N° d'ordre : 3455

THÈSE

présentée devant

l'Université de Rennes 1

pour obtenir le grade de

Docteur de l'Université de Rennes 1
Mention Informatique

par

Thomas Chatain

préparée à l'IRISA
Équipe d'accueil : Distribcom
École Doctorale : Matisse
Composante universitaire : IFSIC

Symbolic Unfoldings of High-Level Petri Nets and Application to Supervision of Distributed Systems

Dépliages symboliques de réseaux de Petri de haut niveau et application à la supervision des systèmes répartis

soutenue le 23 novembre 2006 devant la commission d'examen

Thomas Jensen	DR CNRS (IRISA)	Président du jury
Paul Gastin	Professeur (ENS Cachan)	Rapporteur
Oded Maler	DR CNRS (VERIMAG)	Rapporteur
Claude Jard	Professeur (ENS Cachan)	Directeur de thèse
Victor Khomenko	Research Fellow (University of Newcastle upon Tyne)	Examinateur
Kim G. Larsen	Professeur (Aalborg University)	Examinateur

Contents

Acknowledgements

I would like to thank the members of the jury, and particularly the reviewers, for having accepted to evaluate this work. Their remarks helped me to improve this document.

During this thesis, I worked with several researchers, not only in IRISA, and I found this very exciting. I would like to thank all of them. Of course my supervisor Claude Jard is concerned here. Working with him was a real pleasure, and still now I attach very high importance to his advice.

Finally I want to thank my wife Amélie for her patience.

Introduction

Distributed systems have been developing very quickly for a few decades. Some of them are so large and complex that it is impossible in practice to get comprehensive understanding of their structure and their behavior. Telecommunications networks are examples of systems that evolve continuously and whose users have very little knowledge about the components that are involved in the services they expect.

Many networks are so complex that even a specialist has a very abstract view of them and cannot easily predict the response of the system to simple inputs.

Moreover, because of the number of components that take part in the network, failures cannot be avoided and the system has to be designed so that they have as little impact as possible on the activity of the whole system. But, as the faulty component communicates with others, it is crucial that the parts that are affected by a failure are determined quickly. Then the system can be reconfigured so that it continues without the faulty components. This means that the network has to be equipped with a supervision architecture in order to detect the errors. For this purpose many components are now designed so that they emit alarms when some particular conditions are met.

Nevertheless inferring the causes of the failures remains a challenging problem. One reason for this is the large number of alarms that are emitted. Consequently the supervisor has to select the ones that report an actual defect. Moreover some bad states may result from a complex scenario that involves several components. Even if each component behaves as expected, bad states like deadlocks can be reached.

Finally sometimes the causes of a failure can only be inferred from a set of alarms, by reconstructing a part of the history of the system. This is the problem that motivates our work.

In order to infer the trajectory of the system from a set of observed alarms, the supervisor needs a model of the system and a description of the way the alarms are emitted.

This kind of models are extensively used by formal methods like test, verification or validation that provide a way to master the complexity of the systems. Concerning distributed systems, a class of models are particularly well suited for the representation of concurrency. Petri nets are one of the most widespread among these "true concurrency" models. It is a low-level model where the states of the components of the system are enumerated, but there exist numerous high-level extensions that allow to describe actions that access data, or to specify time constraints. These extensions are often required to model real systems.

But despite the use of true concurrency models, most of the techniques dedicated to the study of distributed systems suffer from the state space explosion problem due to the interleavings of concurrent actions.

Unfoldings provide a solution to this problem. They are based on a causal semantics, or partial order semantics, defined as an alternative to the sequential semantics. The executions are represented by *processes*, in which the actions, called *events*, are only partially ordered by a *causality* relation. Concurrent events are not ordered by this relation, which avoids computing the interleavings.

Among the other techniques that aim at reducing the state space for the verification of concurrent systems, the most famous are probably the partial order reduction techniques [56]. The idea is to explore only a subset of states and transitions that is relevant in the framework of the verification of properties that are not concerned with commutation of concurrent actions. The main difficulty is the identification of concurrency inside the executions. These techniques are relatively efficient for model-checking, but they do not aim at giving an explicit representation of concurrency like unfoldings do.

