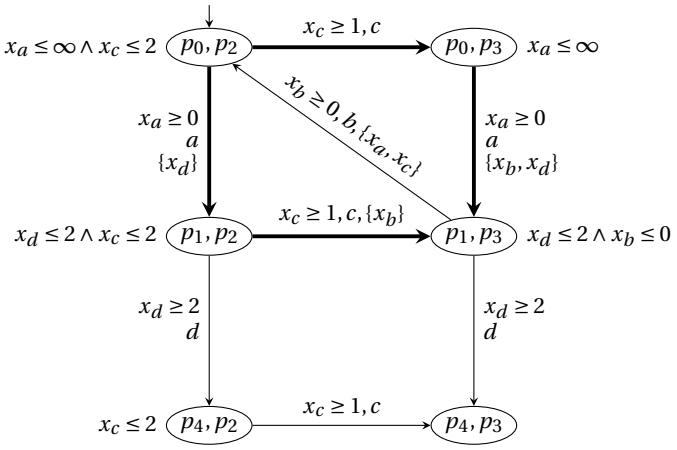


Fig. 2.12: *The semantics of the TPN of Fig. 2.11 as a timed automaton*

Sequential Semantics as a Timed Automaton We can also define the semantics of a TPN as a TA called marking TA and introduced in [GRR06]. Indeed, the marking TA of the TPN (P, T, F, M_0, efd, lfd) is the TA $(L, \ell_0, X, \Sigma, E, Inv)$ such that

- $L \subseteq 2^P$ is the set of reachable markings,
- $\ell_0 = M_0$,
- each clock $x_t \in X$ is associated with one transition t,
- $\Sigma = T$,
- $E = \{(M, g, t, r, M') \mid M' = (M \backslash {}^\bullet t) \cup t^\bullet, g \equiv x_t \ge efd(t), r = \{x_{t'} \mid \uparrow enabled(t', M, t)\}\}$,
- for each reachable marking $M \in L$, $Inv(M) \equiv \bigwedge_{\cdot t \subseteq M} \big(x_t \le lfd(t)\big)$.

A timed word is accepted by a TPN iff it is accepted by its marking TA. Figure 2.12 shows the marking TA of the TPN presented in Fig. 2.11. We note that concurrency is not explicit in this automaton, as it naturally gives 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.

Known Results Below we summarize some known results of interest. Most of the problems that were decidable for PNs without time are now undecidable. Only the k-boundedness is decidable.

Whether a marking M is reachable (i.e. whether some state (M', ν') such that $M' = M$ is reachable) is undecidable [JLL77] (it is shown that TPNs can simulate deterministic input-free 2-counter machines). Hence, the reachability of a state is also undecidable.

Because of this, boundedness and liveness are also undecidable.

However, k-boundedness is decidable by the construction of a finite abstraction called the state class graph [BD91].

Tools TPNs are becoming a well-accepted model for distributed real-time systems, and very mature tools for their simulation and verification exist.

TINA (TIme petri Net Analyzer) is a toolbox for the edition and analysis of PNs, with possibly inhibitor arcs and read arcs, TPNs, with possibly priorities and stopwatches [BRV04]. In particular, TINA can compute the S-invariants of a net.

Roméo is a software for TPN analysis. It performs analysis on TPNs and on one of their extension to scheduling. Roméo also deals with parameters and stopwatches [GLMR05, LRST09].

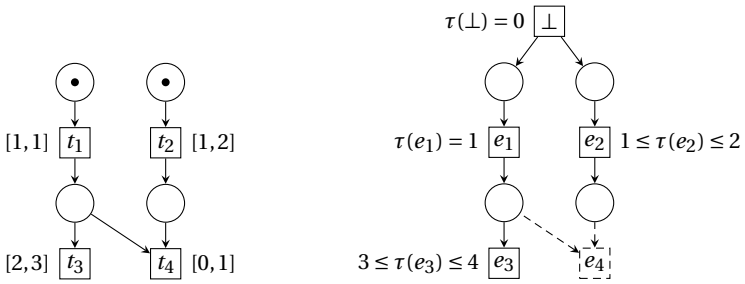
CPN Tools is a tool for editing, simulating, and analyzing Colored Petri Nets. The tool handles several variants of timed nets [JKW07].

Time Processes for Time Petri Nets

We now summarize some results about time processes for TPNs. These results give the first steps toward the unfolding of TPNs. They were introduced in [AL97].

Causal nets are defined as conflict-free ON. That is, a *causal net* $CN = (B, E, F)$ is a finitary , acyclic net, where $\forall b \in B, |{}^\bullet b| = 1 \wedge |b^\bullet \leq 1|$.

The relation between a TPN and a causal net is given by a net homomorphism, as for branching processes of a PN. When a causal net is associated with a TPN in this way, it is called *causal process* of the TPN. However, in order to represent a run of a TPN, a causal process (CN, π) must be equipped with a *timing function* $\tau : E \to \mathbb{R}_{\geq 0}$ that gives the *occurrence times* of the events. A *time process* of a TPN will be a causal process equipped with a timing function τ that has to be a *valid timing*.



(a) A time Petri net

(b) A causal process (without e_4), and some constraints over the occurrence times of its events

Fig. 2.13: *In addition to the constraints given in Fig. (b), a valid timing τ has to satisfy $\tau(e_3) \leq \tau(e_2) + 1$.*

We first define some notions that will help us define a valid timing. If B' is a set of conditions, and transition t is enabled by $\pi(B')$, the *time of enabling* for t in B' is defined as

$$toe(B', t) = \max(\{\tau(^\bullet b) \mid b \in B' \wedge \pi(b) \in {}^\bullet t\}).$$

The set of *earlier events* for an event e is

$$Earlier(e) = \{e' \in E \mid \tau(e') < \tau(e)\}.$$

The authors notice that owing to dependencies between events of a TPN, the notion of *valid timing* has to take into account several points. First, of course the occurrence time of an event must be in the interval defined by its time of enabling and its earliest and latest firing delays. These constraints are written in the example of Fig. 2.13(b). But, the occurrence of an event also depends on the events that could enable its conflicting events. For example, consider the TPN of Fig. 2.13(a), and the causal process of Fig. 2.13(b). Any timing function τ of the causal process has to satisfy $\tau(e_1) = 1$, $3 \leq \tau(e_3) \leq 4$, and $1 \leq \tau(e_2) \leq 2$. Moreover, for t_3 to be able to fire, the timing has to satisfy $\tau(e_3) \leq \tau(e_2) + 1$, otherwise t_4 has to fire before t_3 is firable. Here, the only valid timing for the given process is $\tau(e_2) = 2$ and $\tau(e_3) = 3$.

Therefore, timing constraints add causal constraints between events (in our example, e_3 can occur only if e_2 occurs late enough), that are not represented in the partial order defined by the causal net. That is why the occurrence date of an event e also involves the events enabled by $C_e = Cut(Earlier(e))$.

Finally, we are able to give the following definitions.

Definition 18 (Valid Timing, Time Process [AL97])**.** Let (P, T, F, M_0, efd, lfd) be a TPN, (CN, π) its causal process, where $CN = (B, E, G)$, and $\tau : E \to \mathbb{R}_{\geq 0}$ a timing

function. τ is a *valid timing* of the causal process if $\tau(\bot) = 0$ and

$$\forall e \in E, \tau(e) \geq toe({}^\bullet e, \pi(e)) + efd(\pi(e)) \tag{2.1}$$

$$\forall e \in E, \forall t \in En(\pi(C_e)), \tau(e) \leq toe(C_e, t) + lfd(t). \tag{2.2}$$

A *time process* of a TPN is a triple (CN, π, τ) where (CN, π) is a causal process of the TPN and τ is a valid timing of the causal process.

Intuitively, a time process represents an execution of the TPN (up to a given time). The image by π of a *timed linearization* of a time process gives a timed word accepted by the TPN. A timed linearization assigns valid time stamps to the events and imposes a total order over them, compatible with the partial order given by the causal net, and the chronological order given by the time stamps. For example $(e_1, 1)(e_2, 2)(e_3, 3)$ is a timed linearization of the time process of Fig. 2.13(a).

Moreover, one implicit property of a valid timing is the temporal completeness defined below. The temporal completeness ensures that all the concurrent parts of a causal process have advanced until the same time.

Definition 19 (Temporal Completeness [AL97])**.** Let (P, T, F, M_0, efd, lfd) be a TPN, (CN, π) its causal process, where $CN = (B, E, G)$. Then, the causal process (CN, π) is *temporally complete* with respect to timing function τ iff

$$\forall t \in En(\pi(Cut(E))), \max\{\tau(e) \mid e \in E\} \leq toe(Cut(E), t) + lfd(t).$$

Final Remark We recalled the work of [AL97], where the authors define valid timings for arbitrary causal processes. But every causal process cannot be assigned a valid timing.

Furthermore, contrary to the untimed case where the unfolding can be computed by looking independently at the concurrent parts, this cannot be done in the same manner in the timed case, because of the aforementioned dependencies. Nevertheless, a method for computing a finite complete prefix of a TPN has been defined in [CJ06].

Part I

Concurrency in Timed Models

Chapter 3

Centralized vs Distributed Timed Systems

This chapter is an introduction to Part I that focuses on concurrency in timed systems. It also introduces some notions that will be used in Chapters 4 and 5. So far, the different formalisms we presented were described using *sequential* notions such as timed words, timed languages and timed transition systems. Yet it is obvious that formalisms such as NTA and TPN can be used to model *distributed* timed systems. Here, we want to distinguish clearly between the formalisms and notions used to describe centralized timed systems, and those used to describe distributed timed systems. We also observe that we miss some notions to describe the distributed semantics of NTA and TPN.

Some formalisms were introduced for the study and the modeling of distributed timed systems, with a focus on the distributed nature and the *concurrency*. Timed extensions of message sequence charts emphasize the complex dependencies between the order imposed by the causality and the one imposed by the timing constraints [ABG07]. Another formalism, called distributed timed automata, considers NTA where each clock belongs to one automaton, and can be reset by this automaton only [ABG+08]. Hence the automata are more independent. After a formalization of independency, some works also define partial order reduction techniques for timed systems [Min99, BJLY98, LNZ05]. Lastly, we address the problem of implementing distributed timed systems on a distributed architecture, and in particular the problem of shared clocks.

In a second section, we present some extensions of NTA that focus on the *distribution* and the *interactions*. Then we introduce general notions, based on partial order, for describing the executions of distributed timed systems and comparing their behavior.

Organization of the Chapter We first present, in Section 3.1, the problem and the motivations of using distributed semantics to study models of distributed timed systems. Then we recall and compare some related works. Lastly, in Section 3.2, we present notions to describe and compare the distributed behavior of distributed timed systems.

3.1 Problem and Related Work

3.1.1 Problem

Distributed timed systems are timed systems with several components (or processes) that may perform local actions or synchronize with each other. We focus on two formalisms for such systems: NTA and TPN. Usually, the semantics of these formalisms is presented with sequential notions, such as timed words, TTS, product automaton (for NTA), or marking TA (for TPN). Indeed, there are only a few works on distributed semantics for *timed* systems.

Sequential vs Distributed Semantics

A sequential semantics describes the executions of a model as a total order over events. Therefore, such semantics is suitable for describing models of *centralized systems* such as TA. Indeed, a TA alone is considered as a sequential component, in the sense that it is supposed to model a single process that performs local actions only. But a sequential semantics is unsuitable for describing models of distributed systems such as NTA and TPN, because it is not able to describe the *distribution of actions* over the different processes, and the resulting partial ordering of events.

Moreover, when we want to preserve concurrency, we consider that a product automaton is not a good representation of an NTA and that a marking TA is not a good representation of a TPN. With such representations, the structural *separation of processes*, that comes from the physical separation, is lost. Hence the actions and variables (in particular clocks) are no longer distributed over the processes, and they all become local.

Therefore, in order to represent the behavior of NTA and TPN as models of distributed timed systems, we need a partial order semantics which reflects the distribution of actions over the processes. In Subsection 3.2.2, we present the notions of *timed trace* and *distributed timed language*. These notions are useful to characterize the fact that different representations of a same distributed timed system in different formalisms are equivalent.

Comparison and Translation of Formalisms

Several formalisms to model distributed real-time systems coexist in the literature. With each formalism comes a series of dedicated simulation and verification tools. This naturally entails a need to compare the expressiveness of the different formalisms, and to translate models from one formalism to another when possible.

The first formal comparisons of the expressiveness of these models focused on the preservation of the sequential behavior of the models, using notions like timed language equivalence or timed bisimilarity. They do not consider preservation of concurrency.

In Chapter 4, we want to compare and translate NTA and TPN while considering concurrency, i.e. the number of components and the *distribution of actions* over the components.

Implementability

The focus on the preservation of the number of components and the distribution of actions also finds a motivation when one considers implementing a distributed model on a distributed architecture.

Implementability of NTA NTA are widely used to model distributed real-time systems. Quite often in the literature, the automata are allowed to share clocks, i.e. the transitions of one automaton may be guarded by a condition on the value of clocks reset by another automaton. This is a problem when one considers implementing such model in a distributed architecture, since reading clocks a priori requires communications which are not explicitly described in the model.

In Chapter 5, we focus on the following question: given an NTA $A_1 \parallel A_2$ where A_2 reads some clocks reset by A_1, does there exist an NTA $A_1' \parallel A_2'$ without shared clocks, and with the same behavior as the initial NTA? For this, we allow the automata to exchange information during synchronizations only, in particular by copying the value of their neighbor's clocks. In order to formalize this problem, we need extended notions of TTS and timed bisimulation. To this purpose, we will define the notion of *contextual timed transition system*, which represents the behavior of A_2 when in parallel with A_1, and the associated notion of *contextual timed bisimulation*.

3.1.2 Related Work

Distributed Semantics for Timed Systems

In the untimed context, Mazurkiewicz traces [DR95] are defined using an independency relation that arises naturally from the distribution of actions. However, in the presence of time such relation would have less nice properties because even actions that occur in two totally independent processes may be ordered by their occurrence time. These orders induced by causality and by time stamping of events appear in [ABG07], where timed MSCs (Message Sequence Charts) and MSCs with timing constraints are considered, and in [ABG+08] where the authors consider distributed timed automata with independently evolving clocks.

In [PBV11], a simple extension of TPN that eases the specification of time dependent systems in a compositional approach is defined. Components can be easily derived from TPN specifications, without penalty for the future analysis of the whole system or for analysis of the individual components.

While partial order semantics are well known for untimed systems, they have been very little studied for distributed real-time systems.

Partial Order Reduction Techniques for Timed Systems Defining distributed semantics for timed systems enables the use of partial order techniques that improve their analysis. Partial order reductions were defined for both TPN and NTA. Partial order reductions for (N)TA were proposed in [Min99, BJLY98, LNZ05]. Some of them are detailed below.

An independency relation among the actions of a timed automaton, using a diamond property that takes time into account is defined in [LNZ05, NQ06]. This relation enables the use of partial order reduction techniques that avoid the combinatorial explosion in the analysis of timed automata. In [Pag96] a notion of independency between transitions is defined: two transitions are independent if they can be fired in any order and the resulting states are the same. Then this idea is lifted to the symbolic semantics and used for the detection of deadlocks.

Another approach, for time Petri net, is presented in [YSSC93, YS97], where an efficient model checking algorithm for the verification of real-time systems based on the partial order approach is presented.

Lastly, a local-time semantics for NTA is presented in [BJLY98]: local clocks evolve independently and are resynchronized at synchronization points. Then a method for partial order reduction in a symbolic reachability algorithm is presented.

Translations

The expressiveness of timed extensions of Petri nets and (N)TA has been compared in several works [BCH+05b, BR08, BHR08, Srb08].

Several transformations have been proposed and we observe the following. (i) The transformations mainly rely on natural structural equivalences between the basic elements of the formalisms. For instance, a location of a TA corresponds to a place of a TPN, a transition of a TPN corresponds to a tuple of synchronized transitions of an NTA, and the time interval associated with a transition of a TPN becomes a pair (guard, invariant) in a TA. (ii) Beyond these natural equivalences, there is no obvious one to one conversion in general. 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 do not consider preservation of concurrency. In [CR06], a structural transformation from TPN to NTA extended with integer variables is defined. This transformation builds a timed automaton per transition of the TPN and preserves weak timed bisimilarity. In the other direction, [BCH+08] shows that there are timed automata that are not weakly timed bisimilar to any TPN. In [BJS09], the authors propose a translation from bounded timed arc Petri nets to NTA, based on the decomposition of the net in sequential components that communicate through handshake synchronizations (in the UPPAAL style). In [SY96], 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 mechanism of deadlines, which hides the communications between components that would be necessary to implement it.

Some works have also translated extensions of MSCs into (N)TA or variants of TA. In [AGMK10], the authors translate regular collections of time-constrained MSCs into a special class of event-clock automata [AFH99] (an alternative to TA), thus permitting an algorithmic solution to the model checking problem. Lastly, in [LBD+10], a timed extension of live sequence charts (an extension of MSCs) is used in two approaches: (i) for modeling a system as a set of live sequence charts (LSCs), and (ii) for specifying a requirement that has to be satisfied by a system (modeled as an NTA or a set of LSCs). In order to implement LSCs in UPPAAL, the authors propose (i) a translation of the set of LSCs into an NTA so that each instance line is translated into one TA, and (ii) a translation of the LSC into an observer TA.

Locality of Clocks and Implementability

Locality of Clocks The semantics of time in distributed systems has been debated in the introduction. The idea of localizing clocks has already been considered in the model of *distributed timed automata*, and some authors [ABG+08, DL07, BJLY98] have even suggested to use local-time semantics with independently evolving clocks. In this thesis, we stay in the classical setting of perfect clocks evolving at the same speed. This is a key assumption that provides an implicit synchronization and lets us know some clock values without reading them.

A study of shared clocks is conducted in [LMST03], where the authors investigate the power of shared clocks for an extension of TA called *concurrent timed automata* (CTA). In particular, they prove that simulating shared clocks implies, in general, an exponential growth of the CTA with diagonal free clock constraints and constant updates.

Another extension of timed automata, *timed cooperating automata* is presented in [LMSP00]. In this extension, automata view the current state of other automata and the time elapsed since their activation.

Behavioral Equivalences for Distributed Timed Systems History-preserving bisimulations, behavioral equivalence relations for distributed untimed systems were defined in [BDKP91, vGG01]. Nevertheless, we are not aware of behavioral equivalences for distributed timed systems.

The notion of contextual timed bisimulation we present in Chapter 5 deals with the knowledge of agents in distributed systems. This is also the aim of epistemic logics [HFMV95], which have been extended to real-time [WL04, LPW07, Dim09]. Our notion of contextual TTS also resembles the technique of parti-

tioning states based on observation, used in timed games with partial observability [BDMP03, DLLN09].

Robustness and Implementability of Timed Automata Other works consider problems that come from the implementation platform. Because of the digital nature and the imprecision of the hardware, some properties that hold on the model do not hold on its implementation [BLM+11]. This observation leads to defining *robust timed automata*, as in [GHJ97]. Also, an implementable semantics, called almost ASAP semantics is considered in [DDR04], where it is observed that, with the usual semantics, some specifications cannot be implemented on a hardware, no matter how fast it is. However these works do not consider distributed systems.

3.2 Modeling Distribution and Interaction

In order to formalize preservation of concurrency in real-time models, we take into account the distribution of actions over a set of processes. Each process represents a component which has its own alphabet of actions. When an action belongs to several processes, it represents a synchronization, otherwise it is a local action.

We also consider the distribution of clocks: in some NTA, clocks may be local i.e. clocks are reset and read by only one automaton, whereas in so-called *distributed* TA [ABG+08, DL07], clocks can be reset by only one automaton (their owner) but can be read by any automaton.

3.2.1 Extensions of Networks of Timed Automata

Focusing on NTA as a formalism for distributed real-time systems imposes considering locality and distribution of actions and clocks (or other variables when the syntax is extended).

Below we consider two extended syntaxes for NTA. These syntaxes have been used in two works presented in Chapters 4 and 5.

Local Syntax vs Global State Syntax

We call *local syntax* the common syntax for NTA, but with local clocks only, i.e. every clock can be read and reset by only one automaton. This is a restriction of the syntax of NTA, where usually no such assumption is made. Thus, in this local syntax, invariants are still of the form $i ::= x \leq k \mid x < k \mid i \wedge i$, as defined in Subection 2.2.2.

We also define an extended syntax that we call *global state syntax* (and that will be used in Chapter 4), in which clocks can be read by any automaton, but

reset by only one automaton, and invariants are now of the following form.

$$i ::= x \leq k \mid x < k \mid i \wedge i \mid \ell \mid i \vee i$$

The two last constructors are not standard. In an invariant, "ℓ" is true if ℓ is a current location, that is, invariants are evaluated according to the global state of the system (current locations and valuation) and not only to the valuation. We denote by $\mathscr{B}(X, L)$ the set of such constraints over the set of clocks X and the set of locations L. Other operators that do not extend the expressiveness of i can be used, such as

- the negation of a location: $\neg \ell_i \equiv \bigvee_{\ell \in L_i \setminus \{\ell_i\}} \ell$,
- the implication: $\ell \Rightarrow (x \leq k) \equiv \neg \ell \vee (x \leq k)$, and
- the minimum of a set of clocks: $\min_{i \in I}(x_i) \leq k \equiv \bigvee_{i \in I}(x_i \leq k)$.

This extended syntax does not change the expressiveness w.r.t. the sequential semantics. Indeed, in the associated product automaton, clocks are local and the states of all TA are known, therefore there is no need to read a location ℓ in an invariant. Moreover, whatever the considered semantics, a simple construction with ε-transitions shows that operator $\vee$ does not increase the expressiveness. But we will show in Chapter 4 that, if we consider the *distributed* timed language (see Subsection 3.2.2), the global state syntax enhances the expressiveness of the NTA with local clocks.

Although it is not generally allowed to share active locations in timed automata, there are several variants of timed automata that can handle such a feature. For example, NTA can be extended with shared variables as in UPPAAL [BDL04], and a boolean variable can be associated with each location and used to denote whether the location is enabled (active). In [LMSP00], the authors propose another variant, Timed Cooperating Automata, a parallel composition of sequential automata where the edges can be guarded with timing constraints of the form $q = \tau$ (location q is enabled for τ time units), $q[\tau]$ (location q is enabled for at least τ time units), $q\{\tau\}$ (location q is disabled for at most τ time units) or boolean combinations of these terms.

A Special Case of Updatable Timed Automata

Updatable Timed Automata (UTA) were introduced in [BDFP00a] and further investigated in [BDFP00b, BDFP04]. Unlike standard TA that allow only resets of clocks (sometimes denoted as $x := 0$ where x is a clock), UTA are constructed with updates of the following form: $x :\sim k \mid x :\sim y + k$, where x and y are clocks, $\sim \in \{<, \leq, =, \neq, \geq, >\}$, and $k \in \mathbb{N}$. They also allow *diagonal* clock constraints, that are constraints of the form $x - y \leq k$, where x and y are clocks, and $k \in \mathbb{N}$, but we will not consider them.

We will consider only UTA with deterministic updates (the operator $\sim$ is the equality) and without diagonal constraints. For this subclass, it is shown in [BDFP00a], that the emptiness problem is decidable, which enables the verification of such TA. Also, one of the results presented in [BDFP00b] is that any UTA of this subclass is strongly timed bisimilar to a classical TA, i.e. to a TA with the standard syntax presented in Subsection 2.2.2. Hence, UTA with deterministic updates are as expressive as classical TA. In fact, to be more precise, with only updates of the form $x := k$ and $x := y$ (the updates we will consider), the results are the same if diagonal constraints are considered.

Although the subclass of UTA we consider is not more expressive than classical TA, it allows to represent in a concise way systems that cannot be modeled in a natural way with classical TA. Subclasses of UTA have been implemented in UPPAAL, with a technique presented in [BBFL03].

Lastly, we will see in Chapter 5 that when NTA are considered with a distributed semantics, networks of UTA are more expressive than classical NTA.

Two Extensions that Increase the Expressiveness of NTA with Local Clocks The two extensions we presented were used in two studies of the distributed semantics of NTA.

First, in Chapter 4 we show how to translate a TPN into an NTA with the global state syntax. We also prove that we cannot preserve the distributed semantics of the TPN if we restrict ourselves to the local syntax where clocks are not shared.

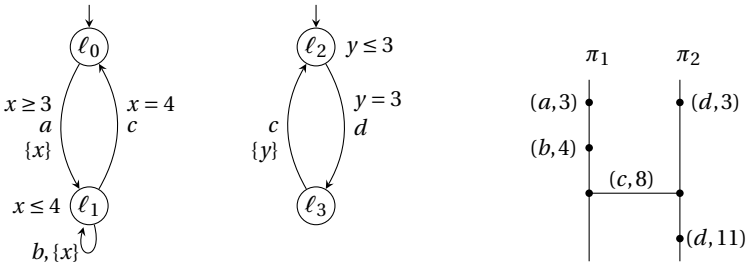
Second, in Chapter 5, we show how to decide whether a distributed NTA $A_1 \parallel A_2$, can be translated into an NTA $A_1' \parallel A_2'$ without shared clocks, but with clock copies (i.e. updates of the form $x := y$, where y can be a clock of a neighbor automaton) on synchronizations. We then present a construction of such $A_1' \parallel A_2'$ when it exists. In this work also, we show that in general, we cannot achieve the construction of $A_1' \parallel A_2'$ if we allow only classical NTA with a local syntax.

Below, we present *timed traces*, a distributed semantics for timed systems.

3.2.2 Timed Traces and Distributed Timed Bisimulations

A sequential semantics is not adapted to describe distributed systems because the information about the distribution of actions over the different components is lost.

We define timed traces as a partial order representation of executions of our models for real-time distributed systems. Timed traces provide an alternative to timed words, and take the distribution of actions into account. They generalize timed words and represent the executions of either an NTA or a TPN on which processes have been identified.



(a) A network of timed automata (b) A timed trace

Fig. 3.1: *A network of timed automata and a generated timed trace (one possible associated timed word is* $(d,3)(a,3)(b,4)(c,8)(d,11)$*)*

Timed Traces Given a distributed timed system over alphabet Σ, with its set of processes $\Pi = \{\pi_1, \ldots, \pi_n\}$, we can describe the runs of the system as *timed traces*. With this definition, each action $a \in \Sigma$ is associated with a set of processes, $proc(a) \subseteq \Pi$, that always perform it together and simultaneously, therefore it may be local or shared (synchronization). *Events* (action occurrences) are partially ordered since two events on disjoint sets of processes may not be causally ordered. The set of events is denoted by E, and for $e \in E$, $\lambda(e) \in \Sigma$ is the name of the action associated with event e.

We first give some preliminary notations:

- $\Sigma_i = \{a \in \Sigma \mid \pi_i \in proc(a)\}$ denotes the alphabet of process π_i, and
- $E_i = \{e \in E \mid \lambda(e) \in \Sigma_i\}$ denotes the set of events that occur on process π_i.

Definition 20 (Timed Trace, Distributed Timed Language)**.** A *timed trace* over the alphabet Σ and the finite set of processes $\Pi = \{\pi_1, \ldots, \pi_n\}$ is a tuple $W = (E, \preccurlyeq, \lambda, \delta, proc)$ where

- E is a countable set of *events*,
- $\preccurlyeq \subseteq (E \times E)$ is a *partial order* over E such that, for any event e, the set $\{e' \in E \mid e' \preccurlyeq e\}$ is finite, and for any i in $[1..n]$, $\preccurlyeq_{|\pi_i} = \preccurlyeq \cap (E_i \times E_i)$ is a *total order* on E_i.
- $\lambda : E \to \Sigma$ is a labeling function,
- $\delta : E \to \mathbb{R}_{\geq 0}$ assigns a date to every event such that, if $e_1 \preccurlyeq e_2$, then $\delta(e_1) \leq \delta(e_2)$;
- $proc : \Sigma \to 2^{\Pi}$ is the *distribution of actions* that maps each action to a subset of Π.

A *distributed timed language* is a set of timed traces.

Observe that two unordered events e_i and e_j are necessarily performed by two disjoint sets of processes, i.e. satisfy $proc(\lambda(e_i)) \cap proc(\lambda(e_j)) = \emptyset$. Two unordered events are called *concurrent*.

Graphical Representation Figure 3.1 gives a representation of a timed trace. Each process is represented by a vertical line, and each event is represented by either a dot or dots connected by a horizontal line, according to whether it occurs on one process or on several processes. Each event $e \in E$ is also labeled by the pair $(\lambda(e), \delta(e))$. Moreover, events are ordered along each process from the top to the bottom of the line, and we can see that events on different processes are not always ordered. For example, $(a,3) \preccurlyeq (b,4)$, $(b,4)$ and $(d,3)$ are not ordered, and $(b,4) \preccurlyeq (d,11)$ because $(c,8)$ takes them apart by transitivity.

Timed Linearization and Projection A *timed linearization* of a timed trace is a possible execution expressed as a timed word which respects both the causal order prescribed by the partial order, and the chronological order imposed by the time stamping. For example, $(d,3)(a,3)(b,4)(c,8)(d,11)$ is a timed linearization of the timed traces of Fig. 3.1(b). For a given timed trace, there can be several timed linearizations. In the given example, $(a,3)$ and $(d,3)$ can be switched.

The *projection* of a timed trace W onto process π_i, denoted by $W_{|\pi_i}$ is defined as the projection of any linearization of W, w, onto Σ_i, denoted by $w_{|\Sigma_i}$:

- if $w = \varepsilon$, then $w_{|\Sigma_i} = \varepsilon$
- if $w = (a,\theta) \cdot w'$, then $w_{|\Sigma_i} = \begin{cases} (a,\theta) \cdot w'_{|\Sigma_i} & \text{if } a \in \Sigma_i \\ w'_{|\Sigma_i} & \text{otherwise} \end{cases}$

Graphically, this corresponds to taking all events, in the process line π_i, from the top to the bottom. For example, if W is the timed trace of Fig. 3.1(b), then, $W_{|\pi_1} = (a,3)(b,4)(c,8)$.

Definition as a Timed Word and a Distribution of Actions A timed trace W can be defined as a tuple $(w, proc)$ where w is a timed linearization of W.

This means that, given a timed word $w = (a_0,d_0)\dots(a_n,d_n)\dots$ accepted by an NTA, and the distribution of actions *proc* over the automata, we can build an accepted timed trace for the NTA. Namely, $E = \{e_0,\dots,e_n,\dots\}$, λ and δ are such that, for each $i \geq 0$, $\lambda(e_i) = a_i$ and $\delta(e_i) = d_i$, and $\preccurlyeq$ is the transitive closure of the relation $\preccurlyeq'$ defined as: for any events e_i and e_j, $e_i \preccurlyeq' e_j \iff \big(i \leq j \wedge proc(\lambda(e_i)) \cap proc(\lambda(e_j)) \neq \emptyset\big)$.

Hence, a distributed timed language can also be defined as a timed language and a distribution of actions.

Definition as a Juxtaposition of Timed Words The *juxtaposition* of n timed words, $w_1 \parallel w_2 \parallel \cdots \parallel w_n$ is the timed trace W, over the set of processes $\Pi = \{\pi_1, \ldots, \pi_n\}$, such that for each i in $[1..n]$, $W_{|\pi_i} = w_i$.

Hence, if (a,θ) appears in the timed words $w_{i_1}, \ldots, w_{i_k}$, then $proc(a) = \{\pi_{i_1}, \ldots, \pi_{i_k}\} \subseteq \Pi$, and there is an event $e \in E$ such that $\lambda(e) = a$ and $\delta(e) = \theta$. There are as many events with this label and date as there are (a,θ) in any w_{i_j} with $j \in [1..k]$. Lastly, $\preccurlyeq$ is the transitive closure of the union of the causal orders imposed by the words $w_1, \ldots, w_n$.

This juxtaposition is defined such that for any timed trace W over $\pi_1, \ldots, \pi_n$, $W = W_{|\pi_1} \parallel \cdots \parallel W_{|\pi_n}$. For example, $(a,3)(b,4)(c,8) \parallel (d,3)(c,8)(d,11)$ gives the timed trace of Fig. 3.1(b).

These notions are similar to the notion of *histories*, for untimed concurrent systems, introduced in [Shi85]. The main idea is to represent non-sequential processes by a collection of *individual histories* of components running concurrently. Such collection is called *global history*.

Distributed Timed Bisimulation

A natural notion that follows from the definition of a timed language as a timed word together with a distribution of actions, is the definition of a distributed timed bisimulation. As in the sequential case, distributed timed bisimulation is a finer behavioral equivalence than distributed timed language equivalence.

Definition 21 (Distributed Timed Bisimulation)**.** Let $\mathscr{S}_1$ and $\mathscr{S}_2$ be two distributed timed systems over n processes and the same alphabet, S_1 and S_2 their respective TTS, and $proc_1$ and $proc_2$ their respective distributions of actions. Then, $\mathscr{S}_1$ and $\mathscr{S}_2$ are in distributed strong (resp. weak) timed bisimulation if

1. S_1 and S_2 are strongly (resp. weakly) timed bisimilar, and
2. there is a bijection bij between the processes of $\mathscr{S}_1$ and those of $\mathscr{S}_2$ such that $bij \circ proc_1 = proc_2$ (same distribution of actions over the processes).

Distributed Timed Bisimulations and Local Clocks Assume $\mathscr{S}_1$ and $\mathscr{S}_2$ are two networks of n timed automata with the same distribution of actions (the i^{th} automata of $\mathscr{S}_1$ and $\mathscr{S}_2$ have the same alphabet). If the individual timed automata are pairwise timed bisimilar (i.e the i^{th} automata of $\mathscr{S}_1$ and $\mathscr{S}_2$ are timed bisimilar), then $\mathscr{S}_1$ and $\mathscr{S}_2$ are distributed timed bisimilar.

The other direction does not hold because of the synchronizations that can occur only if all the concerned components are ready to synchronize.

Distributed Timed Bisimulations and Shared Clocks When there are shared clocks, we cannot perform a pairwise comparison of the individual timed automata. Indeed, since they read clocks of their neighbors, their behavior may depend on the one of their neighbors.

Therefore, if we want to pairwise compare the timed automata, this distributed timed bisimulation is not helpful because we have to somehow include the context of the automata (i.e. their neighbors) in this comparison. That is why we introduce the notion of *contextual timed bisimulation* in Chapter 5. We present briefly this notion below.

Contextual Timed Bisimulations are used to compare two timed automata when they are put in parallel with a same timed automaton of which they may read the clocks. This bisimulation is based on the notion of contextual timed transition system that represents the knowledge of an automaton about another automaton, and the possible current state of this other automaton.

A Distributed Semantics for Distributed Timed Systems

The notion of timed trace will be particularly useful to compare systems modeled by TPN or NTA, while considering the distribution of actions over the processes, that is the concurrency.

In an NTA, it is clear that each automaton corresponds to a process. But in a TPN, the identification of the processes is not so straightforward. In the next chapter, we aim at identifying the processes in a TPN (in fact in the untimed underlying PN), as a first step of a concurrency-preserving translation from TPN to NTA. The decomposition of a PN in processes is based on the idea of S-components presented in Subsection 2.1.2.

Chapter 4

A Concurrency-Preserving Translation from Time Petri Nets to Networks of Timed Automata

In this chapter, we propose a translation between two popular formalisms for the modeling of distributed real-time systems: TPN and NTA. These formalisms have different histories but were both designed to model *distributed* real-time systems. Moreover they both handle *urgency*, which is a key feature without which most real-time systems cannot be modeled correctly.

Preservation of Concurrency Here we focus on the preservation of concurrency. Since both TPN and NTA were designed to model distributed systems, we consider that not only their sequential behavior as timed transition systems is relevant, but also their distributed behavior. This 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, and a transformation should preserve this structure. Our translation preserves the distribution of actions, that is we require that if the TPN represents the product of several components (called processes), then each process should have its counterpart as one timed automaton in the resulting NTA.

Motivation Our motivation for this is twofold: 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.

Formalization In order to formalize preservation of concurrency in the context of real-time models, we use the notions of timed traces and of distributed timed bisimulations introduced in Subsection 3.2.2. That is, we take into account the distribution of actions over a set of processes, each process representing a component which has its own alphabet of actions.

Organization of the Chapter In Section 4.1 we recall how to identify the processes in a Petri net, and then present the translation procedure. We propose a translation from a 1-bounded TPN to a distributed timed bisimilar NTA. Then, in Section 4.2, we show that this translation cannot be done without using an extended syntax which allows, in particular, shared clocks. We also define conditions under which our translation can be adapted to avoid using shared clocks. Finally, in Section 4.3, we discuss extensions and limitations of our translation.

4.1 Translation from Time Petri Net to Network of Timed Automata

We define a structural transformation from (a class of) TPN to NTA which preserves timed traces. That is, we require that, if the TPN represents the product of several components (called processes), then each process has its counterpart as one timed automaton in the resulting NTA and the distribution of actions among the components is preserved.

To this end, we first discuss how to identify processes in a TPN. The structure of each process gives a natural transformation into an automaton. Then we focus on the timed constraints and show how to equip the automata with clocks, guards and invariants so that the resulting NTA preserves the timed traces (and is in strong distributed timed bisimulation with the initial TPN as well).

We show that in general this transformation is possible only if we allow the automata to read the states of their neighbors (see global state syntax in Subsection 3.2.1), which we interpret as a dependency between the processes, that was hidden in the TPN. Notice also that the decomposition of a PN into components is not always possible. However, we believe that most PNs that model real systems are decomposable. It is also known (see [DE95]) that well-formed free-choice nets are decomposable in strongly connected components.

4.1.1 S-subnets as Processes for Petri Nets

Identifying processes in a TPN is not as immediate as in an NTA. But, 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.

We first define S-subnets as the processes of a Petri net, and the decomposition of a Petri net into S-subnets. Then we show how we can find this decomposition. We borrow the main ideas from [DE95], where the authors give a method (introduced in [Hac72]) to decompose a live and bounded free-choice net into such components and we adapt this method to decompose more general nets.

Decomposition into S-Subnets

Since the notion of process involves only the structure and does not depend on any time property, for the decomposition in processes, we consider only the structure of a PN, i.e. the net (P, T, F) where P is the set of places, T is the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs.

We already mentioned, in Subsection 2.1.2, that S-nets (nets such that $\forall t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$) can be seen as automata (places are locations and transitions are edges). Therefore, we want to decompose a net N into S-nets that cover the net. To do so, we introduce the notion of *S-subnet*.

Definition 22 (S-subnet)**.** A P-closed subnet (P', T', F') of a net $N = (P, T, F)$, with $P' \neq \emptyset$ is an S-subnet of N if it is an S-net.

Note that the notion of S-subnet generalizes the notion of S-component presented in [DE95] (see Definition 3) because we do not impose that the subnet is strongly connected. However, we are looking for *minimal* S-subnets w.r.t. the set inclusion of their generating places, and these S-subnets are always connected (see Proposition 23 below).

A net $N = (P, T, F)$ is *decomposable* iff there exists a set of minimal S-subnets $\{N_1, \ldots, N_n\}$ with $N_i = (P_i, T_i, F_i)$, such that $\bigcup_{i \in [1..n]} P_i = P$. In this case, the set of S-subnets is called a *cover* of N (and $\bigcup_{i \in [1..n]} T_i = T$ because the S-subnets are P-closed).

We are also looking for *minimal covers*, i.e. covers such that if one S-subnet is removed, then the net is no longer covered.

First we state a proposition that relates S-invariants and S-subnets, similarly to the proposition "S-components induce minimal S-invariants" stated in [DE95].

Proposition 23 (Connected S-subnets induce minimal S-invariants)**.**

1. *The characteristic function of the set of places of an S-subnet is an S-invariant.*
2. *The characteristic function of the set of places of a* connected *S-subnet is a minimal S-invariant.*

Proof. The proof from [DE95, p 96] can be adapted. Let $N' = (P', T', F')$ be an S-subnet, and $I : P \to \{0, 1\}$ the characteristic function of P'. Then, for each transition $t \in T$, $|{}^\bullet t \cap P'| = |t^\bullet \cap P'|$ (1 if $t \in T'$, and 0 otherwise), i.e. $\sum_{p \in {}^\bullet t} I(p) = \sum_{p \in t^\bullet} I(p)$, which characterizes an S-invariant.

Now assume N' is connected and there exists a non-zero S-invariant $I_1 \subseteq I$. We prove that $I_1 = I$, which implies that I is minimal. Let p_1 and p_2 be two arbitrary places of N'. Since N' is a connected S-net, there is a path from p_1 to p_2 or a path from p_2 to p_1 in N'. And, as in [DE95, p 96], we use this path to show that $I_1(p_1) = I_1(p_2)$. This means that, either for any place $p \in P'$, $I_1(p) = 0$ (i.e. $I_1 = \mathbf{0}$), or for any place $p \in P'$, $I_1(p) = 1$ (i.e. $I_1 = I$). Since I_1 is non-zero, $I_1 = I$. □

Below we give a necessary and sufficient condition so that a net is decomposable.

Proposition 24. *A net (P,T,F) is decomposable iff there exists a set of minimal S-invariants $\{X_1,\dots X_n\}$ such that*

- $\forall i \in [1..n], X_i : P \to \{0,1\}$, *(1)*
- $\forall i \in [1..n], \forall t \in T, \sum_{p \in {}^\bullet t} X_i(p) \in \{0,1\}$ *(2)*
- $\forall p \in P, \sum_{i \in [1..n]} X_i(p) \geq 1$ *(the set covers the net).* *(3)*

Proof. ($\Rightarrow$) Assume P is decomposable, then there exists a set of n minimal S-subnets $N_i = (P_i, T_i, F_i)$, with $i \in [1..n]$, such that $\bigcup_i P_i = P$. Consider the characteristic function of P_i, $X_i : P \to \{0,1\}$. By Proposition 23, the X_i are minimal S-invariants, and moreover, for any transition t, $\sum_{p \in {}^\bullet t} X_i(p) = |P_i \cap {}^\bullet t|$ equals 1 if $t \in T_i$, and 0 otherwise. Lastly, since each place is in at least one subset of places, for each place p, $\sum_{i \in [1..n]} X_i(p) \geq 1$.

($\Leftarrow$) Assume now that there exists a set of minimal S-invariants $\{X_1,\dots,X_n\}$ which satisfies the three conditions of Proposition 24. We show that the n subnets generated by each $\langle X_i \rangle$ with i in $[1..n]$, are minimal S-subnets that cover N. We denote them by $N_i = (P_i, T_i, F_i)$, with $P_i = \langle X_i \rangle$ and $T_i = {}^\bullet\langle X_i \rangle = \langle X_i \rangle^\bullet$. By construction, N_i is a P-closed subnet of N. N_i is minimal because X_i is minimal. Moreover, since for each place p, $X_i(p) \in \{0,1\}$, $p \in \langle X_i \rangle$ implies that $X_i(p) = 1$, and $p \notin \langle X_i \rangle$ implies that $X_i(p) = 0$. That is, for each transition t, $|{}^\bullet t \cap P_i| = |{}^\bullet t \cap \langle X_i \rangle| = \sum_{p \in {}^\bullet t} X_i(p) = 1$ or 0, from (2). If $t \in T_i = \langle X_i \rangle^\bullet$, then ${}^\bullet t \cap \langle X_i \rangle \neq \emptyset$ and we must have $|{}^\bullet t \cap \langle X_i \rangle| = 1$, i.e. $|t^\bullet \cap P_i| = 1$. Hence N_i is an S-net. Lastly, the n S-subnets cover the net because for each place p, $\sum_{i \in [1..n]} X_i(p) \geq 1$, which implies that there exists i in $[1..n]$ such that $p \in \langle X_i \rangle$, that is $\bigcup_{i \in [1..n]} \langle X_i \rangle = P$. □

Thus, when the net is decomposable, there exists a set $\{X_1,\dots X_n\}$ of minimal S-invariants that is a minimal cover of the net. Such a set gives a decomposition of the net in the S-subnets generated by the minimal S-invariants. Note that this decomposition is not unique and that a place may be shared by several S-subnets, as shown by the examples below.

The number of tokens in an S-subnet is constant. Thus, a connected S-subnet initially marked with one token represents an automaton where the active location is the marked place. Such subnet is called a *process*. If the S-subnet is initially marked with m tokens, then it corresponds to m processes with the same structure but not necessarily starting in the same place, and these processes do not synchronize with each others. To simplify, for now we only consider 1-bounded PNs, but we explain how the procedure can be extended to k-bounded PNs in Subsection 4.3.1. Lastly, notice that the conservation of the number of tokens in each S-subnet implies that unbounded PNs are not decomposable.

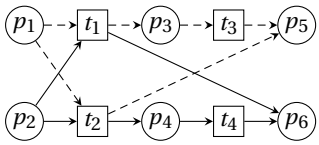
Decomposition Algorithm Some algorithms for the computation of minimal S-invariants can be found in [CS89] where they are called p-semiflows. Therefore, it is possible to compute the set **X** of minimal S-invariants with values in $\{0,1\}$ from a given incidence matrix **N**. Hence, Algorithm 1 below describes how a net can be decomposed. We first determine the minimal S-invariants with values in $\{0,1\}$. We keep only those that generate S-nets (condition 2 of Prop. 24): invariant X is kept if, for any transition t, $\sum_{p\in {}^\bullet t} X(p) \in \{0,1\}$ (1 if t is in the subnet generated by X, and 0 otherwise). If these invariants cover the net (condition 3 of Prop. 24), then the net is decomposable, and we keep a set of S-invariants that forms a minimal cover.

Data: incidence matrix **N**
Result: minimal set **S** of minimal S-subnets that covers the net if the net is decomposable,
empty set otherwise
begin
  S $\leftarrow \emptyset$;
  X $\leftarrow$ set of minimal S-invariants $X : P \rightarrow \{0,1\}$, computed from **N**, s.t. for any $t \in T$, $\sum_{p\in {}^\bullet t} X(p) \in \{0,1\}$;
  if **X** *does not cover the net* **then**
    return **S**;
  endif
  foreach X *in* **X** **do**
    if **X** $\setminus \{X\}$ *covers the net* **then**
      X $\leftarrow$ **X** $\setminus \{X\}$;
    endif
  endfch
  foreach X *in* **X** **do**
    $S \leftarrow$ subnet generated by X;
    S $\leftarrow$ **S** $\cup \{S\}$;
  endfch
  return **S**;
end

Algorithm 1: Decomposition algorithm

Decomposition Examples Below are three examples of decomposition procedure. In Example 1, the net is decomposable, the decomposition is unique and some places belong to several components. In Example 2, the net is decomposable, the decomposition is not unique, and places belong to only one component. Lastly, in Example 3, the net is not decomposable.

Ex 1. The first example was given in Subsection 2.1.2, where we gave the decomposition of the net shown in Fig. 2.4 (which is well-formed and free-choice,



(a) A decomposable net (the different line types denote the arcs of the different components)

	t_1	t_2	t_3	t_4
p_1	−1	−1	0	0
p_2	−1	−1	0	0
p_3	1	0	−1	0
p_4	0	1	0	−1
p_5	0	1	1	0
p_6	1	0	0	1

(b) Its incidence matrix

Fig. 4.1: *A net which is decomposable in S-subnets and its incidence matrix*

and thus decomposable in S-components). Let us apply Algorithm 1 to this example. With the incidence matrix given in Fig. 2.4, we obtain the following minimal S-invariants: X_1 = [1 1 0 0 0 0 0], X_2 = [0 0 1 1 0 1 1], and X_3 = [0 0 1 1 1 0 0]. These S-invariants all generate S-subnets, and they cover the net, therefore the net is decomposable. They also form a minimal cover, therefore they give a decomposition of the net. Hence the net is decomposable in the three S-subnets generated by the sets of places $\{p_1, p_2\}$ (X_1), $\{p_3, p_4, p_6, p_7\}$ (X_2), and $\{p_3, p_4, p_5\}$ (X_3), see Fig. 2.5.