Although high-level extensions of Petri nets are commonly used to model real systems, the unfolding technique has not been much studied in the framework of high-level true concurrency models. One reason may be that unfoldings have been mostly used for model-checking, and many properties are decidable only in low-level models.

In this document we are concerned with the application of unfoldings to supervision of distributed systems. And we will show that for this application it is not necessary to restrict oneself to low-level models. We carry on with the approach of [8, 9] which uses unfoldings of low-level Petri nets to compute the explanations as processes of a constrained model that is built from a model of the supervised system.

The interest of using unfoldings for supervision is not only to avoid the state space explosion problem, but also to highlight the causal dependencies between the events that are involved in the explanations. This allows us to find easily the causes of the failures.

We adapt this approach to the case of high-level extensions of Petri nets like colored Petri nets, dynamic nets and time Petri nets. The first part of this work is the definition of symbolic unfoldings for these extensions. The combination of symbolic techniques with unfoldings allows a compact representation of families of executions that share the same structure. Then we extend the unfolding-based diagnosis approach and show the usefulness of symbolic unfoldings for supervision of distributed systems.

Organization of the Document

The first contribution of this work is the definition of unfoldings for several high-level extensions of Petri nets. Chapter 1 presents the state-of-the-art concerning unfoldings of low-level Petri nets, and Chapters 2 and 3 describe the extensions of the unfolding technique.

In Chapter 2, we present our contribution to the definition of unfoldings of colored Petri nets and dynamic nets. The model of colored Petri nets is a classical extension of Petri nets that allows to model data aspects. The unfoldings that we propose are based on the symbolic representation of families of executions that share the same structure.

The model of dynamic nets is a very powerful extension of colored Petri nets, which allows to represent systems whose structure evolves during the execution. These dynamic aspects are modeled by adding or removing transitions during the execution. The problem with the unfoldings of this model is to know which transitions are in the net after a given execution. The originality of our approach is to represent transitions as particular tokens so that they appear in the marking and can be added or removed dynamically.

Chapter 3 is dedicated to symbolic unfoldings of timed models. The classical model of time Petri nets is handled first, then we deal with a more general timed model, that combines the features of colored Petri nets with time constraints. First we explain why the techniques that are used to handle untimed models cannot be immediately adapted to a timed framework. Then we present our original approach, which is based on the definition of a concurrent operational semantics for safe time Petri nets in order to reduce the implicit synchronization due to the progress of time. We show the existence of a finite complete prefix for symbolic unfoldings of safe time Petri nets.

After defining symbolic unfoldings for these high-level extensions of Petri nets, we focus on their application to the problem of supervision of distributed systems. This is the purpose of Chapter 4. The use of symbolic unfoldings for the supervision of distributed systems is original and we try to show

the interest of this approach. An original aspect of this approach is the construction of a model that includes both the representation of the system and of the observation architecture.

The last chapter of this thesis details algorithms that compute the unfoldings and diagnoses that are defined in the preceding chapters. These algorithms have been implemented; we describe the implementation.

Notations

$\mathbb{N}$ denotes the set of natural numbers, $\mathbb{R}$ denotes the set of nonnegative real numbers and $\mathbb{Q}$ denotes the set of nonnegative rational numbers.

Sequences. The length of a sequence σ is denoted $|\sigma|$. We write $\sigma_1 \cdot \sigma_2$ for the concatenation of two sequences. Sometimes letters are considered as sequences of length 1. The empty sequence is denoted ϵ.

Sets. We denote $|E|$ the cardinality of a set E, and 2^E the set of subsets of E. We denote $E \times F$ the Cartesian product of E and F.

Mappings. We denote $E \longrightarrow F$ the set of mappings from a set E to a set F. Each mapping is coded as a set $\phi \subseteq E \times F$ such that for all $e \in E$, there exists a unique $f \in F$, denoted $\phi(e)$, such that $(e, f) \in \phi$. We often write $\phi : E \longrightarrow F$ instead of $\phi \in E \longrightarrow F$. Slightly abusing the notations, we often consider that $\phi \in E \longrightarrow F$ when $\phi \in E' \longrightarrow F$ and E and E' are sets such that $E \subseteq E'$.

We denote ϕ^{-1} the inverse of a bijection ϕ. We denote $\phi_{|E}$ the restriction of a mapping ϕ to a set E. The restriction has higher priority than the inverse: $\phi^{-1}_{|E} = (\phi_{|E})^{-1}$.

We denote by $\circ$ the usual composition of mappings.

Multisets. The multisets over a set E are defined as mappings from E to $\mathbb{N}$, that indicate how many times each element of E occurs in the multiset. The set of multisets over E is denoted $E^{\oplus}$. Finite multisets can be described by an enumeration of their elements, between $\{\!|\ldots|\!\}$. For example the multiset A over $\{x, y, z\}$ such that $A(x) \stackrel{\text{def}}{=} 1$, $A(y) \stackrel{\text{def}}{=} 2$ and $A(z) \stackrel{\text{def}}{=} 0$ can be denoted by $\{\!|x, y, y|\!\}$.

We compare multisets using $\leq$:

$$A \leq B \quad \text{iff} \quad \forall x \in E \quad A(x) \leq B(x) \ .$$

We denote $+$ and $-$ the addition and difference between multisets. We shall write $A-B$ only when $B \leq A$. We sometimes consider a set E as the multiset $E \times \{1\}$ over E that contains one instance of each element of E. For each multiset A over E, and for each $\phi : E \longrightarrow F$, we write $\{\!|\phi(x) \mid x \in A|\!\}$ to denote the multiset over F defined as $\sum_{x \in E} A(x)\{\!|\phi(x)|\!\}$.

Extension of the Notation $\longrightarrow$ to Multisets. Sometimes it will be convenient to extend the notation $E \longrightarrow F$ and to write $A \longrightarrow F$ also when A is a multiset and F is a set.

$$A \longrightarrow F \stackrel{\text{def}}{=} \{\phi \in (A \times B)^{\oplus} \mid A = \{\!|a \mid (a,b) \in \phi|\!\}\} \ .$$

Chapter 1

True Concurrency Models and Unfoldings: State of the Art

This chapter presents the state-of-the-art in unfoldings of low-level Petri nets. It also introduces notations that we will reuse throughout this document. We deal with the case of Petri nets with read arcs in Section 1.4 because we use read arcs in other chapters.

1.1 Petri Nets

Petri nets [83] are one of the most widespread models in the field of modeling and analysis of asynchronous distributed systems. The state of the system is represented by *tokens* that are positioned in *places*. The semantics is based on the *firing* of *transitions*. When a transition fires, it consumes (or deletes) tokens and creates new ones, in general, in other places. This allows an explicit representation of concurrency: if transitions consume different tokens, they can fire independently, without any synchronization. Therefore we consider Petri nets as a "true concurrency model".

1.1.1 Definition

A *Petri net* is a 5-tuple $(P, T, pre, post, M_0)$ where P is a set of *places*, T is a set of *transitions*, *pre* and *post* map each transition $t \in T$ to its *preset* often denoted ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \in P^\oplus$ and its *postset* often denoted $t^\bullet \stackrel{\text{def}}{=} post(t) \in P^\oplus$.

$M_0 \in P^\oplus$ is the *initial marking*. A *marking* of a Petri net is a multiset $M \in P^\oplus$ of places. We say that these places are *marked*, or contain *tokens*.

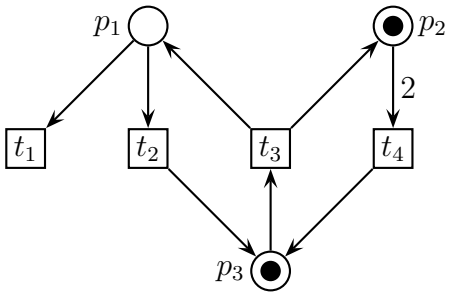


Figure 1.1: A Petri net. The places are p_1, p_2, p_3 and the transitions are t_1, t_2, t_3, t_4. The initial marking is $\{\!|p_2, p_3|\!\}$. The label on the arc from p_2 to t_4 indicates that t_4 consumes two tokens in p_2.