Ex 2. As a second example, we want to decompose the net shown in Fig. 4.1(a). With the incidence matrix given in Fig. 4.1(b), we obtain the following minimal S-invariants: X_1 = [1 0 1 0 1 0], X_2 = [1 0 0 1 0 1], X_3 = [0 1 1 0 1 0] and X_4 = [0 1 0 1 0 1]. These S-invariants all generate S-subnets. The net is covered, therefore decomposable, and there are two minimal covers $\{X_1, X_4\}$ and $\{X_2, X_3\}$, therefore two decompositions. The two components of the decomposition given by $\{X_1, X_4\}$ are denoted in Fig. 4.1(a) by different line types: the arcs of the S-subnet generated by $\{p_1, p_3, p_5\}$ (X_1) are represented by dashed lines, and those of the one generated by $\{p_2, p_4, p_6\}$ (X_4) are represented by plain lines. In the second possible decomposition, p_1 and p_2 are switched.

Ex 3. Consider the net of Fig. 4.2. Any S-subnet N' containing p_2 must also contain its input and output transitions t_1 and t_2 (an S-subnet is P-closed).

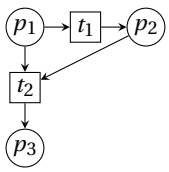


Fig. 4.2: *A non decomposable net*

Then it must contain an input place for t_1 and an output place for t_2 (an S-subnet is an S-net), which are necessarily p_1 and p_3. This means that the only candidate for being a S-subnet containing p_2 is the entire net, but it is not an S-net since t_2 has two input places. This can also be seen by computing the S-invariants from the incidence matrix: there is no non-zero solution with values in $\{0,1\}$ (but there are some with values in $\mathbb{N}$, for example [1 1 2]). Therefore, this net is not decomposable.

Size of the Decomposition

Assume net $N = (P,T,F)$ is decomposable in n connected S-subnets $N_1,\ldots,N_n$, such that $N_i = (P_i,T_i,F_i)$ is the subnet generated by P_i. The number of places in the decomposition is equal to $\sum_{i\in[1..n]}|P_i|$ and is at most $|P|^2$ because a place may be shared by several components and no more than $|P|$ components are needed to cover the net. And the number of transitions is $\sum_{i\in[1..n]}|T_i|$ and is at most $|T|\times|P|$ for the same reason. But these upper bounds are seldom reached since generally there are fewer components and few places and transitions are duplicated in all components.

4.1.2 Translation Procedure

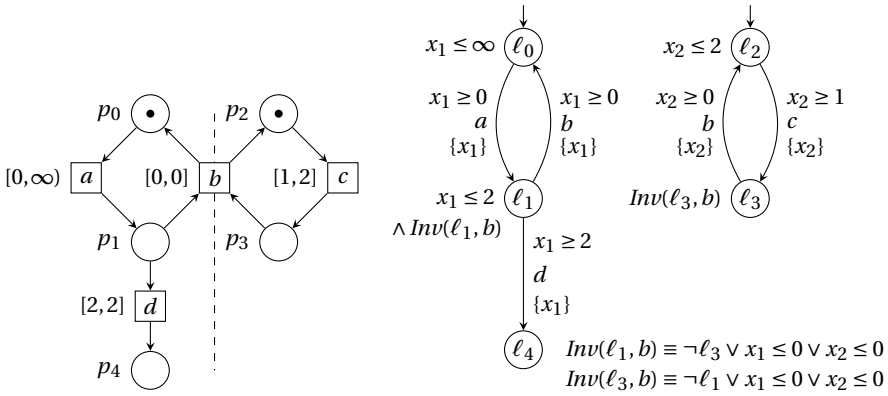
A TPN can be translated in a TA which accepts the same timed words (see marking TA in Fig. 2.12). But we would like to translate it in an NTA which accepts the same timed traces. Below, we propose a structural translation from a TPN to an NTA, based on the decomposition in processes presented above. Therefore, this translation deals with TPN whose untimed support is *decomposable*.

Moreover, in this subsection, we consider only TPN whose untimed support is 1-bounded, in order to simplify the explanation, but the procedure can easily be extended to TPN whose untimed support is k-bounded and still decomposable, as explained in Subsection 4.3.1. In Subsection 4.3.2, we will discuss an extension to deal with bounded TPNs whose untimed support is unbounded and therefore not decomposable.

Procedure

Our procedure translates a TPN $\mathcal{N}$ into an NTA and relies on a decomposition of the untimed support of $\mathcal{N}$ into connected S-subnets (that may be obtained using Algorithm 1). Therefore, our procedure is not (at least directly) applicable if the net is not decomposable. We also require that each S-subnet is initially marked with one token (we discuss the case when S-subnets are not marked, or marked with more than one token in Subsection 4.3.1). In Fig. 4.3(a), we give an example TPN, the dashed line denotes the decomposition of the untimed support.

Each S-subnet determines a process in the time Petri net and will be translated into a timed automaton. We focus now on the treatment of time con-



(a) A TPN and its two processes (b) Resulting NTA

Fig. 4.3: *Translation of the TPN of Fig. (a) into the NTA of Fig. (b)*

straints in order to get a network of timed automata which has the same distributed timed language as $\mathcal{N}$. This involves three 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. Time is added by equipping each automaton with a single clock x_i. This clock is reset on each edge, thus its value gives the time elapsed in the current location. On each edge, if $[a,b]$ is the firing interval of the corresponding transition, we add a guard $x_i \geq a$, and if the transition has only one input place, we add an invariant $x_i \leq b$ on the source location.

3. Lastly, we have to deal with the transitions that have several input places. Such transitions have to fire if they are enabled and their latest firing delay is reached. On our example, see Fig. 4.3(b), we can stay in (ℓ_1, ℓ_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 $lfd(b) = 0$). Thus, we add $Inv(\ell_1, b) \equiv \ell_3 \Rightarrow (x_1 \leq 0 \vee x_2 \leq 0) \equiv \neg\ell_3 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ and $Inv(\ell_3, b) \equiv \ell_1 \Rightarrow (x_1 \leq 0 \vee x_2 \leq 0) \equiv \neg\ell_1 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ in the invariants of ℓ_1 and ℓ_3 (actually we only need to add this "global" invariant to the invariant of one of the source locations concerned by the synchronization).

Formally, a TPN $\mathcal{N} = (P, T, F, M_0, efd, lfd)$ with n processes can be translated in the NTA $A_1 \parallel \cdots \parallel A_n$ with, for all i in $[1..n]$, $A_i = (L_i, \ell_i^0, X, \Sigma_i, E_i, Inv_i)$ where

- $L_i = P_i$ (places of the i^{th} subnet),
- ℓ_i^0 is such that $\{\ell_i^0\} = P_i \cap M_0$,
- $X = \{x_1, \ldots, x_n\}$,

- $\Sigma_i = T_i$ (transitions of the i^{th} subnet),
- E_i is the set of edges (p, g, t, r, p') s.t. $t \in T_i$, $\{p\} = {}^\bullet t \cap P_i$, $\{p'\} = t^\bullet \cap P_i$, $g \equiv x_i \geq efd(t)$, and $r = \{x_i\}$,
- $Inv_i : P_i \to \mathscr{B}(X, P)$ assigns invariants to locations s.t. $\forall p \in P_i$, $Inv_i(p) \equiv \bigwedge_{t \in p^\bullet} Inv(t)$, where $Inv(t) \equiv (\bigwedge_{p' \in {}^\bullet t} p') \Rightarrow \min_{k \in I_t}(x_k) \leq lfd(t)$, with $I_t = \{i \in [1..n] \mid t \in T_i\}$ the set of indices of the subnets that contain t.

That is, $Inv_i(p)$ ensures that we cannot exceed the latest firing delay of an enabled transition which is in the post-set of p. Notice that $Inv_i(p)$ uses the global state syntax presented in Subsection 3.2.1: automaton A_i can read the clocks of the other automata, but cannot reset them and it can also read the current location of the other automata in its invariants.

In the sequel, we first prove that this translation is correct w.r.t. the preservation of the distributed timed language and we discuss the size of the resulting NTA, then we show that the use of the extended syntax is necessary and we identify some cases when the local syntax is sufficient.

Correctness of the Translation

Proposition 25. *The initial decomposable time Petri net $\mathscr{N}$ and the network of timed automata $\mathscr{S}$ which results from the translation are in distributed timed bisimilation.*

Proof. A marking of $\mathscr{N}$ can be identified with a vector of current locations of $\mathscr{S}$. A place may correspond to several locations in the NTA, but in this case, if it is active in one automaton, then it is active in all the automata where it appears. Indeed, for any transition t, any place in $t^\bullet$ is in a component (because the net is covered) and t is also in this component (because the components are P-closed). Therefore, the firing of t in $\mathscr{N}$ corresponds to a synchronization on t in $\mathscr{S}$.

For any i in $[1..n]$, we note $p_i = M \cap P_i$ the location of automaton A_i associated with marking M. We first show the following equivalence:

$$v \models \bigwedge_{1 \leq i \leq n} Inv_i(p_i) \iff \big(\forall t \in T, {}^\bullet t \subseteq M \implies v(t) \leq lfd(t)\big) \tag{4.1}$$

Indeed, by construction, $Inv_i(p_i) \equiv \bigwedge_{t \in p_i^\bullet} \big((\bigwedge_{p \in {}^\bullet t} p) \Rightarrow \min_{k \in I_t}(x_k) \leq lfd(t)\big)$. Thus, $v \models \bigwedge_{1 \leq i \leq n} Inv_i(p_i)$ is equivalent to $\forall t \in T$ s.t. $({}^\bullet t \cap M \neq \emptyset) \wedge ({}^\bullet t \subseteq M)$, $\min_{k \in I_t}\big(v(x_k)\big) \leq lfd(t)$. Then ${}^\bullet t \cap M \neq \emptyset$ can be removed, and by construction, when t is enabled, $v(t) = \min_{k \in I_t}\big(v(x_k)\big)$.

Moreover the guard $g_i(t)$ associated with the edge labeled by t in automaton A_i, is built so that $g_i(t) \equiv x_i \geq efd(t)$, and again, when t is enabled, $v(t) =$

$\min_{i\in I_t}\big(v(x_i)\big)$, which gives:

$$\forall t \in T, {}^\bullet t \subseteq M \implies \left(v \models \bigwedge_{i\in I_t} g_i(t) \iff \nu(t) \geq efd(t) \right) \tag{4.2}$$

Then we define a relation $\mathscr{R}$ between states of $\mathscr{S}$ and states of $\mathscr{N}$ as follows:

$$(M, v)\ \mathscr{R}\ (M, \nu) \iff \left(\forall t \in T, {}^\bullet t \subseteq M \implies \nu(t) = \min_{i\in I_t}\big(v(x_i)\big) \right)$$

Note that $\mathscr{R}$ is not a bijection because the clocks of the automata do not correspond to the clocks of the transitions, and a state of $\mathscr{N}$ may correspond to several states of $\mathscr{S}$. We want to show that $\mathscr{R}$ is a timed bisimulation.

We first observe that $(M_0, v_0)\ \mathscr{R}\ (M_0, \nu_0)$ and we show that, from any correspondent states, $(M, v)\ \mathscr{R}\ (M, \nu)$, the same executions are possible.

Delay step Assume that there exists $d \in \mathbb{R}_{\geq 0}$ such that $(M, v) \xrightarrow{d} (M, v + d)$. Then, $\forall d' \in [0, d], v + d' \models \bigwedge_{1\leq i\leq n} Inv_i(p_i)$. Equation 4.1 implies that $\nu + d'$ is an admissible valuation for marking M, and $(M, v + d)\ \mathscr{R}\ (M, \nu + d)$.

Similarly, if there exists $d \in \mathbb{R}_{\geq 0}$ such that $(M, \nu) \xrightarrow{d} (M, \nu + d)$, then, $(M, v + d)$ is also an admissible state for $\mathscr{S}$ and $(M, v + d)\ \mathscr{R}\ (M, \nu + d)$.

Action step Assume now that there exists an action t such that $(M, v) \xrightarrow{t} (M', v')$, and I_t is the set of indices of the processes that perform t. Then, there exists $e = (e_1, \ldots, e_n) \in (E_1 \cup \{\bullet\}) \times \cdots \times (E_n \cup \{\bullet\})$ s.t. $\forall i \in [1..n]$,

$$\begin{cases} \text{if } i \notin I_t, \text{ then } e_i = \bullet \text{ and } p_i = p'_i \\ \text{otherwise, } e_i = (p_i, g_i, t, r_i, p'_i) \text{ s.t.} \begin{cases} p_i \in {}^\bullet t \wedge p'_i \in t^\bullet, \\ g_i \equiv x_i \geq efd(t), \\ r_i = \{x_i\} \end{cases} \end{cases}$$

and $v \models \bigwedge_{i\in I_t} g_i$, $v' = v[\bigcup_{i\in I_t} r_i]$, and $v' \models \bigwedge_i Inv_i(p'_i)$.

$(M, v)\ \mathscr{R}\ (M, \nu)$ implies that transition t is firable from (M, ν), because it is enabled (${}^\bullet t = \{p_i \mid i \in I_t\}$) and its firing delays are respected (because of (4.1) and (4.2)). This transition leads to state (M'', ν') s.t. $M'' = (M \backslash {}^\bullet t) \cup t^\bullet = M'$, and $\forall t' \in T, \nu'(t') = \begin{cases} 0 & \text{if } \uparrow enabled(t', M, t), \\ \nu(t') & \text{otherwise.} \end{cases}$

By construction, $\forall i \in [1..n], v'(x_i) = 0$ if $i \in I_t$, and $v'(x_i) = v(x_i)$ otherwise. That is, for each transition t', $\min_{i\in I_{t'}}\big(v'(x_i)\big) = 0$ if $I_{t'} \cap I_t \neq \emptyset$ and $\min_{i\in I_{t'}}\big(v'(x_i)\big) = \min_{i\in I_{t'}}\big(v(x_i)\big)$ otherwise.

Then, for each *enabled* transition t', we distinguish two cases:

1. t' is newly enabled by the firing of t from marking M ($\uparrow enabled(t', M, t)$ holds). That means that the last token to enable t' has been created by t, that is, $I_{t'} \cap I_t \neq \emptyset$. Therefore, $\nu'(t') = 0 = \min_{i\in I_{t'}}\big(v'(x_i)\big)$.

2. t' was enabled before the firing of t. That implies $I_{t'} \cap I_t \neq \emptyset$ (because there is one token by process and the tokens in ${}^\bullet t'$ have not been moved by the firing of t). Therefore, $\nu'(t') = \nu(t') = \min_{i \in I'_t}\big(v(x_i)\big) = \min_{i \in I_{t'}}\big(v'(x_i)\big)$.

Therefore, ν' is an admissible valuation for M' and $(M', v') \mathscr{R} (M', \nu')$.

Similarly, if there exists $t \in T$ such that $(M, \nu) \xrightarrow{t} (M', \nu')$, then we can take synchronization t: $(M, v) \xrightarrow{t} (M', v')$, such that this synchronization is shared by the automata whose indices are in I_t, and for any i, $v'(x_i) = 0$ if $i \in I_t$ and $v'(x_i) = v(x_i)$ otherwise. That is, for any transition t', $\min_{i \in I_{t'}}\big(v'(x_i)\big) = 0$ if $I_t \cap I_{t'} \neq \emptyset$, and $\min_{i \in I_{t'}}\big(v'(x_i)\big) = \min_{i \in I_{t'}}\big(v(x_i)\big)$ otherwise. Therefore, if t' is enabled, $\min_{i \in I_{t'}}\big(v'(x_i)\big) = \nu'(t')$, and $(M', v') \mathscr{R} (M', \nu')$.

We have shown that $\mathscr{R}$ is a timed bisimulation between the TTS of $\mathscr{N}$ and $\mathscr{S}$. Moreover, there is a bijection between the processes of $\mathscr{N}$ and those of $\mathscr{S}$ and we have the same distribution of actions between the processes. Therefore, $\mathscr{N}$ and $\mathscr{S}$ are in distributed timed bisimulation and in particular, they accept the same distributed timed language. □

Size of the Network of Timed Automata

Once the decomposition is computed, we directly have the structure of the timed automata. Thus the NTA has at most $|P|^2$ locations and $|T| \times |P|$ edges (see paragraph just before Subsection 4.1.2). The number of edges is exactly $\sum_{t \in T} |I_t|$.

Then, the timing information is provided by as many clocks as processes, that is at most $|P|$ clocks. There is one clock comparison on each edge, because the guards are of the form $x_i \geq \mathit{lfd}(t)$. Moreover, each $\mathit{Inv}(t)$ contains $|I_t|$ clock comparisons (because the min ranges over $|I_t|$ clocks). $\mathit{Inv}(t)$ can be attached only to one of the input places of t because a state is legal as long as the valuation satisfies all the invariants of the current locations, thus, if t is enabled and one of its input places carries $\mathit{Inv}(t)$, $\mathit{lfd}(t)$ cannot be overtaken. Therefore, if we attach each $\mathit{Inv}(t)$ to only one of the input places of t, we have $\sum_{t \in T} |I_t|$ clock comparisons in the invariants. To conclude, the size of the timing information given by the clock comparisons is proportional to the number of edges.

4.2 Know thy Neighbor!

Our translation produces a network of timed automata which accepts the same distributed timed language (and which is timed bisimilar). But we use an extended syntax (see global state syntax in Subsection 3.2.1) in which each automaton can read the state (location and clock) of the other automata. We show that the use of this syntax is necessary.

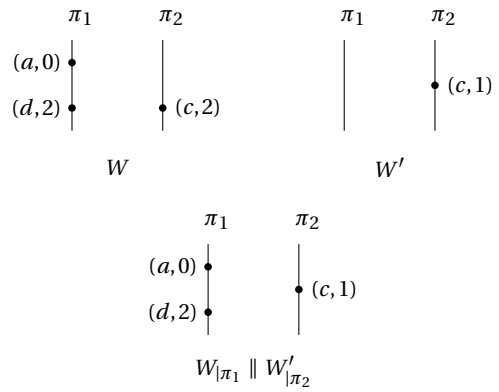


Fig. 4.4: *Two accepted timed traces and one non accepted timed trace for the TPN of Fig. 2.11*

4.2.1 Need for an Extended Syntax

Proposition 26. *Given a TPN $\mathcal{N}$ with its processes, in general, there does not exist any NTA $\mathcal{S}$ using the local syntax such that $\mathcal{N}$ and $\mathcal{S}$ have the same distributed timed language.*

For example, Fig. 4.4 shows two timed traces W and W' representing the beginning of two possible runs, without synchronization, for the TPN $\mathcal{N}$ of Fig. 2.11. Any NTA $\mathcal{S}$ using the local syntax and accepting W and W' would also accept the timed trace built by composing the projection of W onto π_1 and the projection of W' onto π_2 (see Fig. 4.4). But this timed trace is not accepted by $\mathcal{N}$.

We first formalize this in a lemma stated below. Let $\mathcal{S}$ be a network of n TA $A_1 \parallel \cdots \parallel A_n$, we denote by $R_\theta(\mathcal{S})$ the set of all timed traces representing admissible runs of $\mathcal{S}$, without synchronization, and stopping at date θ.

Lemma 27. *Let $\mathcal{S}$ be a network of n timed automata using the local syntax, then, for any timed traces $W_1, \ldots, W_n \in R_\theta(\mathcal{S})$ (not necessarily different), the timed trace defined by $W_{1|\pi_1} \parallel \cdots \parallel W_{n|\pi_n}$ is also in $R_\theta(\mathcal{S})$.*

Proof of Lemma 27. In θ, the automata have not yet synchronized, that is their runs stopping at date θ are independent (because clocks are not shared), and they could have performed any other admissible sequence of actions, stopping at date θ, without synchronization. □

Proof of Proposition 26. Assume that the two automata corresponding to the two processes of the TPN $\mathcal{N}$ of Fig. 4.3(a) are not able to read the current location and the clock of the other automaton. Then, for any two timed traces W and W', representing two admissible runs without synchronization, stopping at date θ, the timed trace $W_{|\pi_1} \parallel W'_{|\pi_2}$ represents also an admissible run.

If we choose, as in Fig. 4.4, $W = (w, proc)$ and $W' = (w', proc)$, with $w = (a,0)(d,2)(c,2)$, $w' = (c,1)$ and $proc = \{(a,\{\pi_1\}),(b,\{\pi_1,\pi_2\}),(c,\{\pi_2\}),(d,\{\pi_1\})\}$

(with $\theta = 2$), then $W_{|\pi_1} \parallel W'_{|\pi_2} = \big((a,0)(c,1)(d,2), proc\big)$ (see Fig. 4.4) should represent an admissible run for $\mathscr{S}$ and $\mathscr{N}$. Which is false because b must be performed immediately after c has been. Therefore, the local syntax (see Subsection 3.2.1) must be extended. □

4.2.2 TPN with Good Decompositional Properties

Proposition 26 states that in general any NTA $\mathscr{S}$, having the same distributed timed language as a given TPN $\mathscr{N}$, uses the global state syntax defined in Subsection 3.2.1, i.e. the automata of $\mathscr{S}$ 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. Still, 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.

We did not find an algorithm that decides in general if TPN $\mathscr{N}$ has this property and we do not know if it is decidable. However, we present a simple sufficient condition, which can be detected by reachability analysis on the marking TA of $\mathscr{N}$. We show how our construction can be easily adapted in this case, to avoid reading information about other automata.

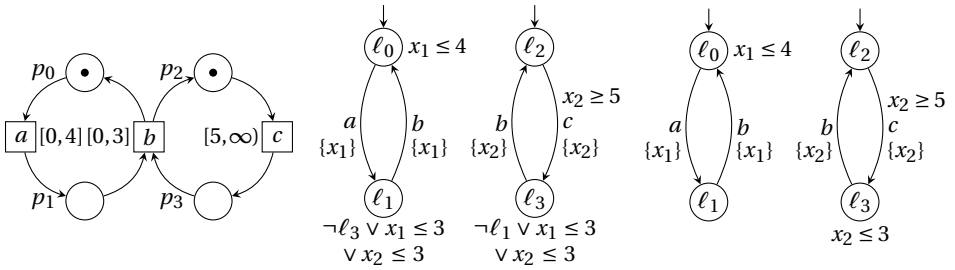
A Class of TPN with Good Decompositional Properties Below, we present a class of TPN that can be decomposed in NTA with the local syntax.

Proposition 28. *Let $\mathscr{N}$ be a 1-bounded TPN which is decomposable, and such that for any transition t, there exists a place p in $^\bullet t$ which is always the last place to be marked among $^\bullet t$ when t becomes enabled, then there exists an NTA $\mathscr{S}$ with the local syntax and with the same distributed timed language as $\mathscr{N}$.*

Proof. We use the same translation as before and choose to add $Inv(t)$ only in $Inv_i(p)$ (this can be done, as explained in the third step of the translation). By construction, $Inv(t) \equiv \big((\bigwedge_{p' \in {}^\bullet t} p') \Rightarrow \min_{k \in I_t}(x_k) \leq lfd(t)\big)$. In this case, $(\bigwedge_{p' \in {}^\bullet t} p')$ is always true in $Inv_i(p)$ – because if p is marked, then all places in $^\bullet t$ are marked – and $\min_{k \in I_t}(v(x_k)) = v(x_i) = v(t)$. Therefore, for any i in $[1..n]$ and for any place p in P_i, $Inv_i(p)$ can be expressed with the local syntax. □

This property can be expressed in CTL [CE81] and checked on the marking TA: for any transition t and for any place $p \in {}^\bullet t$ we check whether the formula $AG(p \in M \Leftrightarrow {}^\bullet t \subseteq M)$ is satisfied (the formula has to hold for at least one place of $^\bullet t$).

For example, consider the TPN of Fig. 4.5(a). Without studying the timing constraints, the translation gives the NTA of Fig. 4.5(b), where the invariants of locations ℓ_1 and ℓ_3 read the state of the other automaton. But when we look at the timing constraints, we can see that location ℓ_1 is always activated before



(a) Initial TPN

(b) Result of the procedure of Subsection 4.1.2

(c) Result of the translation when observing that b is always enabled as soon as p_3 is marked

Fig. 4.5: *A TPN that can be translated in an NTA with the local syntax*

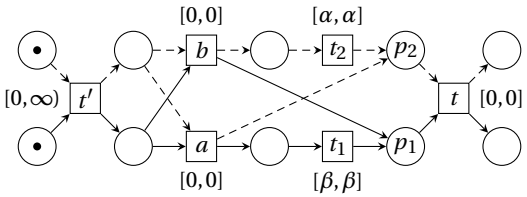
location ℓ_3, i.e. $\ell_3 \Rightarrow \ell_1$, that is b is enabled as soon as ℓ_3 is marked. Therefore, the invariant associated with b can be placed on ℓ_3 only and simplified. Indeed, there is no need to read ℓ_1 since we know it is marked and no need to read x_1 since $\min(x_1, x_2) = x_2$. Eventually, we get the NTA of Fig. 4.5(c).

More Complex Examples We believe that the class of TPN with good decompositional properties that we described above captures most of the practical cases in which one can avoid reading information about other components. The idea is that most often, when there is a variable delay before several components synchronize on a common action, this delay is due to one of the components (which may typically be waiting for some input), while the other components are simply waiting; then the invariant that triggers the synchronization can be associated to the component that is responsible for the delay, and it will not need to read any information about the state of the others: it can assume that the others are ready to synchronize.

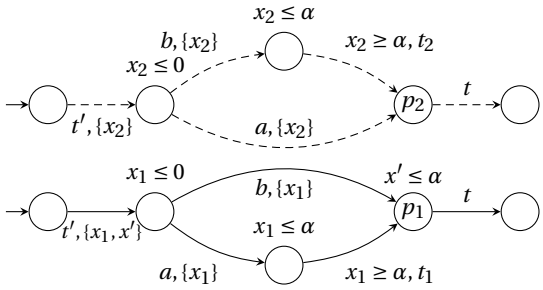
On the other hand, if the delay is really due to several components, then it is very likely that none of the components have enough information locally to be able to trigger the synchronization without reading information about the state of the others. This observation is not always verified: we now show an example of that, but we are convinced that this scenario is not very likely to occur in practice.

Consider the example depicted in Fig. 4.6(a), where α and β are parameters for the values of the constants. This TPN can be decomposed into two components (see the very similar example of Fig. 4.1(a)). These two components will be translated into two automata A_1 (plain lines in the figure), with clock x_1, and A_2 (dashed lines), with clock x_2. Here, after the occurrence of t', either a occurs or b occurs.

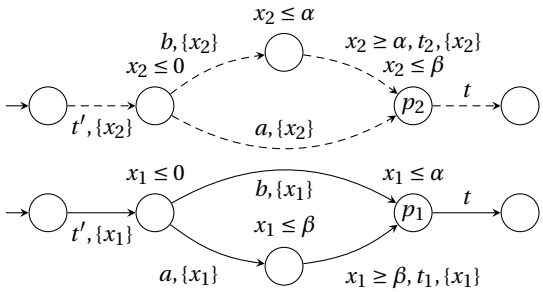
For the first example, we assume that $\alpha = \beta$. Then, regardless of whether a or b occurs, t will be enabled α time units after the firing of t'. Therefore



(a) Initial TPN



(b) $\alpha = \beta$. NTA with the local syntax but one more clock



(c) $\alpha \neq \beta$. NTA with the local syntax

Fig. 4.6: *A TPN that can be translated in an NTA with the local syntax. The arcs of the two components are drawn differently*

a clock x' can be added in one of the automata, reset when t' fires, and used in the invariant of one of the input locations of t as the condition $x' \leq \alpha$ (see Fig. 4.6(b)).

Now, let us assume that $\alpha \neq \beta$. If a occurs, then p_2 is marked immediately, and p_1 is marked β time units later. In this case, p_1 must be disabled immediately and p_2 must be disabled after β time units. If b occurs, then p_1 is marked immediately, and p_2 is marked α time units later. In this case, p_2 must be disabled immediately and p_1 must be disabled after α time units. Therefore, in order to respect the latest firing delay of t, when t is enabled, it suffices to attach $x_2 \leq \beta$ to p_2 and $x_1 \leq \alpha$ to p_1 (see Fig. 4.6(c)).

4.3 Discussion and Extensions

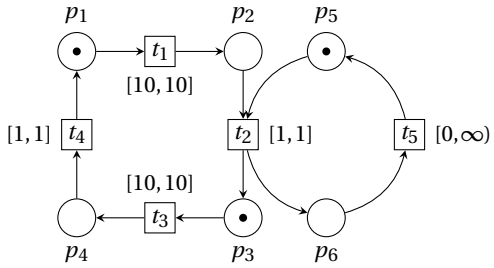
4.3.1 Dealing with Safe TPNs whose Untimed Support is Decomposable and k-Bounded

The translation procedure was given for TPNs whose untimed support is a decomposable PN such that each S-subnet is initially marked with one token, but we argued that decomposable PNs whose S-subnets are not marked or marked with more than one token can also been handled. Here, we use this observation to translate safe TPNs whose untimed support is decomposable and such that the S-subnets may not be marked or be marked with more than one token. Below, we describe the procedure on an example.

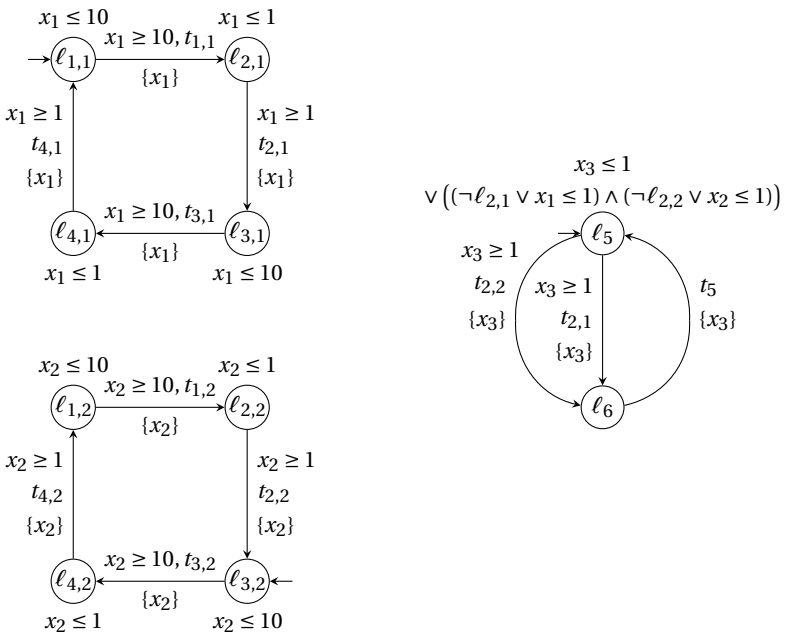
Consider a net such that an S-subnet is initially marked with more than one token. The untimed support of the safe TPN of Fig. 4.7(a) is decomposable into the two S-subnets generated by $\{p_1, p_2, p_3, p_4\}$ and $\{p_5, p_6\}$. Since one S-subnet is initially marked with two tokens, it corresponds to two processes π_1 and π_2 with the same structure. Moreover, since a transition needs only one token in each of its input places to be enabled, π_1 and π_2 need not know the state of each other. That is, each one of them will model the course of one token in the net.

In Fig. 4.7(b), we labeled differently the actions in the first two automata, to denote that they do not synchronize with each other. And since the third process synchronizes on t_2, the edge labeled by t_2 in the associated automaton is duplicated to denote the two possible synchronizations with $t_{2,1}$ and $t_{2,2}$.

Notice that this approach also applies to Petri nets such that an S-subnet N_i is not marked initially (and hence will never be marked). There is no process corresponding to N_i and there will be no corresponding TA. Moreover, for any other S-subnet N_j that shares a transition with N_i, this transition will never fire. This is ensured by the fact that edges are duplicated in as many versions as possible synchronizations: since there is no possible synchronization, there will be no edge denoting this transition in the TA associated with N_j.



(a) Initial TPN with two S-subnets but three processes



(b) Resulting NTA where two automata have the same structure but different initial locations

Fig. 4.7: *A safe TPN whose support is a decomposable PN such that one S-subnet is initially marked with 2 tokens, and its translation into an NTA*

4.3.2 Dealing with Safe TPNs whose Untimed Support is Unbounded

The conservation of the weighted sum of the tokens in an S-invariant (for any reachable marking M, and any S-invariant I, $I \cdot M = I \cdot M_0$, see [DE95]) shows that unbounded PNs are not decomposable. Moreover, not all 1-bounded PNs are decomposable, although we think that, most models of real systems are.

However, our method can be adapted to some temporally 1-bounded TPNs whose untimed support is unbounded. The idea is to modify the underlying unbounded net so that it becomes decomposable and to adapt the timing information in the NTA to preserve the semantics of the original TPN $\mathcal{N}$. We use complementary places: for a place p, the complementary place $\bar{p}$, is built such that ${}^\bullet\bar{p} = p^\bullet \setminus {}^\bullet p$, $\bar{p}^\bullet = {}^\bullet p \setminus p^\bullet$, and $\bar{p}$ is marked iff p is not. For a place p, let the predicate $NC(p)$ denote that p is not covered by any S-subnet, i.e.

$$NC(p) \iff \big(\forall X : P \to \{0,1\}, X \cdot \mathbf{N} = \mathbf{0} \implies X(p) = 0\big).$$

For the TPN of Fig. 4.8(a), this predicate holds for place p_s only.

Then, we can transform the untimed unbounded PN $\mathcal{N}_{untimed} = (P, T, F, M_0)$ into a bounded PN $\mathcal{N}'_{untimed} = (P', T, F', M'_0)$ where

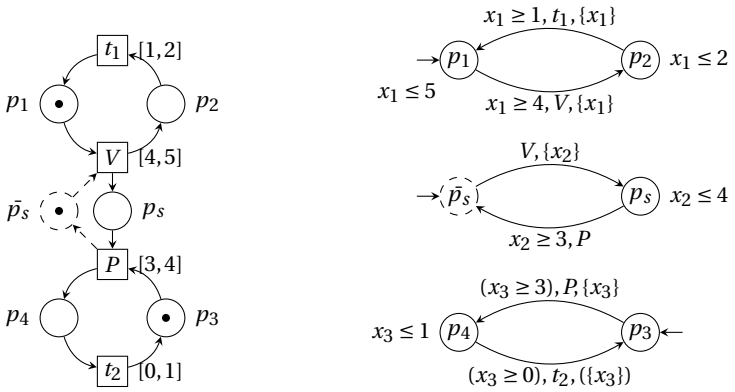
- $P' = P \cup \{\bar{p} \mid NC(p)\}$, i.e. for each place p that is not covered by any S-subnet, a complementary place $\bar{p}$ is added,
- $F' = F \cup \{(\bar{p}, t) \mid NC(p) \wedge (t,p) \in F\} \cup \{(t, \bar{p}) \mid NC(p) \wedge (p,t) \in F\}$,
- $M'_0 = M_0 \cup \{\bar{p} \mid NC(p) \wedge p \notin M_0\}$.

For example, consider the 1-bounded TPN $\mathcal{N}$ of Fig. 4.8(a) without the dashed items (taken from [LR06]). Its untimed support is unbounded, but the timing constraints prevent there being more that one token in p_s. Even though the net is not decomposable without modification, in the structure of the net, we can identify three parts: the S-subnets generated by $\{p_1, p_2\}$, and $\{p_3, p_4\}$ and the subnet generated by $\{p_s\}$ which is not a valid component, because it is not an S-net. Therefore, we add a complementary place to p_s to make the untimed PN 1-bounded, by restricting the number of tokens in place p_s to 1. With this new place, V has to wait for the occurrence of P before occurring again. That is, the boundedness that was ensured by the timing constraints in $\mathcal{N}$, is now ensured in the untimed PN $\mathcal{N}'_{untimed}$ by the complementary places. Notice also that the following proposition holds.

Proposition 29. *A timed run of $\mathcal{N}$ from which the occurrence dates are removed is a run of $\mathcal{N}'_{untimed}$.*

Proof. We define a relation $\mathscr{R}$ which associates a (valid) state (M, ν) of $\mathcal{N}$ with a state M' of $\mathcal{N}'_{untimed}$, and show that $\mathscr{R}$ is a simulation. Namely,

$$(M, \nu)\ \mathscr{R}\ M' \iff M' = M \uplus \{\bar{p} \mid NC(p) \wedge p \notin M\}.$$



(a) A temporally 1-bounded TPN (without $\bar{p}_s$)

(b) Resulting NTA where information between parenthesis can be removed

Fig. 4.8: *Translation of a structurally unbounded TPN*

First, $(M_0, \nu_0)\ \mathscr{R}\ M'_0$ holds. Second, assume that t is firable from state (M, ν) which is $\mathscr{R}$-related to state M'. Then t is also enabled in $M' = M \cup \{\bar{p} \mid NC(p) \wedge p \notin M\}$. Indeed, in $\mathscr{N}'_{untimed}$, if there is a complementary place $\bar{p}$ in the input places of t, then in $\mathscr{N}$, $p \in t^\bullet \setminus {}^\bullet t$, and since the TPN $\mathscr{N}$ is 1-bounded, $p \notin M$ and $\bar{p} \in M'$. We denote by ${}^{\bullet\prime}t$ (resp. $t^{\bullet\prime}$) the pre-set (resp. post-set) of t in $\mathscr{N}'_{untimed}$. By definition of $\mathscr{N}'_{untimed}$, $\bar{p} \in {}^{\bullet\prime}t \iff p \in t^\bullet \setminus {}^\bullet t$ and $\bar{p} \in t^{\bullet\prime} \iff p \in {}^\bullet t \setminus t^\bullet$.

When t fires in $\mathscr{N}$, it leads to state (M_1, ν_1) such that $M_1 = (M \setminus {}^\bullet t) \cup t^\bullet$ (regardless of ν_1). And when t fires in $\mathscr{N}'_{untimed}$, it leads to marking M'_1 defines as follows.

$$\begin{aligned} M'_1 &= (M' \setminus {}^{\bullet\prime}t) \cup t^{\bullet\prime} \\ &= \big((M \uplus \{\bar{p} \mid NC(p) \wedge p \notin M\}) \setminus ({}^\bullet t \uplus \{\bar{p} \mid NC(p) \wedge p \in t^\bullet \setminus {}^\bullet t\})\big) \\ &\quad \cup (t^\bullet \uplus \{\bar{p} \mid NC(p) \wedge p \in {}^\bullet t \setminus t^\bullet\}) \end{aligned}$$

Since $\{\bar{p} \mid NC(p)\}$ is disjoint from M, ${}^\bullet t$ and $t^\bullet$, this can be simplified in

$$M'_1 = \big((M \setminus {}^\bullet t) \cup t^\bullet\big) \uplus \{\bar{p} \mid NC(p) \wedge p \in \big(\overline{M} \setminus (t^\bullet \setminus {}^\bullet t)\big) \cup ({}^\bullet t \setminus t^\bullet)\}$$

and lastly, since $\big(\overline{M} \setminus (t^\bullet \setminus {}^\bullet t)\big) \cup ({}^\bullet t \setminus t^\bullet) = (\overline{M} \cup {}^\bullet t) \setminus t^\bullet = \overline{(M \setminus {}^\bullet t) \cup t^\bullet} = \overline{M_1}$,

$$M'_1 = M_1 \uplus \{\bar{p} \mid NC(p) \wedge p \notin M_1\}$$

Therefore $(M_1, \nu_1)\ \mathscr{R}\ M'_1$. □

But if the timing delays of $\mathscr{N}$ are added to $\mathscr{N}'_{untimed}$, both TPNs will not have the same timed semantics. For instance, on our example, the timed word $(V,4)(t_1,5)(P,7)(t_2,8)(V,9)$ is no longer accepted. However, the transformation is only used to find a decomposition of the net and now our translation can be adapted.

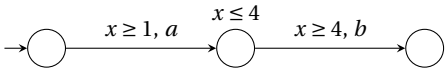


Fig. 4.9: *A TA that cannot be translated in a time S-net with one token*

Proposition 30. *Let $\mathcal{N}$ be a 1-bounded TPN whose untimed support is unbounded. If the net $\mathcal{N}'_{untimed}$ defined above is decomposable, then there exists an NTA with the same distributed timed language as $\mathcal{N}$.*

Proof. If $\mathcal{N}'_{untimed}$ is decomposable, we choose a decomposition such that each $\{p, \bar{p}\}$ forms a component. Then we adapt the translation: each component corresponds to an automaton and the timing information is added in the same way as in Subsection 4.1.2, but without considering the new places because the time spent in these places is not relevant for the semantics of the TPN. That is, for each new place $\bar{p}$, there is no clock reset in the ingoing edges of $\bar{p}$, no guard on the outgoing edges of $\bar{p}$, no invariant on $\bar{p}$, and $\bar{p}$ appears in no invariant. In this way, we get an NTA with the same distributed timed language as the initial TPN. □

In the context of our example, this results in the NTA of Fig. 4.8(b). We decide to attach $Inv(P)$ to p_s, and since we notice that, in $\mathcal{N}$, if p_s is marked, then p_3 is also marked (i.e., in the NTA $\min(x_2, x_3) = x_2$), we simplify this invariant, $Inv(P) \equiv (p_s \wedge p_3) \Rightarrow \min(x_2, x_3) \leq 4$ in $Inv(P) \equiv p_s \Rightarrow x_2 \leq 4$, and therefore $Inv(p_s) \equiv Inv(P) \wedge p_s \equiv x_2 \leq 4$.

4.3.3 Reverse Translation

Let us now consider the reverse translation, i.e. a translation from an NTA to a TPN. There exist translations, for example in [BCH+08], from a TA into a weakly timed bisimilar TPN, but we want to preserve the distributed timed language, that is, when we translate an NTA into a TPN, we want to preserve the mapping between the processes. This implies that we should be able to translate each automaton in a TPN which is an S-net with one token and then compose the obtained nets.

A time S-net with one token is less expressive than a TA with one clock because it can be translated in a TA with one clock which accepts the same timed language. Thus, it is less expressive than a TA with two clocks, according to [HKWT95]. We can even strengthen this by proving that some TA with one clock cannot be translated in finite time S-net with one token (see Prop. 31). Therefore, only a very small class of TA can be translated.

Proposition 31. *Time S-nets with one token are strictly less expressive than TA with one clock.*

Proof. Assume that the TA A of Fig. 4.9 can be translated in a finite time S-net with one token which accepts the same timed language, called $\mathcal{N}$. Then, in $\mathcal{N}$, finitely many states can be reached after having fired an a. We denote these states by $s_i = (\{p_i\}, \mathbf{0})$ with $i \in [1..n]$. The clocks of the enabled transitions have been reset.

Now, assume that we can reach s_i by firing a at some date θ_1. Then, the only possible continuation from s_i is to delay during $d_1 = 4 - \theta_1$ and fire b. That is, (a, θ_1) is the only possible way to reach s_i (otherwise, we would have another possible continuation from s_i).

Therefore, each state s_i can only be reached by executing a at one date θ_i, and from each s_i only one continuation is possible. This implies that $\mathcal{N}$ has a *finite number* of admissible runs whereas A has *infinitely* many. Thus, A cannot be translated in a time S-net with one token. □

If we impose for example that each TA has one clock which is reset on each edge, that the invariant are of the form $x \leq n$ and that the guards are of the form $x \geq m$, then the TA can be translated into time S-nets, but even in this simple case, the composition of these components into a TPN with the same semantics as the initial NTA is not always possible. And in general, the composition of TPNs is not easy and significantly increases the complexity of components, as presented in [PBV11].

4.3.4 Conclusion and Outlook

Usability in Practice Although our translation only works for TPNs whose untimed support is bounded, and does not always give a model in the UPPAAL style (with handshake synchronizations), it generally produces networks with fewer automata than the translation proposed in [CR06], because their translation produces $n+1$ automata for an initial net with n transitions. We also think that our translation gives an NTA which is more readable, since the components are clearly identified, and their structure is close to the original model.

Regarding the number of clocks, we also generally have fewer clocks because we have one clock by process instead of one clock by transition. But as mentioned in [CR06], UPPAAL only considers the active clocks during the verification. In our case, in a given state, all clocks are active and with the translation of [CR06], the number of active clocks is equal to the number of enabled transitions in the corresponding marking ([CR06, Theorem 3]). Therefore, we can have fewer active clocks if there are some conflicts.

Lastly, we have shown an extension of the translation procedure to deal with some bounded TPNs whose support cannot be decomposed. Once we get the structure of the automata, the method that assigns the time constraints can be applied with only some minor modifications.

Towards Identification of Concurrency in Timed Systems This work is a starting point for a more advanced study of concurrency in timed systems. Indeed, concurrency in timed systems involves both causality and the time stamping of events. Transitions that appear as concurrent in an untimed model may not remain independent when time constraints are added. First, time constraints may easily force a temporal ordering between them. But, even worse, the occurrence of a transition may have consequences on apparently concurrent transitions due to time constraints: this is what happens in our TPN of Fig. 4.3(a) where firing c after delay 1 from marking $\{p_1, p_2\}$ prevents d from firing (because it forces b to fire earlier). In our translation, the necessity to allow the automata to read the states of their neighbors highlights these complex dependencies between different processes.

Avoiding Shared Variables

We showed that, in general, we have to allow shared variables in the resulting network of timed automata. Indeed, shared locations can be represented as shared boolean variables that code whether the location is active. However, we consider shared variables as implicit communications that need to be made explicit, in order to implement the model in a distributed architecture. Now, we want to decide, given a network of two timed automata, whether there exists another network of two timed automata, without shared clocks, and such that each individual component has the same behavior as the associated component in the original network. This is the objective of the next chapter.

Chapter 5

Avoiding Shared Clocks in Networks of Timed Automata

Timed automata [AD94] are one of the most famous formal models for real-time systems. Networks of Timed Automata are a natural generalization to model real-time distributed systems. In this formalism, each automaton has a set of clocks that constrain its real-time behavior. But quite often in the literature, the automata 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 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 multi-core architecture is not straightforward since reading clocks a priori requires communications which are not explicitly described in the model.

Here we are concerned with the expressive power of shared clocks according to the distributed nature of the system. We are aware of only one previous study about this aspect, presented in [LPW07]. Our purpose is to identify NTA where sharing clocks could be avoided, i.e. NTA which syntactically use shared clocks, but whose semantics could be achieved by another NTA without shared clocks. To simplify, we look at NTA made of two automata A_1 and A_2 where only A_2 reads clocks reset by A_1. The first step is to formalize which aspect of the semantics we want to preserve in this setting. Then the idea is essentially to detect cases where A_2 can avoid reading a clock because its value does not depend on the actions that are local to A_1 and thus unobservable to A_2. To generalize this idea we have to compute the knowledge of A_2 about the state of A_1. We show that this knowledge is maximized if we allow A_1 to communicate its state to A_2 each time they synchronize on a common action.