1.1.2 Graphical Representation

Petri nets can be seen as directed bipartite graphs, whose vertices are either places or transitions, and whose arcs indicate the consumption or creation of tokens by the transitions. This offers an easy way to represent the nets graphically. Traditionally the places are represented by circles and the transitions by rectangles. The presets and postsets of the transitions become directed arcs from the places to the transitions and from the transitions to the places. Multiplicities are sometimes represented by integers on these arcs. The markings are represented by tokens in the places.

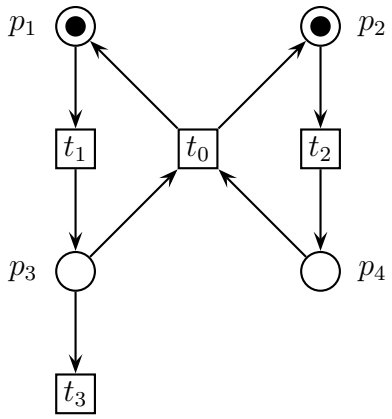
Figure 1.1 shows an example of a Petri net.

1.1.3 Interleaving Semantics

A Petri net starts in its *initial marking* M_0. A transition $t \in T$ is *enabled* in a marking M if all of its input places contain enough tokens: ${}^\bullet t \leq M$. Then t can *fire* from M, which leads to the state $M' \stackrel{\text{def}}{=} (M - {}^\bullet t) + t^\bullet$. We say that t *consumes* the tokens in ${}^\bullet t$ and *creates* the tokens in $t^\bullet$.

We call *firing sequence* any sequence $\sigma \stackrel{\text{def}}{=} (t_1, \ldots, t_n)$ of transitions where there exist markings $M_1, \ldots, M_n$ such that for all $i \in \{1, \ldots, n\}$, firing t_i from M_{i-1} is possible and leads to M_i. In fact the markings $M_1, \ldots, M_n$ are unique; thus M_n is denoted $RS(\sigma)$ and called *the marking reached after* σ. The empty firing sequence is denoted ϵ, and the set of all the firing sequences of a Petri net N is denoted Σ_N (or Σ when no confusion can occur).

A marking M is *reachable* if there exists a firing sequence σ such that $M = RS(\sigma)$.



$\sigma_1 = (t_1, t_2, t_3)$
$\sigma_2 = (t_1, t_2, t_0, t_1)$
$\sigma_3 = (t_2, t_1, t_0, t_1)$

Figure 1.2: A safe Petri net and some of its firing sequences.

A Petri net is called *k-bounded* if for all reachable marking $M \in RS(\Sigma)$ and for all place $p \in P$, $M(p) \leq k$. A 1-bounded Petri net is also called *safe*, and then its markings can be represented by sets rather than multisets.

Examples

In the initial marking of the Petri net of Figure 1.1, only t_3 is enabled. Notice that t_4 cannot fire because it needs two tokens in p_2. Firing t_3 leads to the marking $\{|p_1, p_2, p_2|\}$. Now there are two tokens in p_2, so t_4 is enabled. t_1 and t_2 are also enabled. As t_1 and t_4 consume distinct tokens, they can fire concurrently. On the other hand, t_1 and t_2 consume the same token; we say that they are in conflict. If t_2 fires, the marking $\{|p_2, p_2, p_3|\}$ is reached. Then t_4 can still fire, which leads to $\{|p_3, p_3|\}$.

It is clear that the Petri net is not safe. It is not even bounded: for example, firing n times t_3 and t_2 puts $n + 1$ tokens in p_2.

On the other hand the net of Figure 1.2 is safe.

1.1.4 Marking Graph

A classical way to represent the behavior of a Petri net is to define its marking graph. The nodes of this graph are the reachable markings of the net, and each time a transition t can fire from a marking M and lead to a marking M', an arc labeled by t connects M to M'. If the net is bounded (and the sets of places and transitions are finite), then its marking graph is finite. In Figure 1.3 , we show the marking graph of the safe Petri net of Figure 1.2.

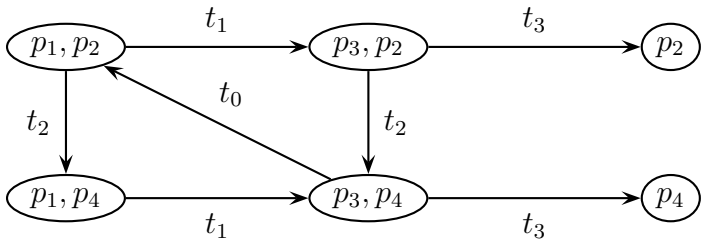


Figure 1.3: The marking graph of the safe Petri net of Figure 1.2. The initial marking is on top left.

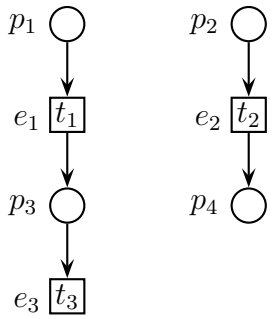
1.1.5 Other True Concurrency Models

Although Petri nets are one of the most widespread, there exist other models that handle concurrency in a comparable way. The most popular are probably process algebras and networks of automata. In this document we deal with several extensions of Petri nets but it seems to be possible to adapt a large part of this work to other true concurrency models. As a matter of fact the unfolding technique that we use was first developed for Petri nets. Only recently it has been applied to other models. Let us mention [45] for networks of automata and [69] for process algebra.

1.2 Partial Order Representation of the Runs: Processes

Although Petri nets are certainly a true concurrency model, their semantics is usually given in terms of firing sequences. But naturally a firing sequence defines a linear order on the actions, as if all of them were executed by a single sequential machine.

In the case of the Petri net of Figure 1.2, the firing of t_1 and t_2 are ordered by the sequential semantics, although these actions are concurrent. This ordering of concurrent actions leads to large marking graphs: each time two (or more) concurrent transitions can fire from the same marking, we observe a pattern that we call a "diamond". We see an example of diamond on the left of Figure 1.3: t_1 and t_2 can fire concurrently from the initial marking $\{p_1, p_2\}$. If we order them as t_1, t_2, we reach first $\{p_3, p_2\}$ and then $\{p_3, p_4\}$. In the order t_2, t_1, the intermediate marking is $\{p_1, p_4\}$, but both trajectories finally meet in the marking $\{p_3, p_4\}$.



$$e_1 = (\{(\bot, p_1)\}, t_1)$$
$$e_2 = (\{(\bot, p_2)\}, t_2)$$
$$e_3 = (\{(e_1, p_3)\}, t_3)$$

Figure 1.4: A process of the safe Petri net of Figure 1.2. This process corresponds to the firing sequence σ_1 of Figure 1.2.

In order to avoid these drawbacks of the sequential semantics, it is sometimes profitable to get rid of the interleavings and to propose a representation of the behaviors that shows clearly the concurrency inside the executions.

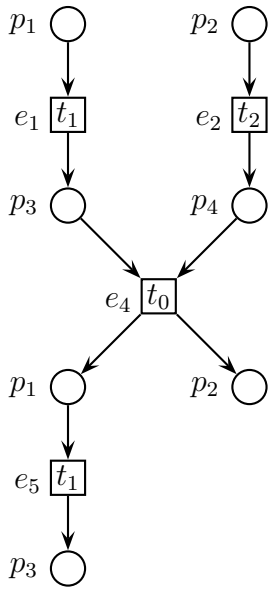
For this reason Petri nets were equipped with a causal semantics [57, 17, 18, 90]. We call it also partial order semantics. The executions are described as *processes* in which the actions are represented by *events*, that are only partially ordered by a *causality* relation. Two events that are not causally related one to the other are explicitly shown as *concurrent*.

Let us mention also that processes can be represented graphically in a very similar way as Petri nets themselves. The events are represented like the transitions. The places are replaced by *conditions* that code the presence of a token in a place at a given time during the execution. The oriented graph that we obtain has some properties; for instance it is acyclic because each new occurrence of a transition is represented by a new event. On this graph the causality and concurrency relations are easy to read: two events are causally related if there exists a path from one to the other. Otherwise they are concurrent.