In order to formalize our problem we need an appropriate notion of behavioral equivalence between two NTA. We explain why classical comparisons

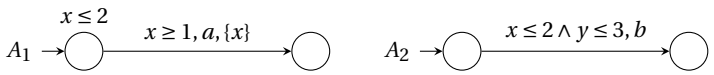


Fig. 5.1: *A_2 could avoid reading clock x which belongs to A_1.*

based on the sequential semantics, like timed bisimulation, are not sufficient here. We need a notion that takes the distributed nature of the system into account. That is, 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. We formalize this idea by the notion of contextual timed transition systems (contextual TTS).

Then we express the problem of avoiding shared clocks in terms of contextual TTS and we give a characterization of the NTA for which shared clocks can be avoided. Finally we effectively construct an NTA without shared clocks with the same behavior as the initial one, when this is possible. A possible interest is to allow a designer to use shared clocks as a high-level feature in a model of a protocol, and rely on our transformation to make it implementable.

Organization of the Chapter This chapter is organized as follows. Section 5.1 presents the problem of avoiding shared clocks on examples and raises the problem of comparing NTA component by component. For this, the notion of contextual TTS is developed in Section 5.2. The problem of avoiding shared clocks is formalized and characterized in terms of contextual TTS. Section 5.3 presents our construction. Then, Section 5.4 discusses some extensions.

5.1 Need for Shared Clocks: Problem Setting

In this chapter, we always assume that the TA we deal with are *non-Zeno*, i.e. they are such that for every infinite timed word w generated by a run, time diverges (i.e. $\delta(w) = \infty$). This is a common assumption for TA.

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.

To start with, let us focus on 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.

A first remark is that our problem makes sense only if we insist on the distributed nature of the system, made of two separate components. On the other hand, if the composition operator is simply used as a convenient syntax for describing a system that is actually implemented on a single sequential component, then a simple product automaton would perfectly describe the system and

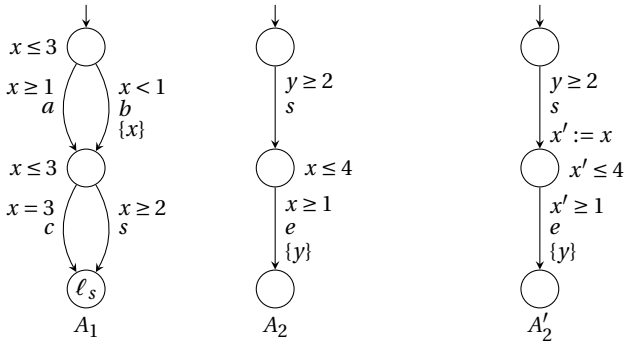


Fig. 5.2: *A_2 reads x which belongs to A_1 and A'_2 does not.*

every clock becomes local.

So, let us consider the example of Fig. 5.1, 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.

5.1.1 Transmitting Information during Synchronizations

Consider now the example of Fig. 5.2. Here also A_2 reads clock x which is reset by A_1, and here also this reading could be avoided. The idea is that A_1 could 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 Fig. 5.2, while preserving the behavior of the NTA, but also the behavior of A_2 w.r.t. A_1.

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.

But we saw that A'_1 also needs to transmit the exact value of its clocks. For

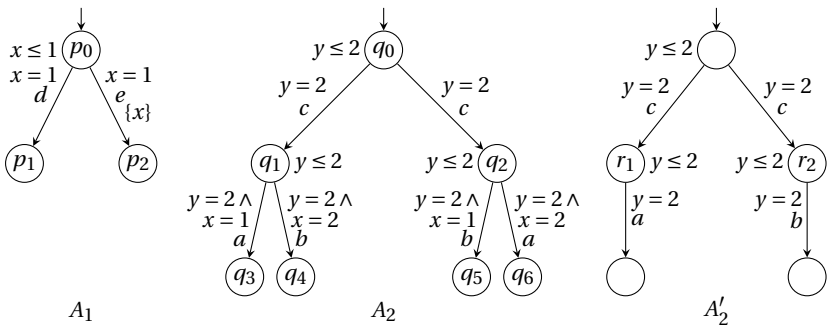


Fig. 5.3: *A_2 needs to read the clocks of A_1 and* $\mathrm{TTS}(A_1 \parallel A_2) \sim \mathrm{TTS}(A_1 \parallel A'_2)$.

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 Fig. 5.2). This is a special case of updatable timed automata, defined in [BDFP04], and recalled in Subsection 3.2.1. Moreover, as shown in [BDFP04], the class we consider, with diagonal-free constraints and updates with equality (they allow other operators) is not more expressive than classical TA for the sequential semantics (any updatable TA of the class is bisimilar to a classical TA), and the emptiness problem is PSPACE-complete.

Semantics $\mathrm{TTS}(A_1 \parallel A_2)$ can be defined as previously, with the difference that the synchronizations are now defined by: $((\ell_1, \ell_2), v) \xrightarrow{a} ((\ell'_1, \ell'_2), v')$ iff $\ell_1 \xrightarrow{g_1, a, r_1}_1 \ell'_1$, $\ell_2 \xrightarrow{g_2, a, r_2, u}_2 \ell'_2$ where u is a partial function from X_2 to X_1, $v \models g_1 \wedge g_2$, $v' = (v[r_1 \cup r_2])[u]$, and $v' \models Inv(\ell'_1) \wedge Inv(\ell'_2)$. The valuation $v[u]$ is defined by $v[u](x) = v(u(x))$ if $u(x)$ is defined, and $v[u](x) = v(x)$ otherwise.

Here, we choose to apply the reset $r_1 \cup r_2$ before the update u, because we are interested in sharing the state reached in A_1 after the synchronization, and r_1 may reset some clocks in $C_1 \subseteq X_1$.

5.1.2 Towards a Formalization of the Problem

We want to know whether A_2 really needs to read the clocks reset by A_1, or if another NTA $A'_1 \parallel A'_2$ could achieve the same behavior as $A_1 \parallel A_2$ without using shared clocks. It remains to formalize what we mean by "having the same behavior" in this context.

First, we impose that the locality of actions is preserved, i.e. A'_1 uses the same set of local actions as A_1, and similarly for A'_2 and A_2. For the synchronizations, we have explained earlier why we allow A'_1 and A'_2 to use a larger synchronization alphabet than A_1 and A_2. The correspondence between both alphabets will be done by a mapping ψ (this point will be refined later).

Now we have to ensure that the behavior is preserved. The first idea that

comes in mind is to impose bisimulation between $\psi(\text{TTS}(A'_1 \parallel A'_2))$ (i.e. $\text{TTS}(A'_1 \parallel A'_2)$ with synchronization actions relabeled by ψ) and $\text{TTS}(A_1 \parallel A_2)$. But this is not sufficient, as illustrated by the example of Fig. 5.3 (where ψ is the identity). Intuitively A_2 needs to read x when in q_1 (and similarly in q_2) at time 2, because this reading determines whether it will perform a or b, and the value of x cannot be inferred from its local state given by q_1 and the value of y. Anyway $\text{TTS}(A_1 \parallel A'_2)$ is bisimilar to $\text{TTS}(A_1 \parallel A_2)$, and A'_2 does not read x. For the bisimulation relation $\mathscr{R}$, it is sufficient to impose $(p_1, q_1)\ \mathscr{R}\ (p_1, r_1)$ and $(p_2, q_1)\ \mathscr{R}\ (p_2, r_2)$.

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 behave the same. As a matter of fact, when A_2 fires one edge labeled by c, it has not read x yet, and there is still a possibility to fire a or b, whereas when A'_2 fires one edge labeled by c, there is no more choice afterwards. Therefore we need a relation between A'_2 and A_2, and in the general case, a relation between A'_1 and A_1 also.

5.2 Contextual Timed Transition Systems

As we are interested in representing a partial view of one of the components, we need to introduce another notion, that we call *contextual timed transition system*. This resembles the powerset construction used in game theory to capture the knowledge of an agent about another agent [Rei84].

Notations $\mathbb{S} = \Sigma_1 \cap \Sigma_2$ denotes the set of common actions. Q_1 denotes the set of states of $\text{TTS}(A_1)$. When $s = ((\ell_1, \ell_2), v)$ is a state of $\text{TTS}(A_1 \parallel A_2)$, we also write $s = (s_1, s_2)$, where $s_1 = (\ell_1, v_{|X_1})$ is in Q_1, and $s_2 = (\ell_2, v_{|X_2 \setminus X_1})$, where $v_{|X}$ is v restricted to X. $\text{TW}_0(\Sigma)$ denotes the set of finite timed ε-words of duration 0 over Σ, i.e. $\text{TW}_0(\Sigma) = \{w \mid \delta(w) = 0 \wedge \lambda(w) \in \Sigma_\varepsilon^*\}$. Lastly, $\mathit{Paths}(\Sigma, d)$ denotes the set of finite paths of duration d over Σ_ε.

Definition 32 ($\text{UR}(s)$)**.** Let $\text{TTS}(A_1) = (Q_1, s_0, \Sigma_1, \to_1)$ and $s \in Q_1$. The set of states of A_1 reachable from s by local actions in 0 delay (and therefore not observable by A_2) is denoted by $\text{UR}(s) = \{s' \in Q_1 \mid \exists w \in \text{TW}_0(\Sigma_1 \setminus \Sigma_2) : s \xrightarrow{w}_1 s'\}$.

5.2.1 Contextual TTS

Contextual States

The states of this contextual TTS are called *contextual states*. They can be regarded as possibly infinite sets of states of $\text{TTS}(A_1 \parallel A_2)$ for which A_2 is in the same location and has the same valuation over $X_2 \setminus X_1$. A_2 may not be able to distinguish between some states (s_1, s_2) and (s'_1, s_2). In $\text{TTS}_{A_1}(A_2)$, these states are grouped into the same contextual state. Since we are interested in the case where $X_2 \cap X_1 \neq \emptyset$, it may happen that A_2 is able to perform a local action or delay from (s_1, s_2) and not from (s'_1, s_2), even if these states are grouped in a same contextual state.

Definition 33 (Contextual TTS)**.** Let $\mathrm{TTS}(A_1 \parallel A_2) = (Q, q_0, \Sigma_1 \cup \Sigma_2, \Rightarrow)$. Then, the *TTS of A_2 in the context of A_1*, denoted by $\mathrm{TTS}_{A_1}(A_2)$, is the TTS $(S, s_0, (\Sigma_2 \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1), \rightarrow)$, where

- $S = \{(S_1, s_2) \mid \forall s_1 \in S_1, (s_1, s_2) \in Q\}$,
- $s_0 = (S_1^0, s_2^0)$, s.t. $(s_1^0, s_2^0) = q_0$ and $S_1^0 = \mathrm{UR}(s_1^0)$,
- $\rightarrow$ is defined by
  - Local action: for any $a \in \Sigma_{2,\varepsilon} \setminus \mathbb{S}$, $(S_1, s_2) \xrightarrow{a} (S_1', s_2')$ iff $\exists s_1 \in S_1 : (s_1, s_2) \overset{a}{\Rightarrow} (s_1, s_2')$, and $S_1' = \{s_1 \in S_1 \mid (s_1, s_2) \overset{a}{\Rightarrow} (s_1, s_2')\}$
  - Synchronization: for any $(a, s_1') \in \mathbb{S} \times Q_1$, $(S_1, s_2) \xrightarrow{a, s_1'} (\mathrm{UR}(s_1'), s_2')$ iff $\exists s_1 \in S_1 : (s_1, s_2) \overset{a}{\Rightarrow} (s_1', s_2')$
  - Local delay: for any $d \in \mathbb{R}_{\geq 0}$, $(S_1, s_2) \xrightarrow{d} (S_1', s_2')$ iff $\exists s_1 \in S_1, \rho \in \mathit{Paths}(\Sigma_1 \setminus \Sigma_2, d) : (s_1, s_2) \overset{\rho}{\Rightarrow} (s_1', s_2')$, and $S_1' = \{s_1' \mid \exists s_1 \in S_1, \rho \in \mathit{Paths}(\Sigma_1 \setminus \Sigma_2, d) : (s_1, s_2) \overset{\rho}{\Rightarrow} (s_1', s_2')\}$

For example, consider A_1 and A_2 of Fig. 5.3. The initial state is $\big(\{(p_0, 0)\}, (q_0, 0)\big)$. From this contextual state, it is possible to delay 2 time units and reach the contextual state $\big(\{(p_1, 2), (p_2, 1)\}, (q_0, 2)\big)$. Indeed, during this delay, A_1 has to perform either e and reset x, or d. Now, from this contextual state, we can take an edge labeled by c, and reach $\big(\{(p_1, 2), (p_2, 1)\}, (q_1, 2)\big)$. Lastly, from this new state, a can be fired, because it is enabled by $((p_2, 1), (q_1, 2))$ in the TTS of the NTA, and the reached contextual state is $\big(\{(p_2, 1)\}, (q_3, 2)\big)$.

Unrestricted Contextual TTS

We say that there is no restriction in $\mathrm{TTS}_{A_1}(A_2)$ 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 above, there is a restriction in $\mathrm{TTS}_{A_1}(A_2)$ because we have seen that a is enabled only by $((p_2, 1), (q_1, 2))$, and not by all states merged in $\big(\{(p_1, 2), (p_2, 1)\}, (q_1, 2)\big)$. Formally, we use the predicate $\mathit{noRestriction}_{A_1}(A_2)$ defined as follows.

Definition 34 ($\mathit{noRestriction}_{A_1}(A_2)$)**.** The predicate $\mathit{noRestriction}_{A_1}(A_2)$ holds iff for any reachable state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$, both

- $\forall a \in \Sigma_{2,\varepsilon} \setminus \mathbb{S}, (S_1, s_2) \xrightarrow{a} (S_1', s_2') \iff \forall s_1 \in S_1, (s_1, s_2) \overset{a}{\Rightarrow} (s_1, s_2')$, and
- $\forall d \in \mathbb{R}_{\geq 0}, (S_1, s_2) \xrightarrow{d} \iff \forall s_1 \in S_1, \exists \rho \in \mathit{Paths}(\Sigma_1 \setminus \Sigma_2, d) : (s_1, s_2) \overset{\rho}{\Rightarrow}$

Remark 35. If A_2 does not read X_1, then $\mathit{noRestriction}_{A_1}(A_2)$.

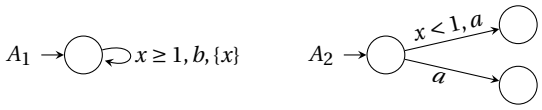


Fig. 5.4: $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2) \approx \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$*, although there is a restriction in* $\mathrm{TTS}_{A_1}(A_2)$

Sharing of Information on the Synchronizations Later we assume that during a synchronization, A_1 is allowed to transmit all its state to A_2, that is why, in $\mathrm{TTS}_{A_1}(A_2)$, we distinguish the states reached after a synchronization according to the state reached in A_1. We also label the synchronization edges by a pair $(a, s_1) \in \mathbb{S} \times Q_1$ where a is the action and s_1 the state reached in A_1.

For the sequel, let $\mathrm{TTS}_{Q_1}(A_1)$ (resp. $\mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$) denote $\mathrm{TTS}(A_1)$ (resp. $\mathrm{TTS}(A_1 \parallel A_2)$) where the synchronization edges are labeled by (a, s_1), where $a \in \mathbb{S}$ is the action, and s_1 is the state reached in A_1.

First remark the following.

Remark 36. Let $A_1 \parallel A_2$ be such that $X_1 \cap X_2 = \emptyset$. Then $\mathrm{TTS}(A_1) \otimes \mathrm{TTS}(A_2)$ is isomorphic to $\mathrm{TTS}(A_1 \parallel A_2)$. This is not true in general when $X_1 \cap X_2 \neq \emptyset$. For example, in Fig. 5.2, taking b at time 0.5 and e at time 1 is possible in $\mathrm{TTS}(A_1) \otimes \mathrm{TTS}(A_2)$ but not in $\mathrm{TTS}(A_1 \parallel A_2)$, since b resets x which is tested by e.

We can now state a nice property of unrestricted contextual TTS that is similar to the distributivity of TTS over the composition when considering TA with disjoint sets of clocks (as stated in the above remark). We say that a TA is *deterministic* if it has no ε-transition and for any location ℓ and action a, there is at most one edge labeled by a from ℓ.

Lemma 37. *If there is no restriction in* $\mathrm{TTS}_{A_1}(A_2)$*, then* $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2) \approx \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$*. Moreover, when* A_2 *is deterministic, this condition becomes necessary.*

The example of Fig. 5.4 shows that the reciprocal does not hold when A_2 is not deterministic. In order to prove Lemma 37, we first present two propositions. The first one relates the reachable states of $\mathrm{TTS}_{A_1}(A_2)$ with those of $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$.

Proposition 38.

1. *For any reachable state* (S_1, s_2) *of* $\mathrm{TTS}_{A_1}(A_2)$*,*
   $s_1 \in S_1 \implies (s_1, (S_1, s_2))$ *is a reachable state of* $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$
2. $noRestriction_{A_1}(A_2)$ *iff*
   for any reachable state (S_1, s_2) *of* $\mathrm{TTS}_{A_1}(A_2)$*,*
   $s_1 \in S_1 \iff (s_1, (S_1, s_2))$ *is a reachable state of* $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$

Proof. (1) For any reachable state (S_1, s_2), let us denote by $P(S_1, s_2)$ the fact that for any $s_1 \in S_1$, $(s_1, (S_1, s_2))$ is reachable in $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$. We give a recursive proof. First, the initial state (S_1^0, s_2^0) satisfies $P(S_1^0, s_2^0)$ because for any $s_1 \in S_1^0 = \mathrm{UR}(s_1^0)$, $\exists w \in \mathrm{TW}_0(\Sigma_1 \setminus \Sigma_2) : s_1^0 \xrightarrow{w}_1 s_1$ and hence $(s_1^0, (S_1^0, s_2^0)) \xrightarrow{w} (s_1, (S_1^0, s_2^0))$. Then, assume some reachable state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$ satisfies $P(S_1, s_2)$ and show that any state (S_1', s_2') reachable in one step from (S_1, s_2) also satisfies $P(S_1', s_2')$. There can be three kinds of steps from (S_1, s_2) in $\mathrm{TTS}_{A_1}(A_2)$.

1. If for some $a \in \Sigma_{2,\varepsilon} \setminus \mathbb{S}$, $(S_1, s_2) \xrightarrow{a} (S_1', s_2')$, then for any $s_1' \in S_1' \subseteq S_1$, $(s_1', (S_1, s_2)) \xrightarrow{a} (s_1', (S_1', s_2'))$, i.e. $P(S_1', s_2')$ holds.

2. If for some $(a, s_1') \in \mathbb{S} \times Q_1$, $(S_1, s_2) \xrightarrow{a, s_1'} (S_1', s_2')$, then $S_1' = \mathrm{UR}(s_1')$, and for some $s_1 \in S_1$, $(s_1, (S_1, s_2)) \xrightarrow{a, s_1'} (s_1', (S_1', s_2'))$. By the same reasoning as for (S_1^0, s_2^0), for any $s_1'' \in S_1' = \mathrm{UR}(s_1')$, $\exists w \in \mathrm{TW}_0(\Sigma_1 \setminus \Sigma_2) : (s_1', (S_1', s_2')) \xrightarrow{w} (s_1'', (S_1', s_2'))$. Hence $P(S_1', s_2')$ holds.

3. If for some $d \in \mathbb{R}_{\geq 0}$, $(S_1, s_2) \xrightarrow{d} (S_1', s_2')$, then $\exists d_1 \leq d : (S_1, s_2) \xrightarrow{d_1} (S_1^1, s_2^1) \wedge \exists s_1^1 \in S_1^1, s_1 \in S_1 : (s_1, s_2) \overset{d_1}{\Rightarrow} (s_1^1, s_2^1)$, that is $(s_1^1, (S_1^1, s_2^1))$ is reachable, and by time-determinism, $(S_1^1, s_2^1) \xrightarrow{d-d_1} (S_1', s_2')$.

For the third case, take d_1 small enough (but strictly positive) so that $S_1^1 = \{s_1' \mid \exists s_1 \in S_1 : (s_1, s_2) \overset{d_1}{\Rightarrow} (s_1^1, s_2^1) \wedge s_1' \in \mathrm{UR}(s_1^1)\}$. That is, after some local actions that take no time, A_1 is able to perform a delay d_1 during which no local action is enabled (such d_1 exists because of the non-zenoness assumption). With such d_1, any state $s_1' \in S_1^1$ is such that $s_1' \in \mathrm{UR}(s_1^1)$ for some s_1^1 so that $(s_1^1, (S_1^1, s_2^1))$ is reachable. Therefore, $\exists w \in \mathrm{TW}_0(\Sigma_1 \setminus \Sigma_2) : (s_1^1, (S_1^1, s_2^1)) \xrightarrow{w} (s_1', (S_1^1, s_2^1))$ and hence $P(S_1^1, s_2^1)$ holds.

Since A_1 is not Zeno, any delay in $\mathrm{TTS}_{A_1}(A_2)$ can be cut into a finite number of such smaller global delays. Hence, for any (S_1, s_2) that satisfies $P(S_1, s_2)$, for any $d \in \mathbb{R}_{\geq 0}$ such that $(S_1, s_2) \xrightarrow{d} (S_1', s_2')$, $P(S_1', s_2')$ holds.

(2, $\Rightarrow$) (1) already gives that $\forall s_1 \in S_1$, $(s_1, (S_1, s_2))$ is a reachable state. So it remains to prove that, when $\mathit{noRestriction}_{A_1}(A_2)$, if $(s_1, (S_1, s_2))$ is a reachable state, then $s_1 \in S_1$. We say that a reachable state $s = (s_1, (S_1, s_2))$ satisfies $P(s)$ iff $s_1 \in S_1$.

Assume $\mathit{noRestriction}_{A_1}(A_2)$ and $s = (s_1, (S_1, s_2)$ is a reachable state that satisfies $P(s)$. Then, any state s' reachable in one step from s by some local action or delay $a \in (\Sigma_{1,\varepsilon} \cup \Sigma_{2,\varepsilon}) \setminus \mathbb{S} \cup \mathbb{R}_{\geq 0}$ or by some synchronization $(a, s_1') \in \mathbb{S} \times Q_1$ matches one of the following cases.

- if $a \in \Sigma_{1,\varepsilon} \setminus \Sigma_2$ is a local action in A_1, then $s' = (s_1', (S_1, s_2))$ such that $s_1' \in \mathrm{UR}(s_1) \subseteq S_1$ (by construction, $s_1 \in S_1 \implies \mathrm{UR}(s_1) \subseteq S_1$),

- if $a \in \Sigma_{2,\varepsilon} \setminus \Sigma_1$ is a local action in A_2, then $s' = (s_1, (S_1, s_2'))$,

- if $a \in \mathbb{R}_{\geq 0}$, then $s' = (s'_1, (S'_1, s'_2))$, where s'_1 such that $(s_1, s_2) \stackrel{a}{\Rightarrow} (s'_1, s'_2)$ is in $S'_1 = \{q'_1 \mid \exists q_1 \in S_1, \rho \in Paths(\Sigma_1 \setminus \Sigma_2, a) : (q_1, s_2) \stackrel{\rho}{\Rightarrow} (q'_1, s'_2)\}$,
- if $(a, s'_1) \in (\mathbb{S} \times Q_1)$, then $s' = (s'_1, (\mathrm{UR}(s'_1), s'_2))$.

Therefore, any state s' reached in one step from s also satisfies $P(s')$, and recursively, since the initial state $s_0 = (s^0_1, (\mathrm{UR}(s^0_1), s^0_2))$ satisfies $P(s_0)$, any reachable state s of $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$ satisfies $P(s)$.

$(2, \Leftarrow)$ By contradiction, assume there is a restriction in state (S_1, s_2) for local delay or action $a \in (\Sigma_{2,\varepsilon} \setminus \Sigma_1) \cup \mathbb{R}_{\geq 0}$ i.e. a is possible from some state (s'_1, s_2) but not from another state (s_1, s_2) such that $s'_1, s_1 \in S_1$. Then, after performing a from $(s_1, (S_1, s_2))$, that is reachable according to Proposition 38, we reach state $(s_1, (S'_1, s'_2))$ such that $s_1 \notin S'_1$. □

Proposition 39. *If $noRestriction_{A_1}(A_2)$ then, for any timed ε-word w over $(\Sigma_2 \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1)$, there exists at most one S_1 such that, for some s_2, $(S^0_1, s^0_2) \xrightarrow{w} (S_1, s_2)$ in $\mathrm{TTS}_{A_1}(A_2)$ (i.e. S_1 is uniquely determined by w, whatever the structure of A_2).*

Proof. Assume $noRestriction_{A_1}(A_2)$, we show that, for any (S^1_1, s^1_2) reachable in $\mathrm{TTS}_{A_1}(A_2)$, for any local action, synchronization, or delay in $(\Sigma_{2,\varepsilon} \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1) \cup \mathbb{R}_{\geq 0}$, there is at most one S_1 such that, for some s_2, (S_1, s_2) is a successor of (S^1_1, s^1_2) by this action.

Indeed, by construction, and since there is no restriction,

- any successor of (S^1_1, s^1_2) by a local action is of the form (S^1_1, s'_2),
- any successor of (S^1_1, s^1_2) by a synchronization (a, s'_1) is of the form $(\mathrm{UR}(s'_1), s'_2)$,
- any successor of (S^1_1, s^1_2) by a delay d is of the form (S_1, s'_2) with $S_1 = \{s'_1 \mid \exists \rho \in Paths(\Sigma_1 \setminus \Sigma_2, d), s_1 \in S^1_1 : s_1 \xrightarrow{\rho}_1 s'_1\}$.

Therefore, for any possible action or delay, S_1 does not depend on the state of A_2, and is uniquely determined by this action or delay.

Since (S^0_1, s^0_2) is unique, for any timed ε-word w over $(\Sigma_2 \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1)$, either w does not describe a valid path in $\mathrm{TTS}_{A_1}(A_2)$, or there exists a unique S_1 such that for some s_2, $(S^0_1, s^0_2) \xrightarrow{w} (S_1, s_2)$ in $\mathrm{TTS}_{A_1}(A_2)$. □

We can now prove Lemma 37.

Proof of Lemma 37. Assume $noRestriction_{A_1}(A_2)$, and define relation $\mathscr{R}$ as $(s_1, (S_1, s_2)) \mathscr{R} (s'_1, s'_2) \stackrel{def}{\iff} s_1 = s'_1 \wedge s_2 = s'_2$, for any reachable states $(s_1, (S_1, s_2))$ of $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$ and (s'_1, s'_2) of $\mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$. By Proposition 38, since $(s_1, (S_1, s_2))$ is reachable, $s_1 \in S_1$. We show that $\mathscr{R}$ is a strong timed bisimulation.

First, the initial states are $\mathscr{R}$-related: $(s^0_1, (S^0_1, s^0_2)) \mathscr{R} (s^0_1, s^0_2)$. Then, if $(s_1, (S_1, s_2)) \mathscr{R} (s'_1, s'_2)$, four kinds of steps are possible:

- if for some $a \in \Sigma_{1,\varepsilon} \setminus \Sigma_2$, $(s_1,(S_1,s_2)) \xrightarrow{a} (s_1',(S_1,s_2))$, then $(s_1,s_2) \overset{a}{\Rightarrow} (s_1',s_2)$ and $(s_1',(S_1,s_2)) \mathscr{R} (s_1',s_2)$, and conversely.

- if for some $a \in \Sigma_{2,\varepsilon} \setminus \Sigma_1$, $(s_1,(S_1,s_2)) \xrightarrow{a} (s_1,(S_1,s_2'))$, then, $\forall s_{11} \in S_1$, $(s_{11},s_2) \overset{a}{\Rightarrow} (s_{11},s_2')$ (because $noRestriction_{A_1}(A_2)$), and in particular, $(s_1,s_2) \overset{a}{\Rightarrow} (s_1,s_2')$ and $(s_1,(S_1,s_2')) \mathscr{R} (s_1,s_2')$, and conversely.

- if for some $(a,s_1') \in \mathbb{S} \times Q_1$, $(s_1,(S_1,s_2)) \xrightarrow{a,s_1'} (s_1',(S_1',s_2'))$, then $(s_1,s_2) \overset{a,s_1'}{\Longrightarrow} (s_1',s_2')$ and $(s_1',(S_1',s_2')) \mathscr{R} (s_1',s_2')$, and conversely.

- if for some $d \in \mathbb{R}_{\geq 0}$, $(s_1,(S_1,s_2)) \xrightarrow{d} (s_1',(S_1',s_2'))$, then $(s_1,s_2) \overset{d}{\Rightarrow} (s_1',s_2')$ (because $noRestriction_{A_1}(A_2)$), and $(s_1',(S_1',s_2')) \mathscr{R} (s_1',s_2')$, and conversely.

Now assume A_2 is deterministic. Let relation $\mathscr{R}$ be a strong timed bisimulation between $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$ and $\mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$.

By contradiction, assume there is a restriction in $\mathrm{TTS}_{A_1}(A_2)$. Then there is a reachable state (S_1,s_2) of $\mathrm{TTS}_{A_1}(A_2)$, and a local delay or action $a \in (\Sigma_2 \setminus \Sigma_1) \cup \mathbb{R}_{\geq 0}$ such that, for some $s_1,s_1' \in S_1$, (s_1,s_2) enables a in $\mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$, whereas (s_1',s_2) does not.

By definition of a bisimulation, there also exist two states $(p_1,(P_1,p_2))$ and $(p_1',(P_1',p_2'))$ such that $(p_1,(P_1,p_2)) \mathscr{R} (s_1,s_2)$ and $(p_1',(P_1',p_2')) \mathscr{R} (s_1',s_2)$. That is, in particular, $(p_1',(P_1',p_2'))$ does not enable a. Moreover, these states can be chosen so that they are reached by the same timed word over $(\Sigma_2 \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1)$, and since A_2 is deterministic, $p_2 = p_2' = s_2$.

Now, we can assume that (S_1,s_2) is chosen so that it is the first state with a restriction along an initial path. Then, the paths to (P_1,s_2) and (P_1',s_2) generate the same timed word over $(\Sigma_2 \setminus \mathbb{S}) \cup (\mathbb{S} \times Q_1)$, and by Proposition 39, $P_1 = P_1' = S_1$.

Therefore, we have shown the existence of a state $(p_1',(S_1,s_2))$ in $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$ that does not enable a, which means that (S_1,s_2) does not enable a in $\mathrm{TTS}_{A_1}(A_2)$. This contradicts the fact that there exists $s_1 \in S_1$ such that (s_1,s_2) enables a. □

We are now in condition to formalize our problem.

5.2.2 Need for Shared Clocks Revisited

We have argued in Section 5.1.2 that the existence of a NTA $A_1' \parallel A_2'$ without shared clocks and such that $\psi(\mathrm{TTS}_{Q_1'}(A_1' \parallel A_2')) \sim \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$ is not sufficient to capture the idea that A_2 does not need to read the clocks of A_1. We are now equipped to define the relations we want to impose on the separate components, namely $\psi(\mathrm{TTS}_{Q_1'}(A_1')) \sim \mathrm{TTS}_{Q_1}(A_1)$ and $\psi(\mathrm{TTS}_{A_1'}(A_2')) \sim \mathrm{TTS}_{A_1}(A_2)$. And since we have seen the importance of labeling the synchronization actions in contextual TTS by labels in $\mathbb{S} \times Q_1$ rather than in $\mathbb{S}$, the correspondence between

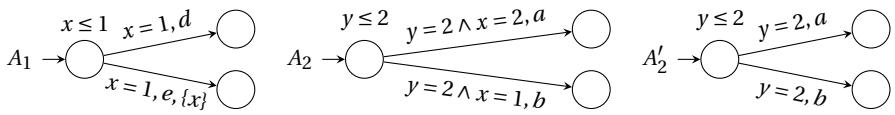


Fig. 5.5: *A_2 needs to read the clocks of A_1 and* $\mathrm{TTS}_{A_1}(A_2) \sim \mathrm{TTS}_{A_1}(A'_2)$.

the synchronization labels of $A'_1 \parallel A'_2$ with those of $A_1 \parallel A_2$ is now done by a mapping $\psi : \mathbb{S}' \times Q'_1 \to \mathbb{S} \times Q_1$.

This settles the problem of the example of Fig. 5.3 where $\mathrm{TTS}_{A_1}(A'_2) \not\sim \mathrm{TTS}_{A_1}(A_2)$ (here $A'_1 = A_1$), but as shown in Fig. 5.5, a problem remains. In this example, we can see that A_2 needs to read clock x of A_1 to know whether it has to perform a or b at time 2, and yet $\mathrm{TTS}_{A_1}(A_2) \sim \mathrm{TTS}_{A_1}(A'_2)$ (here also $A'_1 = A_1$). The intuition to understand this is that the contextual TTS merge too many states for the two systems to remain differentiable. However we remark that here, the first condition that we have required in Section 5.1, namely the global bisimulation between $\psi(\mathrm{TTS}(A'_1 \parallel A'_2))$ and $\mathrm{TTS}(A_1 \parallel A_2)$, does not hold.

Formalization

Now we show that the conjunction of global and local bisimulations actually gives the good definition.

Definition 40 (Need for shared clocks)**.** Given $A_1 \parallel A_2$ such that A_1 does not read the clocks of A_2, *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), using the same sets of local actions and a synchronization alphabet $\mathbb{S}'$ related to the original one by a mapping $\psi : \mathbb{S}' \times Q'_1 \to \mathbb{S} \times Q_1$, and such that

1. $\psi(\mathrm{TTS}_{Q'_1}(A'_1 \parallel A'_2)) \sim \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$ and
2. $\psi(\mathrm{TTS}_{Q'_1}(A'_1)) \sim \mathrm{TTS}_{Q_1}(A_1)$ and
3. $\psi(\mathrm{TTS}_{A'_1}(A'_2)) \sim \mathrm{TTS}_{A_1}(A_2)$.

Notice that this does not mean that the clock constraints that read X_1 can simply be removed from A_2 (see Fig. 5.2).

Lemma 41. *When $\mathit{noRestriction}_{A_1}(A_2)$ holds, any NTA $A'_1 \parallel A'_2$ without shared clocks and that satisfies items 2 and 3 of Definition 40 also satisfies item 1.*

Proof. When $\mathit{noRestriction}_{A_1}(A_2)$ holds, then by Lemma 37, $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2) \approx \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$. So for any NTA $A'_1 \parallel A'_2$ satisfying items 2 and 3 of Definition 40, we have $\psi(\mathrm{TTS}_{Q'_1}(A'_1)) \otimes \psi(\mathrm{TTS}_{A'_1}(A'_2)) \sim \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$. It remains to show that $\psi(\mathrm{TTS}_{Q'_1}(A'_1 \parallel A'_2)) \approx \psi(\mathrm{TTS}_{Q'_1}(A'_1)) \otimes \psi(\mathrm{TTS}_{A'_1}(A'_2))$. Remark that applying ψ to the labels before doing the product allows more synchronizations than applying ψ on the TTS of the system since ψ may merge different labels. We show that, in our case, the two resulting TTS are bisimilar anyway.

For this, let $\mathscr{R}_1$ be a bisimulation relation between $\psi(\mathrm{TTS}_{Q_1'}(A_1'))$ and $\mathrm{TTS}_{Q_1}(A_1)$, and $\mathscr{R}_2$ be a bisimulation relation between $\psi(\mathrm{TTS}_{A_1'}(A_2'))$ and $\mathrm{TTS}_{A_1}(A_2)$. We will build inductively a bisimulation $\mathscr{R}$ between $\psi(\mathrm{TTS}_{Q_1'}(A_1' \parallel A_2'))$ and $\psi(\mathrm{TTS}_{Q_1'}(A_1')) \otimes \psi(\mathrm{TTS}_{A_1'}(A_2'))$ such that for any (q_1,q_2) and (r_1,r_2) such that $(q_1,q_2)\ \mathscr{R}\ (r_1,r_2)$, there exists a state s_1 of $\mathrm{TTS}_{Q_1}(A_1)$ and a state s_2 of $\mathrm{TTS}_{A_1}(A_2)$ such that $q_1\ \mathscr{R}_1\ s_1$ and $r_1\ \mathscr{R}_1\ s_1$ and $q_2\ \mathscr{R}_2\ s_2$ and $r_2\ \mathscr{R}_2\ s_2$. The inductive definition of $\mathscr{R}$ is as follows. The initial states (which are the same in both sides) are in relation; $\mathscr{R}$ is preserved by delays; $\mathscr{R}$ is preserved by playing local actions. The key is the treatment of synchronizations: when $(q_1,q_2)\ \mathscr{R}\ (r_1,r_2)$ and $q_1 \xrightarrow{a_1} q_1'$ in $\mathrm{TTS}_{Q_1}(A_1)$ and $q_2 \xrightarrow{a_2} q_2'$ in $\mathrm{TTS}_{A_1}(A_2)$ with $\psi(a_1)=\psi(a_2)=a$, then the existence of the s_1 and s_2 mentioned earlier ensures that there exists a state (r_1',r_2') in $\psi(\mathrm{TTS}_{Q_1'}(A_1' \parallel A_2'))$ such that $(r_1,r_2) \xrightarrow{a} (r_1',r_2')$, and we set $(q_1',q_2')\ \mathscr{R}\ (r_1',r_2')$ for any such (r_1',r_2').

□

A Criterion to Decide the Need for Shared Clocks

We are now ready to give a criterion to decide whether shared clocks are necessary.

Theorem 42. *When $\mathrm{noRestriction}_{A_1}(A_2)$ holds, A_2 does not need to read the clocks of A_1. When A_2 is deterministic, this condition becomes necessary.*

Proof of Theorem 42, necessary condition when A_2 is deterministic. Like in the proof of Lemma 41, we show that for any NTA $A_1' \parallel A_2'$ satisfying items 2 and 3 of Definition 40, $\psi(\mathrm{TTS}_{Q_1'}(A_1' \parallel A_2')) \sim \mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$. But, by Lemma 37, when A_2 is deterministic and $\mathrm{TTS}_{A_1}(A_2)$ has restrictions, $\mathrm{TTS}_{Q_1}(A_1) \otimes \mathrm{TTS}_{A_1}(A_2)$ is not timed bisimilar to $\mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$ (not even weakly timed bisimilar since there are no ε-transitions). Hence any NTA $A_1' \parallel A_2'$ satisfying items 2 and 3 of Definition 40, does not satisfy item 1. □

We remark from the proof that when there is a restriction in $\mathrm{TTS}_{A_1}(A_2)$, even infinite A_1' and A_2' would not help. Next section will be devoted to the constructive proof of the direct part of this theorem.

The counterexample in Fig. 5.4 also works here to argue that the conditions of Lemma 41 and Theorem 42 are not necessary when A_2 is not deterministic. Indeed A_2' with only one unguarded edge labeled by a and $A_1' = A_1$ satisfy the three items of Definition 40 but there is a restriction in $\mathrm{TTS}_{A_1}(A_2)$.

5.3 Constructing a Network of Timed Automata without Shared Clocks

This section is dedicated to proving Theorem 42 by constructing suitable A_1' and A_2'. To simplify, we assume that in A_2, the guards on the synchronizations do not read X_1.

5.3.1 Construction

First, our A'_1 is obtained from A_1 by replacing all the labels $a \in \mathbb{S}$ on the synchronization edges of A_1 by $(a, \ell_1) \in \mathbb{S} \times L_1$, where ℓ_1 is the output location of the edge. Therefore the synchronization alphabet between A'_1 and A'_2 will be $\mathbb{S}' = \mathbb{S} \times L_1$, which allows A'_1 to transmit its location after each synchronization.

Then, the idea is to build A'_2 as a product $A_{1,2} \otimes A_{2,mod}$, 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. For this, as long as the automata do not synchronize, $A_{1,2}$ will evolve, simulating a run of A'_1 that is compatible with what A'_2 knows about A'_1. And, as soon as A'_1 synchronizes with A'_2, A'_2 updates $A_{1,2}$ to the actual state of A'_1. 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 $A_1 \parallel A_2$. To check that this is the case, 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. Therefore, if ☹ is not reachable, they can be removed so that A'_2 does not read the clocks of A'_1. More precisely, a contradiction happens when $A_{2,mod}$ is in a given location and the guard of an outgoing edge is true according to $A_{1,2}$ and false according to A'_1, or vice versa, or when the invariant of the current location is false according to A'_1 (whereas it is true according to $A_{1,2}$, since $A_{2,mod}$ reads the clocks of $A_{1,2}$).

Namely, $\mathscr{S}_{mod} = A'_1 \parallel (A_{1,2} \otimes A_{2,mod})$ where $A_{1,2}$ and $A_{2,mod}$ are defined as follows. $A_{1,2} = (L_1, \ell_1^0, X'_1, \mathbb{S}', E'_1, Inv'_1)$, where

- each clock $x' \in X'_1$ is associated with a clock $c(x') = x \in X_1$ (c is a bijection from X'_1 to X_1). For any clock constraint γ, γ' denotes the clock constraint where any clock x of X_1 is substituted by x' of X'_1.
- $\forall \ell \in L_1, Inv'_1(\ell) = Inv_1(\ell)'$
- $E'_1 = \{\ell_1 \xrightarrow{g', \varepsilon_a, r'} \ell_2 \mid \exists a \in \Sigma_{1,\varepsilon} \setminus \Sigma_2 : \ell_1 \xrightarrow{g,a,c(r')} \ell_2 \in E_1\}$
  $\cup \{\ell \xrightarrow{\mathbf{tt},(a,\ell_2),c} \ell_2 \mid \ell \in L_1 \wedge a \in \mathbb{S} \wedge \exists \ell_1 \xrightarrow{g,a,r} \ell_2 \in E_1\}$
  where c denotes the assignment of any clock $x' \in X'_1$ with the value of its associated clock $c(x') = x \in X_1$ (written $x' := x$ in Fig. 5.6).

$A_{2,mod} = (L_2 \cup \{☹\}, \ell_2^0, X_2 \cup X'_1 \cup X_1, (\Sigma_2 \setminus \Sigma_1) \cup \mathbb{S}', E'_2, Inv'_2)$, where

- $\forall \ell \in L_2, Inv'_2(\ell) = Inv_2(\ell)'$ and $Inv'_2(☹) = \mathbf{tt}$,
- $E'_2 = \{\ell_1 \xrightarrow{g',a,r} \ell_2 \mid \ell_1 \xrightarrow{g,a,r} \ell_2 \in E_2 \wedge a \notin \mathbb{S}\}$
  $\cup \{\ell_1 \xrightarrow{g,(a,\ell),r} \ell_2 \mid \ell_1 \xrightarrow{g,a,r} \ell_2 \in E_2 \wedge a \in \mathbb{S} \wedge \ell \in L_1\}$
  $\cup \{\ell \xrightarrow{\neg Inv_2(\ell),\varepsilon,\emptyset} ☹ \mid \ell \in L_2\}$
  $\cup \{\ell \xrightarrow{g' \wedge \neg g,\varepsilon,\emptyset} ☹ \mid \ell \xrightarrow{g,a,r} \ell' \in E_2 \wedge a \notin \mathbb{S}\}$
  $\cup \{\ell \xrightarrow{\neg g' \wedge g,\varepsilon,\emptyset} ☹ \mid \ell \xrightarrow{g,a,r} \ell' \in E_2 \wedge a \notin \mathbb{S}\}$.

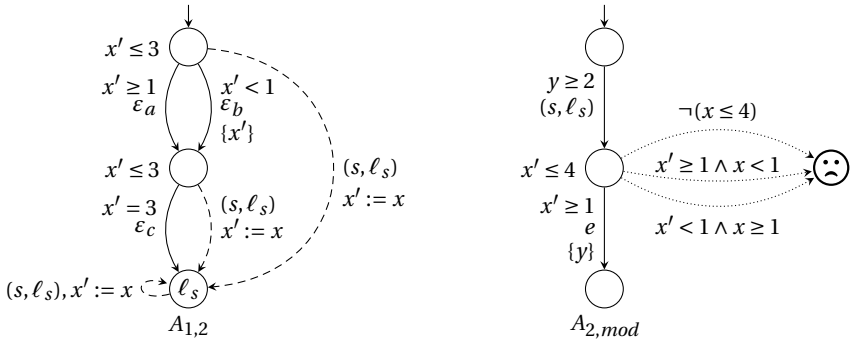


Fig. 5.6: *$A_{1,2}$ and $A_{2,mod}$ for the example of Fig. 5.2*

For the example of Fig. 5.2, $A_{1,2}$ and $A_{2,mod}$ are pictured in Fig. 5.6.

We now prove the correspondence between a state of $\mathscr{S}_{mod}$ and two states of $\mathrm{TTS}(A_1 \parallel A_2)$ that are merged into the same state of $\mathrm{TTS}_{A_1}(A_2)$. This is stated in the following proposition. A state of $\mathscr{S}_{mod}$ is denoted as $(s_1, s_{1,2}, s_2) = \big((\ell_1, v_{|X_1}), (\ell_{1,2}, v_{|X'_1}), (\ell_2, v_{|X_2 \setminus X_1})\big)$. For a given state of $A_{1,2}$, $s_{1,2} = (\ell_{1,2}, v_{|X'_1})$, we denote by $s'_{1,2}$ the state $(\ell_{1,2}, v')$, where $v' : X_1 \to \mathbb{R}_{\geq 0}$ is defined as: for any $x \in X_1$, $v'(x) = v(x')$ (i.e. $s'_{1,2}$ is a state of A_1). Reciprocally, for a given state of A_1, $s'_{1,2} = (\ell_{1,2}, v')$, $s_{1,2}$ denotes the state $(\ell_{1,2}, v)$, where $v : X'_1 \to \mathbb{R}_{\geq 0}$ is defined as: for any $x' \in X'_1$, $v(x') = v'(x)$.

Proposition 43. *Let $(s_1, s_{1,2}, s_2)$ be a state of $\mathscr{S}_{mod}$. If along one path that leads to $(s_1, s_{1,2}, s_2)$ no edge leading to ☹ is enabled, then there exists S_1 such that (S_1, s_2) is a reachable state of $\mathrm{TTS}_{A_1}(A_2)$ and s_1 and $s'_{1,2}$ are both in S_1.*

Conversely, let (S_1, s_2) be a reachable state of $\mathrm{TTS}_{A_1}(A_2)$, and s_1 and $s'_{1,2}$ be some states in S_1. Then $(s_1, s_{1,2}, s_2)$ is a state of $\mathscr{S}_{mod}$.

Proof. Let $(s_1, s_{1,2}, s_2)$ be a reachable state of $\mathscr{S}_{mod}$, such that there is a path ρ from the initial state $(s^0_1, s^0_{1,2}, s^0_2)$ to $(s_1, s_{1,2}, s_2)$ that does not enable any edges leading to ☹ (except maybe from $(s_1, s_{1,2}, s_2)$). We give a recursive proof. First, for the initial state $(s^0_1, s^0_{1,2}, s^0_2)$ of $\mathscr{S}_{mod}$, s^0_1 and $s^{0\prime}_{1,2}$ are both in S^0_1 such that (S^0_1, s^0_2) is the initial state of $\mathrm{TTS}_{A_1}(A_2)$. Now, assume this is true for some $(p_1, p_{1,2}, p_2)$ visited along ρ. That is, there exists P_1 such that (P_1, p_2) is reachable and $p_1, p'_{1,2} \in P_1$. Then, the next state s' visited along ρ is reached after one of the following steps:

- local action in A'_1: $s' = (q_1, p_{1,2}, p_2)$ such that $q_1 \in \mathrm{UR}(p_1) \subseteq P_1$,
- local action in $A_{1,2}$: $s' = (p_1, q_{1,2}, p_2)$ such that $q'_{1,2} \in \mathrm{UR}(p'_{1,2}) \subseteq P_1$,
- local action in A_2: $s' = (p_1, p_{1,2}, q_2)$ such that there exists S'_1 such that (S'_1, q_2) is reachable from (P_1, q_2) by the same action, and, since no edge leading to ☹ is enabled, both (p_1, p_2) and $(p'_{1,2}, p_2)$ enable this step in $\mathrm{TTS}(A_1 \parallel A_2)$. Therefore, $p_1, p'_{1,2} \in S'_1$.