Figures 1.4 and 1.5 show two processes of the safe Petri net of Figure 1.2. On the right of the processes, we have written the canonical coding of the events that we describe in Section 1.2.1 below.

Minor Restriction on the Structure of the Petri Nets

We need a minor and classical restriction on the structure of the Petri nets in order to deal with processes: we require that all the transitions consume tokens ($^\bullet t \neq \emptyset$) and that the initial marking and the postsets of the transitions are sets, i.e. they do not contain several occurrences of the same place. These nets are called "semi-weighted nets" in [74, 5]. Notice that



$$e_1 = (\{(\bot, p_1)\}, t_1)$$
$$e_2 = (\{(\bot, p_2)\}, t_2)$$
$$e_4 = (\{(e_1, p_3), (e_2, p_4)\}, t_0)$$
$$e_5 = (\{e_4, p_1\}, t_1)$$

Figure 1.5: A process of the safe Petri net of Figure 1.2. This process corresponds to either σ_2 or σ_3.

1. Every safe Petri net where each transition can fire from some reachable state, satisfies our requirement;

2. Every Petri net that does not satisfy our requirement can easily be transformed into a Petri net that satisfies it. If a transition t does not consume any token, it suffices to add a fake initially marked place both as input and output of the transition. If a transition t creates n tokens in a place p, it suffices to add n intermediate places $p_1, \ldots, p_n$ and n intermediate transitions $t_1, \ldots, t_n$, to replace the n occurrences of p in $t^\bullet$ by $\{p_1, \ldots, p_n\}$ and to set ${}^\bullet t_i \stackrel{\text{def}}{=} p_i$ and $t_i{}^\bullet \stackrel{\text{def}}{=} p$ for all $i \in \{1, \ldots, n\}$. The same trick can be used to cope with the restriction on the initial marking.

The use of this restriction will appear clearly in the description of the coding of the processes. For the moment we can say informally that the restriction on the presets of the transitions allows us to distinguish between several events that are different occurrences of a transition, by the conditions that they consume; whereas the restriction on the postsets allows us to identify a condition by the event that created it and the place it corresponds to.

1.2.1 Coding of the processes

This section is a preliminary to the next sections 1.2.2 and 1.2.3: we describe how the processes are coded. We use a canonical coding like in [44]. In Figures 1.4 and 1.5, the coding of the events is written on the right.

Each process will be a set E of *events*. We denote $E_\perp \stackrel{\text{def}}{=} E \cup \{\perp\}$ the set E augmented with a special event, called the *initial event* and denoted $\perp$, which is considered as the origin of the runs. Each event $e \in E$ is a pair (B, t) that codes an occurrence of the transition t (denoted $\tau(e)$) in the process. B (which is denoted ${}^\bullet e$) is a set of pairs $b \stackrel{\text{def}}{=} (f, p) \in E_\perp \times P$. Such a pair is called a *condition* and refers to the token that has been created by the event f (denoted ${}^\bullet b$) in the place p (denoted $place(b)$). Each token of the initial marking is represented by a condition b which is created by the initial event: ${}^\bullet b = \perp$.

We say that the event $e \stackrel{\text{def}}{=} ({}^\bullet e, \tau(e))$ *consumes* the conditions in ${}^\bullet e$. Symmetrically the set $\{(e, p) \mid p \in \tau(e)^\bullet\}$ of conditions that are *created* by e is denoted $e^\bullet$. For the initial event $\perp$ we set $\tau(\perp) \stackrel{\text{def}}{=}$ -, ${}^\bullet\perp \stackrel{\text{def}}{=} \emptyset$ and $\perp^\bullet \stackrel{\text{def}}{=} \{(\perp, p) \mid p \in M_0\}$.

For each set B of conditions, we denote $Place(B) \stackrel{\text{def}}{=} \{\!|place(b) \mid b \in B|\!\}$, and when the restriction of $place$ to B is injective, we denote $place^{-1}_{|B}$ its inverse, and for each $P \subseteq Place(B)$, $Place^{-1}_{|B}(P) \stackrel{\text{def}}{=} \{place^{-1}_{|B}(p) \mid p \in P\}$.