- synchronization: $s' = (q_1, q_{1,2}, q_2)$ such that there exists $S'_1 = \mathrm{UR}(q_1)$ such that (S'_1, q_2) is reachable from (P_1, q_2) by the same action, and $q_1 = q'_{1,2} \in S'_1$.

By recursion, $(s_1, s_{1,2}, s_2)$ also satisfies the property, that is, there exists S_1 such that (S_1, s_2) is reachable and $s_1, s'_{1,2} \in S_1$.

Conversely, let denote by $P(S_1, s_2)$ the fact that for any reachable state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$, for any states $s_1, s'_{1,2} \in S_1$, $(s_1, s_{1,2}, s_2)$ is a reachable state of $\mathscr{S}_{mod}$. First, for any $s_1, s'_{1,2} \in S^0_1 = \mathrm{UR}(s^0_1)$, $(s_1, s_{1,2}, s^0_2)$ is a reachable state, because by construction, $A_{1,2}$ can only mimic (as long as there is no synchronization) one possible behavior of A_1 to reach $s_{1,2}$ from s^0_1, therefore $P(S^0_1, s^0_2)$ holds. Assume that for some reachable state (S_1, s_2) $P(S_1, s_2)$ holds. Then any state reachable in one step from (S_1, s_2) is reached by one of the following steps.

- If for some $a \in \Sigma_{2,\varepsilon} \setminus \mathbb{S}$, $(S_1, s_2) \xrightarrow{a} (S'_1, s'_2)$, then for any $s_1, s'_{1,2} \in S'_1 \subseteq S_1$, $(s_1, s'_{1,2}, s_2) \xrightarrow{a} (s_1, s'_{1,2}, s'_2)$, i.e. $P(S'_1, s'_2)$ holds.

- If for some $(a, s'_1) \in \mathbb{S} \times Q_1$, $(S_1, s_2) \xrightarrow{a, s'_1} (S'_1, s'_2)$, then $S'_1 = \mathrm{UR}(s'_1)$, and for any $s_1, s'_{1,2} \in S'_1$, $(s_1, s_{1,2}, s'_2)$ can be reached from some $(p_1, p_{1,2}, s_2)$ such that $p_1, p'_{1,2} \in S_1$. Indeed, in $\mathscr{S}_{mod}$, synchronization $((a, \ell'_1), s'_1)$ resets $A_{1,2}$ in the same state as A_1 and then A_1 performs some local actions while $A_{1,2}$ also performs some local actions mimicking one possible behavior of A_1 (that is why $s'_{1,2} \in S'_1$). Hence $P(S'_1, s'_2)$ holds.

- If for some $d \in \mathbb{R}_{\geq 0}$, $(S_1, s_2) \xrightarrow{d} (S'_1, s'_2)$, then we use the same reasoning as for a synchronization. Since $A_{1,2}$ is built so that it mimics any possible behavior of A_1 between synchronizations, any state $s'_{1,2} \in S'_1$ reachable by A_1 during this delay corresponds to a state $s_{1,2}$ reachable by $A_{1,2}$. Hence $P(S'_1, s'_2)$ also holds.

By recursion, $P(S_1, s_2)$ holds for any reachable state (S_1, s_2). □

Lastly, the following lemma will be used to prove the direct part of Theorem 42.

Lemma 44. ☹ *is reachable in* $\mathscr{S}_{mod}$ *iff there is a restriction in* $\mathrm{TTS}_{A_1}(A_2)$.

Proof. Assume ☹ is not reachable in $\mathscr{S}_{mod}$. From Proposition 43, we know that for any state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$, for any s_1, $s'_{1,2}$ in S_1, there is a corresponding state $s = \big((\ell_1, v_{|X_1}), (\ell_{1,2}, v_{|X'_1}), (\ell_2, v_{|X_2 \setminus X_1})\big) = (s_1, s_{1,2}, s_2)$ of $\mathscr{S}_{mod}$. Moreover, for any such s, if there is an outgoing edge towards ☹ from ℓ_2, then this edge is never enabled. That is, for any time constraint γ read in ℓ_2 in the original system $\mathscr{S}$ (invariant of ℓ_2 or guard of an outgoing edge with a local action), $v_{|X_2 \cup X_1} \models \gamma \iff v_{|(X_2 \setminus X_1) \cup X'_1} \models \gamma'$. Hence for any enabled step from (S_1, s_2), s_1 and $s'_{1,2}$ are in the same restriction. Therefore, $\mathit{noRestriction}_{A_1}(A_2)$.

Assume ☹ is reachable in $\mathscr{S}_{mod}$. From Proposition 43, we know that for any state $s = \big((\ell_1, v_{|X_1}), (\ell_{1,2}, v_{|X_1'}), (\ell_2, v_{|X_2 \setminus X_1})\big) = (s_1, s_{1,2}, s_2)$ of $\mathscr{S}_{mod}$, reached after a path that does not enable edges leading to ☹ (except maybe from this last state), there is a corresponding state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$ such that s_1 and $s'_{1,2}$ are both in S_1. If ☹ can be reached, then consider a path that reaches ☹ and such that no edge leading to ☹ was enabled before along the path. The last state s of $\mathscr{S}_{mod}$ visited before ☹ is such that for some time constraint γ evaluated at s from ℓ_2, $v_{|X_2 \cup X_1} \models \gamma$ and $v_{|(X_2 \setminus X_1) \cup X_1'} \not\models \gamma'$ (or conversely). Therefore, a local action or local delay is possible from (s_1, s_2) and not from $(s'_{1,2}, s_2)$. Hence (S_1, s_2) is a state with a restriction. □

We now give a first simple case for which Theorem 42 can be proved easily. We say that A_1 has no urgent synchronization if for any location, when the invariant reaches its limit, a local action is enabled. Under this assumption, we can show that $A'_2 = A_{1,2} \otimes A'_{2,mod}$, where $A'_{2,mod}$ is $A_{2,mod}$ without location ☹ (that is never reached according to Lemma 44) and its ingoing edges, is suitable. Indeed, we can show that A'_2 does not read X_1 and is such that $\psi(\mathrm{TTS}_{A'_1}(A'_2)) \sim \mathrm{TTS}_{A_1}(A_2)$, where for any $((a, \ell_1), s_1) \in \mathbb{S}' \times Q'_1$, $\psi(((a, \ell_1), s_1)) = (a, s_1)$. Obviously, item 2 of Definition 40 holds, and Lemma 41 says that item 1 also holds.

When A_1 has urgent synchronizations, this construction allows one to check the absence of restriction in $\mathrm{TTS}_{A_1}(A_2)$, but it does not give directly a suitable A'_2. We define the construction of A'_2 for the general case in Subsection 5.3.3.

Proof of Theorem 42, direct part, when no urgent synchronization in A_1.
Assume $noRestriction_{A_1}(A_2)$. We consider $A'_2 = A_{1,2} \otimes A'_{2,mod}$ where $A'_{2,mod}$ is $A_{2,mod}$ without ☹ (that is never reached according to Lemma 44) and its ingoing edges. Therefore, $A'_{2,mod}$ does not read X_1 and neither does $A'_2 = A_{1,2} \otimes A'_{2,mod}$. Below we show that A'_2 is a suitable candidate because $\psi(\mathrm{TTS}_{A'_1}(A'_2)) \sim \mathrm{TTS}_{A_1}(A_2)$ ($\psi(\mathrm{TTS}_{Q'_1}(A'_1)) \sim \mathrm{TTS}_{Q_1}(A_1)$ obviously holds).

Let $\mathscr{R}$ be the relation such that for any reachable state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$, and any reachable state (S'_1, s'_2) of $\psi(\mathrm{TTS}_{A'_1}(A'_2))$,

$$(S_1, s_2)\ \mathscr{R}\ (S'_1, s'_2) \overset{def}{\iff} \begin{cases} s_2 = (\ell_2, v_2) \text{ and } s'_2 = ((\ell_{1,2}, \ell_2), v'_2) \text{ s.t.} \\ \forall x \in X_2 \setminus X_1,\ v_2(x) = v'_2(x) \\ S_1 = S'_1 \end{cases}$$

i.e. A_2 and $A'_{2,mod}$ are both in ℓ_2 and their local clocks have the same value, and A_1 and A'_1 are in indistinguishable states (states merged in a same contextual state S_1). Obviously, the initial states, (S_1^0, s_2^0) and $(S_1^0, s_2^{0\prime})$, are $\mathscr{R}$-related. Since there is no marked state in $\mathrm{TTS}_{A_1}(A_2)$ (resp. in $\mathrm{TTS}_{A'_1}(A'_2)$), for any state $s = (S_1, s_2)$ (resp. $s' = (S'_1, s'_2)$) of this TTS, all time constraints read by automaton 2 in ℓ_2 (invariant of ℓ_2 and guards of the outgoing edges) have the same truth value for all the states (s_1, s_2) such that $s_1 \in S_1$ (resp. $s_1 \in S'_1$). In the sequel, we say that valuation V of s (resp. V' of s') satisfies constraint g, when the valua-

tions of all states (s_1, s_2) in s (resp. in s') satisfy g. Assume now that for some reachable states (S_1, s_2) and (S'_1, s'_2), $(S_1, s_2) \mathscr{R} (S'_1, s'_2)$.

Local Action If $a \in \Sigma_{2,\varepsilon} \setminus \Sigma_1$ is enabled from (S_1, s_2), then, there is an associated edge in A_2, $\ell_2 \xrightarrow{g,a,r} p_2$ such that guard g is satisfied by V. Let g' be the guard on the corresponding outgoing edge $(\ell_{1,2}, \ell_2) \xrightarrow{g',a,r} (\ell_{1,2}, p_2)$ in A'_2. g uses clocks in X_2, and by construction, g' has the same form but with clocks in $(X_2 \setminus X_1) \uplus X'_1$. $(S_1, s_2) \mathscr{R} (S'_1, s'_2)$ says that v_2 and v'_2 coincide on $X_2 \setminus X_1$, and since ☹ is never reached in $\mathscr{S}_{mod}$, V satisfies the constraints of g on X_1 iff V' satisfies the constraints of g' on X'_1. That is, $V \models g \iff V' \models g'$. Therefore A'_2 can also perform a from (S_1, s'_2) and the states reached in both systems are $\mathscr{R}$-related: $(S_1, q_2) \mathscr{R} (S_1, q'_2)$, because $q_2 = (p_2, v_2[r])$ and $q'_2 = ((\ell_{1,2}, p_2), v'_2[r])$. This also holds reciprocally.

Synchronization Assume for some $(a, s'_1) \in \mathbb{S} \times Q_1$, $(S_1, s_2) \xrightarrow{a,s'_1} (S'_1, q_2)$. That is, there is an edge $\ell_2 \xrightarrow{g_2,a,r_2} p_2$ in A_2 such that $v_2 \models g_2$ and $q_2 = (p_2, v_2[r_2])$ and, for some $(\ell_1, v_1) \in S_1$, an edge $\ell_1 \xrightarrow{g_1,a,r_1} p_1$ in A_1 such that $v_1 \models g_1$ and $s'_1 = (p_1, v_1[r_1]) \in S'_1$. Hence, synchronization $((a, p_1), s'_1)$ is also enabled from state (S_1, s'_2) because $A_{2,mod}$ is in the same location as A_2, and has the same clock values over $X_2 \setminus X_1$, and A'_1 is also in some state of S_1, therefore, there is also the same state $(\ell_1, v_1) \in S_1$ which enables (a, p_1). We do not consider $A_{1,2}$ because it is always ready to synchronize. Moreover, the state reached in $\psi(\mathrm{TTS}_{A'_1}(A'_2))$ after this synchronization is (S'_1, q'_2) such that $(S'_1, q_2) \mathscr{R} (S'_1, q'_2)$, because $q_2 = (p_2, v_2[r_2])$ and $q'_2 = \big((p_{1,2}, p_2), (v'_2[r_2])[c]\big)$ where c denotes the copy of the clocks of X_1 into their associated clocks of X'_1 and therefore c modifies only clocks that we do not consider in relation $\mathscr{R}$, and $r_2 \subseteq C_2 \subseteq (X_2 \setminus X_1)$ resets the same clocks in both systems. And reciprocally.

Local Delay Assume for some $d \in \mathbb{R}_{\geq 0}$, $(S_1, s_2) \xrightarrow{d} (S'_1, q_2)$. Then, $V + d \models Inv_2(\ell_2)$, and since ☹ is never reached in $\mathscr{S}_{mod}$, $V + d \models Inv_2(\ell_2) \iff V' + d \models Inv'_2(\ell_2)$. That is, the same delay is enabled from (S_1, s'_2) while $A_{1,2}$ may perform some local steps: $(S_1, s'_2)(\xrightarrow{g_0,\varepsilon,r_0})^* \xrightarrow{d_0} (\xrightarrow{g_n,\varepsilon,r_n})^* \dots \xrightarrow{d_n} (S''_1, q'_2)$, where $\sum_{i=0}^{n} d_i = d$, g_i is a guard over X'_1 and r_i is a reset included in X'_1. This works because we assumed that A_1 has no urgent synchronization (and so does A'_1). Therefore, $A_{1,2}$ cannot force a synchronization.

Reciprocally, if we can perform a delay d from (S_1, s'_2), then $V' + d \models Inv'_2(\ell_2) \wedge Inv'_1(\ell_{1,2})$. And since $V + d \models Inv_2(\ell_2) \iff V' + d \models Inv'_2(\ell_2)$, we can perform the same delay from (S_1, s_2).

Moreover, we reach equivalent states in both systems. Indeed, A_2 and $A'_{2,mod}$ stay in the same location, the clocks in $X_2 \setminus X_1$ increase their value by d, and the set of states of A_1 and A'_1 becomes $S'_1 = S''_1 = \{s'_1 \mid \exists s_1 \in S_1, \rho \in Paths(\Sigma_1 \setminus \Sigma_2, d) :$

$(s_1, s_2) \overset{\rho}{\Rightarrow} (s_1', q_2)\}$.

Therefore, $\mathscr{R}$ is a weak timed bisimulation and $\psi(\mathrm{TTS}_{A_1'}(A_2')) \sim \mathrm{TTS}_{A_1}(A_2)$. Lastly, by Lemma 41, $\psi(\mathrm{TTS}_{Q_1'}(A_1' \parallel A_2')) \sim \mathrm{TTS}_{Q_1}(A_1 \parallel A_2)$ also, and A_2 does not need to read X_1. □

In the example of Fig. 5.2, ☹ is not reachable in $\mathscr{S}_{mod}$ (see Fig. 5.6), therefore A_2 does not need to read X_1. For an example where ☹ is reachable, consider the same example with an additional edge $\xrightarrow{tt,f,\{x\}}$ from the end location of A_1 to a new location. Location ☹ can now be reached in $\mathscr{S}_{mod}$, for example consider a run where s is performed at time 2 leading to a state where $v(x) = 2$ and $v(x') = 2$, and then A_1 immediately performs f and resets x, leading to a state where the valuation v' is such that $v'(x) = 0$ and $v'(x') = 2$, and satisfies guard $x' \geq 1 \wedge x < 1$ in $\mathscr{S}_{mod}$. Therefore, with this additional edge in A_1, A_2 needs to read X_1. Indeed, without this edge, A_2 knows that A_1 cannot modify x after the synchronization, but with this edge, A_2 does not know whether A_1 has performed f and reset x, while this may change the truth value of its guard $x \geq 1$.

5.3.2 Complexity

PSPACE-hardness The reachability problem for timed automata is known to be PSPACE-complete [AD90]. We will reduce this problem to our problem of deciding whether A_2 needs to read the clocks of A_1. Consider a timed automaton A over alphabet Σ, with some location ℓ. Build the timed automaton A_2 as A augmented with two new locations ℓ' and ℓ'' and two edges, $\ell \xrightarrow{tt,\varepsilon,\emptyset} \ell'$ and $\ell' \xrightarrow{x=1,a,\emptyset} \ell''$, where x is a fresh clock, and a is some action in Σ. Let A_1 be the one of Fig. 5.4 with an action $b \notin \Sigma$. Then, ℓ is reachable in A iff A_2 needs to read x which belongs to A_1. Therefore the problem of deciding whether A_2 needs to read the clocks of A_1 is also PSPACE-hard.

PSPACE-membership Moreover, we can show that when A_2 is deterministic, our problem is in PSPACE. Indeed, by Theorem 42 and Lemma 44, ☹ is not reachable iff $noRestriction_{A_1}(A_2)$ iff A_2 does not need to read the clocks of A_1. Since the size of the modified system on which we check the reachability of ☹ is polynomial in the size of the original system, our problem is in PSPACE.

5.3.3 Dealing with Urgent Synchronizations

If we use exactly the same construction as before and allow urgent synchronizations, the following problem may occur. 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 of the construction 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. This choice of a run of A_1 is as valid as the others, and it prevents the system from having to deal with the subtle situation that we described above. Below, we describe the construction of $A_{1,2}$ in two cases:

1. There is always a run with maximal duration between two synchronizations
2. It may happen that, between two synchronizations, there is no run with maximal duration because of some strict time constraints.

Case 1: There is always a Run with Maximal Duration between two Synchronizations

Consider automaton A_1 in Fig. 5.7. We can see that, for the urgent synchronization to happen as late as possible, $A_{1,2}$ has to fire b at time 1, so that it can then wait 3 time units before synchronizing, although it is still able to synchronize at any time (we add the same dashed edges as in Fig. 5.6). This can be generalized for any A_1. The idea is essentially to force $A_{1,2}$ to follow the appropriate finite or ultimately periodic path in the region automaton [AD94] of A_1 (see end of Subsection 2.2.2 for a brief summary on the region automaton). The construction is described below and illustrated by Fig. 5.8 and 5.9.

$A_{1,2}$ is now built over the region automaton [AD94] of A_1. Transitions labeled by some $a \in \mathbb{S}$ are treated separately like in the original construction. The problem now is to constrain $A_{1,2}$ to take one of the most time consuming runs between two synchronizations.

The first step is to build the region automaton of A_1, and remove the synchronizations. Then, from the initial state, and from each state after a synchronization, we want to compute the most time consuming path. Let s be one of these states. If one of the paths from s has a loop, then there is an infinite run from s with local actions, and since we consider non-Zeno TA, time diverges and this run is valid. If no path from s contains a loop, then the paths from s are finite and there is a finite number of such paths. It is possible to compute, for each

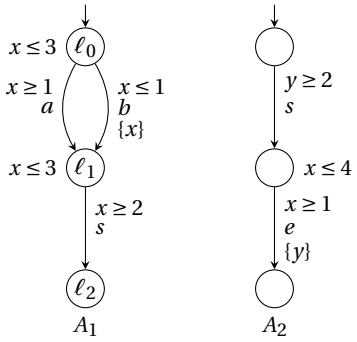


Fig. 5.7: *A_1 has an urgent synchronization.*

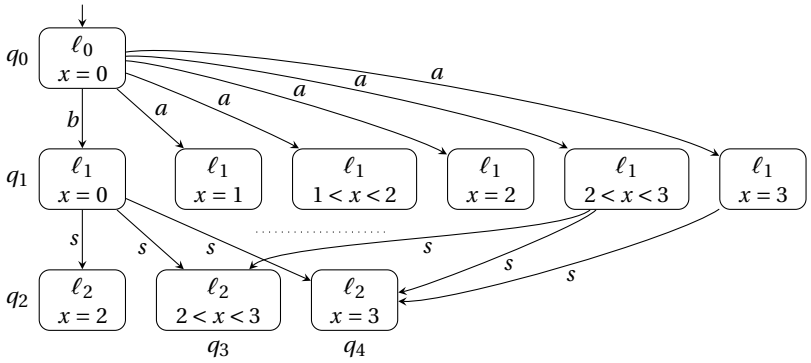


Fig. 5.8: *Region automaton of A_1 of Fig. 5.7. For readability, some of the edges labeled by s are not represented.*

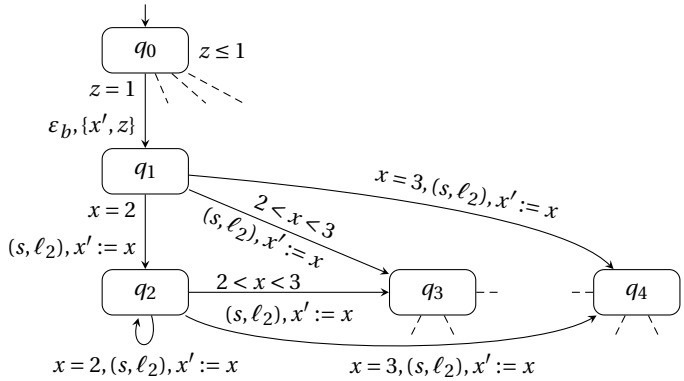


Fig. 5.9: *$A_{1,2}$ associated with A_1 of Fig. 5.7. Dashed lines denote synchronization edges that are not represented (similar to those represented from q_1 and q_2). $A_{2,mod}$ is the same as the one of Fig. 5.6.*

path, the supremum of the duration of the path: just sum the maximal delays in each locations so that the path is possible (including the time spent in the last location). Then we choose the path with maximal duration (the duration may be infinite if the last location has no invariant). If for each action in the path, it is possible to ensure a firing time that makes the run the most time consuming, then we impose these firing times using a fresh clock and the appropriate guards and invariants. By assumption, this is what happens in the case we are dealing with. To summarize, the algorithm to keep only paths with maximal duration is given below.

Data: timed automaton A_1
Result: region automaton of A_1 without the synchronizations and with only one path of maximal duration between two synchronizations
begin
  $R(A_1) \leftarrow (S, s_0, \Sigma_{1,\varepsilon} \setminus \mathbb{S}, \rightarrow)$ region automaton of A_1 without the synchronizations;
  $Init \leftarrow \{s \in S \mid s \text{ is a source state}\}$;
  foreach *s in Init* **do**
    if *There is no marked path from s* **then**
      if *There is a path from s with a loop* **then**
        mark edges of this path until a state $s' \in S$ with an outgoing edge that is already marked is met;
      else
        compute one path with maximal duration from s by summing the maximal delays in each state;
        mark edges of this path until a state $s' \in S$ with an outgoing edge that is already marked is met;
      endif
    endif
  endfch
  remove all unmarked edges from $R(A_1)$;
  return $R(A_1)$;
end

Algorithm 2: Keeping only paths with maximal duration between two synchronizations

Lastly, for each synchronizing edge in A_1, and each corresponding output state in the region automaton, we add synchronizing edges from all locations, to the location associated with this output state. These edges are labeled by "$\gamma(R), (a, \ell_1), c$", where $\gamma(R)$ is the constraint that describes the region R associated with the target state, a is the synchronization label in A_1, ℓ_1 is the output location of the synchronization in A_1, and c is the copy of clock values. It is important to keep the different states of the region automaton that can be reached after a synchronization, for the paths with maximal duration before the next synchronization have to be computed and enforced also from these states (there

could be other synchronization edges in A_1). That is why, when it synchronizes, $A_{1,2}$ has to read the region of the current valuation of A'_1 to move towards the corresponding state.

Finally, for our example automaton A_1 in Fig. 5.7, we get the region automaton of Fig. 5.8. After the synchronizations are removed, 6 final states can be reached from the initial state, with 6 possible paths. For each one of them, we compute the most time consuming one (we sum the maximal delays in each location, so that the path is possible and we add the maximal delay in the last location). All paths with action a have maximal duration of 3, and the path with action b has maximal duration of 4, when b is performed at time 1.

Therefore, we impose the firing of a silent action associated with b at time 1 in $A_{1,2}$, with adequate timing constraints, using a new clock, z, as in Fig. 5.9. Then we add the synchronizations as explained before, although here it is not necessary to keep the three states after the synchronization, since there are no other synchronization after. Below, we give the formal definition of $A_{1,2}$.

Definition of $A_{1,2}$ Assume (S, s_0, E) is a structure that stores the region automaton of A_1, without the synchronization edges, and with only the edges that are in the most time consuming paths computed as explained earlier. That is, S (resp. s_0) is the set of states (resp. the initial state) of the region automaton of A_1, and $E \subseteq S \times (\mathbb{N} \times E_1) \times S$ stores edges in the form $s \xrightarrow{d,e} s'$ where d is the delay that has to be performed in $\ell(s)$, the location associated with state s, before performing edge e labeled by some action in $\Sigma_{1,\varepsilon} \setminus \mathbb{S}$. Note that, by Algorithm 2, for each state s, there is at most one such edge in E). Then, $A_{1,2} = (S, s_0, X_1 \cup C'_1 \cup \{z\}, \mathbb{S}', E'_1, Inv'_1)$ where

- C'_1 is the set of clocks associated with C_1 as previously, and clocks in X_1 will be read on the synchronizations only,
- $E'_1 = \{s \xrightarrow{z=d,\varepsilon,r'\cup\{z\}} s' \mid \exists s \xrightarrow{d,e} s' \in E : e = (\ell(s) \xrightarrow{g,a,c(r')} \ell(s'))\}$
  $\cup \{s \xrightarrow{\gamma,(a,\ell_2),c} s' \mid s \in S \wedge \gamma \equiv \gamma(R(s')) \wedge a \in \mathbb{S} \wedge \exists \ell_1 \xrightarrow{g,a,r} \ell_2 \in E_1\}$
  where $\gamma(R(s'))$ is the clock constraint that describes the region of state s', and c still denotes the assignment of any clock $x' \in C'_1$ with the value of its associated clock $c(x') = x \in C_1$ (written $x' := x$).
- $\forall s \in S$, $Inv'_1(s) \equiv z \leq d$ if $\exists s \xrightarrow{d,e} s' \in E$, and $Inv'_1(s) \equiv \mathit{tt}$ otherwise.

We can now prove the direct way of Theorem 42 in this setting where A_1 may have urgent synchronizations, and the most time consuming runs between two synchronizations exist. First, let us recall some notations. $\mathcal{S}_{mod} = A'_1 \parallel (A_{1,2} \otimes A_{2,mod})$, with the same A'_1 and $A_{2,mod}$ as before, $A'_2 = A_{1,2} \otimes A'_{2,mod}$ where $A'_{2,mod}$ denotes $A_{2,mod}$ without location ☹, and ψ is such that for any $((a, \ell_1), s_1) \in \mathbb{S}' \times Q'_1$, $\psi(((a, \ell_1), s_1)) = (a, s_1)$.

Proof of Theorem 42, when most time consuming runs before synchronization exist. We show that when $noRestriction_{A_1}(A_2)$ holds, A_2 does not need to read the clocks of A_1, because then, the constructed $A'_1 \parallel A'_2$ satisfies satisfy Definition 40, i.e. has no shared clocks and

1. $\psi(\text{TTS}_{Q'_1}(A'_1 \parallel A'_2)) \sim \text{TTS}_{Q_1}(A_1 \parallel A_2)$ and
2. $\psi(\text{TTS}_{Q'_1}(A'_1)) \sim \text{TTS}_{Q_1}(A_1)$ (this still holds because A'_1 has not changed)
3. $\psi(\text{TTS}_{A'_1}(A'_2)) \sim \text{TTS}_{A_1}(A_2)$.

First, we can prove that ☹ is reachable in $\mathscr{S}_{mod}$ iff there is a restriction in $\text{TTS}_{A_1}(A_2)$, as we proved Lemma 44. Indeed, what works when $A_{1,2}$ simulates any run of A_1 also works when $A_{1,2}$ simulates a fixed run of A_1.

Then, we can prove that, if ☹ is not reachable (i.e. if there is no restriction in $\text{TTS}_{A_1}(A_2)$), then $\psi(\text{TTS}_{A'_1}(A'_2)) \sim \text{TTS}_{A_1}(A_2)$. We use the same relation $\mathscr{R}$ as in the previous proof in 5.3.1, that is, $\mathscr{R}$ is the relation such that for any reachable state (S_1, s_2) of $\text{TTS}_{A_1}(A_2)$, and any reachable state (S'_1, s'_2) of $\psi(\text{TTS}_{A'_1}(A'_2))$,

$$(S_1, s_2)\ \mathscr{R}\ (S'_1, s'_2) \overset{def}{\iff} \begin{cases} s_2 = (\ell_2, v_2) \text{ and } s'_2 = ((\ell_{1,2}, \ell_2), v'_2) \text{ s.t.} \\ \forall x \in X_2 \setminus X_1, v_2(x) = v'_2(x) \\ S_1 = S'_1 \end{cases}$$

The proof of this bisimulation follows the same steps as the proof in 5.3.1, except now we know that $A_{1,2}$ cannot force a synchronization by construction, and not by assuming that there is not urgent synchronization in A_1.

Then, by Lemma 41, $\psi(\text{TTS}_{Q'_1}(A'_1 \parallel A'_2)) \sim \text{TTS}_{Q_1}(A_1 \parallel A_2)$ also. □

Case 2: There is not always a Run with Maximal Duration between two Synchronizations

Now, we show how to adapt the previous construction when there are strict time constraints and there is no path with maximal duration before an urgent synchronization. For example, consider automaton A_1 of Fig. 5.10 that has an urgent synchronization and such that there is no path with maximal duration before this synchronization is taken: as previously, b has to be performed as late as possible, but because of the strict inequality $x < 1$ on the edge labeled by b, it is not possible to enforce this.

Here also, the construction of the region automaton and the computation of the paths with maximal duration is useful. But, if suitable firing times do not exist, then the idea is to follow one of the paths that reach the last region. In our example, the supremum of the duration of the path with b is 4, and is greater than the supremum of any other paths (the paths with a have a maximal duration of 3). Therefore, b has to be performed while x is in the region defined by $0 < x < 1$, which is the region that ensures that the last region is reached. We

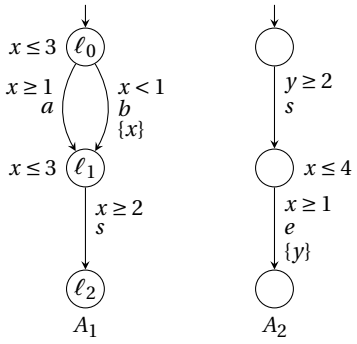


Fig. 5.10: *A_1 has an urgent synchronization and there is no path with maximal duration before this synchronization.*

enforce that $A_{1,2}$ performs the actions in the regions met along the computed path, with adequate timing constraints using a fresh clock.

Now, when $A_{1,2}$ reaches a state where it has to synchronize, if A'_1 is not ready to synchronize (i.e. A'_1 is not in the location before the synchronization), then this means that A'_1 took a more time consuming path (and not necessarily the same actions). Then $A_{2,mod}$ can stop using the values of the clocks of $A_{1,2}$ to evaluate the truth value of its time constraints, and simply take their truth value according to the last region that makes the invariant of the urgent synchronization true (i.e. the region of its current valuation), since it would still be in this region if it had been more time consuming. Note that, if ☹ is not reachable, this means that, if $A_{1,2}$ had performed a more time consuming run (for example the actual run followed by A'_1), then $A_{2,mod}$ would have been able to perform the same run. Therefore, "stopping" the clocks in their current region has no side effects.

In the construction, this results in new synchronization edges, performed by $A_{1,2}$ and $A_{2,mod}$, when $A_{1,2}$ has not been time consuming enough (i.e. when the invariant is about to expire). In our example, the synchronization labeled by $final\_region_\alpha$, guarded by $x' = 3$, notifies $A_{2,mod}$ that $A_{1,2}$ has reached the maximal region that satisfies the invariant $x' \leq 3$ (if the invariant was $x' < 3$, then the guard would be $2 < x' < 3$). Observe that these synchronizations will become local actions of A'_2 when the product will be done. Lastly, after each synchronization of this kind, $A_{2,mod}$ enters a duplicated version of itself, where the truth value of the guards is evaluated according to the last region of the path computed in the region automaton, see Fig. 5.11. This duplicated version can still reach location ☹, and the constraints on the edges leading to ☹ are also evaluated according to the last region.

If a synchronization happens when $A_{2,mod}$ is in one of its duplicated versions,then $A_{2,mod}$ can go back to its initial version, as depicted in Fig. 5.11. We now give the formal definition of $A_{1,2}$ and $A_{2,mod}$.

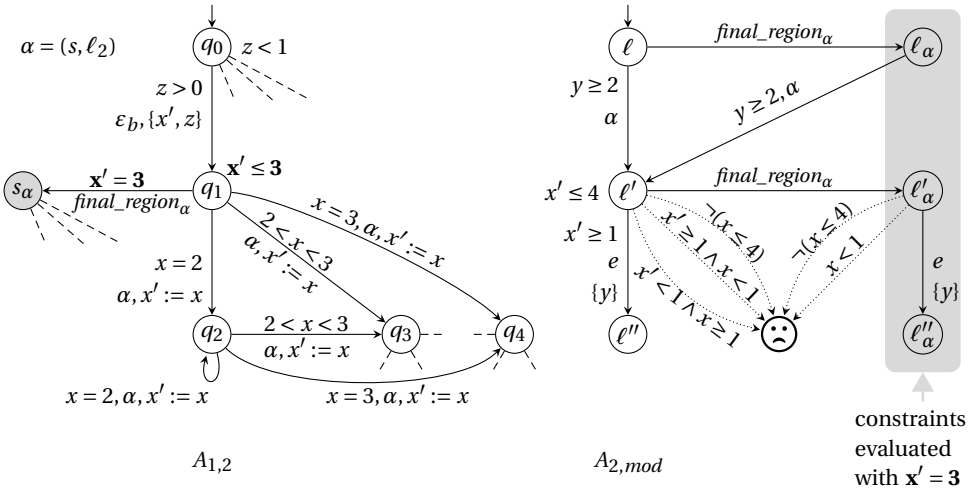


Fig. 5.11: *$A_{1,2}$ and $A_{2,mod}$ for the NTA of Fig. 5.10. Dashed lines denote synchronization edges that are not represented (similar to those represented from q_1 and q_2).*

Definition of $A_{1,2}$ and $A_{2,mod}$ Assume (S, s_0, E) is a structure that stores the region automaton of A'_1, without the synchronization edges, and with only the edges that are in the most time consuming paths in the following form: $E \subseteq S \times (\mathbb{N} \times \mathbb{N} \times E_1) \times S$ and for each edge $s \xrightarrow{d_m, d_M, e} s'$, d_m and d_M are the minimal and maximal delays to perform in $\ell(s)$ before taking this edge. If $d_m = d_M$, then the edge must be performed after a delay of d_m time units.

Let us first define some notations. $\mathbb{S}'_{urg} \subseteq \mathbb{S}'$ denotes the set of urgent synchronizations in A'_1, and for each state $s \in S$ such that $\ell(s)$ is a location before some urgent synchronization $a \in \mathbb{S}'_{urg}$, $\gamma(a) = \max(Inv_1(\ell(s))'$ denotes the constraint in $\mathscr{B}(X'_1)$ associated with the maximal region that makes $Inv_1(\ell(s))'$ true (like $x' = 3$ is the maximal region that makes invariant $x' \leq 3$ true in Fig. 5.11).

Then $A_{1,2} = (L_{1,2}, s_0, X'_1 \cup X_1 \cup \{z\}, \Sigma_{1,2}, E'_1, Inv'_1)$, where

- $L_{1,2} = S \cup \{s_a \mid a \in \mathbb{S}'_{urg}\}$, s.t. s_a is the location reached after performing $final\_region_a$,
- X'_1 is in bijection with X_1 as previously, and clocks in X_1 will be read on the synchronizations only,
- $\Sigma_{1,2} = \mathbb{S}' \cup \{final\_region_a \mid a \in \mathbb{S}'_{urg}\}$,
- $E'_1 = \{s \xrightarrow{z=d_m, \varepsilon, r' \cup \{z\}} s' \mid \exists s \xrightarrow{d_m, d_m, e} s' \in E : e = (\ell(s) \xrightarrow{g,a,c(r')} \ell(s'))\}$
  $\cup \{s \xrightarrow{z>d_m, \varepsilon, r' \cup \{z\}} s' \mid \exists s \xrightarrow{d_m, d_M, e} s' \in E : d_m \neq d_M \wedge e = (\ell(s) \xrightarrow{g,a,c(r')} \ell(s'))\}$
  $\cup \{s \xrightarrow{\gamma, (a, \ell_2), c} s' \mid s \in S \wedge \gamma \equiv \gamma(R(s')) \wedge a \in \mathbb{S} \wedge \exists \ell_1 \xrightarrow{g,a,r} \ell_2 \in E_1\}$
  $\cup \{s \xrightarrow{\gamma(a), final\_region_a} s_a \mid a \in \mathbb{S}'_{urg}\}$

where $\gamma(R(s'))$ is the clock constraint that describes the region of state s', and c denotes the assignment of any clock $x' \in C'_1$ with the value of its associated clock $c(x') = x \in C_1$.

- $\forall s \in S$, $Inv'_1(s) \equiv z \leq d_m$ if $\exists s \xrightarrow{d_m,d_m,e} s' \in E$,
  $Inv'_1(s) \equiv z < d_M$ if $\exists s \xrightarrow{d_m,d_M,e} s' \in E$ and $d_m \neq d_M$,
  $Inv'_1(s) = Inv_1(\ell(s))'$ if $\ell(s)$ is before an urgent synchronization,
  and $Inv'_1(s) \equiv \boldsymbol{tt}$ otherwise.

$A_{2,mod}$ is now defined as follows: $A_{2,mod} = (L_{2,mod}, \ell^0_2, X_2 \cup X'_1 \cup X_1, \Sigma_{2,mod}, E'_2, Inv'_2)$, where

- $L_{2,mod} = L_2 \cup \{\frownie\} \cup \{\ell_a \mid \ell \in L_2 \wedge a \in \mathbb{S}'_{urg}\}$, s.t. ℓ_a is the duplicated version of ℓ, that $A_{2,mod}$ reaches when it synchronizes on $final\_region_a$ with $A_{1,2}$,
- $\Sigma_{2,mod} = (\Sigma_2 \setminus \Sigma_1) \cup \mathbb{S}' \cup \{final\_region_a \mid a \in \mathbb{S}'_{urg}\}$,
- $E'_2 = \{\ell \xrightarrow{g',b,r} \ell' \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell_a \xrightarrow{g' \wedge \gamma(a),b,r} \ell'_a \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell \xrightarrow{g,(b,\ell_1),r} \ell' \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \in \mathbb{S} \wedge \ell_1 \in L_1\}$
  $\cup \{\ell_a \xrightarrow{g,(b,\ell_1),r} \ell' \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \in \mathbb{S} \wedge \ell_1 \in L_1\}$
  $\cup \{\ell \xrightarrow{\neg Inv_2(\ell),\varepsilon,\emptyset} \frownie \mid \ell \in L'_2\}$
  $\cup \{\ell \xrightarrow{g' \wedge \neg g,\varepsilon,\emptyset} \frownie \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell_a \xrightarrow{(g' \wedge \gamma(a)) \wedge \neg g,\varepsilon,\emptyset} \frownie \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell \xrightarrow{\neg g' \wedge g,\varepsilon,\emptyset} \frownie \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell_a \xrightarrow{\neg(g' \wedge \gamma(a)) \wedge g,\varepsilon,\emptyset} \frownie \mid \ell \xrightarrow{g,b,r} \ell' \in E_2 \wedge b \notin \mathbb{S}\}$
  $\cup \{\ell \xrightarrow{final\_region_a} \ell_a \mid \ell \in L_2 \wedge a \in \mathbb{S}'_{urg}\}$,
- $\forall \ell \in L_2, Inv'_2(\ell) = Inv_2(\ell)'$,
  $\forall \ell_a \in L'_2, Inv'_2(\ell_a)$ is $Inv_2(\ell)'$ evaluated with the clock values given by $\gamma(a)$, and $Inv'_2(\frownie) \equiv \boldsymbol{tt}$.

Gray boxes denote the new locations, edges and invariants yielded by the new communications between $A_{1,2}$ and $A_{2,mod}$, that drive $A_{2,mod}$ into duplicated versions of itself.

Then, we can prove a proposition similar to Proposition 43 with this construction that handles the general case. That is, we will prove the correspondence between a state of $\mathscr{S}_{mod}$ and two states of $\text{TTS}(A_1 \parallel A_2)$ that are merged into the same state of $\text{TTS}_{A_1}(A_2)$. A state of $\mathscr{S}_{mod}$ is denoted as $(s_1, s_{1,2}, s_2) = \big((\ell_1, v_{|X_1}), (\ell_{1,2}, v_{|X'_1}), (\ell_2, v_{|X_2 \setminus X_1})\big)$. For a given state of $A_{1,2}$, $s_{1,2} = (\ell_{1,2}, v_{|X'_1})$, we denote by $s'_{1,2}$ the state $(\ell'_{1,2}, v')$, where if $\ell_{1,2} \in L_1$, then $\ell'_{1,2} = \ell_{1,2}$ and $v' : X_1 \to \mathbb{R}_{\geq 0}$ is defined as: for any $x \in X_1$, $v'(x) = v(x')$, otherwise, $\ell_{1,2} = \ell_a$ for

some $a \in \mathbb{S}'_{urg}$, and then $\ell'_{1,2} = \ell$ such that ℓ is the location before *final_region*$_a$ in $A_{1,2}$, and ν' is some valuation that satisfies $\gamma(a)$, and such that for any $x \in X_1$ that is not constrained by $\gamma(a)$, $\nu'(x) = \nu(x')$. Reciprocally, for a given state of A_1, $s'_{1,2} = (\ell_{1,2}, \nu')$, $s_{1,2}$ denotes the state $(\ell_{1,2}, \nu)$, where $\nu : X'_1 \to \mathbb{R}_{\geq 0}$ is defined as: for any $x' \in X'_1$, $\nu(x') = \nu'(x)$.

Proposition 43 bis (general case)**.** *Let $(s_1, s_{1,2}, s_2)$ be a state of $\mathscr{S}_{mod}$. If along one path that leads to $(s_1, s_{1,2}, s_2)$ no edge leading to ☹ is enabled, then there exists S_1 such that (S_1, s_2) is a reachable state of $\mathrm{TTS}_{A_1}(A_2)$ and s_1 and $s'_{1,2}$ are both in S_1.*

Conversely, let (S_1, s_2) be a reachable state of $\mathrm{TTS}_{A_1}(A_2)$, and s_1 and $s'_{1,2}$ be some states in S_1. Then $(s_1, s_{1,2}, s_2)$ is a state of $\mathscr{S}_{mod}$.

The proof is again by recursion, in the same line as the original proof of Proposition 43. We use this proposition to prove Lemma 44 that we recall below, also with a similar proof as the original one.

Lemma 44 (reminder)**.** ☹ *is reachable in $\mathscr{S}_{mod}$ iff there is a restriction in* $\mathrm{TTS}_{A_1}(A_2)$.

Lastly, this lemma is used to prove the direct part of Theorem 42 in the general case. Below, we recall Theorem 42, and prove its direct way (the other way was proven just after its enunciation).

Theorem 42 (reminder)**.** *When noRestriction$_{A_1}(A_2)$ holds, A_2 does not need to read the clocks of A_1. When A_2 is deterministic, this condition becomes necessary.*

Proof. As above, we can prove that, if there is no restriction in $\mathrm{TTS}_{A_1}(A_2)$ (i.e. if ☹ is not reachable), then $\psi(\mathrm{TTS}_{A'_1}(A'_2)) \sim \mathrm{TTS}_{A_1}(A_2)$, where we consider that all actions *final_region*$_a$ for some $a \in \mathbb{S}'_{urg}$ are unobservable. Let $\mathscr{R}$ be the relation defined as follows: for any reachable state (S_1, s_2) of $\mathrm{TTS}_{A_1}(A_2)$, and any reachable state (S'_1, s'_2) of $\psi(\mathrm{TTS}_{A'_1}(A'_2))$,

$$(S_1, s_2)\ \mathscr{R}\ (S'_1, s'_2) \overset{def}{\iff} \begin{cases} s_2 = (\ell_2, \nu_2) \text{ and } s'_2 = ((\ell_{1,2}, \ell'_2), \nu'_2) \text{ s.t.} \\ \ell_2 = \ell'_2 \text{ or } \ell'_2 \text{ is one of the duplicated versions of } \ell_2 \\ \forall x \in X_2 \setminus X_1, \nu_2(x) = \nu'_2(x) \\ S_1 = S'_1 \end{cases}$$

Then, we can prove that $\mathscr{R}$ is a bisimulation. □

5.4 Discussion and Extensions

We have shown that in a distributed framework, when locality of actions and synchronizations matter, NTA with shared clocks cannot be easily transformed into NTA without shared clocks. 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 whether the transformation is possible 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 we 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 used in an appropriate way, and more generally gives a flavor of a class of NTA closer to implementation.

Perspectives Our first perspective is 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.

Notice that the set $\mathrm{UR}(s_1)$ used in the definition of contextual TTS is always put in parallel with a state s_2. Therefore, it can be extended to $\mathrm{UR}_{s_2}(s_1)$ that represents the set of states that A_1 can immediately reach from s_1 while A_2 is in s_2. This means that the TTS of A_2 in the context of A_1 can still be defined when A_1 also reads clocks from A_2. However, we do not know whether Theorem 42 is still true with this definition of contextual TTS, because most of the intermediate lemmas and propositions to prove this theorem use $\mathrm{TTS}(A_1)$ that is not defined when A_1 reads clocks from A_2.

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. Even when infinitely precise information is required to achieve the exact semantics of the NTA, it would be interesting to study how this semantics can be approximated using finitely precise information.

Finally, when shared clocks are necessary, one can 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.

Part II

Dependencies between Event Occurrences in Distributed Systems

From Untimed to Timed Settings

Chapter 6

Dependencies between Event Occurrences in Distributed (Real-Timed) Systems

Partial order representations of runs of Petri nets provide an alternative to sequential semantics by exhibiting the concurrency that naturally arises from the Petri net dynamics. *Occurrence nets* are the data structure for the partial order semantics referred to as *unfoldings* [NPW81, Eng91]. They are nets in which all transitions, called *events*, are executable and the *causality* relation induced by the arcs is acyclic. *Conflict* between two events is explicitly represented by a common place, called *condition*, in their causal past. Lastly, two events that are neither causally related nor in conflict are *concurrent*.

These structural relations between events induce logical dependencies between event occurrences: the occurrence of an event e in a run implies that all its causal predecessors also occur, and that no event in conflict with e occurs. But these structural relations do not express all the logical dependencies between event occurrences in maximal runs: in particular, the occurrence of e in any maximal run may imply the occurrence of another event that is not a causal predecessor of e, in that run. The *reveals* relation has been introduced to express this dependency between two events [Haa10]. It expresses dependencies such as "if event e occurs, then event f has occurred or will necessarily occur".

Furthermore, other more complex relations may exist between the events of a TPN. Indeed, when time is added, the dependencies between events may depend on the time at which the events occur. For example, it is possible that for event e to occur, another concurrent event f must have occurred in a given time interval, and if f occurs outside this time interval, then e cannot occur.

Such dependencies have been described in previous works about TPN unfoldings [CJ06, Tra09]. However, they have never been studied deeply and a better understanding of these relations could help improving the unfolding of TPN, in particular by providing a canonical structure.

In Chapter 7, we generalize the reveals relation to express more general dependencies, involving more than two events, and we introduce a logic to express them as Boolean formulas. We then present a synthesis procedure from a formula. Lastly, in Chapter 8, we lift these relations to a timed setting, and discuss

how to use them for TPN unfolding.

Organization of the Chapter In this chapter, we start by presenting, in Section 6.1, the problem and related work. In Section 6.2 we present the reveals relation introduced in [Haa10], and discuss different semantics. In Section 6.3 we define *logical independency* as opposed to concurrency which we consider as a *structural independency*. We also present the converse well-foundedness of the reveals relation over the set of facets. Lastly, we define tight nets as an explicit representation of the logical dependencies between the events of a finite occurrence net.

6.1 Problem and Related Work

In the structure of an unfolding, concurrency, causality and conflict are explicitly represented. Other binary relations, that describe the logical dependencies between event occurrences, are not represented in the structure of the unfolding. For example, the reveals relation expresses dependencies such as "if event e occurs, then event f has occurred or will necessarily occur".

In this part, we intend to study the dependencies between the events of a TPN, in order to define compact and canonical unfoldings of TPN.

Unfoldings were first defined for 1-bounded ordinary Petri nets, and were then extended to other classes of Petri nets, in particular to time Petri nets, and also to networks of timed automata.

6.1.1 Logical Characterization of Runs

Motivation

Several works have highlighted dependencies between event occurrences. First, in untimed systems, when considering occurrence nets, some dependencies are not described by the structural relations, and some concurrent events may not be independent. Second, when time is added, even more complex dependencies between event occurrences exist. That is why, in particular, unfolding a TPN is much more difficult than unfolding a PN.

Improving the unfolding of TPN, in order to get a more compact structure is one of the motivation for studying and formalizing the complex dependencies created by time.

Another motivation for studying implicit dependencies is to make them explicit in what we called *tight nets*. A tight net has the same set of runs as the initial ON and can be viewed as a pre-computation for a further analysis of the dependencies.

Related Work

Conflict Detection in STG Signal transition graphs (STG) are used to model asynchronous logic circuits, they are Petri nets whose transitions are interpreted as rising and falling edges of signals. In [KKY04], the authors build *configuration constraints* by considering the causal closure and the conflict-freeness of the configurations (or general runs). These constraints are then used for checking some properties that ensure the implementability of the STG as a logic circuit.

Reveals Relation The *reveals* relation was introduced in [Haa10] to express logical dependencies between two events. Knowledge of reveals facilitates in particular the analysis of partially observable systems, in the context of diagnosis, testing, or verification: an event b revealed by a need not be observable if a is, the occurrence of b can be *inferred*.

The reveals relation was recalled in Subsection 6.2.1. In [Haa10], an equivalent definition for the reveals relation was given.

$$e \rhd f \iff \#[f] \subseteq \#[e]$$

This definition was proved equivalent to Definition 45, where only maximal runs are considered. This corresponds to a setting where weak fairness [Vog95] is assumed, i.e. any enabled event has to occur or to be disabled. Later, we will consider different semantics, and show that the two definitions are no longer equivalent.

Release Relation In [Tra09], a conflict relation called *direct conflict* is defined. Since we already defined a different direct conflict relation in Subsection 2.1.3, we will call it *minimal conflict*. This relation, *conf* is defined as follows.

$$e_1 \mathit{conf}\ e_2 \iff \begin{cases} {}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset \\ \forall e < e_1, \neg(e \# e_2) \\ \forall e < e_2, \neg(e \# e_1) \end{cases}$$

That is, e_1 and e_2 are in conflict, and the conditions in ${}^\bullet e_1 \cup {}^\bullet e_2$ are pairwise concurrent.

Notice that two events are in conflict iff two of their ancestors are in minimal conflict. Since these minimal conflicts are the root of all conflicts, it is enough to solve these minimal choices, in order to solve all choices. Then, a second relation, called release relation ("relation de libération" in the manuscript in French) is defined.

$$(e_1, e, e_2) \in \mathit{lib} \iff \begin{cases} e_1 \mathit{co}\ e_2 \\ e_1 \mathit{conf}\ e \wedge e \mathit{conf}\ e_2 \end{cases}$$

When there exists e such that $(e_1, e, e_2) \in \mathit{lib}$, then $e_1\ \mathit{lib}\ e_2$. This relation represents additional dependencies between two events: when e_1 occurs it releases the conflict between e_2 and e (and symmetrically), as shown in Fig. 6.1.

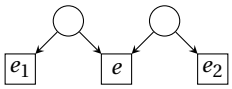


Fig. 6.1: *e_1 releases the conflict between e_2 and e.*

This release relation is then used for defining time branching processes i.e branching processes equipped with a timing function that assigns a valid time to a conflict-free set of events that forms a process, and an infinite time to other events. When computing the time of an event, two cases are distinguished, depending on whether the minimal conflicts have been released or not.

Comparison of the Relations e_1 *lib* e_2 does not mean that if e_1 occurs, then e_2 necessarily occurs. For example, another event in conflict with e_2 could occur. For e_1 to reveal e_2, all the minimal conflicts in the past of e_2 (the root of all conflicts with e_2) have to be released by some event in the past of e_1. Then the inclusion $\#[e_2] \subseteq \#[e_1]$ holds. Hence, the link between the release relation and the reveals relation can be expressed as follows.

$$\left(\forall f_2 \in \lceil e_2 \rceil, \forall e \; conf \; f_2, \exists f_1 \in \lceil e_1 \rceil : (f_1, e, f_2) \in lib\right) \iff e_1 \rhd e_2$$

Lastly, notice that the conflict relation plays a role in all these implicit dependencies.

6.1.2 Unfoldings of TPN

Motivation

The analysis methods based on the construction of the state space (for instance TTS or state class graphs) use the sequential semantics of the time Petri net. Therefore, all interleavings are represented, and the concurrency is lost.

In order to preserve the concurrency prescribed by the (time) Petri net, other methods are needed. Unfoldings are one of these methods for they preserve the partial order of events and thus the concurrency.

Related Work

Unfoldings of Petri Nets The representation of all runs of a Petri net as an *unfolding* [NPW81, Eng91] 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 [McM92, ERV02], for instance to check LTL formulas [Kho03].

Unfoldings have first been defined for ordinary 1-bounded nets. They have been extended to high-level (or colored) nets [KK03, Kho03, SK04]. Symbolic unfoldings for high-level nets were also defined in [EHP+02, CJ04, CF10], and their interest for the diagnosis of distributed systems has been argued. Unfoldings of

petri nets with read arcs have also been defined and applied to their verification in [VSY98, RSB11, RS12].

Unfoldings of Time Petri Nets Because time creates additional causality relations between events, time processes are not sufficient to build the unfolding of a TPN by looking independently at the concurrent parts, like in the untimed case. Hence building time processes implies to look at the global state of the net. Nevertheless, a constructive method of a complete finite prefix of a TPN has been introduced in [CJ06].

In this work, the authors defined *local firing conditions* that say whether a transition t can be fired at a given date θ, according to a partial marking L. In practice, in order to compute the firing time of a transition, one has to consider the tokens that it consumes, but also the tokens that can enable a conflicting transition. Time processes are then extended with read arcs that add causality relations between conditions and events. Lastly, the *symbolic unfolding* is defined by replacing the fixed timing functions by symbolic firing times described by inequalities.

A more compact unfolding has been defined in [Tra09], where it is noticed that, with the previous method, some events are duplicated many times. They limit the duplications of events to some situations where a conflict corresponds to a degree of indeterminism in the past of events.

Unfoldings have also been defined for other classes of TPN, like parametric stopwatch Petri nets [TGJ+10].

Lastly, unfoldings of NTA have also been defined in [BHR06, CCJ06].

6.2 Several Semantics for the Reveals Relation

Maximal and General Runs Below we call *run* of an ON a configuration, and *maximal run* a maximal configuration. We write Ω_{gen} for the set of all runs and Ω_{max} for the set of maximal runs.

Structural vs Logical Relations The structure of an occurrence net defines three relations over its events: *causality*, *concurrency* and *conflict*.

But these structural relations do not express all the logical dependencies between the occurrence of events in maximal runs. A central fact is that concurrency is not always a logical independency: it is possible that the occurrence of an event implies the occurrence of another one, which is structurally concurrent. This happens with events a and c in Fig. 6.2(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.

6.2.1 Reveals Relation and Facets Abstraction

Reveals Relation The reveals relation expresses dependencies between events such as "if e occurs, then f has already occurred or will occur eventually" in the sense that any run that contains e also contains f.

Definition 45 (Reveals relation [Haa10])**.** We say that event e *reveals* event f, and write $e \rhd f$, iff $\forall \omega \in \Omega_{max}, (e \in \omega \implies f \in \omega)$.

Note that $\rhd$ is transitive.

Link with the Structural Relations The structural dependencies are somewhat connected with the logical ones. Below we describe some connections. First, since runs are causally closed, the following property is trivial.

Property 46. For any events e and f, $f \leq e \implies e \rhd f$.

Property 47 ($\#$-inheritance under $\rhd$)**.** The conflict relation is inherited under the reveals relation: for any events a, b, c, $a \# b$ and $c \rhd b$ together imply $a \# c$.

Proof. Assume a run contains a and c. Then, because $c \rhd b$, it also contains b, which contradicts $a \# b$. □

Facets Abstraction

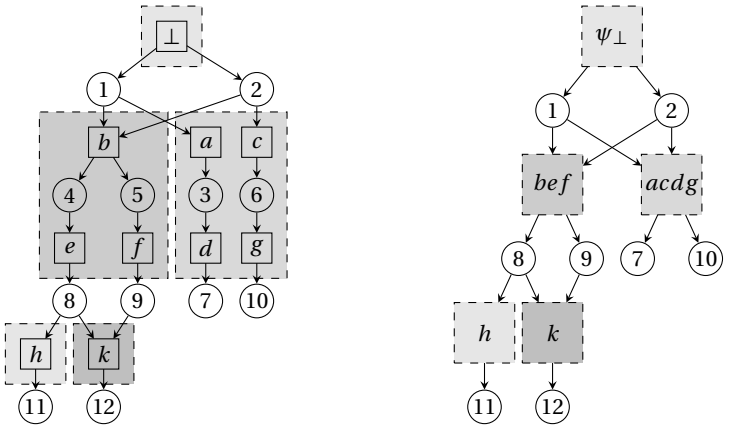
Definition 48 (Facet[Haa10])**.** Let $\sim$ be an equivalence relation defined as: $\forall e, f \in E, e \sim f \overset{def}{\iff} (e \rhd f) \wedge (f \rhd e)$, then a *facet* of an ON is an equivalence class of $\sim$.

For example, in Fig. 6.2(a), the ON has five facets: $\{\bot\}$, $\{a, c, d, g\}$, $\{b, e, f\}$,$\{h\}$ and $\{k\}$. If ψ is a facet, then for any run ω and for any event e such that $e \in \psi$, $e \in \omega$ iff $\psi \subseteq \omega$. In this sense, facets can be seen as atomic sets of events.

The causality relation, $\leq$, and the conflict relation, $\#$, naturally extend to the set of facets as follows: $\forall \psi_1, \psi_2 \in \Psi$,

$$\begin{aligned} \psi_1 \leq \psi_2 &\overset{def}{\iff} \exists e_1 \in \psi_1, e_2 \in \psi_2 : e_1 \leq e_2 \\ \psi_1 \# \psi_2 &\overset{def}{\iff} \exists e_1 \in \psi_1, e_2 \in \psi_2 : e_1 \# e_2 \end{aligned}$$

The set of facets equipped with $\leq$ and $\#$ is an event structure [Haa10].



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

(b) The corresponding reduced ON.

Fig. 6.2: *An ON and its reduction through the facet abstraction*

Reduced Occurrence Nets 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 events. In the sequel, we consider reduced ONs [Haa10], i.e. ONs reduced by contracting the facets into events.

For example, in Fig. 6.2(a), the reduced ON is obtained by contracting, for each facet, the squared events into an event. With the maximal semantics, this gives the reduced ON of Fig. 6.2(b). From now on, runs are thus considered as conflict-free and causally closed sets of *facets*.

Definition 49 (Reduced occurrence net)**.** A *reduced ON* is an ON (B, Ψ, F) such that $\forall \psi_1, \psi_2 \in \Psi, \psi_1 \sim \psi_2 \iff \psi_1 = \psi_2$ (i.e. such that $\triangleright$ is antisymmetric).

Several choices of Ω

An important line of research is to go beyond the maximal semantics considered here. In [Haa10], $\triangleright$ was defined over maximal runs only. But it can easily be defined on an arbitrary set of runs Ω: just replace Ω_{max} by Ω in Definition 45. Then the reveals relation depends on the set of runs Ω that we consider. Two natural choices are the set of maximal runs Ω_{max} and the set of all runs Ω_{gen}.

Other sets of runs could also be considered, like runs of time Petri nets. But this extension is not trivial since time creates complex dependencies between events. For example, it may happen that the occurrence of an event f depends on the occurrence date of an apparently unrelated event e, because if e happens early enough, it triggers an event in conflict with f.

In Chapter 8, we extend the reveals relation to the runs of a deterministic timed ON, i.e. an ON such that the date of an event is uniquely determined: if

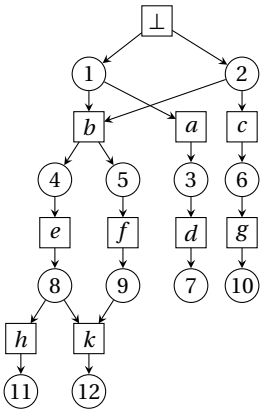


Fig. 6.3: *An occurrence net*

e occurs, it occurs at time τ_e. We explain how these ON describe the runs of a subclass of time Petri nets (also called deterministic), and we discuss how to lift the reveals relation to the runs of (unfoldings of) general TPN.

Below, we compare the maximal and general semantics. Then we discuss about the timed semantics.

6.2.2 Maximal and General Semantics

We study two untimed semantics: the maximal semantics and the general semantics. The maximal semantics assumes maximal progress (or weak fairness [Vog95]), whereas the general semantics makes no such assumption, i.e. an execution may stop at any time.

Maximal Semantics

The maximal semantics is the one which inspired the definition of the reveals relation in [Haa10]. Indeed this setting generates rich dependencies between events. The first interesting point with the maximal semantics is a nice characterization of the reveals relation based on the conflict relation. This characterization was actually used as the definition of the reveals relation in [Haa10].

Lemma 50 (Reveals relation: alternative definition for Ω_{max} [Haa10])**.** *For any events e and f, $(e \rhd f \text{ in } \Omega_{max}) \iff \#[f] \subseteq \#[e]$.*

Notice that, with the general semantics, the two definitions are not equivalent. For example, in Fig. 6.3, $d \rhd a$ holds for general runs and therefore also for maximal runs, but $a \rhd d$ and $d \rhd c$ hold for maximal runs only.

Moreover, the following lemma highlights the importance of the conflict relation in the definition of maximal runs.

Lemma 51. *A set of events ω is a maximal run iff $\forall a \in E, a \in \omega \iff \#[a] \cap \omega = \emptyset$.*

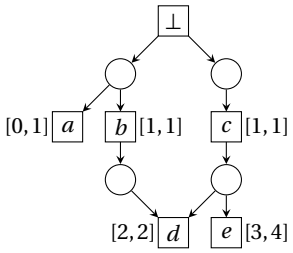


Fig. 6.4: *An occurrence net constrained by time delay intervals*

Proof. If ω is a run and there exists $a \in E \setminus \omega$ that is not in conflict with any event of ω, then $\omega \cup \lceil a \rceil$ is also a run and ω is not maximal. Conversely, a set of events ω which satisfies the equivalence for any event a is conflict-free and $\subseteq$-maximal, and since the conflict is inherited under the causality, ω must also be causally closed. □

General Gemantics

The general semantics allows a lot of possible runs (including the empty set). Moreover, in this setting where we do not assume maximal progress, the reveals relation coincides exactly with the structural causality. Indeed, first by Property 46, we know that for any events e and f, if $f \leq e$, then $e \rhd f$, and second, when there is no progress assumption, $\lceil e \rceil$ is a valid run, and consequently e does not reveal any event outside $\lceil e \rceil$.

6.2.3 Timed Semantics

Beyond the two natural setups presented above, we see many relevant situations where more specific sets of executions have to be considered. One example comes from the modeling of real-time systems. Consider the occurrence net depicted in Fig. 6.4 and assume that its behavior is constrained by the time constraints given by the intervals, which are interpreted like in time Petri nets [Mer74] with dense time. We observe that, because of urgency, the occurrence of b forces the occurrence of d two time units later, i.e. $b \rhd d$ in the time semantics. As a consequence e reveals a. This also makes b and e incompatible, although they are not in conflict in the sense of untimed occurrence nets.

More complex dependencies exist and are described and analyzed in Chapter 8, where we focus on the dependencies in the unfoldings of a simple class of TPN.

6.3 Tight Occurrence Nets

In this section, we focus on the maximal semantics and we study the logical dependencies over facets. We first discuss the difference between concurrency and

logical independency, then show that we have to restrict ourselves to finite reduced ON, and then present tight nets. Tight nets are ON where the logical dependencies are explicitly represented as causalities, therefore in these ON, concurrency and logical independency coincide.

First, since the reduction of an ON by the facet abstraction yields an ON, the concurrency relation, *co*, and the reveals relation, $\triangleright$, naturally extend to the set of facets as well as the causality and conflict relations (recalled in Subsection 6.2.1). For any facets ψ_1 and ψ_2,

$$\begin{aligned} \psi_1 \; co \; \psi_2 &\overset{def}{\iff} \neg(\psi_1 \# \psi_2) \wedge \neg(\psi_1 \leq \psi_2) \wedge \neg(\psi_2 \leq \psi_1) \\ &\iff \psi_1 \neq \psi_2 \wedge \forall e_1 \in \psi_1, e_2 \in \psi_2 : e_1 \; co \; e_2 \\ \psi_1 \triangleright \psi_2 &\overset{def}{\iff} \exists e_1 \in \psi_1, e_2 \in \psi_2 : e_1 \triangleright e_2 \end{aligned}$$

6.3.1 Concurrency vs Logical Independency

Two facets may be causally ordered ($\leq$), in conflict ($\#$) or concurrent (*co*). The conflict relation exactly coincides with the fact that two facets never occur in the same execution. Moreover the causal ordering induces a reveals relation as stated in Property 46. But two concurrent facets are not necessarily logically independent in maximal runs. Hence causality and reveals together give a finer partition of the possible dependencies between two facets that are not in conflict. They can be either:

- causally related (and therefore also related by $\triangleright$),
- concurrent but related by $\triangleright$, or
- logically independent (and hence concurrent).

Formally, we define the *independency relation* among facets, denoted by *ind*, as the complement of the conflict and reveals relations:

$$\begin{aligned} \psi_1 \; ind \; \psi_2 &\overset{def}{\iff} \neg(\psi_1 \# \psi_2) \wedge \neg(\psi_2 \triangleright \psi_1) \wedge \neg(\psi_1 \triangleright \psi_2) \\ &\iff \psi_1 \; co \; \psi_2 \wedge \neg(\psi_2 \triangleright \psi_1) \wedge \neg(\psi_1 \triangleright \psi_2) \end{aligned}$$

That is, two facets are independent if they are neither in conflict nor related by the reveals relation. For example, in Fig. 6.5, facets b and c are concurrent but not independent because c reveals b, and facets a and b are independent. Therefore, if a is in a run, this gives no information on the presence (or absence) of b in the run.

6.3.2 Well-foundedness of the Inverse Reveals Relation

Lemma 52. *In any reduced ON $\mathcal{N} = (B, \Psi, F)$ where there is no infinite set of pairwise concurrent events (in particular in the reduced unfolding of any safe Petri net), the reveals relation, $\triangleright$, is* converse well-founded *on Ψ, i.e. there is no infinite chain of distinct facets $\psi_1 \triangleright \psi_2 \triangleright \ldots$*

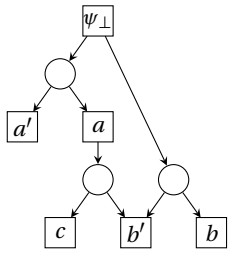


Fig. 6.5: *a ind b, $\neg(b$ ind $c)$ and $\neg(b$ ind $a')$*

Proof. In the proof, we use the alternative characterization of well-foundedness: $\rhd$ is converse well-founded on Ψ iff every nonempty subset S of Ψ has a $\rhd$-maximal facet, i.e. a facet ψ such that for any facet $\psi' \in S$, $\psi' \neq \psi \Rightarrow \neg(\psi \rhd \psi')$.

Assume first that the set $S \subseteq \Psi$ is conflict-free, and consider the set S' of the facets of S that have no strict causal predecessor in S. Because causality is well-founded, S' is not empty. Moreover, by definition, the facets of S' are pairwise concurrent. Thus, by hypothesis, S' is finite. Therefore there must be a facet ψ that is $\rhd$-maximal in S'. It remains to show that ψ is also $\rhd$-maximal in S. Let ψ' be a facet of S such that $\psi \rhd \psi'$. By construction of S' there exists a facet ψ'' in S' such that $\psi'' \leq \psi'$. By Property 46, this implies that $\psi' \rhd \psi''$, and by transitivity of $\rhd$, we get $\psi \rhd \psi''$. Since ψ is $\rhd$-maximal in S', ψ'' must equal ψ. Then we have $\psi \rhd \psi' \rhd \psi$, which implies that ψ equals ψ' by construction of the facets.

If $S \subseteq \Psi$ is not conflict-free, then for any facet $\chi \in S$, the subset of S, $S_\chi = \{\chi' \in S \mid \chi \rhd \chi'\}$ is conflict-free and hence has a $\rhd$-maximal facet ψ. Moreover, by construction of S_χ, ψ does not reveal any facet in S, therefore, ψ is also $\rhd$-maximal in S. □

We do not know if this lemma still holds without the hypothesis that there is no infinite set of pairwise concurrent events.

Anyway, Lemma 52 does *not* imply that any facet reveals only finitely many other facets. As a counterexample, consider the reduced ON of Fig. 6.6: facet ψ_3, associated with transition t_3, reveals all the facets $\psi_{1,i}$, $i \in \mathbb{N}^*$, associated with transition t_1.

Remark 53. For any *finite* reduced ON (B, Ψ, F), the triple $(\Psi, \rhd^{-1}, \#)$ is an *event structure* because:

1. $\rhd^{-1}$ is a partial order on Ψ,
2. For all $x \in \Psi$, $\{y \in \Psi \mid x \rhd y\}$ is finite,
3. $\# \subseteq \Psi \times \Psi$ is an irreflexive and symmetric relation, and for all $x, y, z \in \Psi$, $x \# y$ and $z \rhd y$ together imply $x \# z$ (Property 47).

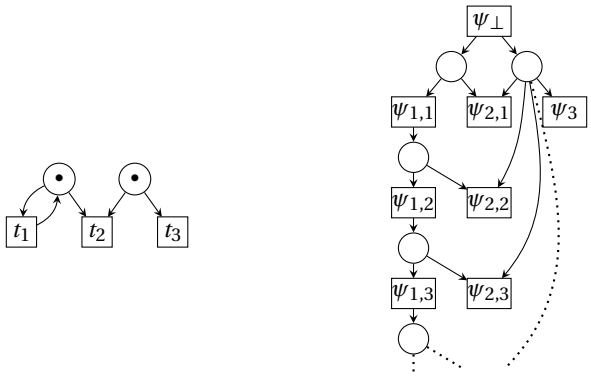


Fig. 6.6: *A Petri net and its unfolding (which is already a reduced ON)*

6.3.3 Tight (Occurrence) Nets

A tight (occurrence) net is a reduced ON in which all binary logical dependencies among facets (given by the reveals relation) are represented as causalities.

Definition 54 (Tight net)**.** A *tight net* is an ON (B, E, F) such that $\forall e, f \in E,\ e \triangleright f \iff f \leq e$.

Tight nets constitute a natural and canonical class of occurrence nets, of interest in their own right as representations of logical dependencies (as opposed to temporal ones). Moreover, all nets obtained as the output of the synthesis procedure defined in Section 7.2 are tight.

Proposition 55. *Every tight net is a reduced ON.*

Proof. If two events e and f of a tight net are in the same facet, then we have $e \triangleright f \wedge f \triangleright e$, which is equivalent to $f \leq e \wedge e \leq f$ because the net is tight. This implies $e = f$ by antisymmetry of $\leq$. □

Remark 56. In a tight net, *ind* is equivalent to *co*, and therefore the observation of the independency relation is easier than in a general reduced ON.

Remark 57. If we consider the set of general runs, then, since any ON is reduced, and for any events e and f, $e \triangleright f \iff f \leq e$, any ON is tight.

We will show in Section 7.3.1 that it is possible to transform any finite reduced ON in a canonical tight net which accepts the same set of maximal runs Ω_{max}. This canonical tight net gives an efficient representation of the reveals relation. The example of Fig. 6.6, shows that the assumption of finiteness is necessary.

6.3.4 Facets as Labeled Partial Orders

When the ON corresponds to the unfolding of a safe PN (P, T, F, M_0), and π is the homomorphism that relates the ON to the PN, it can be interesting to study the shape of facets in terms of partial orders labeled by transitions and places, To do so, we associate with each facet ψ the corresponding labeled partial order $lpo(\psi)$. We show that the set of labeled partial orders corresponding to facets of the unfolding of a safe Petri net, is finite.

Lemma 58. *Let $\Pi = (O, \pi)$, with $O = (B, E, G)$, be the unfolding of the safe PN $N = (P, T, F, M_0)$, $O_r = (B_r, \Psi, G_r)$ the reduced ON associated with O, Then the image of Ψ by lpo is finite.*

Proof. First, we show that any cut C of O_r enables finitely many facets: any facet enabled by C has at least one event that consumes only conditions of C. But there are finitely many such events, and each event belongs to a single facet. Thus C enables finitely many facets.

Now we need some definitions from [ERV02]. For a given cut (maximal set of pairwise concurrent conditions) C of a branching process $\Pi = (O, \pi)$, $\Uparrow C = (O', \pi')$ is the part of Π "lying after" C: O' is the unique subnet of O whose set of nodes is $\{x \mid x \notin \lceil C \rceil \wedge \forall y \in \lceil C \rceil, \neg(x \# y)\}$ where $\lceil C \rceil = \bigcup_{b \in C} \{y \in B \cup E \mid y < b\}$ ($<$ is now defined on $B \cup E$ and not only on E), and π' is the restriction of π to the nodes of O'. According to Proposition 4.3 in [ERV02], if Π is the unfolding of (P, T, F, M_0), then $\Uparrow C$ is the unfolding of $(P, T, F, \pi(C))$ (up to isomorphism).

We associate with each set of conditions $B' = \{b_1, \ldots, b_m\} \subseteq B$ the multiset of places $m(B') = \{\pi(b_1), \ldots, \pi(b_m)\}$. In fact, we will consider only co-sets or cuts, therefore $m(B')$ will be a subset of a marking or a marking, i.e. a set of places and not a multiset places: $m(B') = \pi(B') \subseteq P$.

Furthermore, for any cuts of O_r, C_1 and C_2, such that $\pi(C_1) = \pi(C_2) = M$, $\Uparrow C_1$ and $\Uparrow C_2$ are isomorphic (they are isomorphic to the unfolding of (P, T, F, M)) and thus $\{m(\psi) \mid {}^\bullet\psi \subseteq C_1\} = \{m(\psi) \mid {}^\bullet\psi \subseteq C_2\}$, i.e. the images by m of the sets of facets enabled by C_1 and C_2 are the same. This image is a finite set of multisets of T characterized by M. Hence, any reachable marking $M \subseteq P$ is mapped with a finite set of multisets of T: $T_M = \{m(\psi) \mid \pi({}^\bullet\psi) \subseteq M\}$. Therefore $\{m(\psi) \mid \psi \in \Psi\} = \bigcup_{M \in R(N)} \{m(\psi) \mid \psi \in \Psi \wedge \pi({}^\bullet\psi) \subseteq M\} = \bigcup_{M \in R(N)}$, where $R(N)$ is the set of reachable markings of N, is finite since it is a finite union of finite sets. □

Towards a Logical Characterization of Runs of Occurrence Nets

Some logical dependencies such as "if a and b occur then c occurs" cannot be expressed by the binary reveals relation. In the next chapter, starting from this observation, we define an *extended reveals relation*. We also introduce a logic that allows us to describe sets of runs, and solve the following synthesis problem: given a logical formula φ, is there an ON whose set of runs is described by φ?

Chapter 7

Synthesis of Tight Occurrence Nets

The structure of an occurrence net induces three relations over its events, *causality*, *conflict* and *concurrency* [NPW81]. The causality and conflict relations induce logical dependencies between event occurrences: the occurrence of an event e in a run implies that all its causal predecessors also occur, and that no event in conflict with e ever occurs.

We mentioned in Subsection 6.2.1, that, when only maximal runs are considered, the structural relations do not express all the logical dependencies between event occurrences. Moreover, we also observe that, in this context, concurrency is not a logical independency: it is possible that the occurrence of an event implies the occurrence of another one, which is structurally concurrent. This happens for concurrent events a and c in Fig. 6.3. The *reveals* relation between events was introduced in [Haa10] to express these implicit dependencies between two events. The equivalence classes of events that mutually reveal each other are called *facets*; contracting facets into single events creates a *reduced* occurrence net whose set of maximal executions is in bijection with that of the initial occurrence net.

While the focus in [Haa10] was on the *binary reveals* relation, in this chapter, we embed the relation in a more general logical framework. Starting from the observation that the reveals relation corresponds to logical implication between the occurrence of events, we consider general boolean formulas where the atoms express the occurrence of events. The resulting logic, which we call ERL, captures dependencies in occurrence nets. We then show how to build a logical formula that describes all logical dependencies between the occurrence of events. Then we ask which formulas are satisfied by all the runs of an occurrence net. An important result is that the logical dependencies between events, with the maximal progress assumption, are not only binary: there are logical dependencies that cannot be deduced from binary dependencies. This leads us to define an extended reveals relation.

Lastly, we solve the synthesis problem that arises: given an ERL formula over events (or facets), does this formula describe the set of possible runs of an occurrence net? We propose a method for synthesizing an occurrence net from an ERL formula. As a corollary, this allows us to identify a canonical occurrence

net to represent the equivalence class of all occurrence nets that have the same logical dependencies between events.

Organization of the Chapter First, Section 7.1 introduces the ERL logic, capable of capturing general logical dependencies between events. ERL formulas can be interpreted with respect to a set of acceptable runs of an occurrence net. Section 7.2 explains how to build an ERL formula that describes the dependencies between the events of a given occurrence net, and then solves the problem of synthesis of tight occurrence nets from ERL formulas. Section 7.3 presents a few extensions. In particular it shows that, while synthesizing an occurrence net from an ERL formula, the causality in the net can be freely chosen provided it is compatible with the reveals relation induced by the formula; it also discusses synthesis under non-maximal semantics.

7.1 A Logic for Occurrence Nets

We introduce 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.

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 reveal 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 Fig. 6.5: causality gives only the dependencies $a < c$ and $a < b'$, plus the trivial ones involving $\psi_\perp$. With the reveals relation we get $c \rhd b$ and $a' \rhd b$. 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 (containing $\psi_\perp$) that satisfies these constraints, is a maximal run? The answer is no: for instance $\{\psi_\perp, a, b\}$ 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.

Our logic is designed so that it allows us to express this kind of complex dependencies between event occurrences, and to define an appropriate *extended reveals relation*.

7.1.1 Syntax and Semantics

As any propositional logic, the *alphabet* consists of a set of variables E (including $\perp$), the constants $\boldsymbol{tt}$ and $\boldsymbol{ff}$, and the logival connectives $\wedge$ and $\neg$.

Well-formed formulas are called *ERL formulas* and defined inductively with the following BNF grammar:

$$\varphi ::= \boldsymbol{tt} \mid \boldsymbol{ff} \mid e \mid \neg\varphi \mid \varphi \wedge \varphi, \text{ where } e \in E$$

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

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

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 φ and for any set of sets of events Γ,

$$\Gamma \models \varphi \text{ iff } \forall \gamma \in \Gamma, \gamma \models \varphi$$

i.e. the formula is satisfied by all sets of events. Notice that, $\Gamma \not\models \varphi$ iff $\exists \gamma \in \Gamma : \gamma \not\models \varphi$ (which is different from $\Gamma \models \neg\varphi$).

We define the set $[\![\varphi]\!]$ as $[\![\varphi]\!] \stackrel{def}{=} \{\gamma \subseteq E \mid \gamma \models \varphi\}$. We write $\varphi \equiv \varphi'$ when $[\![\varphi]\!] = [\![\varphi']\!]$.

Extended Reveals Relation

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

$$\begin{array}{ll} & \bigwedge_{i\in I}(b_{i,1} \vee b_{i,2} \vee \cdots \vee b_{i,n_i} \vee \neg a_{i,1} \vee \neg a_{i,2} \vee \cdots \vee \neg a_{i,m_i}) \\ \text{iff} & \bigwedge_{i\in I}\big((a_{i,1} \wedge a_{i,2} \wedge \cdots \wedge a_{i,m_i}) \rightarrow (b_{i,1} \vee b_{i,2} \vee \cdots \vee b_{i,n_i})\big) \\ \text{iff} & \bigwedge_{i\in I}(\bigwedge_{a\in A_i} a \rightarrow \bigvee_{b\in B_i} b), \end{array}$$

where $A_i = \{a_{i,1}, \ldots, a_{i,m_i}\}$ and $B_i = \{b_{i,1}, \ldots, b_{i,n_i}\}$. And since for any set of runs Ω,

$$\begin{array}{ll} & \Omega \models \bigwedge_{i\in I}(\bigwedge_{a\in A_i} a \rightarrow \bigvee_{b\in B_i} b) \\ \text{iff} & \forall i \in I, \Omega \models \bigwedge_{a\in A_i} a \rightarrow \bigvee_{b\in B_i} b \\ \text{iff} & \forall i \in I, \forall \omega \in \Omega, A_i \subseteq \omega \Rightarrow B_i \cap \omega \neq \emptyset, \end{array}$$

we can 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 and that are satisfied by a set of runs Ω iff whenever all events in A occur in a run $\omega \in \Omega$, then at least one event in B occurs in ω. This leads us to define the *extended reveals relation*.

Definition 59 (Extended reveals relation)**.** Let $\Omega \subseteq 2^E$ be a set of runs, and A, B two sets of events, A reveals B written $A \twoheadrightarrow B$, iff $\forall \omega \in \Omega$, $A \subseteq \omega \Rightarrow B \cap \omega \neq \emptyset$

In this notation, Ω becomes implicit. Notice that $\neg(A \twoheadrightarrow B)$ means $\Omega \not\models \bigwedge_{a\in A} a \rightarrow \bigvee_{b\in B} b$ i.e. $\exists \omega \in \Omega : A \subseteq \omega \wedge B \cap \omega = \emptyset$.

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

Proposition 60. *In the maximal semantics and the general semantics, conflicts can be expressed using this extended reveals relation:* $\{a,b\} \rightarrowtriangle \emptyset \iff a \# b$.

This equivalence comes directly from the definition of runs. We should however consider it as a strong property of these two semantics and notice that only one direction would hold, for instance, in the timed semantics evoked in Subsection 6.2.3: in the example of Fig. 6.4, events b and e are incompatible ($\{b,e\} \rightarrowtriangle \emptyset$), although they are not in conflict in the sense of untimed occurrence nets ($\neg(b \# e)$).

Remark 61. The extended reveals relation is not transitive: in general $A \rightarrowtriangle B \wedge B \rightarrowtriangle C$ does not imply $A \rightarrowtriangle C$. Indeed, the extended reveals relation is interpreted as a conjunction of events in the left part and as a disjunction of events in the right part.

7.1.2 Minimal and Immediate Constraints

Expressions of the form $A \rightarrowtriangle B$ are called *constraints*. We notice that some constraints can be deduced from others by monotonicity and by inheritance, which leads us to define *minimal constraints*.

Monotonicity Properties

First, the extended reveals relation has the following monotonicity properties:

Left Monotonicity Property. $\forall A,B,C \in 2^E, A \rightarrowtriangle C \wedge A \subseteq B \Rightarrow B \rightarrowtriangle C$.
Indeed, $A \subseteq B \Leftrightarrow \Omega \models \bigwedge_{b\in B} b \rightarrow \bigwedge_{a\in A} a$, and $\rightarrow$ is transitive.

Right Monotonicity Property. $\forall A,B,C \in 2^E, A \rightarrowtriangle C \wedge C \subseteq B \Rightarrow A \rightarrowtriangle B$.
Indeed, $C \subseteq B \Leftrightarrow \Omega \models \bigvee_{c\in C} c \rightarrow \bigvee_{b\in B} b$, and $\rightarrow$ is transitive.

Therefore, we begin by considering the constraints $A \rightarrowtriangle B$ where the sets A and B are minimal.

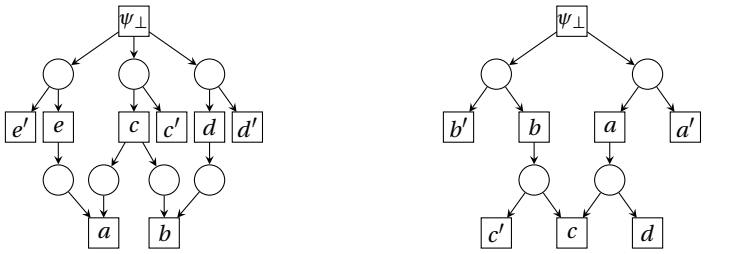
Definition 62 (Minimal reveals relation)**.** We define the *minimal reveals relation*, $\rightarrowtriangle_m$, as: $\forall A,B \in 2^E$,

$$A \rightarrowtriangle_m B \overset{def}{\iff} (A \neq B) \wedge (A \rightarrowtriangle B) \wedge (\nexists B' \subsetneq B : A \rightarrowtriangle B') \wedge (\nexists A' \subsetneq A : A' \rightarrowtriangle B)$$

i.e. if one event is removed from the left part or the right part, the reveals relation is lost.

For example, in Fig. 6.5, $\{a,b\} \rightarrowtriangle_m \{c\}$ because none of the following constraints holds: $\{a\} \rightarrowtriangle \{c\}$, $\{b\} \rightarrowtriangle \{c\}$, $\emptyset \rightarrowtriangle \{c\}$ and $\{a,b\} \rightarrowtriangle \emptyset$.

Intuitively the minimal reveals provides a more precise description than the extended reveals. Indeed, if $A \rightarrowtriangle_m B$, we know that for each $b \in B$, there is a run that contains A and b and no other event in B (otherwise $A \rightarrowtriangle B \setminus \{b\}$). Similarly, for each $a \in A$, there is a run that contains $A \setminus \{a\}$ and no event of B (otherwise $A \setminus \{a\} \rightarrowtriangle B$).



(a) $\{c,d\} \rightarrowtriangle \{b\}$ is an immediate constraint and $\{a,d\} \rightarrowtriangle \{b\}$ is not.

(b) $\{a\} \rightarrowtriangle \{c,d\}$ is an immediate constraint and $\{a\} \rightarrowtriangle \{b,d\}$ is not.

Fig. 7.1: *Immediate constraints*

Deduction Through a Singleton

Moreover, the following properties also hold:

Left Inheritance Property. $\forall A,B \in 2^E, (A \cup \{d\} \rightarrowtriangle B) \wedge (d' \rhd d) \Rightarrow A \cup \{d'\} \rightarrowtriangle B$

Right Inheritance Property. $\forall A,B \in 2^E, (A \rightarrowtriangle B \cup \{d\}) \wedge (d \rhd d') \Rightarrow A \rightarrowtriangle B \cup \{d'\}$

We can now identify the extended reveals relations that are minimal w.r.t. deduction through a singleton.

Definition 63 (Immediate reveals relation)**.** We define the *immediate reveals relation*, $\rightarrowtriangle_i$, as: $\forall A,B \in 2^E$,

$$A \rightarrowtriangle_i B \overset{def}{\iff} \begin{cases} A \rightarrowtriangle_m B \\ \wedge\ \forall a \in A, \nexists a' \in E \setminus \{A \cup B\} : (a \rhd a' \wedge A_{a'/a} \rightarrowtriangle B) \\ \wedge\ \forall b \in B, \nexists b' \in E \setminus \{A \cup B\} : (b' \rhd b \wedge A \rightarrowtriangle B_{b'/b}) \end{cases}$$

where $A_{a'/a}$ denotes $A \cup \{a'\} \setminus \{a\}$.

For example, in Fig. 7.1(a), $\{a,d\} \rightarrowtriangle_m \{b\}$ is not an immediate constraint because $a \rhd c$ and $\{c,d\} \rightarrowtriangle \{b\}$. And in Fig. 7.1(b) $\{a\} \rightarrowtriangle_m \{b,d\}$ is not an immediate constraint because $c \rhd b$ and $\{a\} \rightarrowtriangle \{c,d\}$.

When $\rhd$ is antisymmetric, the conjunction of all immediate constraints implied by some formula φ, is equivalent to φ (by definition of the immediate constraints).

7.1.3 Properties of the Extended Reveals Relation

When we consider the maximal or the general semantics, a set of events that never occur together necessarily contains two events in conflict.

Lemma 64. *If we consider Ω_{gen} or Ω_{max}, for any set of events A, $A \rightarrowtriangle_m \emptyset \Rightarrow |A| = 2$.*

Proof. The reveals relation $A \rightarrowtriangle \emptyset$ implies that there exists no general run $\omega \in \Omega_{gen}$ such that $A \subseteq \omega$: in the general semantics, this holds simply by definition of the reveals relation; in the maximal semantics, the definition says that there exists no *maximal* run $\omega \in \Omega_{max}$ such that $A \subseteq \omega$, which implies that there exists no general run $\omega \in \Omega_{gen}$ such that $A \subseteq \omega$.

Then in particular $\lceil A \rceil$ is not a general run. Since it is causally closed, the reason why it is not a general run, is that it contains two events a and b that are in conflict. Since a and b are in $\lceil A \rceil$, there exist events a' and b' in A such that $a \in \lceil a' \rceil$ and $b \in \lceil b' \rceil$. By inheritance of the conflict along the causality, a' is in conflict with b', which implies $\{a', b'\} \rightarrowtriangle \emptyset$. And since $\{a', b'\} \subseteq A$ and $A \rightarrowtriangle_m \emptyset$, we must have $A = \{a', b'\}$. Finally, the absence of self-conflicts in occurrence nets guarantees that a' and b' are distinct. □

Remark 65. As well as Prop. 60, the previous lemma should be considered as an important property of the maximal and general semantics, and would not hold, for instance, in the timed semantics (see Subsection 6.2.3) nor for contextual occurrence nets [BCM01, BP96, Win98, Vog02], 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 [EHP+02, CJ04, CF10].

When we consider the set of general runs, Ω_{gen}, we have already noticed that the binary reveals relation is given by the causality: $\forall a, b \in E, \{a\} \rightarrowtriangle \{b\} \iff b \leq a$. Furthermore, we have:

Proposition 66 (Decomposition of reveals relation in the general semantics)**.** *With the general semantics, for any sets of events A and B,*

$$A \rightarrowtriangle B \iff (\exists a \in A,\ b \in B : b \leq a) \vee (\exists a, a' \in A : a \# a').$$

Proof. ($\Leftarrow$) If there exist $a, a' \in A$ such that $a \# a'$, then, no run contains A and for any set of events C, $A \rightarrowtriangle C$. And if there exist $a \in A$ and $b \in B$ such that $b \leq a$, then $a \triangleright b$ and by the monotonicity properties of $\rightarrowtriangle$, $A \rightarrowtriangle B$.
($\Rightarrow$) Assume $A \rightarrowtriangle B$ and A is conflict-free. Denote by $\lceil A \rceil$ the causal past of A i.e. the set $\lceil A \rceil = \cup_{a \in A} \lceil a \rceil$. Since we make no progress assumption, $\lceil A \rceil$ is a valid run. By definition of the extended reveals relation, $\forall \omega \in \Omega_{gen}$, $A \subseteq \omega \Rightarrow \omega \cap B \neq \emptyset$, and in particular, for $\omega = \lceil A \rceil$, this implies that $\lceil A \rceil \cap B \neq \emptyset$ i.e. that there exist $b \in B$ and $a \in A$ such that $b \leq a$. □

Therefore, with general runs, non binary constraints can be decomposed as disjunctions of binary ones, in contrast to the case for Ω_{max}.

Binary Immediate Constraints

Two kinds of binary immediate constraints will be particularly useful in the sequel.

First, we define the *immediate conflict* relation, $\#_i$, as a special case of the immediate reveals relation: for all events a and b, $a \#_i b \overset{def}{\iff} \{a,b\} \rightarrowtriangle_i \emptyset$. For example, in Fig. 7.1(b), a' and c are in conflict but not in immediate conflict because $a' \# a$ and $c \rhd a$. For any formula φ describing the runs of an ON, we have $\#_i \subseteq \#_d$.

Second, we define the *immediate reveals* relation, $\rhd_i$, as: $a \rhd_i b \overset{def}{\iff} \{a\} \rightarrowtriangle_i \{b\}$. For example, in Fig. 7.1(b), $b \rhd_i \psi_\perp$ and $\neg(c \rhd_i \psi_\perp)$.

Remark 67. When $\rhd$ is antisymmetric, the reveals relation is the transitive and reflexive closure of the immediate reveals relation and the conflict relation can be deduced by $\rhd$-inheritance from the immediate conflict relation. Therefore, the conflict relation can be deduced from the immediate reveals relation and the immediate conflict relation: $\# = (\rhd_i^{-1})^* \circ \#_i \circ \rhd_i^*$. That is, $(a,b) \in \# \iff \exists c,d : (a,d) \in \rhd_i^* \wedge (d,c) \in \#_i \wedge (c,b) \in (\rhd_i^{-1})^*$.

7.2 A Synthesis Problem

In Section 7.1 we have introduced ERL logic to describe logical dependencies between events of an occurrence net. Now two synthesis problems arise naturally.

We first show how to build the ERL formula $\Phi^{\mathcal{N}}$ which describes the set of maximal runs of a finite ON $\mathcal{N}$, i.e. such that $\Omega^{\mathcal{N}}_{max} = [\![\Phi^{\mathcal{N}}]\!]$. Then we present a procedure to answer whether there exists a tight net $\mathcal{N}$ such that its set of maximal runs is described by a given ERL formula φ.

This synthesis procedure allows us to understand the power of the logical properties expressed via *reveals*-relations or, equivalently, ERL formulas. They also allow - see below - to identify the canonical shape of occurrence nets with respect to these properties. Note that we restrict our attention in this section to *finite* occurrence nets, i.e. over a fixed finite set of individuals interpreted as events. 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; note in particular the fact, highlighted by Fig. 6.6, that a facet (here ψ_3) may reveal infinitely many others, which means that the procedure below would fail to produce event ψ_3.

Even so, the capability of synthesizing occurrence nets with a given finite set of facets from ERL formulas has potential even in practical terms. In fact, suppose you take any finite occurrence net $\mathcal{O}$ obtained by synthesis from φ, and convert it into a safe Petri net by adding,

- for every maximal run ω of $\mathcal{O}$, a transition t_ω whose pre-set is formed by the maximal conditions of ω,

- an extra place p whose post-set is $\{\bot\}$ and whose pre-transitions are the t_ω,
- and a token on p and no tokens elsewhere.

Then the resulting net $\mathcal{N}$ is a workflow net whose behaviors are concatenations of ωs, i.e. such that the properties satisfied at each workflow round are given by φ.

7.2.1 From Occurrence Nets to ERL Formulas

For a given finite ON $\mathcal{N}$, we start by building $\Phi^{\mathcal{N}}_{gen}$, a formula such that $[\![\Phi^{\mathcal{N}}_{gen}]\!] = \Omega^{\mathcal{N}}_{gen}$, from the characterization of general runs. Then we build $\Phi^{\mathcal{N}}$, a formula such that $[\![\Phi^{\mathcal{N}}]\!] = \Omega^{\mathcal{N}}_{max}$, by adding terms corresponding to the progress assumption to $\Phi^{\mathcal{N}}_{gen}$. The construction of $\Phi^{\mathcal{N}}_{gen}$ is similar to [KKY04], where the authors build what they call "configuration constraints" also by considering the causal closure and the conflict-freeness of the configurations (or general runs).

By definition, a set of events is a general run iff it is closed under causality and conflict-free. That is, for a given finite ON $\mathcal{N} = (B, E, F)$, we can build the formula $\Phi^{\mathcal{N}}_{gen}$ as follows:

$$\Phi^{\mathcal{N}}_{gen} = \underbrace{\bigwedge_{a,b\in E, a<b} (b \to a)}_{\text{causal closure}} \wedge \underbrace{\bigwedge_{a,b\in E, a\#b} (\neg a \vee \neg b)}_{\text{conflict-freeness}}$$

Therefore, for a given finite ON $\mathcal{N}$, $\Phi^{\mathcal{N}}$ can be built as follows:

$$\begin{aligned}
\Phi^{\mathcal{N}} \quad = \quad & \bigwedge_{a,b\in E, a<b} (b \to a) && \text{(causal closure)}\\
\wedge \quad & \bigwedge_{a,b\in E, a\#b} (\neg a \vee \neg b) && \text{(conflict-freeness)}\\
\wedge \quad & \bigwedge_{a\in E} \Big(\underbrace{\big(\bigwedge_{b\in E, b\lessdot a} b \big)}_{a \text{ enabled}} \to \big(a \vee \bigvee_{c\in E, c\#_d a} c \big) \Big) && \text{(progress assumption)}
\end{aligned}$$

The new part is implied by the maximality and stands for "for any event a, if a is enabled, then a or an event in direct conflict with a has to fire".

Since $<$ is the transitive closure of the direct causality $\lessdot$, the first part can be rewritten using only $\lessdot$, and since $\#$ is inherited through $<$, in the second part, we can consider only the direct conflict $\#_d$, and eventually:

$$\Phi^{\mathcal{N}} \equiv \bigwedge_{a,b\in E, a\lessdot b} (b \to a) \wedge \bigwedge_{a,b\in E, a\#_d b} (\neg a \vee \neg b) \wedge \bigwedge_{a\in E} \Big(\big(\bigwedge_{b\in E, b\lessdot a} b \big) \to \big(a \vee \bigvee_{c\in E, c\#_d a} c \big) \Big)$$

Notice that, since $\bot$ has no conflict and no causal predecessor, the third part with $a = \bot$ gives $\boldsymbol{tt} \to \top$ which can be reduced in $\bot$, i.e. $\bot$ is always true (and so is $\psi_\bot$ when we consider reduced ONs).

For example, in Fig. 7.1(b):

$$\begin{aligned}\Phi^{\mathcal{N}} \equiv\ & (c' \to b) \wedge (c \to b) \wedge (c \to a) \wedge (d \to a) \\ & \wedge (\bar{a}' \vee \bar{a}) \wedge (\bar{b}' \vee \bar{b}) \wedge (\bar{c}' \vee \bar{c}) \wedge (\bar{c} \vee \bar{d}) \\ & \wedge \psi_\perp \wedge ((a \wedge b) \to (c \vee c' \vee d)) \\ & \wedge (a \to (c \vee d)) \wedge (b \to (c' \vee c)) \\ & \wedge (\psi_\perp \to (b' \vee b)) \wedge (\psi_\perp \to (a' \vee a)),\end{aligned}$$

where $\bar{a}$ stands for $\neg a$.

We have deliberately omitted terms of the form $a \to \psi_\perp$ that are redundant since $\psi_\perp$ must be true.

Complexity

The formula is built as a conjunction of terms. First, identifying the causalities and the conflicts requires looking at each pair of events $\{a,b\} \subseteq E$. Therefore, this gives $O(n^2)$ terms with two events, where $n = |E|$. Second, there are n terms that describe the progress assumption (one for each event), and these terms are of size $O(n)$. Therefore, the size of the formula is $O(n^2)$.

7.2.2 From ERL formulas to Tight Nets: a Synthesis Procedure

The synthesis problem for PNs has been widely studied. It consists in answering whether, given a behavior, there exists a PN with this behavior. The behavior can be specified as a transition system [Ber93, DR96, BCD02, CCK+08] or a language, be it (i) a sequential language: in [Dar98], the behavior is bounded by two regular languages; or (ii) a finite partial language (finite set of labeled partial orders): [BDLM08]. Most of the time, the synthesis procedure is based on the notion of region [ER89, BD98].

In this paper, we propose another approach and we solve the following synthesis problem: given an ERL formula φ, is there a tight net $\mathcal{N}$ whose behavior is the one specified by φ, i.e. such that the set of maximal runs of $\mathcal{N}$, $\Omega^{\mathcal{N}}_{max}$, is equivalent to $[\![\varphi]\!]$?

In the sequel, we give a procedure to build a net, CN(φ), from an ERL formula φ. First, a set of binary immediate constraints is extracted from φ, then, CN(φ), is built from these constraints. If CN(φ) is a reduced ON, then $\Phi^{\mathrm{CN}(\varphi)}$ is computed and compared with φ. As in the other synthesis procedures, places are used to restrict the behavior of the net and denote dependencies between occurrences of transitions.

Extracting the Immediate Constraints

The set of maximal runs is given by the conflict relation which can be deduced from the immediate reveals relation and the immediate conflict relation (Lemma 51 and Remark 67). Therefore, if there exists a reduced ON $\mathcal{N}$ such that

$\Omega^{max}_{\mathcal{N}} = [\![\varphi]\!]$, then the binary immediate constraints, i.e. expressions of the form $a \rhd_i b$ and $a \#_i b$, are enough to describe $\Omega^{max}_{\mathcal{N}}$ (and thus also to describe φ). That is why we focus on binary immediate constraints.

Our problem is to decide whether binary constraints of the form $a \rhd b$ (respectively $\{a,b\} \twoheadrightarrow \emptyset$) are satisfied by φ. This amounts to deciding whether $\varphi \to (a \to b)$ (respectively $\varphi \to (\neg a \vee \neg b)$) is a tautology. This problem is co-NP-complete and can be solved quite efficiently in practice by SAT-solvers.

Building a Canonical Tight Net

We denote by $\Psi(\varphi)$ the set of variables that appear in φ which is supposed to be "reduced", i.e. such that for any distinct variables $a, b \in \Psi(\varphi)$, $[\![\varphi]\!] \not\models a \leftrightarrow b$. Each binary immediate constraint extracted from φ is represented by a condition connected to the facets that appear in the constraint. The net $\mathrm{CN}(\varphi)$ is defined as follows.

Definition 68 ($\mathrm{CN}(\varphi)$)**.** Let φ be an ERL formula. $\mathrm{CN}(\varphi) = (B, \Psi, F)$ is the finite net such that $\Psi = \Psi(\varphi)$, $B = B_1 \cup B_2$ and $F = F_1 \cup F_2$, where:

- $B_1 = \big\{\{\psi, \psi'\} \mid \psi \#_i \psi'\big\}$,
- $F_1 = \big\{(\{\psi, \psi'\}, \psi) \in B_1 \times \Psi\big\} \cup \big\{(\psi_\perp, \{\psi, \psi'\}) \in \Psi \times B_1\big\}$.

That is, for each constraint of the form $\psi \#_i \psi'$, one condition b is created and connected to $\psi_\perp$, ψ and ψ' such that ${}^\bullet b = \{\psi_\perp\}$ and $b^\bullet = \{\psi, \psi'\}$.

- $B_2 = \big\{(\psi, \psi') \in (\Psi \setminus \{\psi_\perp\})^2 \mid \psi' \rhd_i \psi\big\}$,
- $F_2 = \big\{\big((\psi, \psi'), \psi'\big) \in B_2 \times \Psi\big\} \cup \big\{\big(\psi, (\psi, \psi')\big) \in \Psi \times B_2\big\}$.

That is, for each constraint of the form $\psi' \rhd_i \psi$, one condition b is created and connected to ψ and ψ' such that ${}^\bullet b = \{\psi\}$ and $b^\bullet = \{\psi'\}$. Notice that constraints of the form $\psi \rhd_i \psi_\perp$ are not considered because, if φ describes the maximal runs of a reduced ON, they are already represented by B_1 and F_1. Indeed, in this case, $\psi \neq \psi_\perp$, hence $\#[\psi] \neq \emptyset$, and since ψ reveals only $\psi_\perp$ and $\#[\psi_\perp] = \emptyset$, there exists ψ' s.t. $\psi \#_i \psi'$.

Remark 69. In fact, the set of runs described by φ, i.e. $[\![\varphi]\!]$ is more relevant than φ itself, because for two formulas, φ_1 and φ_2, $\varphi_1 \equiv \varphi_2 \iff \mathrm{CN}(\varphi_1) = \mathrm{CN}(\varphi_2)$. Therefore, we could also define $\mathrm{CN}(\Omega)$ for a given $\Omega \subseteq 2^\Psi$.

Lemma 70. *Let $\mathcal{N}$ be a finite reduced ON, then $\mathrm{CN}(\Phi^{\mathcal{N}})$ is a tight net and $\Phi^{\mathrm{CN}(\Phi^{\mathcal{N}})} \equiv \Phi^{\mathcal{N}}$.*

Proof. First, we show that $\mathrm{CN}(\Phi^{\mathcal{N}})$ is a tight net. We call $\mathscr{C}$ the net $\mathrm{CN}(\Phi^{\mathcal{N}})$. We first show that $\mathscr{C}$ is an ON, then that it is reduced, and lastly that it is a tight net. $\mathcal{N}$ and $\mathscr{C}$ have the same conflict relation, because they have the same reveals relation and the same immediate conflict relation (Remark 67). Moreover $\mathscr{C}$ is built so that $\forall a, b \in \Psi, a \leq_{\mathscr{C}} b \iff b \rhd a$. Therefore, $\mathscr{C}$ is an ON because:

- There is no self-conflict in $\mathscr{C}$, because there is no self-conflict in $\mathcal{N}$.
- $\leq_{\mathscr{C}}$ is equivalent to $\triangleright^{-1}$ therefore it is a partial order.
- $\forall \psi \in \Psi, \{\psi' \mid \psi' \leq_{\mathscr{C}} \psi\}$ is finite because Ψ is finite.
- There is no backward branching by construction.
- $\psi_\perp \in \Psi$ is the only minimal node by construction.

Since $\Phi^{\mathcal{N}}$ is associated with the reduced ON $\mathcal{N}$, it is such that, for any distinct variables $v_1, v_2 \in \Psi, [\![\Phi^{\mathcal{N}}]\!] \not\models v_1 \leftrightarrow v_2$. Therefore, $\mathscr{C}$ is also reduced. Lastly, by construction, $\mathscr{C}$ is a tight net.

Second, we show that $\Phi^{\mathrm{CN}(\Phi^{\mathcal{N}})} \equiv \Phi^{\mathcal{N}}$. By Lemma 51, the set of maximal runs can be defined from the conflict relation only. $\mathcal{N}$ and $\mathrm{CN}(\Phi^{\mathcal{N}})$ have the same conflict relation. Therefore, $\mathcal{N}$ and $\mathrm{CN}(\Phi^{\mathcal{N}})$ have the same set of runs and equivalent associated ERL formulas. □

Notice that $\mathcal{N}$ and $\mathrm{CN}(\Phi^{\mathcal{N}})$ may not accept the same general runs because the facets that are concurrent but related by the reveals relation in $\mathcal{N}$, become causally ordered in $\mathrm{CN}(\Phi^{\mathcal{N}})$.

From Lemma 70, we can derive the following theorem.

Theorem 71. *Let φ be an ERL formula such that for any distinct variables $a, b \in \Psi(\varphi)$, $[\![\varphi]\!] \not\models a \leftrightarrow b$. There exists a reduced ON $\mathcal{N}$ such that $\Phi^{\mathcal{N}} \equiv \varphi$ iff $\mathrm{CN}(\varphi)$ is a reduced ON and $\Phi^{\mathrm{CN}(\varphi)} \equiv \varphi$.*

Proof. ($\Rightarrow$) If there exists a reduced ON $\mathcal{N}$ such that $\Phi^{\mathcal{N}} \equiv \varphi$, then, by Lemma 70 $\mathrm{CN}(\varphi)$ is a candidate.
($\Leftarrow$) $\mathrm{CN}(\varphi)$ is an example of suitable reduced ON. □

Example 4 illustrates that the net $\mathrm{CN}(\varphi)$, obtained by the synthesis from an arbitrary formula φ, may not be a reduced ON. When $\mathrm{CN}(\varphi)$ is a reduced ON, it is called the *canonical tight net* associated with φ (or with $\mathcal{N}$ when ϕ is defined as the formula $\Phi^{\mathcal{N}}$ associated with some reduced occurrence net $\mathcal{N}$).

Examples

We extract a set of binary immediate constraints from φ and build the net $\mathrm{CN}(\varphi)$.

Ex 1. Consider the following formula:

$$\begin{aligned}\varphi = \quad & \psi_\perp \wedge (a \rightarrow b) \wedge (b' \rightarrow a') \\ & \wedge (\bar{a} \vee \bar{a'}) \wedge (\bar{b} \vee \bar{b'}) \\ & \wedge (a \vee a') \wedge (b \vee b')\end{aligned}$$

The set of runs described by φ is $[\![\varphi]\!] = \{\{\psi_\perp, a, b\}, \{\psi_\perp, a', b\}, \{\psi_\perp, a', b'\}\}$.
The binary immediate constraints are: $a \triangleright_i b$, $b' \triangleright_i a'$, $b \triangleright_i \psi_\perp$, $a' \triangleright_i \psi_\perp$,

$a \#_i a'$ and $b \#_i b'$, and the net synthesized from these constraints is given in Fig. 7.2(a). This net is a reduced ON and its set of maximal runs is indeed $[\![\varphi]\!]$.

Ex 2. Consider the following formula:

$$\varphi = \psi_\perp \wedge (\bar{a} \vee \bar{b})$$

The set of runs described by φ is $[\![\varphi]\!] = \{\{\psi_\perp\}, \{\psi_\perp, a\}, \{\psi_\perp, b\}\}$. The binary immediate constraints are: $a \rhd_i \psi_\perp$, $b \rhd_i \psi_\perp$ and $a \#_i b$, and the ON $\mathcal{N}$ synthesized from these constraints is given in Fig. 7.2(b). $\mathcal{N}$ is a reduced ON but $\Omega^{\mathcal{N}}_{max} = \{\{\psi_\perp, a\}, \{\psi_\perp, b\}\} \neq [\![\varphi]\!]$. Therefore, there is no reduced ON $\mathcal{N}$ such that $\varphi \equiv \Phi^{\mathcal{N}}$. We can see that the maximality constraint $a \vee b$ is not respected by φ.

Ex 3. Consider the following formula:

$$\begin{aligned}\varphi = \quad & (\psi_\perp \wedge a \wedge b \wedge \bar{c} \wedge \bar{a'} \wedge \bar{b'} \wedge c') \\ \vee\ & (\psi_\perp \wedge a \wedge \bar{b} \wedge c \wedge \bar{a'} \wedge b' \wedge \bar{c'}) \\ \vee\ & (\psi_\perp \wedge \bar{a} \wedge b \wedge c \wedge a' \wedge \bar{b'} \wedge \bar{c'}) \\ \vee\ & (\psi_\perp \wedge \bar{a} \wedge \bar{b} \wedge \bar{c} \wedge a' \wedge b' \wedge c')\end{aligned}$$

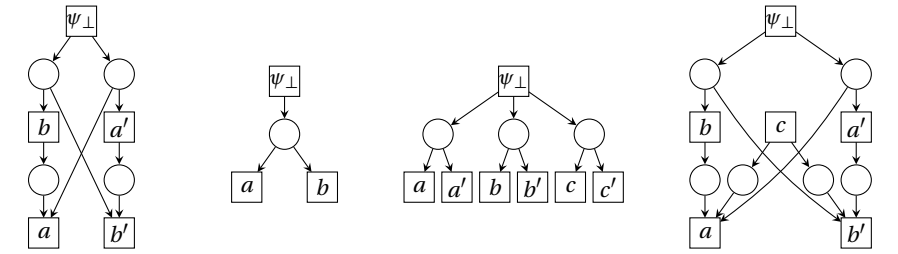
The set of runs described by φ is $[\![\varphi]\!] = \{\{\psi_\perp, a, b, c'\}, \{\psi_\perp, a, b', c\}, \{\psi_\perp, a', b, c\}\}$. The binary immediate constraints are: $a \#_i a'$, $b \#_i b'$, $c \#_i c'$ and for each $\psi \in \Psi \setminus \{\psi_\perp\}$, $\psi \rhd_i \psi_\perp$. The ON $\mathcal{N}$ synthesized from these constraints is given in Fig. 7.2(c). $\mathcal{N}$ is a reduced ON but $\Omega^{\mathcal{N}}_{max} = \{\{\psi_\perp, a, b, c\}, \{\psi_\perp, a', b, c\}, \{\psi_\perp, a, b', c\}, \{\psi_\perp, a, b, c'\}, \{\psi_\perp, a', b', c'\}, \{\psi_\perp, a, b', c'\}, \{\psi_\perp, a', b, c'\}, \{\psi_\perp, a', b', c\}\} \neq [\![\varphi]\!]$. Therefore, there is no reduced ON $\mathcal{N}$ such that $\varphi \equiv \Phi^{\mathcal{N}}$.

Notice that this example illustrates an immediate conflict between a, b and c: $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$ can occur in a run, but $\{a, b, c\}$ cannot, which is not possible in general ONs (see Lemma 64).

Ex 4. Consider the following formula:

$$\begin{aligned}\varphi = \quad & \psi_\perp \wedge (a \to c) \wedge (b' \to c) \wedge (b' \to a') \\ & \wedge (\bar{a} \vee \bar{a'}) \wedge (\bar{b} \vee \bar{b'}) \\ & \wedge (a \vee a') \wedge (b \vee b') \wedge (c \to (a \vee b'))\end{aligned}$$

The set of runs described by φ is $[\![\varphi]\!] = \{\{\psi_\perp, a, b, c\}, \{\psi_\perp, a', b', c\}, \{\psi_\perp, a', b\}\}$. The binary immediate constraints are: $a \rhd_i b$, $a \rhd_i c$, $b' \rhd_i a'$, $b' \rhd_i c$, $b \rhd_i \psi_\perp$, $a' \rhd_i \psi_\perp$, $c \rhd_i \psi_\perp$, $a \#_i a'$ and $b \#_i b'$, and the net synthesized from these constraints is given in Fig. 7.2(d). We can see that this net is not an ON because there are two minimal events, c and $\psi_\perp$. Therefore, there is no reduced ON $\mathcal{N}$ such that $\varphi \equiv \Phi^{\mathcal{N}}$.



(a) $\Phi^{\mathrm{CN}(\varphi)} \equiv \varphi$ (b) $\Phi^{\mathrm{CN}(\varphi)} \not\equiv \varphi$ (c) $\Phi^{\mathrm{CN}(\varphi)} \not\equiv \varphi$ (d) $\mathrm{CN}(\varphi)$ is not an ON

Fig. 7.2: *(a): There is a reduced ON $\mathcal{N}$ such that $\varphi \equiv \Phi^{\mathcal{N}}$. (b) to (d): There is no reduced ON $\mathcal{N}$ such that $\varphi \equiv \Phi^{\mathcal{N}}$.*

Complexity

Identifying the immediate constraints requires looking at each pair of facets $\{a, b\} \subseteq \Psi(\varphi)$, and for each pair, deciding whether the formula $\varphi \to (a \to b)$ (respectively $\varphi \to (\neg a \vee \neg b)$) is a tautology is co-NP-complete.

Once the immediate constraints are computed, the number of places and arcs in $\mathrm{CN}(\varphi)$ is linear in the number of constraints, and therefore at most quadratic in the number of events. The events are simply the variables that appear in the formula. The quadratic bound is reached for a formula of the type $(\psi_1 \vee \cdots \vee \psi_n) \to (\psi'_1 \wedge \cdots \wedge \psi'_n)$ which implies $\psi_i \to \psi'_j$ for all i, j.

7.3 Going Further

7.3.1 Tightening a Reduced ON

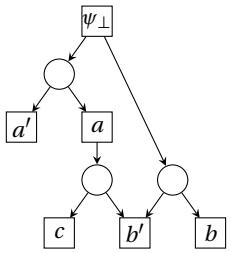
A simple corollary of our synthesis procedures is the following.

Corollary 72. Given any finite reduced ON $\mathcal{N}$, it is always possible to build a tight net $\mathcal{N}'$ such that $\Omega^{\mathcal{N}}_{max} = \Omega^{\mathcal{N}'}_{max}$.

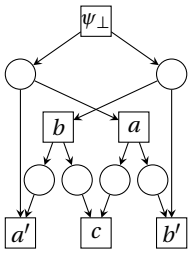
Proof. We can compute $\Phi^{\mathcal{N}}$ as in Subsection 7.2.1, and build the tight net $\mathcal{N}' = \mathrm{CN}(\Phi^{\mathcal{N}})$ as in Subsection 7.2.2. □

The example of Fig. 6.6, shows that the corollary does not hold in general if we drop the assumption of finiteness.

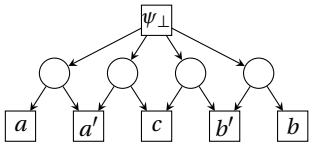
Ex 1. The initial reduced ON, $\mathcal{N}_1$, is depicted in Fig. 7.3(a). The set of maximal runs is $\Omega^{\mathcal{N}_1}_{max} = \{\{\psi_\perp, a, b, c\}, \{\psi_\perp, a, b'\}, \{\psi_\perp, a', b\}\}$ and the binary immediate constraints are $a \rhd_i \psi_\perp$, $b \rhd_i \psi_\perp$, $c \rhd_i a$, $c \rhd_i b$, $a' \rhd_i b$, $b' \rhd_i a$, $a \#_i a'$ and $b \#_i b'$. The canonical tight net obtained by the synthesis from these constraints is represented in Fig. 7.3(b).



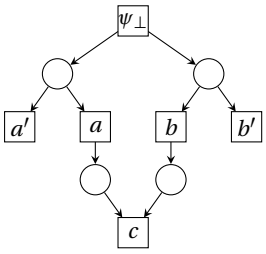
(a) Reduced ON $\mathcal{N}_1$



(b) Canonical tight net associated with $\mathcal{N}_1$



(c) Reduced ON $\mathcal{N}_2$



(d) Canonical tight net associated with $\mathcal{N}_2$

Fig. 7.3: *Examples of reduced ONs with their associated canonical tight net.*

Ex 2. Fig. 7.3(c) and 7.3(d) give another example of a reduced ON and its associated canonical tight net. The set of maximal runs is $\Omega_{max}^{\mathcal{N}_2} = \{\{\psi_\perp, a, b, c\}, \{\psi_\perp, a, b'\}, \{\psi_\perp, a', b\}, \{\psi_\perp, a', b'\}\}$ and the binary immediate constraints are $a \rhd_i \psi_\perp$, $b \rhd_i \psi_\perp$, $c \rhd_i a$, $c \rhd_i b$, $a \#_i a'$ and $b \#_i b'$.

It is a fact that the modifications brought about by (reduction and) tightening are often counter-intuitive and are unconventional net surgeries. At the same time, we believe that the "right" interpretation of these structural modifications should not be sought in the usual form of *temporal* properties. Rather, the reveals relations show *logical* dependencies that can be used for inference properties of the type "if a is known, then b must be the case". Thus the resulting net is in fact drastically changed, to better reflect which deductions are possible e.g. from a partial observation of behaviors.

7.3.2 Characterization of Adequate Formulae

There can be two reasons why an ERL formula φ does not describe the set of maximal runs of any ON: either the formula allows non-maximal runs, or it expresses non-binary minimal conflicts, while all minimal conflicts are binary in occurrence nets under the maximal semantics (see Lemma 64).

It is possible to characterize directly the formulas φ (or equivalently the sets Γ of sets of events) such that $[\![\varphi]\!]$ (respectively Γ) is the set of maximal runs of an ON.

Theorem 73 (Direct characterization of adequate Γ)**.** *Let E be a finite set whose elements are called events, and $\Gamma \subseteq 2^E$ such that any event occurs at least once in Γ, and one event denoted $\bot$ occurs in all the sets of Γ. Then, there exists an ON $\mathcal{N}$ such that $\Omega^{\mathcal{N}}_{max} = \Gamma$, iff*

$$\Gamma = \{\gamma \subseteq E \mid \forall a \in E,\ a \in \gamma \iff \#[a] \cap \gamma = \emptyset\}$$

where the $\#$ relation over E is defined as:

$$a \# b \overset{def}{\iff} \nexists \gamma \in \Gamma : \{a, b\} \subseteq \gamma\,.$$

Proof. By Prop. 60, any ON $\mathcal{N}$ satisfying $\Omega^{\mathcal{N}}_{max} = \Gamma$, has $\#$ as its conflict relation. Then by Lemma 51 Γ is its set of maximal runs.

Now, when $\Gamma = \{\gamma \subseteq E \mid \forall a \in E,\ a \in \gamma \iff \#[a] \cap \gamma = \emptyset\}$, we can define an occurrence net $\mathcal{N} = (B, E, F)$ whose set of events is E, whose set of conditions is $B \overset{def}{=} \{\{e, e'\} \mid e \# e'\}$ and whose flow relation F is defined such that $\bot^\bullet = B$ and for every $e \in E \setminus \{\bot\}$, ${}^\bullet e = \{\{e, e'\} \mid e \# e'\}$ and $e^\bullet = \emptyset$. $\mathcal{N}$ trivially satisfies the conditions for being an occurrence net. Moreover, its set of maximal runs coincides with Γ, by immediate application of Lemma 51. □

Remark 74. Let Γ be a set of sets of events satisfying the condition of Theorem 73. The occurrence nets $\mathcal{N}$ such that $\Omega^{\mathcal{N}}_{max} = \Gamma$ are reduced iff for all distinct events $a, b \in E$, $\Gamma \not\models a \leftrightarrow b$, or equivalently $\#[a] \neq \#[b]$. Indeed, combining the definition of facets and Lemma 50, we get that two events a and b are in the same facet iff $\#[a] = \#[b]$.

7.3.3 Untightened Synthesis

As well as runs are given as *unordered* sets of events, the syntax of ERL logic does not consider the *structural causality* between events. Therefore, the synthesis problem that we solve in Section 7.2 mentions only the logical dependencies between events and not the structural ones. This means that the causalities between events in the synthesized net, which represent the logical dependencies given by the formula, may come from causalities in the original net or from more complex dependencies involving the maximal progress assumption.

Indeed, we decided to represent the logical dependencies as causalities, and that is the reason why we get a tight net. However, we observed in Lemma 51 that the conflict relation gives enough information to define the maximal runs. That is, preserving the conflict relation is preserving the set of maximal runs. Hence, given a set of maximal runs, it is always possible to solve the synthesis problem by building a net with no causality (but the ones required by $\bot$) and only conflicts, like the ones used in the proof of Theorem 73. Fig. 7.3(c) shows an example of such ON. However, with this construction, the reveals relations in the resulting net are all hidden in the conflict relation, whereas our net CN(φ) makes explicit all the binary reveals relations as causalities, which lets us represent as little direct conflicts as possible, i.e. only the immediate conflict.

Between these two choices of representation there is a range of other possible choices which differ by the chosen causality relation (and therefore also by the conflict relation). The point is now to characterize the acceptable choices for the direct causality relation to impose in the net. To answer this question, we introduce a synthesis where a relation $\preceq$ is given, together with a formula φ. The synthesis problem is now: given an ERL formula φ on a set E of events (containing $\bot$) and a partial order relation $\preceq$ on E, is there an ON $\mathcal{N}$ whose behavior is the one specified by φ, and such that the causality in $\mathcal{N}$ matches $\preceq$?

In order to solve this synthesis problem, we adapt the construction $\mathrm{CN}(\varphi)$ of Def. 68 in order to represent only the causalities described by $\preceq$: for each pair of events (e, e') in the transitive reduction $\prec_i$ of $\preceq$, a condition b is created and connected to e and e' (${}^\bullet b = \{e\}$ and $b^\bullet = \{e'\}$). Then, we want to represent as few direct conflicts as possible w.r.t. this imposed causality, and in order to adapt our construction, we define the direct conflict of our synthesized net, similarly to the immediate conflict, but with $\preceq$ instead of $\rhd$.

$$a \#_d b \overset{def}{\iff} a \# b \wedge \nexists c : (c \prec a \wedge c \# b) \vee (c \prec b \wedge c \# a)$$

where $\prec$ denotes the reflexive reduction of $\preceq$.

Notice also that the general conflict relation can be defined with this direct conflict and the relation $\preceq$, as: $\# = \preceq \circ \#_d \circ \preceq^{-1}$. Therefore, the construction of Subsection 7.2.2 can be adapted by replacing $\#_i$ by $\#_d$ and $\rhd_i$ by $\prec_i^{-1}$.

Definition 75 ($\mathrm{CN}(\varphi, \preceq)$)**.** Let φ be an ERL formula over a set E of events (containing $\bot$) and $\preceq$ a partial order relation over E. $\mathrm{CN}(\varphi, \preceq) = (B, E, F)$ is the finite net where E is the set of events, $B = B_1 \cup B_2$ and $F = F_1 \cup F_2$, with:

- $B_1 = \big\{\{e, e'\} \mid e \#_d e'\big\}$,
- $F_1 = \big\{(\{e, e'\}, e) \in B_1 \times E\big\} \cup \big\{(\bot, \{e, e'\}) \in E \times B_1\big\}$.
- $B_2 = \prec_i$,
- $F_2 = \big\{\big((e, e'), e'\big) \in B_2 \times E\big\} \cup \big\{\big(e, (e, e')\big) \in E \times B_2\big\}$.

Then, Lemma 70 and Theorem 71 can be strengthened to:

Lemma 76. *Let $\mathcal{N}$ be a finite ON, and $\preceq$ a partial order relation on E such that $\bot$ is the only minimal event w.r.t. $\preceq$ and $\preceq$ is a subrelation of the reverse of the reveals relation of $\mathcal{N}$. Then $\mathrm{CN}(\Phi^{\mathcal{N}}, \preceq)$ is an ON and $\Phi^{\mathrm{CN}(\Phi^{\mathcal{N}}, \preceq)} \equiv \Phi^{\mathcal{N}}$ and the causality in $\mathcal{N}$ matches $\preceq$.*

Proof. The proof follows the steps of the proof of Lemma 70. □

Now comes the main result of this section, which states that, while synthesizing an ON $\mathcal{N}$ from an ERL formula φ, the causality in $\mathcal{N}$ (denoted $\preceq_{\mathcal{N}}$) can be freely chosen provided it is compatible with the reveals relation induced by φ.

Theorem 77. *Let φ be an adequate formula (i.e. a formula that describes the set of maximal runs of some ON). There exists an ON $\mathcal{N}$ such that $\Phi^{\mathcal{N}} \equiv \varphi$ and $\preceq_{\mathcal{N}} = \preceq$ for any partial order relation $\preceq$ on E, provided $\perp$ is the only minimal event w.r.t. $\preceq$, and $\preceq$ is a subrelation of the reverse reveals relation induced by φ.*

Proof. The existence of $\mathcal{N}$ trivially implies the required conditions on $\preceq$. The other direction is ensured by Lemma 76. □

As previously mentioned, there are two special cases of such synthesis:

- $\preceq = \rhd^{-1}$, then $\#_d = \#_i$ and the resulting net is a tight net.
- $\preceq$ relates simply $\perp$ to any event; then $\#_d = \#$ and the resulting net has no causality but the one linking any event to $\perp$.

Remark 78. For a given set of runs (or ERL formula), less causality implies more direct conflict in the synthesized net: $\preceq_1 \subseteq \preceq_2 \Rightarrow \#_{d2} \subseteq \#_{d1}$.

Ex 1. Consider again the reduced ON $\mathcal{N}_1$, depicted in Fig. 7.3(a). Its associated canonical tight net was built in Example 1. Define now $\preceq$ such that $\prec$ relates a to c and $\psi_\perp$ to every facet (except $\psi_\perp$). Our goal is to build $\mathrm{CN}(\Phi^{\mathcal{N}_1}, \preceq)$. The direct conflicts w.r.t. $\preceq$ are: $a \#_d a'$, $b \#_d b'$, $c \#_d b'$ and $a' \#_d b'$. The reduced ON $\mathcal{N}_1'$ obtained by the synthesis from these constraints is represented in Fig. 7.4(a).

If we define now another $\preceq$ that relates simply $\psi_\perp$ to the other facets, then the direct conflicts are the same as above plus $c \#_d b'$ (actually every conflict becomes direct). And the ON $\mathcal{N}_1''$ obtained by the synthesis from these constraints is represented in Fig. 7.4(b).

Ex 2. Consider now the reduced ON $\mathcal{N}_2$, depicted in Fig. 7.3(c). $\mathcal{N}_2$ is already the result of the synthesis with no causality (except causality from $\psi_\perp$ to every other facet).

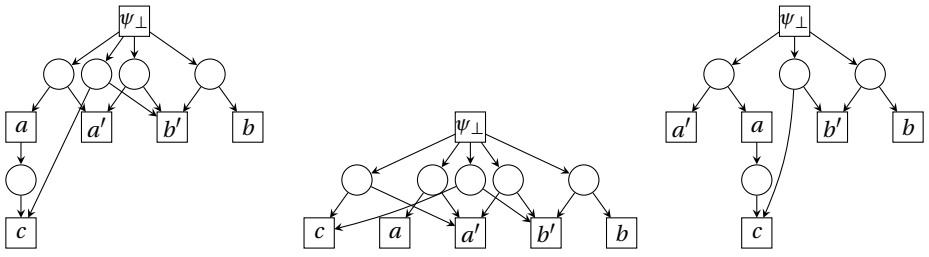
Define now $\preceq$ such that $\prec$ also relates a to c. Then the direct conflicts w.r.t. $\preceq$ are $a \#_d a'$, $b \#_d b'$ and $c \#_d b'$. The reduced ON obtained by the synthesis from these constraints is represented in Fig. 7.4(c).

Synthesis in the General Semantics

We have seen in Subsection 7.2.1 that the set of general runs of an occurrence net can be expressed as the following ERL.

$$\Phi^{\mathcal{N}}_{gen} = \underbrace{\bigwedge_{a,b\in E, a<b} (b \rightarrow a)}_{\text{causal closure}} \wedge \underbrace{\bigwedge_{a,b\in E, a\#b} (\neg a \vee \neg b)}_{\text{conflict-freeness}}$$

Now we show that the problem of synthesizing an occurrence net from an ERL formula can also be solved for the general semantics. More surprisingly,



(a) Reduced ON $\mathcal{N}_1'$ ($a \prec_i c$) (b) Reduced ON $\mathcal{N}_1''$ (c) Reduced ON $\mathcal{N}_2'$ ($a \prec_i c$)

Fig. 7.4: *Examples of synthesis parameterized by a causality relation $\preceq$. According to Def. 75, additional conditions and arcs should connect $\psi_\perp$ to other facets in order to code causality. They are omitted here since this causality is already induced by the conditions used to code the conflicts.*

the procedure for solving it is exactly the same as in Subsection 7.2.2 and Theorem 71 can be adapted.

Theorem 79. *Let φ be an ERL formula such that for any distinct variables $a, b \in \Psi(\varphi)$, $[\![\varphi]\!] \not\models a \leftrightarrow b$. There exists a finite reduced ON $\mathcal{N}$ such that $\Phi_{gen}^{\mathcal{N}} \equiv \varphi$ iff $\mathrm{CN}(\varphi)$ is a reduced ON and $\Phi_{gen}^{\mathrm{CN}(\varphi)} \equiv \varphi$.*

Proof. The steps described in 7.2.2 and 7.2.2, can be repeated. Then we prove that if $\mathcal{N}$ is a finite reduced ON, then $\mathrm{CN}(\Phi_{gen}^{\mathcal{N}})$ is a tight net and $\Phi_{gen}^{\mathrm{CN}(\Phi_{gen}^{\mathcal{N}})} \equiv \Phi_{gen}^{\mathcal{N}}$, as in the proof of Lemma 70, except that, in order to prove that $\mathcal{N}$ and $\mathrm{CN}(\Phi_{gen}^{\mathcal{N}})$ have equivalent formulas (i.e. the same set of general runs), we use the fact that they have the same conflict an causality relations. □

With the general semantics, the set of runs cannot be described with the conflict relation only. But since a net $\mathrm{CN}(\Phi^{\mathcal{N}})$, built from the formula associated with ON $\mathcal{N}$ has the same causality and conflict relations as $\mathcal{N}$, they accept the same set of general runs. Notice also that we have no longer the choice on the causality relation.

Remark 80. In the construction, the immediate conflicts are represented by a condition connected to $\psi_\perp$. This results in a large set of initial conditions. It is possible to improve the construction by representing each immediate conflict $\psi \#_i \psi'$ by a condition connected to any facet ψ_1 such that $\psi \rhd \psi_1$ and $\psi' \rhd \psi_1$. One possible choice would be to consider the $\rhd$-successors of ψ and ψ', defined as $\rhd[\psi, \psi'] = \{\psi_1 \in \Psi \mid \psi \rhd \psi_1 \wedge \psi' \rhd \psi_1\}$, create one condition b_1 for each $\rhd$-minimal facet, ψ_1, in $\rhd[\psi, \psi']$, and connect b_1 to ψ_1, ψ and ψ'. This would define B_1 and F_1. Then, any constraint of the form $\psi' \rhd_i \psi$ would be represented as previously by B_2 and F_2, except that, in B_2, we need to consider only non-redundant conditions. Indeed, if there exists $b \in B_1$ such that $(\psi, b) \in F_1 \wedge (b, \psi') \in F_1$, then $\psi' \rhd_i \psi$ is already represented and can be ignored in B_2.

7.3.4 Conclusion

We have shown how the structural and binary *reveals*-relation from [Haa10] generalizes into a relational framework for the description of *logical* dependencies - as opposed to *temporal* ones - between occurrences of sets of events in occurrence nets. For expressing these properties, a new logic, ERL, has been introduced and studied. In particular, we have solved the problem of synthesis for finite occurrence nets from ERL formulas. The extension to general occurrence nets is a future task, which is not trivial; see Fig. 6.6 and the discussion at the beginning of Section 7.2.

Even if ERL is a logic adapted for partial order semantics, it differs in its aim and structure from the other logics that have been proposed in the literature (for temporal logics for traces and event structures, see e.g. [GK10, Pen95]). First, ERL is not, strictly speaking, a *temporal* logic, since the notions of *before*, *after*, *future*, *until* etc. are of no particular relevance here; in fact, the progression of time is encapsulated in the underlying structure over which one chooses to interpret ERL formulas, and in the choice of admissible runs in that structure: maximal runs, any runs, runs satisfying additional context or timing constraints, etc. In the light of Subsection 7.3.3, causal ordering can be viewed as a refinement of the logical dependencies captured by the ERL formulas.

Thus far, we have intended and used the ERL logic as a means for coding and manipulating *structure* (of occurrence nets) and *knowledge* (observing A reveals B, i.e. gives knowledge about B's occurrence). The results here open some new roads towards efficient verification of system properties, as well as towards *enforcing* such properties through behavior control, or directly through synthesis of systems from logical specifications.

Towards the Timed Setting

So far, we have considered only Petri nets without timing constraints. We now want to extend this study of the logical dependencies between occurrences of events to a timed setting. In the next chapter, we consider a simple timed setting, where we focus on TPN whose firing intervals of transitions are punctual. Even in this setting, the dependencies are much more complex that in the untimed setting. We intend to use these dependencies to define a canonical and minimal unfolding technique for TPN.

Chapter 8

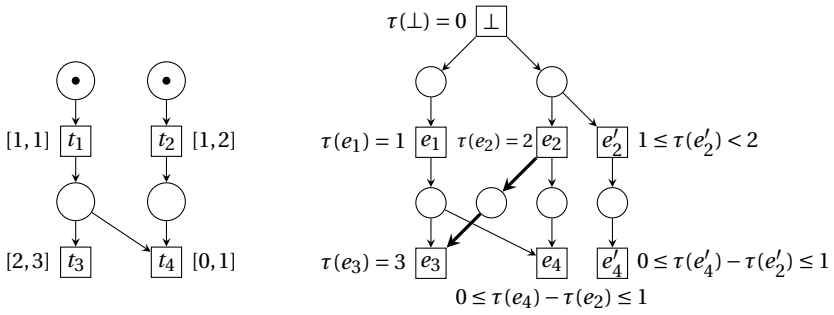
Dependencies between Event Occurrences in Timed Systems

In Subsection 2.3.3, we recalled timed processes introduced in [AL97] on an example that is drawn again in Fig. 8.1(a). We showed that there can be complex dependencies between event occurrences. For example, event e_3 can occur only if event e_2 occurs late enough (at time 2), because if e_2 occurs too early (in the time interval $[1,2)$), then event e_4 in conflict with e_3 will fire.

One way to represent these relations in an unfolding of the TPN of Fig. 8.1(a) is to duplicate events according to their different firing intervals and add new conditions and arcs to represents the causalities added by time. For example, in Fig. 8.1(b), e_2 and e_4 are duplicated, and the fact that e_3 can only occur if e_2 occurs at time 2 is represented as a new condition in the post-set of the version of e_2 that occurs at time 2 and in the pre-set of e_3. In [CJ06], these additional causal relations are represented by read arcs, and the processes are called *extended processes*. The dependencies between events in TPN unfoldings are mentioned in several works [AL97, CJ06, Tra09], but there is not yet a unified framework for their study and their representation.

In this chapter, we study the dependencies in a simplified timed setting, where we consider TPN whose time intervals are punctual. Therefore an event can occur only at a given time, and the time constraints of the unfolding are easier to compute. The unfolding of such TPN is an ON whose events have a fixed occurrence time. But, even in this setting, there are still complex dependencies between event occurrences, and the construction of valid time processes still requires some analysis. Indeed, some causal processes of the untimed ON do not correspond to a (prefix of) a process of the ON with dated events. This means that the prefixes of these causal processes may be extended in invalid processes if no special attention is paid.

In particular, we observe that for an event e to occur, the causal past of e is not always sufficient (see Fig. 8.2). Indeed, there may be some events in minimal conflict with e (like e' in Fig. 8.2) that become enabled and have to fire before e can fire. These events that can prevent e from firing when it is enabled are called *preventing events*. Therefore, we are interested in defining for an event



(a) A time Petri net (b) Its unfolding

Fig. 8.1: *An unfolding of the TPN of Fig. (a): events are duplicated, and new arcs represent the dependency "e_2 at time 2 is a cause of e_3 at time 3".*

e, the configurations that can be extended by e, i.e. the configurations whose final conditions enable e and such that for any extension of these configurations, it is possible to avoid the events that prevent e. These configurations contain the causal past of e and some other events with their causal past, that we call *enabling pasts* of e.

Organization of the Chapter We first give preliminary definitions, and present some properties of processes and pre-processes. Then we give alternative characterizations of processes and pre-processes, define the notion of enabling past, and relate it with the extended reveals relation. Lastly, we argue that the notion of minimal enabling past is a suitable notion for the definition of an algorithm that builds valid processes, since, in some sense, this notion is the counterpart of the notion of causal past in the untimed setting.

8.1 Preliminary Definitions

Let us first give the notations we will use, and present our motivation.

Notations For a given configuration ω of an ON $ON = (B, E, F)$, $Cut(\omega) = \omega^\bullet \setminus {}^\bullet\omega$ is the set of final conditions of ω, $En(Cut(\omega))$ is the set of events enabled by the final conditions of ω, and $En(\omega) = \{e \mid {}^\bullet{}^\bullet e \subseteq \omega\}$ is the set of events enabled along ω (for any ω, $\bot \in En(\omega)$). For an event e, $\lceil e \rceil$ is the causal past of e, $[e] = \lceil e \rceil \setminus \{e\}$ is the strict causal past of e, and $conf(e) \stackrel{def}{=} \{e' \in E \mid e' \; conf \; e\}$. For a set of events F, $conf(F) = \cup_{e \in F} conf(e)$. For a causal net $CN = (B_{CN}, E_{CN}, F_{CN})$ that is a prefix of ON, and a function f whose domain is E (resp. $B \cup E$), f_{CN} denotes the restriction of f to E_{CN} (resp. $B_{CN} \cup E_{CN}$).

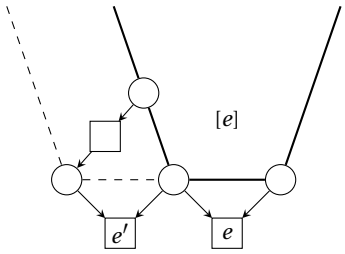


Fig. 8.2: *[e] is not sufficient to know whether e can occur, because e′ may have to fire before e becomes firable.*

8.1.1 Motivation

Consider Fig. 8.2 and assume the events of configuration $[e]$ have been assigned some firing times given by function τ. Then e is enabled at time $t_e = \max\{\tau(f) \mid f \in {}^{\bullet\bullet}e\}$ and has to fire between $t_e + efd(e)$ and $t_e + lfd(e)$, or to be disabled. However, assume now that $[e]$ can be extended with events assigned with time stamps, so that there exists an event e' such that e' *conf* e, that is now enabled by the final conditions, and such that $\max\{\tau(f) \mid f \in {}^{\bullet\bullet}e'\} + lfd(e') < t_e + efd(e)$. Then, from this configuration, e has no chance to fire unless an event can disable e'. Therefore, in order to extend $([e], \tau)$ with e and its occurrence time, we have to be sure that those events e' that can prevent e from firing can be avoided.

This observation leads us to define *preventing events*, *disabling events*, and *releasing events*, and shows the differences that exist between the untimed setting and the timed setting.

Untimed vs Timed Setting

- In the untimed setting, any configuration can be extended into a valid process. That is not true when timing constraints are considered: there are some configurations that are not included in any process. The notion of *pre-process* as been defined in [CJ06] to describe the configurations included in some process.

- In the untimed setting, any configuration that contains the causal past of an event e, and does not disable e can be extended by e. That is, the causal past of an event is the minimal configuration that ensures that e can fire. That is not true when time is considered. In this chapter, we will define *enabling pasts* as the notion associated with the causal pasts in the untimed setting.

8.1.2 Simplified Setting: Punctual Time Petri Nets

To simplify, we consider TPNs with punctual time intervals. This simplifies the unfolding and the computation of the valid timings, but still, defining a valid

process (see Subsection 2.3.3, Definition 18) requires some analysis because of dependencies and incompatibilities that time creates between events.

Punctual Time Petri Nets

A *punctual time Petri net* is a tuple (P, T, F, M_0, d) where (P, T, F, M_0) is a Petri net, and $d : T \to \mathbb{R}_{\geq 0}$ assigns a firing delay to each transition.

A *time branching process* associated with the punctual TPN N is a tuple (ON, π, τ), where (ON, π) is a branching process of N, $\tau : E \to \mathbb{R}_{\geq 0}$ associates an occurrence time with each event, and such that

1. for any event e of ON, there exists a causal process (CN, π_{CN}), where CN is a prefix of ON that contains e, and such that $(CN, \pi_{CN}, \tau_{CN})$ is a time process of N.

2. for any time process $(CN, \pi_{CN}, \tau_{CN})$ of N such that $\max(\tau_{CN}) \leq \max(\tau)$, CN is a prefix of ON.

The first condition means that any event e is firable in some execution. The second condition means that the time branching process represents all the possible executions that stop at time $\max(\tau)$ or before.

This structure resembles time branching processes of [Tra09], except that it is not supposed to represent one valid execution, but a set of executions, and there is no event e such that $\tau(e) = \infty$.

Lastly, for a configuration ω, we define $\tau(\omega) \stackrel{def}{=} \max\{\tau(e) \mid e \in \omega\}$.

Disabling and Preventing Events As previously mentioned, the minimal conflicts play an important role in computing the time constraints of valid processes. In our setting, we distinguish two kinds of minimal conflict according to the occurrence times of events. Below we formally define *disabling* and *preventing* events.

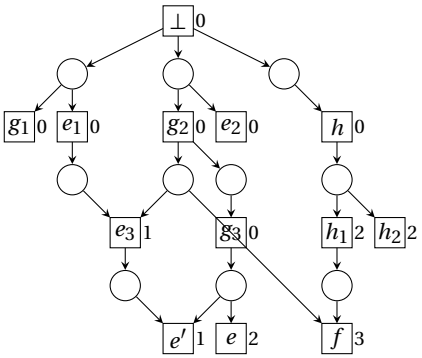
Definition 81. For any event e, we define the following sets:

- $Dis(e) \stackrel{def}{=} \{e' \mid e' \; conf \; e \wedge \tau(e') \leq \tau(e)\}$ is the set of events that can disable e.

- $Pre(e) \stackrel{def}{=} \{e' \mid e' \; conf \; e \wedge \tau(e') < \tau(e)\}$ is the set of events that can disable e and that e cannot disable.

For example, in Fig. 8.3, $Dis(e_1) = \{g_1\}$, $Pre(e) = \{e'\}$ and $Pre(f) = \{e_3\}$.

The set $Dis(e)$ represents the set of events that need to be considered to know whether event e can occur when it is enabled. In fact, any event e' such that $\tau(e') > \tau(e) \wedge e' \; conf \; e$ need not be considered because e' cannot disable e and therefore has no influence on e. If e does not occur in ω and e' occurs, then this means that another conflict with e has occurred (either e is never enabled or it is disabled by another direct conflict which is in $Dis(e)$).



Some configurations:
$\omega_1 = \{\bot, e_1, g_2, g_3, e, h, h_1, f\}$
$\omega_2 = \{\bot, e_1, g_2, g_3, e, h, h_1, e_3\}$
$\omega_3 = \{\bot, e_1, g_2, g_3, e', h, h_1, e_3\}$
$\omega_4 = \{\bot, g_1, g_2, g_3, h, e\}$

Fig. 8.3: *Not every configuration is a pre-process (ω_1 and ω_2 are not).*

Since for any events e and e', $e \; conf \; e' \iff e \in Dis(e') \vee e' \in Pre(e)$, the set $Pre(e)$ has a "parallel" meaning. It represents the set of events on which e has no influence. Hence, if $e' \in Pre(e)$ and e and e' are enabled, then e has no chance to fire unless another event e'', that can disable e' occurs (i.e. $e'' \in Dis(e')$), but the occurrence of e'' will be known before $\tau(e)$.

Later we show that, although the knowledge of $[e]$ may not be sufficient to deduce whether e is able to fire, we can always come to such a deduction with the knowledge of $[e]$ and *some events that occur strictly earlier than e*. This result is stated in Lemma 100. Let us now define processes and pre-processes of a time branching process.

Processes and Pre-Processes

Causal processes represent the executions of untimed branching processes. The executions of time branching processes are represented by *time processes*. Here, we do not need to consider the occurrence times, since they are fixed. To simplify the notations, we also omit conditions and arcs, and define *processes* as configurations associated with time processes. That is, processes are configurations satisfying additional properties that we formalize in the definition below, which is an adaptation of the definition of time processes of [AL97].

Definition 82 (Process, Pre-Process)**.** Let $\omega \subseteq E$ be a configuration of a time branching process, then ω is a *process* iff it satisfies the following condition:

$$\forall e \in En(\omega), e \notin \omega \implies \omega \cap Dis(e) \neq \emptyset \vee \tau(e) \geq \tau(\omega)$$

A *pre-process* is a configuration included in a process.

This means that, any event e enabled in a process ω and that has not fired was either disabled before its firing time by the firing of a minimal conflict (i.e. by an event in $Dis(e)$), or has not yet overtaken its firing time.

We observe that some configurations (i.e. some processes of the untimed branching process) are not pre-processes of the time branching process. This

means that the pre-processes that are included in these configurations may be extended in invalid processes if no special attention is paid. For example, the configuration ω_1 of Fig. 8.3 is not a process because e_3 was enabled and should have fired before f. A "consistent" pre-process that contains f has to contain g_1 to ensure that e_3 will not fire.

Temporal Completeness An intrinsic property of time processes, the *temporal completeness*, was recalled in Subsection 2.3.3. Below, we give a simplified definition in our setting, and recall that processes are temporally complete, i.e. that for any process ω, events that are not in ω are in conflict with ω or have an occurrence time greater than or equal to $\tau(\omega)$ (the time until which ω has progressed). Hence any event e which is not in conflict with ω and whose occurrence time is less than $\tau(\omega)$ is necessarily in ω (see Lemma 87 below).

Definition 83 (Temporal Completeness)**.** Let ω be a configuration. ω is *temporally complete* iff for any event $e \in En(Cut(\omega))$, $\tau(e) \geq \tau(\omega)$. We write $TempComp(\omega)$ when ω is temporally complete.

We also define the temporal completeness of a process ω until a given time $\theta \geq \tau(\omega)$: ω is (temporally) complete until θ if we can delay in the final configuration of ω until time θ.

Definition 84 (Temporal Completeness until θ)**.** A process ω is temporally complete until time θ such that $\theta \geq \tau(\omega)$ iff $\forall e \in En(Cut(\omega)), \tau(e) \geq \theta$.

Next Events For a configuration ω, we now consider the events that are enabled by the final conditions of ω and that should fire if ω was temporally complete. We define the following set: $Next(\omega) \stackrel{def}{=} \{e \mid e \in En(Cut(\omega)) \wedge \tau(e) < \tau(\omega)\}$.

Remark 85. Notice that $Next(\omega) = \emptyset$ iff ω is temporally complete.

Lemma 86 ([AL97])**.** *Any process is temporally complete.*

Lemma 87 (Temporal Completeness)**.**

1. *Let ω be a configuration, then $TempComp(\omega)$ is equivalent to*
   $\forall e \in E, (\omega \cap \#[e] = \emptyset \wedge \tau(e) < \tau(\omega)) \implies e \in \omega$

2. *For any process ω,*
   $\forall e \in E, (\omega \cap \#[[e]] = \emptyset \wedge \omega \cap Dis(e) = \emptyset \wedge \tau(e) < \tau(\omega)) \implies e \in \omega$

Proof. (1) Assume that for some event e, $\omega \cap \#[e] = \emptyset \wedge \tau(e) < \tau(\omega) \wedge e \notin \omega$. Then, since $\omega \cap \#[e] = \emptyset$ and $e \notin \omega$, there exists an event $f \in \lceil e \rceil$ such that $f \in En(Cut(\omega))$. Moreover f is such that $\tau(f) \leq \tau(e) < \tau(\omega)$, which means that $TempComp(\omega)$ does not hold.

In the other direction, for any $f \in En(Cut(\omega))$, both $\omega \cap \#[f] = \emptyset$ and $f \notin \omega$. Assume the right part of the equivalence holds, then $\tau(f) \geq \tau(\omega)$. This implies $TempComp(\omega)$.

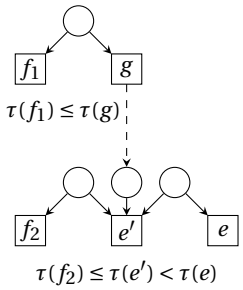


Fig. 8.4: *Any process that contains e also contains f_1 or f_2.*

(2) First, we prove that for any process ω, $\forall e \in E, \omega \cap \#[[e]] = \emptyset \wedge \tau(e) < \tau(\omega) \Rightarrow e \in En(\omega)$. Assume there exists $e \in E$ such that

$$\omega \cap \#[[e]] = \emptyset \wedge \tau(e) < \tau(\omega) \wedge e \notin En(\omega) \qquad (H(e))$$

Then, since $e \notin En(\omega)$, there exists $e' \in {}^{\bullet\bullet}e$ such that $e' \notin \omega$. From the definition of a process, we have $e' \notin \omega \Rightarrow \big(e' \notin En(\omega)\big) \vee \big(\omega \cap Dis(e') \neq \emptyset\big) \vee \big(\tau(e') \geq \tau(\omega)\big)$. Since $\tau(e') \leq \tau(e) < \tau(\omega)$, and $Dis(e') \subseteq \#[[e]]$, this can be simplified in $e' \notin \omega \Rightarrow e' \notin En(\omega)$, thus, e' is such that $\omega \cap \#[[e']] = \emptyset \wedge \tau(e') < \tau(\omega) \wedge e' \notin En(\omega)$, i.e. e' satisfies $H(e')$. That is, if e satisfies $H(e)$, then $\exists e' \in {}^{\bullet\bullet}e$ s.t. e' satisfies $H(e')$. By recursion, we finally get that if e satisfies $H(e)$, then $\perp$ satisfies $H(\perp)$, which it does not because $\perp \in En(\omega)$. Therefore, $\forall e \in E, \omega \cap \#[[e]] = \emptyset \wedge \tau(e) < \tau(\omega) \Rightarrow e \in En(\omega)$.

Hence, $\omega \cap \#[[e]] = \emptyset \wedge \omega \cap Dis(e) = \emptyset \wedge \tau(e) < \tau(\omega)$ implies $e \in En(\omega) \wedge \omega \cap Dis(e) = \emptyset \wedge \tau(e) < \tau(\omega)$, which implies $e \in \omega$ by definition of a process. □

Remark 88. Similarly, if ω is a process that is temporally complete until time θ, then $\forall e \in E, (\omega \cap \#[[e]] = \emptyset \wedge \omega \cap Dis(e) = \emptyset \wedge \tau(e) < \theta) \Rightarrow e \in \omega$. The proof is similar to the proof of Lemma 87. This means that $\forall e \in En(\omega)$, $e \notin \omega$ implies $\omega \cap Dis(e) \neq \emptyset$ or $\tau(e) \geq \theta$. That is, for any event e enabled in ω, if e has not occurred, then either e was disabled, or the occurrence time of e is greater than the time θ until which ω has progressed.

Releasing Events Another property of processes is that, for any event e that occurs, the preventing events of e are disabled. Indeed, if one event in $Pre(e)$ is not disabled, then it will be enabled before e, and will fire. We distinguish two ways of disabling a preventing event e': either a minimal conflict with an event g in the strict causal past of e' occurred and e' was never enabled (we will prove that we can consider that this minimal conflict is in $Dis(g)$) or e' was enabled, but an event in $Dis(e')$ occurred. These two cases are depicted in Fig. 8.4. Releasing events will have a special interest as stated in Theorem 91 below. After this theorem, we also argue that these events also ensure that a configuration is in some sense a "consistent" pre-process.

Definition 89 (Releasing Events)**.** For any events e and e' such that $e' \in Pre(e)$, the set of events that *release* e from e' is defined as follows.

$$\begin{aligned} Rel(e,e') &\stackrel{def}{=} \left\{f \in E \mid \left(\exists g \in \lfloor e' \rfloor : f \in Dis(g)\right) \vee f \in Dis(e')\right\} \\ &= \left\{f \in E \mid \exists g \in \lceil e' \rceil : f \in Dis(g)\right\} \end{aligned}$$

For a configuration ω, we also define the following predicates:

$$\begin{aligned} Release(\omega,e) &\stackrel{def}{\iff} \forall e' \in Pre(e), \omega \cap Rel(e,e') \neq \emptyset \\ Release(\omega) &\stackrel{def}{\iff} \forall e \in \omega, Release(\omega,e) \end{aligned}$$

Therefore, $Release(\omega,e)$ means that for any event $e' \in Pre(e)$, configuration ω contains an event that releases e from e'. For example, in Fig. 8.4, $Rel(e,e') = \{f_1, f_2\}$, and a configuration ω satisfies $Release(\omega,e)$ iff it contains f_1 or f_2.

8.2 Study of Dependencies between Events

In this section, thanks to the notions we defined above, we study the dependencies between events in the unfolding of punctual TPNs. We first give alternative definitions of a process, and show that the releasing events are particularly interesting because they ensure that a configuration is valid (i.e. is a pre-process). We then define enabling pasts of an event e as pre-processes that ensure that a configuration can be extended by event e, and study their properties. Lastly, we relate enabling past and the extended reveals relation defined over the set of processes of a punctual TPN.

8.2.1 Characterization of Processes and Pre-processes

First, let us highlight the interest of disabling events and show that the other minimal conflicts can be ignored.

Lemma 90 (Disabling Events)**.** *For any process ω, for any events $e \in \omega$ and $e' \in Pre(e)$, there exists an event $e'' \leq e'$ such that $\omega \cap Dis(e'') \neq \emptyset$.*

Proof. By contradiction, assume that there exists $e' \in Pre(e)$ such that for any $e'' \leq e'$, $\omega \cap Dis(e'') = \emptyset$. Since $e' \notin \omega$ and $\tau(e') < \tau(\omega)$, by temporal completeness there exists $e'' \leq e'$ such that for some $f \in \omega$, f *conf* e''. Moreover, by assumption, $f \notin Dis(e'')$, i.e. $e'' \in Pre(f)$.

By definition of a process, for any $f \in \omega$ and $e'' \in Pre(f)$, $e'' \notin \omega$ (conflict-freeness). But any such e'' is such that $\tau(e'') < \tau(\omega)$, and since $TempComp(\omega)$ (by Lemma 86), $\omega \cap \#[[e'']] \neq \emptyset \vee \omega \cap Dis(e'') \neq \emptyset$ (by Lemma 87.2). By assumption, $\omega \cap Dis(e'') = \emptyset$, therefore there exists $e''' < e''$ such that for some $f' \in \omega$, f' *conf* e''' and $e''' \in Pre(f')$ (again because by assumption, $\omega \cap Dis(e''') = \emptyset$). Let us denote by $P(e'')$ the following property: $\exists e''' < e'', f' \in \omega : e''' \in Pre(f')$. Using again the definition of a process and the first assumption, we get that e''' also satisfies $P(e''')$. By recursion, we finally arrive at a direct successor of $\perp$ and see that the property cannot be satisfied since $\perp$ has no conflict. □

We are now ready to give alternative definitions of a process.

Theorem 91 (Process, alternative definitions)**.** *Let ω be a configuration, then the following statements are equivalent:*

1. *ω is a process*
2. *$TempComp(\omega) \wedge Release(\omega)$*
3. *$TempComp(\omega) \wedge \big(\forall e \in \omega, \forall e' \in Pre(e), e' \notin En(\omega) \vee \omega \cap Dis(e') \neq \emptyset\big)$*
4. *$TempComp(\omega)$ and ω is a pre-process*

Proof. $1 \Rightarrow 2$ is direct from Lemma 86 and Lemma 90, and $1 \Rightarrow 4$ is direct from Lemma 86. Below, we first show that $2 \Rightarrow 3 \Rightarrow 1$, then that $4 \Rightarrow 1$.

($2 \Rightarrow 3$) $Release(\omega)$ implies that $\forall e \in \omega, \forall e' \in Pre(e), \omega \cap conf([e']) \neq \emptyset \vee \omega \cap Dis(e') \neq \emptyset$. For any e', $\omega \cap conf([e']) \neq \emptyset$ implies $e' \notin En(\omega)$.

($3 \Rightarrow 1$) By Lemma 87.1, the temporal completeness is equivalent to $\forall e, e \notin \omega \Rightarrow \omega \cap \#[e] \neq \emptyset \vee \tau(e) \geq \tau(\omega)$. Thus, $\forall e \in En(\omega), e \notin \omega \Rightarrow conf(e) \cap \omega \neq \emptyset \vee \tau(e) \geq \tau(\omega)$. That is, for any e such that $e \in En(\omega) \wedge e \notin \omega \wedge \tau(e) < \tau(\omega)$, $conf(e) \cap \omega$ is not empty, and it remains to show that $\omega \cap Dis(e)$ is not empty either. If $Dis(e) \cap \omega = conf(e) \cap \omega$, then it is not empty, and otherwise, $\exists e_1 \in (conf(e) \cap \omega) \setminus Dis(e)$, i.e. $e \in Pre(e_1)$ and we can use the second term of the conjunction of item 3 by considering e_1 instead of e: $\forall e_1 \in \omega, \forall e \in Pre(e_1), e \in En(\omega) \Rightarrow \omega \cap Dis(e) \neq \emptyset$.

($4 \Rightarrow 1$) Assume ω is a temporally complete pre-process. Then, for any $e \in En(\omega) \setminus \omega$, $\omega \cap conf(e) \neq \emptyset \vee \tau(e) \geq \tau(\omega)$ (temporal completeness). We want to show that ω is a process, i.e. that for any $e \in En(\omega) \setminus \omega$, $\omega \cap Dis(e) \neq \emptyset \vee \tau(e) \geq \tau(\omega)$. If $\omega \cap Dis(e) = \omega \cap conf(e)$ then we have finished. Otherwise, there exists e_1 in $\omega \cap (conf(e) \setminus Dis(e))$ (i.e. $e \in Pre(e_1)$). Therefore, any process ρ such that $\omega \subseteq \rho$ contains e_1 and satisfies item 3, i.e. $\rho \cap Dis(e) \neq \emptyset$. For any $e' \in \rho \cap Dis(e)$, $\rho \cap \#[e'] = \emptyset$ because ρ is conflict-free, and hence $\omega \cap \#[e'] = \emptyset$ also. That is, e' satisfies $\omega \cap \#[e'] = \emptyset$ and $\tau(e') \leq \tau(e) < \tau(e_1) \leq \tau(\omega)$, and, since ω is temporally complete, e' is also in ω. Hence, $\omega \cap Dis(e) \neq \emptyset$ and ω is a process. □

For example, configuration ω_1 of Fig. 8.3 is temporally complete but is not a process because $e \in \omega_1$, and there is no event in ω_1 that releases e from $e' \in Pre(e)$.

Sufficient Condition for a Configuration to be a Pre-process

We show below in Lemma 93 that $Release(\omega)$ ensures that a configuration ω is a pre-process. Moreover $Release(\omega)$ is a local property that can be checked easily on a configuration, contrary to the non local conditions given in [CJ06]. With this condition, it is easy to build valid pre-processes by extending configurations. We first prove the following proposition that will help us prove Lemma 93.

Proposition 92. *Let ω be a configuration such that $Release(\omega)$. Then, for any event e, $\omega \cap \#[e] \neq \emptyset \Rightarrow \exists g \leq e : \omega \cap Dis(g) \neq \emptyset$*

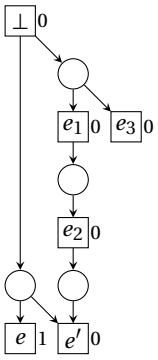


Fig. 8.5: *$\{\bot, e, e_1\}$ is a configuration but is not a pre-process (no process contains both e and e_1).*

Proof. By contradiction, assume that for some e, $\omega \cap \#[e] \neq \emptyset$ and $\forall g \leq e : \omega \cap Dis(g) = \emptyset$. Then, $\exists g \leq e, f \in \omega : g \in Pre(f)$. But since $Release(\omega, f)$, $\exists g_1 \leq g : \omega \cap Dis(g_1) \neq \emptyset$, which contradicts the assumption (because $g_1 \leq e$ also). □

Lemma 93. *Let ω be a configuration. If $Release(\omega)$, then ω is a pre-process.*

Proof. Let us denote by t_0 the time $\min\{\tau(g) \mid g \in Next(\omega)\}$. First, we show that for any event f such that $f \in Next(\omega)$ and $\tau(f) = t_0$, $Release(\omega, f)$ holds. Indeed, $\forall f' \in Pre(f), f' \notin En(Cut(\omega))$ (otherwise $f' \in Next(\omega)$ and $\tau(f) \neq t_0$) hence $\omega \cap \#[f'] \neq \emptyset$ (otherwise, $\exists f'' \in [f'] : f'' \in En(Cut(\omega))$ and f would not be such that $\tau(f) = t_0$). By Proposition 92, $\forall f' \in Pre(f), \exists f'' \leq f' : \omega \cap Dis(f'') \neq \emptyset$. Therefore $Release(\omega, f)$ holds, and so does $Release(\omega \cup \{f\})$.

If ω is a process, then ω is a pre-process. Assume ω is not a process, then ω is not temporally complete, i.e. the set $Next(\omega)$ is not empty (see Remark 85). Moreover, for any $e \in Next(\omega)$, $\omega \cap \#[e] = \emptyset$. We build a process containing ω by adding events to ω: while $TempComp(\omega)$ does not hold , take $f \in Next(\omega)$ such that $\tau(f) = t_0$ (then $Release(\omega \cup \{f\})$ holds) and add it to ω. □

The other direction of the above lemma is not true. For instance, for any event e, $\lceil e \rceil$ is a pre-process and it may not satisfy $Release(\lceil e \rceil)$ (see Fig. 8.5). However, for a pre-process ω, $Release(\omega)$ is an interesting property because it ensures a kind of "consistency" of the pre-process: for any event e in ω, ω contains a sufficient information to know that e can fire. When $Release(\omega)$ does not hold, this consistency may not be ensured, for example, in Fig. 8.5 $\lceil e \rceil$ does not satisfies $Release(\lceil e \rceil)$, $e_3 \notin \lceil e \rceil$ and e needs e_3 to be able to fire.

8.2.2 Enabling Pasts

Contrary to what happens when untimed ON are considered, here $[e]$ alone does not always ensure that e is actually able to occur. Indeed, when the final conditions of a pre-process ω enable e, this does not mean that $\omega \cup \{e\}$ is a pre-process

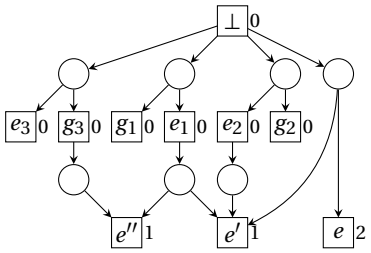


Fig. 8.6: *$\subseteq$-minimal enabling pasts of e:* $\{\bot, g_1\}$, $\{\bot, g_2\}$ *and* $\{\bot, g_3\}$.

(see again $\{\bot, e, e_1\}$ in Fig. 8.5). That is why we want to identify the information that ω has to contain, besides $\lceil e \rceil$, so that $\omega \cup \{e\}$ is a pre-process. Below, we formally define the notion of *enabling past* of e, which is a sufficient information which ensures that e can occur: if a pre-process ω, whose final conditions enable e, contains an enabling past of e, then $\omega \cup \{e\}$ is a pre-process. Therefore, this notion is the counterpart of the notion of causal past in the untimed case.

Definition 94 (Enabling Past)**.** A pre-process E is an *enabling past* of an event e iff for any pre-process ω, $E \subseteq \omega \wedge e \in En(Cut(\omega)) \implies \omega \cup \{e\}$ is a pre-process.

For example, in Fig. 8.3, the $\subseteq$-minimal enabling pasts of f are $\lceil g_1 \rceil$ and $\lceil e \rceil$, and those of e are $\lceil g_1 \rceil$ and $\lceil f \rceil$. However, for any events, e and $e' \in Pre(e)$, observe that an enabling past of e need not disable e' if it ensures the enabling of a disabling event of e'. For example, in Fig. 8.6, $\{\bot, g_3\}$ is an enabling past of e because in any pre-process that enables both e and e', if g_3 occurs then e'' is also enabled and can disable e'.

Remark 95. From the definition, we readily obtain the following properties of enabling pasts.

1. For any event e such that $Pre(e) = \emptyset$, any pre-process (in particular $\emptyset$) is an enabling past of e.
2. For any pre-process E, $E \cup \lceil e \rceil$ is an enabling past of e iff E is an enabling past of e.
3. If E is an enabling past of e, then any pre-process E' such that $E \subseteq E'$ is an enabling past of e.
4. Any process that contains e also contains an enabling past of e.

With the example of in Fig. 8.3, we see that even a $\subseteq$-minimal enabling past of e may reach a time greater than the occurrence time of e: $\lceil f \rceil$ is a $\subseteq$-minimal enabling past of e, and $\tau(f) > \tau(e)$. We consider that these enabling pasts are not satisfactory, and we observe that, by item 4 of Remark 95 above, a process that contains e always contains an enabling past of e, i.e. always contains an enabling past E of e such that $\tau(E) \leq \tau(e)$ (the process may stop at time $\tau(e)$).

Moreover, we will strengthen this result and show that a process that contains e always contains an enabling past E of e such that $\tau(E) < \tau(e)$ (see Lemma 100 below).

First, let us extend item 4 of Remark 95 with the following lemma.

Lemma 96 (Enabling Process)**.** *For any event e, any process π which is complete until $\tau(e)$ and such that $\pi \cap \#[e] = \emptyset$ is an enabling past of e.*

Proof. First, we show that any process π which is complete until $\tau(e)$ and such that $e \in En(Cut(\pi))$ is an enabling past of e. $Pre(e) \cap \pi = \emptyset$ (otherwise e would not be enabled by the final conditions of π), that is for any $e' \in Pre(e)$, $e' \notin En(\pi)$ or $\pi \cap Dis(e') \neq \emptyset$ (see Remark 88). Hence, $\pi \cup \{e\}$ is also a process (it satisfies item 3 of Theorem 91) and π is an enabling past of e.

Second, if π is complete until $\tau(e)$ and such that $\pi \cap \#[e] = \emptyset$, then for any pre-process ω, $\pi \subseteq \omega \wedge e \in En(Cut(\omega))$ implies that there exists a process π' such that $\pi \subseteq \pi' \subseteq \omega$ and $e \in En(Cut(\pi'))$, hence, π' is an enabling past of e (as proved above) and $\omega \cup \{e\}$ is a pre-process. Thus π is also an enabling past of e. □

Proposition 97 (Enabling Past, sufficient condition)**.** *Let E be a pre-process and e an event. If for any pre-process ω, $E \subseteq \omega \implies \omega \cap Pre(e) = \emptyset$, then E is an enabling past of e. In particular, if $Release(E, e)$ then E is an enabling past of e.*

Proof. Let pre-process E be such that for any pre-process ω, $E \subseteq \omega \Rightarrow \omega \cap Pre(e) = \emptyset$. Then, for any pre-process ω such that $E \subseteq \omega \wedge e \in En(Cut(\omega))$, for any process ρ such that $\omega \subseteq \rho$, either (i) $e \in \rho$ or (ii) $e \in En(Cut(\rho))$ or (iii) $\exists f \in \rho : f \ conf \ e$. Below we study these three cases.

(i) $\omega \cup \{e\} \subseteq \rho$ is a pre-process.

(ii) Since $E \subseteq \rho$, no event in $Pre(e)$ can occur in ρ or any extension of ρ. Thus there exists a process ρ' such that $\rho \subseteq \rho'$, $e \in En(Cut(\rho'))$ and ρ' is complete until $\tau(e)$. Then, since $\rho' \cap \#[e] = \emptyset$, by Lemma 96, ρ' is an enabling past of e. Hence, $\rho' \cup \{e\}$ is a pre-process and so is $\omega \cup \{e\} \subseteq \rho' \cup \{e\}$.

(iii) Since $E \subseteq \rho$, $f \notin Pre(e)$, i.e. $\tau(f) \geq \tau(e)$. Therefore, $\tau(\omega) \geq \tau(e)$ and by Lemma 87, $\forall e' \in Pre(e)$, since $\tau(e') < \tau(e) \leq \tau(\rho)$, $\rho \cap \#[[e']] \neq \emptyset$ or $\rho \cap Dis(e') \neq \emptyset$. Hence $Release(\rho, e)$. Thus $\rho' = \rho \setminus \lfloor conf(e) \rfloor$ (ρ without the causal future of the minimal conflicts with e) also satisfies $Release(\rho', e)$, and $\rho' \cup \{e\}$ satisfies $Release(\rho' \cup \{e\})$ and is, by Lemma 93, a pre-process. Therefore, $\omega \cup \{e\} \subseteq \rho' \cup \{e\}$ is also a pre-process.

Therefore E is an enabling past of e. □

These conditions are sufficient but not necessary. For instance, consider Fig. 8.6, where $E = \{\bot, g_3\}$ is an enabling past of e and $\{\bot, g_3, e_1, e_2, e'\}$ is a process, therefore there exists a pre-process which contains E, $\lceil e \rceil$ and $e' \in Pre(e)$. Indeed, the point of an enabling past is not to make all events in $Pre(e)$ impossible, but to make them avoidable. In the example of Fig. 8.6, if e_1, e_2 and e_3 occur, then e' cannot be avoided, therefore, an enabling past of e has to make impossible at least one of these events.

Minimal Enabling Pasts

Since enabling pasts are preserved by addition of events (Remark 95.3), there can be many enabling pasts for a given event e (even infinitely many if the unfolding is infinite). Therefore, in order to consider only relevant enabling pasts, we need a notion of minimality. We observe that the minimality by inclusion is not appropriate. Indeed, if we consider event e, in Fig. 8.3, we see that $\lceil g_1 \rceil$ and $\lceil f \rceil$ are $\subseteq$-minimal enabling pasts of e. But, in any process, if f occurs, g_1 also occurs and furthermore, $\tau(\lceil g_1 \rceil) < \tau(\lceil f \rceil)$. Therefore, we prefer $\lceil g_1 \rceil$ to $\lceil f \rceil$ and we consider $\lceil f \rceil$ as non minimal.

Definition 98 (Minimal Enabling Past)**.** Let E be a $\subseteq$-minimal enabling past of an event e. E is a *minimal* enabling past of e iff there exists a process ω which contains E and e and does not contain another enabling past of e, E' such that $\tau(E') < \tau(E)$.

For example, in Fig. 8.3, $\lceil f \rceil$ is a $\subseteq$-minimal enabling past of e but it is not a minimal enabling past of e. Indeed, in any process that contains f and e, g_1 also occurs and $\lceil g_1 \rceil$ is an enabling past of e such that $\tau(\lceil g_1 \rceil) < \tau(\lceil f \rceil)$. In this example, $\lceil g_1 \rceil$ is the only minimal enabling past of e.

Observe that, an enabling past of e may not contain a minimal enabling past of e. However, by definition, we directly have the following proposition.

Proposition 99. *For any event e, any process ω which contains e also contains a minimal enabling past of e.*

Lastly, we can state the following proposition.

Lemma 100. *For any event e and any minimal enabling past E of e, $\tau(E) < \tau(e)$.*

Proof. Consider any process ω that contains e. By Lemma 90 ,for any event e' in $Pre(e)$, there exists $f_{e'} \in \omega \cap Dis(e'')$, for some $e'' \leq e'$. Hence, for any such $f_{e'}$, $\tau(f_{e'}) \leq \tau(e'') \leq \tau(e') < \tau(e)$. Then, $E = \bigcup_{e' \in Pre(e)} \lceil f_{e'} \rceil$ is an enabling past of e (Proposition 97), and is such that $\tau(E) < \tau(e)$. Moreover, any minimal enabling past is included in some such enabling past. □

Furthermore, if the common assumption of non-zenoness is made, the following proposition holds.

Proposition 101. *For any event e, any minimal enabling past of e is finite.*

We can now relate this notion of enabling past with the reveals relation that we define on the set of processes of a punctual TPN. Indeed, with the interesting sets of events and predicates we have defined above, we understand better the relations between events in time branching processes, i.e. the relations between events in the processes of punctual TPN.

8.2.3 Reveals Relation in Punctual Time Petri Nets

We defined and studied the reveals relation and the extended reveals relation in Chapter 7, in an untimed setting. Time implies new dependencies not visible in the structure of the occurrence net. Independent (concurrent and not $\triangleright$-related) events in the untimed branching process may be logically dependent or incompatible in the time branching process. Indeed, a process of the untimed branching process may not be a process of the time branching process, because some events that are not in structural conflict are no longer compatible when timing constraints are added.

This is what we observe for example in Fig. 8.5, where e and e_1 would be independent without timing constraints, and are incompatible with time constraints: if e_1 occurs, e_2 and e' also occur, and e' disables e.

First, we define $\triangleright$ and $\rightarrowtriangle$ as previously in Definitions 45 and 59, but over the set of time processes Ω_{time}.

Definition 102 (Reveals and Extended Reveals Relations)**.**

- For any events a and b, $a \triangleright b$ iff for any process $\omega \in \Omega_{time}$, $a \in \omega \implies b \in \omega$.
- For any sets of events A and B, $A \rightarrowtriangle B$ iff for any process $\omega \in \Omega_{time}$, $A \subseteq \omega \implies B \cap \omega \neq \emptyset$.

As a special case of this definition, $A \rightarrowtriangle \emptyset$ means that there is no process ω that contains A.

Since this definition is similar to the definition over the runs of an occurrence net, the "logical" properties of the reveals and the extended reveals still hold. For example, $\triangleright$ is still reflexive and transitive, and the left and right inheritance properties of $\rightarrowtriangle$ (see Subsection 7.1.2) are still true. However, the structural properties that are linked with the conflict relation *no longer hold*. When time is added, some events that are not in structural conflict become incompatible, therefore, the structural conflict is no longer the only source of incompatibility between events.

In particular, the inheritance of the conflict relation under the reveals relation (Property 47) does not hold in this case. For example, in Fig. 8.5, $e_3 \# e_1$ and $e \triangleright e_3$ but $\neg(e \# e_1)$. However, e and e_1 never occur together in a run, we say that they are logically incompatible. Observe that this incompatibility is not a binary relation since it may happen that a set of events can never occur although

the events of this set are not pairwise incompatible, like the set $\{e, e_1, e_2, e_3\}$ in Fig. 8.6. That is why Lemma 64 – which states that $A \rightarrow\!\!\triangleright_m \emptyset$ implies $|A| = 2$ – is no longer true in this case either.

Incompatible Events

Definition 103 (Set of events incompatible with e)**.** A set of events E is incompatible with an event e iff $E \not\rightarrow\!\!\triangleright \emptyset$ and $E \cup \{e\} \rightarrow\!\!\triangleright \emptyset$.

When E is incompatible with e, we write $E \in Inc(e)$.

$E \not\rightarrow\!\!\triangleright \emptyset$ means that the events in E can happen together, and $E \cup \{e\} \rightarrow\!\!\triangleright \emptyset$ means that for any process ω, $E \not\subseteq \omega \vee e \notin \omega$. For example, in Fig. 8.6, $\{e_1, e_2, e_3\}$ is incompatible with e.

The set of events E is incompatible with an event e when it makes the occurrence of e impossible. This comes either from a structural conflict with e ($E \cap \#[e] \neq \emptyset$), or from the "unavoidability" of an event $e' \in Pre(e)$ Therefore, the following holds.

$$E \in Inc(e) \iff E \cap \#[e] \neq \emptyset \vee E \cup [e] \rightarrow\!\!\triangleright Pre(e)$$

Theorem 104. *Let E be a pre-process such that $e \in En(Cut(E))$. Then, E is an enabling past of e iff for any set of events F, $(F \in Inc(e) \wedge F \cap \#[e] = \emptyset) \implies E \cup F \rightarrow\!\!\triangleright \emptyset$.*

Proof. ($\Rightarrow$) By contradiction, assume E is an enabling past of e and there exists a set of events F such that $F \in Inc(e) \wedge F \cap \#[e] = \emptyset$ and $E \cup F \not\rightarrow\!\!\triangleright \emptyset$. This means that there exists a process which contains $E \cup F$, therefore $\omega = E \cup \lceil F \rceil$ is a pre-process, and since e is not compatible with F, $e \notin \omega$. Moreover, ω does not contain events in structural conflict with e because F and E are not in structural conflict with e, therefore ω is also such that $e \in En(Cut(\omega))$. That is, there exists a pre-process ω such that $E \subseteq \omega \wedge e \in En(Cut(\omega))$ and yet $\omega \cup \{e\}$ is not a pre-process (because ω contains F which is not compatible with e), which contradicts the definition of enabling past.

($\Leftarrow$) Assume that for any set of events F, $(F \cup \{e\} \rightarrow\!\!\triangleright \emptyset \wedge F \cap \#[e] = \emptyset) \Rightarrow E \cup F \rightarrow\!\!\triangleright \emptyset$. Then, for any pre-process ω such that $E \subseteq \omega \wedge e \in En(Cut(\omega))$, since E is incompatible with any set which is incompatible with e, ω is compatible with e and $\omega \cup \{e\}$ is a pre-process. □

That is, E is an enabling past of e if its occurrence prevents the occurrence of the sets of events logically incompatible with e (those that are in structural conflict with e are already made incompatible by the fact that $e \in En(Cut(\omega))$).

8.3 Conclusion and Perspectives

The processes of classical Petri nets can be constructed recursively by adding events to a configuration, when they are enabled by the final conditions of that configuration. That is, in order to be extended by event e, a configuration need only contain the causal past of e, and no event in minimal conflict with e. This is not true when we consider TPNs. Even in this simplified setting where we focus on punctual TPNs, we have seen how timing constraints create complex dependencies between event occurrences. Building valid processes can no longer be done by looking at the causal pasts, as in the untimed case, and some configurations may not be included in any process.

To address this problem, we have introduced the notion of *enabling past*: a pre-process whose final conditions enable event e and that contains an enabling past of e can safely be extended by e. We observed that there can be infinitely many enabling pasts for a given event, because enabling pasts are preserved by addition of events, and that even $\subseteq$-minimal enabling pasts may reach a time greater than the occurrence time of e. Obviously, we consider that a relevant enabling past of e should not overcome the occurrence time of e. That is captured by the notion of *minimal enabling past* that we consider as satisfactory because

- any minimal enabling past E of e is such that $\tau(E) < \tau(e)$, and
- any process that contains e also contains a minimal enabling past of e.

Another observation is that there can be several minimal enabling pasts for a given event, and these enabling pasts are not necessarily incompatible. For example, in Fig. 8.6, in order to occur, e needs either $[g_1]$ or $[g_2]$ or $[g_3]$, and they can occur all together.

Also, the sets of releasing events of an event e are particularly useful for determining whether a pre-process can be extended by e, because only some events that must have occurred strictly before e have to be considered. Moreover, this gives a local condition in the sense that it can be checked by looking only at the causal predecessors of each preventing event of e.

What is left to be done now, is to define an algorithm to build the processes of a time branching process. Here, we would like to use the notion of enabling past in this way: an event e is added to pre-process ω only if ω contains an enabling past of e. This would be automatically satisfied if the temporal completeness is ensured (Lemma 96), that is if the events are added chronologically. However, we want to be able to add an event as early as possible, and to represent the dependency between this minimal enabling past and the event. But contrary to releasing events, the events of a minimal enabling past may not be local (see g_3 in Fig.8.6). So, releasing events may be a good notion for representing canonical dependencies, although they do not always correspond to minimal enabling past. Besides, they also always occur strictly before e.

In addition, we have defined the reveals relation and the extended reveals relation over the set of processes of a time branching process (this was straightforward). We have discussed the properties that change in this setting, and linked these relations with the notion of enabling past. For a given event e, sets of events incompatible with e have been defined. This incompatibility does only originate from the structural conflict relation, as in the untimed case, and is no longer a binary relation. This is also a first step in a more advanced study of the reveals relation in general TPNs.

In the long term, it is desirable to extend these preliminary results to general TPN, and to define a canonical unfolding of TPN.

Conclusion

Summary

Our works all lie within the scope of modeling and analyzing real-time distributed systems.

We introduced several missing notions for the formalization of our problems. The introduced notions can now be reused to formalize other concurrency-related problems. We first formalized a distributed timed bisimulation that we used to show that it is possible to translate a time Petri net into a network of timed automata that has the same distributed behavior. In this translation, we observed that we have to extend the classical syntax of network of timed automata so that the components can read the state of their neighbors. This represents communications that are hidden in the original time Petri net. This observation led us to consider communications via shared variables in networks of timed automata, and more precisely the problem of shared clocks.

Then we considered the implementation of networks of timed automata on multi-core architectures (i.e. the different automata may be implemented on different processors). In this context, shared clocks are problematic because their implementation requires communications that are not explicitly described in the model. We showed how to avoid shared clocks in network of timed automata when this is possible. For this, we allowed an extended synchronization mechanism such that the automata can transmit their state (current location and current valuation) to their neighbors when they synchronize. This study also required the introduction of new formal notions to prove that the transformation of the network does not change its distributed behavior. We considered networks of two timed automata and formalized the preservation of the local behavior of an automaton in the context of another one.

In the second part, we studied the logical dependencies between event occurrences in occurrence nets. To start with, we considered the untimed setting, and investigated the reveals relation. We then introduced a general framework for the description of more general logical dependencies between event occurrences, and solved a synthesis problem.

Lastly, we considered a simple class of time Petri nets whose time intervals are punctual, with the objective of studying the dependencies between events in this timed setting. Even with this simple class, the dependencies are much more complex than with untimed Petri nets, and this makes it difficult to build valid processes. We addressed this problem by defining enabling pasts: when a

pre-process contains an enabling past of an event e and enables e, it can safely be extended by e. We also studied the reveals relation and the extended reveals relation in this setting, and linked the extended reveals relation to the notion of enabling past. This required the definition of logical incompatibility, as opposed to the conflict relation which we regard as a structural incompatibility.

This thesis studied real-time distributed systems. Our contribution has three main aspects that are summarized below.

Formalization of Behavioral Equivalences for Distributed Timed Systems

- Formalization of distributed timed behavior, and its preservation from one model into the other, with the notions of timed traces, distributed timed language, and distributed timed bisimulation (Chapter 3),
- Formalization of the behavior of one TA in the context of another TA, and behavior comparison, with the notions of contextual timed transition system and contextual timed bisimulation (Chapter 5).

Concurrency and Interactions in Different Formalisms

- Translation from TPN to NTA that preserves concurrency, i.e. the distributed behavior (Chapter 4),
- From a given NTA, construction of an equivalent NTA without shared clocks, but with clock copies and transmission of information on the synchronizations when possible (Chapter 5).

Logical Dependencies between Events

- Deeper study of the reveals relation defined in [Haa10] (Chapter 6),
- Definition of a logical independency between events, as opposed to the concurrency relation in occurrence nets (Chapter 6),
- Definition of tight occurrence nets as a structural representation of the logical (binary) dependencies between events of occurrence nets (Chapter 6),
- Formalization of general logical dependencies between events as logical formulas, and synthesis of an occurrence net from a logical formula that describes its set of runs (Chapter 7),
- Clarification of the dependencies between events in punctual TPNs, with the objective of defining a canonical unfolding (Chapter 8).

General Conclusion and Perspectives

Our work opens the road towards a more advanced study of concurrency in distributed timed systems. We have seen that concurrency in timed systems involves the structural causality but also the timing constraints that add complex dependencies between the processes and between apparently unrelated events. The dependencies between the components of distributed timed systems can be made more explicit in the models thanks to the constructions we have presented. Furthermore, the dependencies between events can be exploited to improve the analysis of these systems.

Even though some communications are explicitly represented in the two formalisms we studied, time Petri nets and networks of timed automata, some dependencies are not directly apparent in these formalisms. This entails the need to make them explicit in order to gain a better understanding of them, and to implement the models correctly on multi-core architectures. A natural extension of the work presented in Chapter 5 is to consider the symmetric case, where the first automaton may also read some clocks that are reset by the second automaton. As mentioned earlier, we think that this extension is feasible by redefining contextual timed transition systems in this case. Other long-term extensions of this work are the consideration of networks of more than two timed automata, and the limitation of the precision of the information transmitted during the synchronizations.

We think that time Petri nets and networks of timed automata are most appropriate formalisms for the modeling of distributed timed systems, and their distributed semantics should be considered for several reasons. These models represent distributed systems and therefore they should be studied as such. For example, it is possible that, for some property, it is relevant to consider only some of the components, and not the whole system. Also, the classical behavioral comparisons, based on the sequential semantics, like timed bisimulations are not sufficient to formalize some problems, like the preservation of the distributed behavior from one model to another model, or the preservation of the local behavior of an automaton in the context of another automaton. Lastly, looking at the local behavior of the components independently could also be used for defining modular and efficient analysis of distributed timed systems.

The logical framework we defined in Chapter 7, with the introduction of the ERL logic, is intended to be used as a tool for manipulating structure of occurrence nets and knowledge. This can have applications in the diagnosis of discrete event systems where some events cannot be observed, but their occurrence can be inferred from the observation of the occurrence of other events. This is also a starting point towards efficient verification of system properties, as well as towards enforcing such properties through behavior control, or directly through synthesis of systems from logical specifications.

Lastly, the work about enabling pasts in the unfoldings of punctual time Petri nets, presented in Chapter 8, is a preliminary work that is intended to be used

in the definition of a canonical unfolding of punctual time Petri nets, and in the longer term of general time Petri nets.

Self References

[BC12a] Sandie Balaguer and Thomas Chatain. Avoiding shared clocks in networks of timed automata. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, Newcastle, UK, September 2012. Springer.

[BC12b] Sandie Balaguer and Thomas Chatain. Avoiding Shared Clocks in Networks of Timed Automata. Rapport de recherche RR-7990, INRIA, June 2012.

[BCH10] Sandie Balaguer, Thomas Chatain, and Stefan Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 77–84, Paris, France, 2010. IEEE Computer Society Press.

[BCH11] Sandie Balaguer, Thomas Chatain, and Stefan Haar. Building tight occurrence nets from reveals relations. In Benoît Caillaud and Josep 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.

[BCH12a] Sandie Balaguer, Thomas Chatain, and Stefan Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. *Formal Methods in System Design*, 2012.

[BCH12b] Sandie Balaguer, Thomas Chatain, and Stefan Haar. Building tight occurrence nets from reveals relations. *Fundamenta Informaticae*, 2012. To appear.

[LBD+10] Shuhao Li, Sandie Balaguer, Alexandre David, Kim G. Larsen, Brian Nielsen, and Saulius Pusinskas. Scenario-based verification of real-time systems using uppaal. *Formal Methods in System Design*, 37(2-3):200–264, 2010.

Bibliography

[ABG07] S. Akshay, Benedikt Bollig, and Paul Gastin. Automata and logics for timed message sequence charts. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4855 of *LNCS*, pages 290–302, New Delhi, India, 2007. Springer.

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *International Conference on Concurrency Theory (CONCUR)*, volume 5201 of *LNCS*, pages 82–97, Toronto, Canada, 2008. Springer.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Automata, Languages and Programming*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.

[AGMK10] S. Akshay, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Model checking time-constrained scenario-based specifications. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, pages 204–215. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiří Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[AIN00] Parosh Aziz Abdulla, S. Purushothaman Iyer, and Aletta Nylén. Unfoldings of unbounded petri nets. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *LNCS*, pages 495–507. Springer, 2000.

[AKY10] Parosh Aziz Abdulla, Pavel Krcál, and Wang Yi. Sampled semantics of timed automata. *Logical Methods in Computer Science*, 6(3), 2010.

[AL97] Tuomas Aura and Johan Lilius. Time processes for time petri-nets. In Pierre Azéma and Gianfranco Balbo, editors, *ICATPN*, volume 1248 of *LNCS*, pages 136–155. Springer, 1997.

[AL00] Tuomas Aura and Johan Lilius. A causal semantics for time petri nets. *Theor. Comput. Sci.*, 243(1-2):409–447, 2000.

[AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed petri nets and bqos. In José Manuel Colom and Maciej Koutny, editors, *ICATPN*, volume 2075 of *LNCS*, pages 53–70. Springer, 2001.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 254–277, Warsaw, Poland, April 2003. Springer.

[BCD02] Eric Badouel, Benoît Caillaud, and Philippe Darondeau. Distributing finite automata through Petri net synthesis. *Journal on Formal Aspects of Computing*, 13:447–470, 2002.

[BCH+05a] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of different semantics for time Petri nets. In Doron A. Peled and Yih-Kuen Tsay, editors, *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *LNCS*, pages 293–307, Taipei, Taiwan, October 2005. Springer.

[BCH+05b] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time petri nets. In Paul Pettersson and Wang Yi, editors, *FORMATS*, volume 3829 of *LNCS*, pages 211–225. Springer, 2005.

[BCH+08] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. When are timed automata weakly timed bisimilar to time Petri nets? *Theoretical Computer Science*, 403(2-3):202–220, 2008.

[BCM01] Paolo Baldan, Andrea Corradini, and Ugo Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1–49, 2001.

[BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[BD98] Eric Badouel and Philippe Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 529–586. Springer Berlin / Heidelberg, 1998.

[BDFP00a] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In *CAV*, volume 1855 of *LNCS*, pages 464–479. Springer, 2000.

[BDFP00b] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of Updatable Timed Automata. In *MFCS*, volume 1893, pages 232–242. Springer, 2000.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, November 1998.

[BDKP91] Eike Best, Raymond R. Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent bisimulations in petri nets. *Acta Inf.*, 28(3):231–264, 1991.

[BDL04] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[BDLM08] Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser. Synthesis of Petri nets from finite partial languages. *Fundam. Inform.*, 88(4):437–468, 2008.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. KRONOS: a model-checking tool for real-time systems. In *CAV*, volume 1427 of *LNCS*, pages 546–550, 1998.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *CAV 2003*, volume 2725 of *LNCS*, pages 180–192. Springer, Heidelberg, 2003.

[Ber93] Luca Bernardinello. Synthesis of net systems. In *ICATPN*, volume 691 of *LNCS*, pages 89–105. Springer, 1993.

[Bes86] Eike Best. Structure theory of petri nets: the free choice hiatus. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 254 of *LNCS*, pages 168–205. Springer Berlin / Heidelberg, 1986.

[BFHJ03] Albert Benveniste, Éric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete event systems: A net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[BGK+96] Johan Bengtsson, W.O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV96*, number 1102 in LNCS, pages 244–256. Springer–Verlag, Jul 1996.

[BHR06] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed unfoldings for networks of timed automata. In Susanne Graf and Wenhui Zhang, editors, *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *LNCS*, pages 292–306, Beijing, China, October 2006. Springer.

[BHR08] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. *Information and Computation*, 206(1):73–107, January 2008.

[BHR09] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Undecidability results for timed automata with silent transitions. *Fundamenta Informaticae*, 92(1-2):1–25, January 2009.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *LNCS*, pages 485–500. Springer, 1998.

[BJS09] Joakim Byg, Kenneth Yrke Joergensen, and Jiří 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.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BLM+11] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, Ocan Sankur, and Claus Thrane. Timed automata can always be made implementable. In Joost-Pieter Katoen and Barbara König, editors, *Pro-*

ceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11), volume 6901 of *LNCS*, pages 76–91, Aachen, Germany, September 2011. Springer.

[BP96] Nadia Busi and G. Michele Pinna. Non sequential semantics for contextual P/T nets. In *Application and Theory of Petri Nets*, volume 1091 of *LNCS*, pages 113–132. Springer, 1996.

[BR08] Marc Boyer and Olivier H. Roux. On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3):225–249, 2008.

[BRV04] Bernard Berthomieu, Pierre-Olivier Ribet, and François 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.

[CCJ06] Franck Cassez, Thomas Chatain, and Claude Jard. Symbolic unfoldings for networks of timed automata. In *ATVA*, volume 4218 of *LNCS*, pages 307–321. Springer, 2006.

[CCK+08] Josep Carmona, Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. A symbolic algorithm for the synthesis of bounded petri nets. In Kees M. van Hee and Rüdiger Valk, editors, *Applications and Theory of Petri Nets*, volume 5062 of *LNCS*, pages 92–111. Springer, 2008.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[CEP95] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1-2):117–136, August 1995.

[CF10] Thomas Chatain and Éric Fabre. Factorization properties of symbolic unfoldings of colored Petri nets. In *Petri Nets*, volume 6128 of *LNCS*, pages 165–184. Springer, 2010.

[CGMP99] Edmund M. Clarke, Orna Grumberg, Marius Minea, and Doron Peled. State space reduction using partial order techniques. *STTT*, 2(3):279–287, 1999.

[CJ04] Thomas Chatain and Claude Jard. Symbolic diagnosis of partially observable concurrent systems. In David de Frutos-Escrig and

Manuel Núñez, editors, *Proceedings of 24th IFIP WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'04)*, volume 3235 of *LNCS*, pages 326–342, Madrid Spain, September 2004. Springer.

[CJ06] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In Susanna Donatelli and P. S. Thiagarajan, editors, *Proceedings of the 27th International Conference on Applications and Theory of Petri Nets (ICATPN'06)*, volume 4024 of *LNCS*, pages 125–145, Turku, Finland, June 2006. Springer.

[CR06] Franck Cassez and Olivier H. Roux. Structural translation from time Petri nets to timed automata. *Journal of Systems and Software*, 2006.

[CS89] José Manuel Colom and Manuel Silva. Convex geometry and semiflows in p/t nets. a comparative study of algorithms for computation of minimal p-semiflows. In Grzegorz Rozenberg, editor, *Applications and Theory of Petri Nets*, volume 483 of *LNCS*, pages 79–112. Springer, 1989.

[CY92] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1:385–415, 1992.

[Dar98] Philippe Darondeau. Deriving unbounded petri nets from formal languages. In *CONCUR*, volume 1466 of *LNCS*, pages 533–548. Springer, 1998.

[DDR04] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. In Rajeev Alur and George J. Pappas, editors, *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *LNCS*, pages 296–310. Springer-Verlag, 2004.

[DE95] Jörg Desel and Javier Esparza. *Free choice Petri nets*. Cambridge University Press, New York, USA, 1995.

[dFERA00] David de Frutos-Escrig, Valentín Valero Ruiz, and Olga Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *Applications and Theory of Petri Nets*, pages 187–206, 2000.

[Dij65] Edsger Wybe Dijkstra. Cooperating sequential processes, 1965. Reprinted in Programming Languages, F. Genuys, ed., Academic Press, New York 1968.

[Dim09] Cătălin 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.

[DL07] Cătălin Dima and Ruggero Lanotte. Distributed time-asynchronous automata. In *ICTAC*, pages 185–200. Springer-Verlag, 2007.

[DLLN09] Alexandre David, Kim G. Larsen, Shuhao Li, and Brian Nielsen. Timed testing under partial observability. In *ICST*, pages 61–70. IEEE Computer Society, 2009.

[DM06] Alastair F. Donaldson and Alice Miller. A computational group theoretic symmetry reduction package for the spin model checker. In Michael Johnson and Varmo Vene, editors, *Algebraic Methodology and Software Technology*, volume 4019 of *LNCS*, pages 374–380. Springer, 2006.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.

[DR96] Jörg Desel and Wolfgang Reisig. The synthesis problem of Petri nets. *Acta Informatica*, 33:297–315, 1996.

[DY00] Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 165–172. IEEE Computer Society, 2000.

[EH08] Javier Esparza and Keijo Heljanko. *Unfoldings: a partial-order approach to model checking*. Springer-Verlag New York Inc, 2008.

[EHP+02] H. Ehrig, K. Hoffmann, J. Padberg, P. Baldan, and R. Heckel. High-level net processes. In *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 191–219. Springer, 2002.

[EN94] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets - a survey. *Elektronische Informationsverarbeitung und Kybernetik*, 30(3):143–160, 1994.

[Eng91] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[ER89] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures. parts I and II. *Acta Informatica*, 27(4):315–368, 1989.

[ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[ES92] Javier Esparza and Manuel Silva. A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theor. Comput. Sci.*, 102(1):185–205, August 1992.

[Esp96] Javier Esparza. Decidability and complexity of Petri net problems - an introduction. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Petri Nets*, volume 1491 of *LNCS*, pages 374–428. Springer, 1996.

[FBHJ05] Éric Fabre, Albert Benveniste, Stefan Haar, and Claude Jard. Distributed monitoring of concurrent and asynchronous systems*. *Discrete Event Dynamic Systems*, 15(1):33–84, 2005.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems*, volume 4202 of *LNCS*, pages 187–199, France, 2006. Springer.

[For04] U.S.-Canada Power System Outage Task Force. Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations. Technical report, April 2004.

[FS02] Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In Javier Esparza and Charles Lakos, editors, *Applications and Theory of Petri Nets*, volume 2360 of *LNCS*, pages 163–181. Springer, 2002.

[GHJ97] Vineet Gupta, Thomas Henzinger, and Radha Jagadeesan. Robust timed automata. In Oded Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *LNCS*, pages 331–345. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0014736.

[GK10] Paul Gastin and Dietrich Kuske. Uniform satisfiability problem for local temporal logics over Mazurkiewicz traces. *Information and Computation*, 208(7):797–816, 2010.

[GLMR05] Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo: A tool for analyzing time Petri nets. In Kousha Etessami and Sriram K. Rajamani, editors, *International Conference on Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer, 1996.

[GRR06] Guillaume Gardey, Olivier H. Roux, and Olivier F. Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming*, 6(3):301–320, 2006.

[GW93] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

[Haa10] Stefan Haar. Types of asynchronous diagnosability and the *reveals*-relation in occurrence nets. *IEEE Transactions on Automatic Control*, 55(10):2310–2320, 2010.

[Hac72] M. Hack. Analysis of production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, USA, 1972.

[HBL+03] Martijn Hendriks, Gerd Behrmann, Kim Guldstrand Larsen, Peter Niebert, and Frits W. Vaandrager. Adding symmetry reduction to uppaal. In Kim Guldstrand Larsen and Peter Niebert, editors, *FORMATS*, volume 2791 of *LNCS*, pages 46–59. Springer, 2003.

[HFMV95] Joseph Y. Halpern, Ronald Fagin, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[HKWT95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 417–428, 1995.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[HSSW68] A.W. Holt, H. Saint, R. Shapiro, and S. Warshall. *Final Report of the Information Systems Theory Project*. Technical Report RADC–TR–68–305, Rome Air Development Center, Griffiss Air Force Base, New York, 1968.

[JGP99] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[JKW07] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3-4):213–254, 2007.

[JLL77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4(3):277 – 299, 1977.

[Kho03] Victor Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, School of Computing Science, University of Newcastle upon Tyne, 2003.

[KK03] Victor Khomenko and Maciej Koutny. Branching processes of high-level petri nets. In Hubert Garavel and John Hatcliff, editors, *TACAS*, volume 2619 of *LNCS*, pages 458–472. Springer, 2003.

[KKY04] Victor Khomenko, Maciej Koutny, and Alex Yakovlev. Detecting state encoding conflicts in STG unfoldings using SAT. *Fundam. Inf.*, 62(2):221–241, 2004.

[KKY06] Victor Khomenko, Maciej Koutny, and Alexandre Yakovlev. Logic synthesis for asynchronous circuits based on STG unfoldings and incremental SAT. *Fundam. Inform.*, 70(1-2):49–73, 2006.

[KM69] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147 – 195, 1969.

[KMY08] Victor Khomenko, Agnes Madalinski, and Alexandre Yakovlev. Resolution of encoding conflicts by signal insertion and concurrency reduction based on STG unfoldings. *Fundam. Inform.*, 86(3):299–323, 2008.

[Kos82] S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 267–281, New York, NY, USA, 1982. ACM.

[KPSP10] Michal Knapik, Wojciech Penczek, Maciej Szreter, and Agata Pólrola. Bounded parametric verification for distributed time petri nets with discrete-time semantics. *Fundam. Inform.*, 101(1-2):9–27, 2010.

[Lau75] K. Lautenbach. Liveness in Petri nets. Technical report, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Germany, July 1975.

[Lip76] R. J. Lipton. The reachability problem requires exponential space. 62, New Haven, Connecticut: Yale University, Department of Computer Science, Research, 1976.

[LL98] François Laroussinie and Kim Guldstrand Larsen. Cmc: A tool for compositional model-checking of real-time systems. In Stanislaw Budkowski, Ana R. Cavalli, and Elie Najm, editors, *FORTE*, volume 135 of *IFIP Conference Proceedings*, pages 439–456. Kluwer, 1998.

[LMSP00] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Adriano Peron. Timed cooperating automata. *Fundamenta Informaticae*, 43(1-4):153–173, 2000.

[LMST03] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Simone Tini. Concurrency in timed automata. *Theor. Comput. Sci.*, 309(1-3):503–527, 2003.

[LNZ05] Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005.

[LPSZ08] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. *SIGPLAN Not.*, 43(3):329–339, March 2008.

[LPW07] Alessio Lomuscio, Wojciech Penczek, and Bozena Wozna. Bounded model checking for knowledge and real time. *Artif. Intell.*, 171(16-17):1011–1038, 2007.

[LR06] Didier Lime and Olivier H. Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Event Dynamic Systems (jDEDS)*, 16(2):179–205, April 2006.

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *TACAS*, volume 5505 of *LNCS*, pages 54–57. Springer, 2009.

[LS00] François Laroussinie and Philippe Schnoebelen. The state-explosion problem from trace to bisimulation equivalence. In Jerzy Tiuryn, editor, *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, volume 1784 of *LNCS*, pages 192–207, Berlin, Germany, March 2000. Springer.

[LT93] Nancy G. Leveson and Clark S. Turner. An investigation of the therac-25 accidents. 26(7):18–41, 1993.

[LW08] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Logic*, 9(2):10:1–10:27, April 2008.

[May81] Ernst Mayr. Persistence of vector replacement systems is decidable. *Acta Informatica*, 15:309–318, 1981.

[McM92] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV*, volume 663 of *LNCS*, pages 164–177. Springer, 1992.

[Mer74] Philip Meir Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine, 1974.

[Min99] Marius Minea. Partial order reduction for model checking of timed automata. In *CONCUR*, volume 1664 of *LNCS*, pages 431–446. Springer, 1999.

[MY96] Oded Maler and Sergio Yovine. Hardware timing verification using KRONOS. In *In Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, pages 12–13. IEEE Press, 1996.

[NHZL01] Peter Niebert, Michaela Huhn, Sarah Zennou, and Denis Lugiez. Local first search - a new paradigm for partial order reductions. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR*, volume 2154 of *LNCS*, pages 396–410. Springer, 2001.

[NPW81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

[NQ06] Peter Niebert and Hongyang Qu. Adding invariants to event zone automata. In *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*, pages 290–305. Springer, 2006.

[OW04] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE Computer Society, 2004.

[Pag96] Florence Pagani. Partial orders and verification of real-time systems. In Bengt Jonsson and Joachim Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 327–346. Springer, 1996.

[PBV11] Florent Peres, Bernard Berthomieu, and François Vernadat. On the composition of time petri nets. *Discrete Event Dynamic Systems*, 21(3):395–424, 2011.

[Pel96] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996.

[Pel08] Radek Pelánek. Fighting state space explosion: Review and evaluation. In Darren D. Cofer and Alessandro Fantechi, editors, *Formal Methods for Industrial Critical Systems*, volume 5596 of *LNCS*, pages 37–52. Springer, 2008.

[Pen95] Wojciech Penczek. Branching time and partial order in temporal logics. In *Time and Logic: A Computational Approach*, pages 179–228. UCL Press, 1995.

[Pet66] Carl Adam Petri. *Communication with automata*. PhD thesis, Universität Hamburg, 1966. Originally published in German: Kommunikation mit Automaten, 1962.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223 – 231, 1978.

[Ram74] Chander Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Dept. Electrical Engineering, Cambridge, 1974.

[Rec11] ITU-T Recommendation. Z.120: Message sequence chart. Technical report, ITU-T, Geneva, 2011.

[Rei84] John Reif. The complexity of two-player games of incomplete information. *Jour. Computer and Systems Sciences*, 29:274–301, 1984.

[RS09] Pierre-Alain Reynier and Arnaud Sangnier. Weak time Petri nets strike back! In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *LNCS*, pages 557–571. Springer, 2009.

[RS12] César Rodríguez and Stefan Schwoon. Verification of petri nets with read arcs. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *LNCS*, pages 471–485. Springer, 2012.

[RSB11] César Rodríguez, Stefan Schwoon, and Paolo Baldan. Efficient contextual unfolding. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *LNCS*, pages 342–357. Springer, 2011.

[RSV11] Anders P. Ravn, Jiří Srba, and Saleem Vighio. Modelling and verification of web services business activity protocol. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *LNCS*, pages 357–371. Springer, 2011.

[RY86] Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105 – 135, 1986.

[Shi85] M. W. Shields. Concurrent machines. *Comput. J.*, 28(5):449–465, 1985.

[SK04] Claus Schröter and Victor Khomenko. Parallel ltl-x model checking of high-level petri nets based on unfoldings. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *LNCS*, pages 109–121. Springer, 2004.

[Srb08] Jiří Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In *FORMATS*, volume 5215 of *LNCS*, pages 15–32. Springer, 2008.

[SY96] Joseph Sifakis and Sergio 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.

[TGJ+10] Louis-Marie Traonouez, Bartosz Grabiec, Claude Jard, Didier Lime, and Olivier H. Roux. Symbolic unfolding of parametric stopwatch petri nets. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *8th International Symposium on Automated Technology for Verification and Analysis (ATVA 2010)*, volume 6252 of *Lecture Notes in Computer Science*, pages 291–305, Singapore, September 2010. Springer.

[TMBK+11] Y. Thierry-Mieg, B. Bérard, F. Kordon, D. Lime, and O. H. Roux. Compositional Analysis of Discrete Time Petri nets. In *1st workshop on Petri Nets Compositions (CompoNet 2011)*, volume 726, pages 17–31, Newcastle, UK, June 2011. CEUR.

[Tra09] Louis-Marie Traonouez. *Vérification et dépliages de réseaux de Petri temporels paramétrés*. PhD thesis, University of Nantes, Nantes, France, November 2009. in French.

[Tri04] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In Kim Larsen and Peter Niebert, editors, *Formal Modeling and Analysis of Timed Systems*, volume 2791 of *LNCS*, pages 182–188. Springer Berlin / Heidelberg, 2004.

[Val89] Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Applications and Theory of Petri Nets*, volume 483 of *LNCS*, pages 491–515. Springer, 1989.

[Val92] Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.

[Val96] Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Petri Nets*, volume 1491 of *LNCS*, pages 429–528. Springer, 1996.

[vG93] Rob J. van Glabbeek. The linear time - branching time spectrum ii. In Eike Best, editor, *CONCUR*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.

[vGG01] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4/5):229–327, 2001.

[Vog95] Walter Vogler. Fairness and partial order semantics. *Inf. Process. Lett.*, 55(1):33–39, 1995.

[Vog02] Walter Vogler. Partial order semantics and read arcs. *Theoretical Computer Science*, 286(1):33–63, 2002.

[VSY98] Walter Vogler, Alexei L. Semenov, and Alexandre Yakovlev. Unfolding and finite prefix for nets with read arcs. In *CONCUR*, volume 1466 of *LNCS*, pages 501–516. Springer, 1998.

[WG93] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In Eike Best, editor, *CONCUR*, volume 715 of *LNCS*, pages 233–246. Springer, 1993.

[Win98] Józef Winkowski. Processes of contextual nets and their characteristics. *Fundamenta Informaticae*, 36(1):71–101, 1998.

[WL04] Bozena Wozna and Alessio Lomuscio. A logic for knowledge, correctness, and real time. In *CLIMA*, volume 3487 of *LNCS*, pages 1–15. Springer, 2004.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR*, volume 458 of *LNCS*, pages 502–520. Springer, 1990.

[YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):187–215, 1997.

[YSSC93] Tomohiro Yoneda, Atsufumi Shibayama, Bernd-Holger Schlingloff, and Edmund Clarke. Efficient verification of parallel real-time systems. In Costas Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *LNCS*, pages 321–332. Springer Berlin / Heidelberg, 1993.

[ZKH02] Tong Zheng, Ferhat Khendek, and Loïc Hélouët. A semantics for timed msc. *Electr. Notes Theor. Comput. Sci.*, 65(7):85–99, 2002.

Résumé substantiel en français

1 Introduction

Contexte. Dans les dernières décennies, le nombre et la complexité des systèmes informatiques ont considérablement augmenté. Certains de ces systèmes contrôlent des processus critiques dont la défaillance peut causer d'importants coûts humains et/ou financiers. C'est pourquoi il est crucial de prescrire tout comportement inattendu. Pour cela des méthodes formelles qui permettent de modéliser et de vérifier ces systèmes sont développées. Ce sont des méthodes algorithmiques qui permettent de vérifier que le (modèle du) système ne peut pas se trouver dans un état jugé mauvais.

Nous nous intéressons ici aux systèmes temps-réel distribués c'est à dire des systèmes qui ont deux caractéristiques importantes :

- L'aspect temps-réel suppose que le système a des contraintes de temps fortes, dans le sens où non seulement les sorties qu'il produit sont importantes, mais aussi les dates auxquelles ces sorties sont produites.

- L'aspect distribué suppose que le système est constitué de plusieurs composants qui communiquent entre eux et qui sont en partie indépendants.

La combinaison de ces deux aspects est source de problèmes qui n'apparaissent pas lorsque seulement un des deux aspects est considéré. Les systèmes temps-réel ont beaucoup été étudiés, surtout depuis les années 90, avec l'introduction du formalisme des automates temporisés ; tout comme les systèmes distribués non temporisés qui bénéficient d'une théorie très complète avec entre autres les algèbres de processus, la théorie des traces, et les réseaux de Petri. Cependant, l'aspect distribué et l'aspect temporisé sont encore rarement étudiés conjointement, malgré l'existence de formalismes spécialement appropriés comme les réseaux de Petri temporels (depuis 1974) et les réseaux d'automates temporisés (depuis 1994). En particulier, ces deux formalismes n'ont pas été comparés en terme de comportement distribué.

Notre objectif est de comprendre les interactions entre le temps et la distribution. Nous considérons surtout la modélisation des systèmes temps-réels distribués, mais nous fournissons des outils qui permettront une vérification plus efficace.

Contributions. Dans la première partie, nous nous intéressons à la formalisation du comportement distribué des modèles de systèmes temps-réel distri-

bués. Cela nous permet de comparer par exemple deux modèles qui utilisent des formalismes différents. Nous définissons une traduction d'un réseau de Petri vers un réseau d'automates temporisés qui préserve le comportement distribué. Nous étudions aussi différentes extensions du formalisme des réseaux d'automates temporisés, et montrons que certaines sont plus expressives lorsque le comportement distribué est considéré, alors qu'elles ne le sont pas lorsque l'on considère le comportement séquentiel. Enfin, nous considérons le problème lié aux horloges partagées dans l'implémentation des réseaux d'automates temporisés.

Dans la deuxième partie, nous étudions les dépendances entre événements dans les dépliages de réseaux de Petri. Nous considérons d'abord le cadre non temporisé, puis un cadre temporisé simplifié, qui nous permet déjà de montrer que l'introduction du temps rend les dépendances beaucoup plus complexes.

2 Formalismes de modélisation

2.1 Systèmes distribués

Produit synchrone de systèmes de transitions. Un système de transitions (étiqueté) est un tuple $S, s_0, \Sigma, \rightarrow$, où S est l'ensemble d'états, s_0 est l'état initial, Σ est l'alphabet (ou ensemble d'étiquettes), et $\rightarrow \subseteq (S \times \Sigma \times S)$ est une relation de transition. Quand $(s, a, s') \in \rightarrow$, nous écrivons $s \xrightarrow{a} s'$.

Le produit synchrone des deux systèmes de transitions $T_1 = (S_1, s_1^0, \Sigma_1, \rightarrow_1)$ et $T_2 = (S_2, s_2^0, \Sigma_2, \rightarrow_2)$, que l'on note $T_1 \otimes T_2$, est le système de transitions $\big(S_1 \times S_2, (s_1^0, s_2^0), \Sigma_1 \cup \Sigma_2, \rightarrow\big)$, où $\rightarrow$ est défini comme :

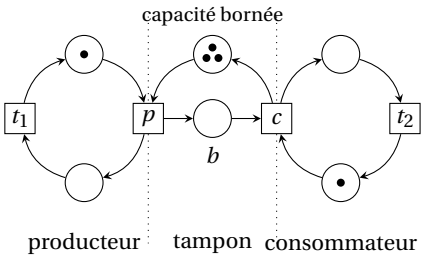
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2)$ ssi $s_1 \xrightarrow{a}_1 s_1'$, pour tout $a \in \Sigma_1 \setminus \Sigma_2$,
- $(s_1, s_2) \xrightarrow{a} (s_1, s_2')$ ssi $s_2 \xrightarrow{a}_2 s_2'$, pour tout $a \in \Sigma_2 \setminus \Sigma_1$,
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ ssi $s_1 \xrightarrow{a}_1 s_1'$ et $s_2 \xrightarrow{a}_2 s_2'$, pour tout $a \in \Sigma_1 \cap \Sigma_2$.

Ainsi un système de transitions représente l'ensemble des actions que peut exécuter un système, et les changements d'état que les actions peuvent produire. Le produit de deux systèmes de transitions représente le comportement de deux systèmes mis en parallèle, comme un seul système séquentiel. Puisque l'opération de produit est associative, il est possible de définir le produit de n systèmes de transitions, avec la définition donnée pour deux systèmes de transitions.

Réseaux de Petri. Les réseaux de Petri nets sont utilisés pour modéliser les systèmes concurrents. Ils ont été utilisés dans de nombreux domaines, comme les réseaux de communication, la gestion des systèmes de production industriels ou la conception de matériel informatique.

Un réseau de Petri est un tuple (P, T, F, M_0), où P et T sont deux ensembles disjoints appelés ensemble de places et ensembles de transitions respectivement, $F \subseteq (P \times T) \cup (T \times P)$ est l'ensemble des arcs qui connectent les places et les transitions, et $M_0 : P \to \mathbb{N}$ est le marquage initial.

Un marquage $M : P \to \mathbb{N}$ représente un état du réseau de Petri, et est graphiquement représenté par $M(p)$ jetons dans chaque place p. Une transition peut être exécutée si ses places d'entrée sont marquées (i.e. contiennent un jeton), ce faisant, elle consomme un jeton dans chacune de ces places d'entrée et crée un jeton dans chacune de ses places de sortie. La figure ci-dessous représente un exemple classique de producteur/consommateur, avec un tampon de capacité bornée à 3.



Les réseaux de Petri bénéficient également de propriétés algébriques, qui permettent, entre autre, d'identifier les différents composants du réseau grâce à de calculs matriciels. Dans notre exemple, un tel calcul montrerait que le réseau a trois composants, comme représenté par les lignes en pointillés.

2.2 Systèmes temporisés séquentiels

Systèmes de transitions temporisés. Les systèmes de transitions temporisés (TTS) sont une notion fondamentale pour la description et la comparaison des comportements des systèmes temporisés séquentiels. Quel que soit le formalisme de modélisation utilisé, le comportement du (modèle du) système peut être décrit avec un TTS.

Un système de transitions temporisé (étiqueté) est un tuple $S, s_0, \Sigma, \to$, où S est l'ensemble d'états, s_0 est l'état initial, Σ est l'alphabet (ou ensemble d'étiquettes), et $\to \subseteq (S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S)$ est une relation de transition. Il représente donc les changements d'états induits par l'exécution d'une action ou le passage du temps.

La notion de bisimulation permet de comparer deux TTS.

Automates temporisés. Les automates temporisés (TA) ont été introduits dans les années 90. Ils sont depuis beaucoup étudiés et implantés dans des outils qui permettent leur utilisation en pratique. Ce sont des automates finis étendus avec la notion d'horloges. Ces horloges peuvent être remises à zéro lorsqu'une

action est exécutée, et leur valeur peut être testée pour savoir si une action est exécutable.

L'ensemble $\mathscr{B}(X)$ des contraintes d'horloge sur l'ensemble d'horloges X est définit par la grammaire suivante.

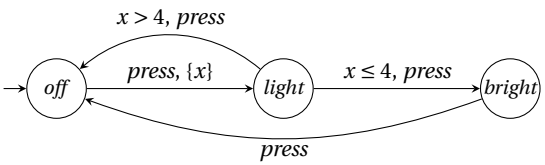
$$g ::= x \bowtie k \mid g \wedge g \mid \textbf{\textit{tt}}, \text{ où } x \in X, k \in \mathbb{N} \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}$$

Un invariant est une contrainte d'horloge qui obéit à la grammaire suivante.

$$i ::= x \leq k \mid x < k \mid i \wedge i \mid \textbf{\textit{tt}}, \text{ où } x \in X, k \in \mathbb{N}$$

Formellement, un TA est un tuple $(L, \ell_0, X, \Sigma, E, Inv)$, où L est l'ensemble fini des localités, ℓ_0 est la localité initiale, X est l'ensemble d'horloges, Σ est l'alphabet ou ensemble d'actions, $E \subseteq L \times \mathscr{B}(X) \times \Sigma \times 2^X \times L$ est l'ensemble d'arcs étiquetés par une garde, une action et un ensemble d'horloges à remettre à 0, et $Inv : L \to \mathscr{B}(X)$ associe un invariant à chaque localité. L'invariant doit être vrai tant que la localité est active.

Par exemple, la figure ci-dessous montre un TA qui modélise une lampe. x est une horloge, les cercles représentent les localités de l'automate et *press* est une action. Ici, après une pression sur le bouton, la lampe s'allume, et elle devient plus lumineuse si le bouton est pressé une deuxième fois avant 4 unités de temps. Sinon elle s'éteint à la prochaine pression.



L'état courant d'un automate est représenté par (ℓ, v) où ℓ est la localité courante et $v : X \to \mathbb{R}_{\geq 0}$ est la valuation courante, c'est-a-dire une fonction qui associe à chaque horloge sa valeur courante.

Le comportement de l'automate peut être décrit grâce à un système de transitions temporisé.

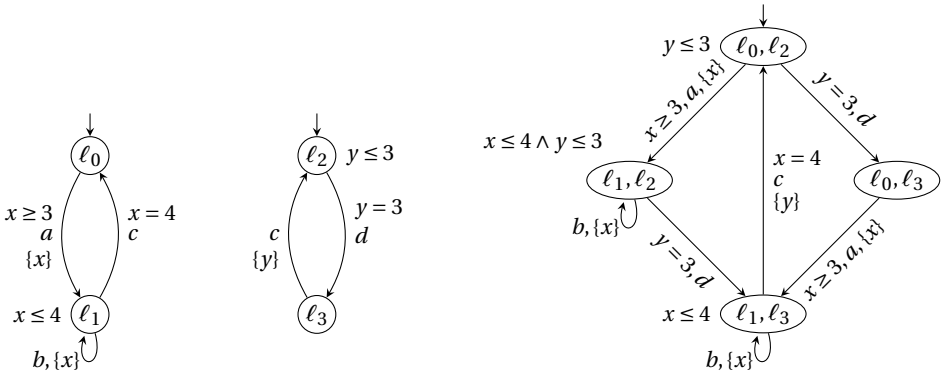
2.3 Systèmes temporisés distribués

Produit synchrone de systèmes de transitions temporisés. Comme pour les systèmes de transitions non temporisés, il est possible de faire le produit de plusieurs systèmes de transitions temporisés pour obtenir un nouveau système de transitions temporisé qui représente le comportement séquentiel des différents systèmes mis en parallèle.

Le produit regroupe les arcs avec les mêmes actions en un seul arc appelé synchronisation.

Réseaux d'automates temporisés. Un réseau d'automate temporisés (NTA) est un ensemble de TA mis en parallèle, $(A_1 \parallel \cdots \parallel A_n)$, où chaque $A_i = (L_i, \ell_0^i, X_i, \Sigma_i, E_i, Inv_i)$ est un automate.

La sémantique d'un NTA peut être définie comme un nouvel automate, appelé automate produit et noté $A_1 \otimes \cdots \otimes A_n$. Pour le construire, on utilise la même idée que précédemment, c'est-à-dire que les actions communes sont exécutées simultanément par tous les automates où l'action apparaît. La figure ci-dessous montre un réseau d'automates temporisés et l'automate produit.



(a) Réseau d'automates temporisés (b) Automate produit

La sémantique du NTA peut aussi être décrite directement par un TTS.

Réseaux de Petri temporels. Les réseaux de Petri ont été étendus avec la notion de temps de plusieurs façons. L'extension que nous considérons s'appelle réseau de Petri temporel, et date de 1974.

Un réseau de Petri temporel (TPN) est un réseau de Petri dans lequel chaque transition est associé à un intervalle de temps qui définit les délais minimum et maximum de tir après sa sensibilisation. Sa sémantique peut être décrite par un système de transitions temporisé et aussi par un automate temporisé appelé automate des marquages.

2.4 Systèmes temporisés distribués vs systèmes temporisés séquentiels

Comme nous l'avons vu, la sémantique des formalismes pour la modélisation des systèmes temporisés distribués est en général décrite par des notions séquentielles, ce que nous ne trouvons pas satisfaisant. C'est pourquoi nous introduisons de nouvelles notions pour décrire la sémantique distribuée de ces formalismes.

Nous définissons d'abord les traces temporisées comme des « mots distribués ». Étant donné un système temporisé distribué sur un alphabet Σ, et son

ensemble de processus $\Pi = \{\pi_1, \ldots, \pi_n\}$, chaque action $a \in \Sigma$ est associée à un ensemble de processus $proc(a) \subseteq \Pi$, qui exécutent toujours cette action ensemble et simultanément. Donc chaque action est soit locale, si elle n'appartient qu'a un seul processus, soit partagée par plusieurs processus.

Les événements (occurrences des actions) sont partiellement ordonnés, puisque deux actions qui ont des ensemble de processus disjoints peuvent être exécutées sans ordre. On note E_i les événements qui ont lieu sur le processus π_i.

Formellement, une trace temporisée sur l'alphabet Σ et l'ensemble de processus $\Pi = \{\pi_1, \ldots, \pi_n\}$ est un tuple $W = (E, \preccurlyeq, \lambda, \delta, proc)$ où

- E est un ensemble dénombrable d'événements,
- $\preccurlyeq \subseteq (E \times E)$ est un ordre partiel sur E tel que pour tout événement e, l'ensemble $e' \in E \mid e' \preccurlyeq e$ est fini, et pour tout $i \in [1..n]$, $\preccurlyeq \cap (E_i \times E_i)$ est un ordre total sur E_i.
- $\lambda : E \to \sigma$ est une fonction d'étiquetage,
- $\delta : E \to \mathbb{R}_{\geq 0}$ assigne une date à chaque événement de façon à ce que, si $e_1 \preccurlyeq e_2$, alors $\delta(e_1) \leq \delta(e_2)$,
- $proc : \Sigma \to 2^{\Pi}$ est la distribution des actions qui associe chaque action à un sous-ensemble de Π.

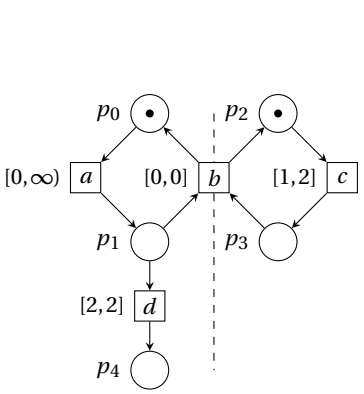
Bisimulation Temporisée Distribuée. Nous définissons aussi la notion de bisimulation temporisée distribuée pour comparer deux systèmes temporisés distribués, avec un même alphabet et un même nombre de processus.

Bisimulation Contextuelle. La bisimulation contextuelle permet de comparer le comportement de deux automates dans le contexte d'un autre même automate.

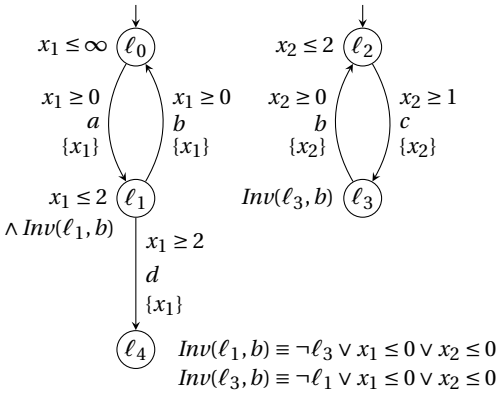
3 Traduction de réseau de Petri vers réseau d'automates temporisés avec préservation de la concurrence

Procédure. La première étape consiste à décomposer le réseau de Petri en composants. Cela se fait grâce à une méthode algébrique qui représente la structure du réseau de Petri comme une matrice. Chaque composant donne directement la structure d'un automate. Il reste donc à ajouter les informations temporelles. Pour cela, nous rajoutons une horloge par automate. Cette horloge est remise à zéro sur chaque arc. Les gardes proviennent directement des délais minimum de tir. L'étape la plus difficile est la traduction des délais maximum de tir en invariants. Dans l'exemple ci-dessous, nous voyons que l'on peut attendre dans ℓ_1 tant que x_1 est inférieur à 2 (x_1 est la date de sensibilisation de d) et, si ℓ_3

est aussi active, tant que le minimum de x_1 et x_2 (date de date de sensibilisation de b) est inférieur à 0.



(a) Un réseau de Petri temporel et ses deux composants



(b) Le réseau d'automates temporisés associé

Connais ton voisin ! La traduction des délais maximums de tirs en invariants nécessite de lire la localité courante de l'automate voisin et son horloge dans les invariants. Nous montrons qu'en général il n'est pas possible de se passer de ces lectures, ce que nous interprétons comme la mise en évidence de dépendances non explicites dans le réseau de Petri temporel initial.

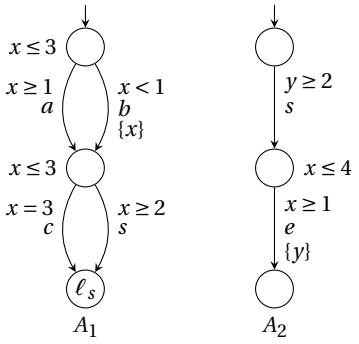
Cette observation nous a aussi conduits à nous intéresser de plus près au problème des horloges partagées dans les réseaux d'automates temporisés.

4 Éviter les horloges partagées dans les réseaux d'automates temporisés

Les horloges partagées sont problématiques lorsque l'on envisage d'implanter un modèle sur une architecture distribuée car elles nécessitent des communications qui ne sont pas explicitement décrites dans le modèle.

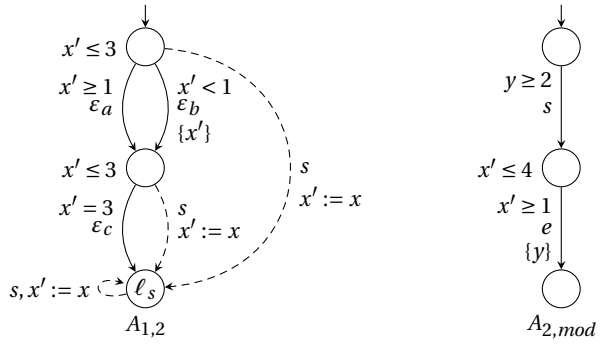
Dans cette partie, nous considérons des réseaux de deux automates temporisés $A_1 \parallel A_2$, tels que A_1 ne lit pas les horloges remises à zéro par A_2, et A_2 peut lire des horloges remises à zéro par A_1 mais pas les remettre à zéro, comme sur l'exemple de la figure ci-dessous. Nous nous demandons si il existe un automate A_2' qui ne lit pas les horloges de A_1, et qui a le même comportement que A_2. Nous autorisons A_2' à recopier la valeur des horloges de A_1 lorsqu'il se synchronise avec celui-ci.

Après avoir formalisé cette équivalence de comportement grâce à la notion de bisimulation contextuelle, nous donnons un critère qui permet de décider si un tel A_2' existe, et nous montrons comment le construire. Nous décrivons



d'abord un cas simple, où il n'y a pas de synchronisation urgente dans A_1, puis nous passons au cas général.

L'idée générale de la construction, est de remplacer A_2 par un produit de deux automates $A_{1,2}$ qui est une copie locale de A_1, et $A_{2,mod}$, qui a la même structure que A_2 mais lit les horloges de $A_{1,2}$ au lieu de lire celles de A_1. À chaque synchronisation, $A_{1,2}$ est remis dans le même état que A_1 (grâce aux copies d'horloges que nous autorisons). La figure ci-dessous donne $A_{1,2}$ et $A_{2,mod}$ pour l'exemple de la figure précédente (cas simple où il n'y a pas de synchronisation urgente dans A_1).



5 Dépendances logiques entre événements

Nous nous intéressons ensuite aux dépendances entre événements dans les réseaux d'occurrence, qui sont la structure obtenue lorsque l'on déplie un réseau de Petri.

Certaines relations proviennent directement de la structure du réseau d'occurrence :

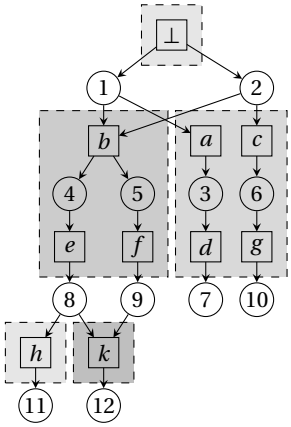
Causalité Deux événements sont causalement reliés lorsqu'il existe un chemin de l'un à l'autre. Formellement, e est une cause de f ssi $e \to^* f$, où $\to$ est la relation qui décrit les arcs du réseau. Dans ce cas, on écrit $e \le f$. Toute

exécution qui contient un événement e contient aussi toutes les causes de e, que l'on appelle le passé causal de e.

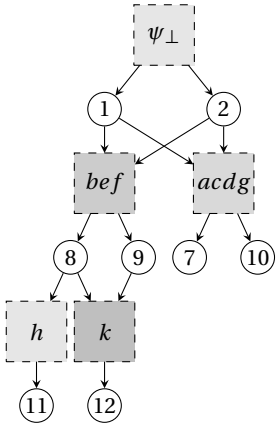
Conflit Deux événements sont en conflit si ils partagent une même condition en entrée. Puisque la condition ne peut être consommée que par un seul des deux événements, ces deux événements n'auront jamais tous les deux lieu dans une même exécution. Lorsque deux événements e et f sont en conflit, on écrit $e \# f$.

Concurrence Enfin, deux événements sont concurrents si ils ne sont ni causalement reliés, ni en conflit. Lorsque deux événements e et f sont concurrent, on écrit $e \; co \; f$.

Par exemple, dans la figure (a) ci-dessous, $b \leq k$, $b \# a$, and $a \; co \; c$.



(a) Un réseaux d'occurrence



(b) Le réseau d'occurrence réduit associé

Nous remarquons que la relation de concurrence ainsi définie n'est pas une relation d'indépendance, car deux événements concurrents peuvent s'influencer par conflits interposés. Par exemple, dans la figure ci-dessus, si c a lieu, b ne peut pas avoir lieu (car il est en conflit avec c) et donc a a aussi lieu, car aucun conflit avec a ne peut avoir lieu. En effet, on suppose que les exécutions sont maximales, c'est-à-dire qu'elles progressent tant que des événements sont sensibilisés (toutes leurs conditions d'entrée sont disponibles). Cette observation a conduit à la définition d'une relation de révélation, comme suit : e révèle f, noté $e \rhd f$, si toute exécution qui contient e contient aussi f. Les réseaux d'occurrence réduits sont la structure obtenue lorsque l'on regroupe les événements qui se révèlent mutuellement. Un exemple de réseau d'occurrence réduit est donné dans la figure (b) ci-dessus.

Enfin, nous définissons les réseaux d'occurrence « tendus », comme les réseaux d'occurrence pour lesquels la relation de causalité et la relation de révélation inversée sont confondues. Dans ces réseaux, toutes les relations logiques

sont donc explicitement représentées par des arcs de causalité. Une conséquence directe est que maintenant, la concurrence est une relation d'indépendance logique. Un réseau d'occurrence tendu est forcément un réseau d'occurrence réduit.

6 Synthèse de réseaux d'occurrence « tendus »

Nous définissons une logique propositionnelle qui permet de décrire les dépendances logiques entre événements. Cela nous conduit à définir la relation de révélation étendue.

Une formule de cette logique peut être utilisée pour décrire l'ensemble des exécutions (générales ou maximales) d'un réseau d'occurrence fini. Réciproquement, nous montrons comment synthétiser, depuis une formule logique, un réseau d'occurrence dont l'ensemble d'exécutions est décrit par cette formule logique (quand un tel réseau existe).

7 Dépendances logiques entre événements dans les systèmes temporisés

Enfin, nous nous intéressons aux dépendances entre événements dans un cadre temporisé simplifié. Même dans ce cadre-là, où nous considérons des intervalles de tirs ponctuels, le temps crée des dépendances complexes entre les événements. Par exemple, il se peut que deux événements e et f, apparemment indépendants, soient incompatibles car f sensibilise un événement e' en conflit avec e, qui doit tirer strictement avant e. Dans cette situation, toute configuration n'est pas valide dans le sens ou certaines configurations ne sont pas incluses dans des processus (exécutions).

C'est pourquoi nous définissons la notion de passé « habilitant » d'un événement e. Cette configuration contient le passé de e et des événements qui assurent que e pourra tirer si il est sensibilisé. Nous expliquons que cette notion nous permet de construire des configurations valides (appelées pré-processus), et donnons des critères suffisants pour vérifier qu'une configuration contient bien un passé habilitant d'un événement e donné. Enfin, nous montrons que cette notion pourrait être utilisée pour définir un dépliage canonique pour la sous-classe de réseaux de Petri temporels que nous considérons, et à long terme un dépliage canonique pour les réseaux de Petri temporels généraux.

8 Conclusion

Notre travail ouvre la voie vers une étude plus avancée de la concurrence dans les systèmes distribués temps-réel. Nous avons montré que la concurrence dans

les systèmes temporisés fait intervenir à la fois la causalité structurelle du modèle et les contraintes de temps qui créent des dépendances entre des événements apparemment indépendants. Les dépendances entre les composants peuvent être explicitées dans les modèles, grâces aux constructions que nous avons présentées. De plus, les dépendances entre événements peuvent être exploitées pour améliorer l'analyse de ces systèmes.

Nous avons aussi vu que les équivalences de comportement habituellement utilisées, basées sur la sémantique séquentielle, ne sont pas adaptées pour traiter des problèmes qui considèrent la distribution du système. Nous nous intéressons à la sémantique distribuée des modèles, ce qui nous conduit à considérer aussi la connaissance d'un composant, c'est-à-dire, ce que ce composant voit du reste du système, et ce qu'il peut en déduire.