To summarize the coding of the processes, it is convenient to define a set D_N, such that all the events that appear in the processes of a Petri net N, are elements of D_N.

Definition 1.1 (D_N). *We define D_N as the smallest set such that*

$$\forall B \subseteq \bigcup_{e \in D_{N\perp}} e^\bullet \quad \forall t \in T \quad Place(B) = {}^\bullet t \implies (B, t) \in D_N \ .$$

Notice that this inductive definition is initialized by the fact that $\perp \in D_{N\perp}$.

Definition 1.2 (final conditions $\uparrow(E)$). *The set of conditions that remain at the end of the process E (meaning that they have been created by an event of E, and no event of E has consumed them) is*

$$\uparrow(E) \stackrel{\text{def}}{=} \bigcup_{e \in E_\perp} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e \ .$$

For example, the process of Figure 1.4 has only one final condition: (e_2, p_4). The final conditions of the process of Figure 1.5 are (e_5, p_3) and (e_4, p_2).

Definition 1.3 (causality, concurrency, causal past). *We define the relation $\rightarrow$ on the events as: $e \rightarrow e'$ iff $e^\bullet \cap {}^\bullet e' \neq \emptyset$. The reflexive transitive closure $\rightarrow^*$ of $\rightarrow$ is called the* causality *relation. Two events of a process that are not causally related are called* concurrent*. For all event e, we define its* causal past $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$*. We extend this notation to conditions as $\lceil b \rceil \stackrel{\text{def}}{=} \lceil {}^\bullet b \rceil$, and to any set A of events or conditions as $\lceil A \rceil \stackrel{\text{def}}{=} \bigcup_{a \in A} \lceil a \rceil$.*

For example in Figure 1.4, e_1 is in the causal past of e_3, but e_1 and e_2 are concurrent. In Figure 1.5, e_2 is in the causal past of e_5 because e_2 is a cause of e_4, which is a cause of e_5.

Notice that, by definition, two events of a process are either concurrent or causally related. However, when one think of unfoldings, one often associate these relations with a third relation called *conflict*. The notion of *conflict* has no meaning at this stage; it will be defined later (see Definition 1.5), when we superimpose several processes.

The set inclusion between processes is called the *prefix* relation.

1.2.2 Processes of Safe Petri Nets

When a Petri net N is safe, we can easily define a mapping Π_N (or simply Π) from its firing sequences to their partial order representation as processes.

The function Π is defined inductively as follows:

- $\Pi(\epsilon) \stackrel{\text{def}}{=} \emptyset$,
- $\Pi((t_1, \ldots, t_{n+1})) \stackrel{\text{def}}{=} \Pi((t_1, \ldots, t_n)) \cup \{e\}$, where the last firing of the sequence is represented by the event $e \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(E)}({}^\bullet t_{n+1}), t_{n+1})$.

For example, when e_5 was placed in the process of Figure 1.5, we had $E = \{e_1, e_2, e_4\}$ and $\uparrow(E) = \{(e_4, p_1), (e_4, p_2)\}$. As the preset of t_1 is $\{p_1\}$, e_5 consumes the condition $(e_4, p_1) = place^{-1}_{\uparrow(E)}(p_1)$.

In order to validate this definition we have to prove that ${}^\bullet t_{n+1} \subseteq Place(\uparrow(E))$ and that $place_{|\uparrow(E)}$ is injective. This is achieved by the following lemma.

Lemma 1.1. *For all firing sequence σ, $\Pi(\sigma)$ is well defined and $RS(\sigma) = Place(\uparrow(\Pi(\sigma)))$.*

Proof. We prove this lemma by induction. $\Pi(\epsilon)$ is well defined and $\uparrow(\Pi(\epsilon)) = \bot^\bullet$, so $Place(\uparrow(\Pi(\epsilon))) = M_0 = RS(\epsilon)$. Assume that the lemma hold for σ, and let t be a transition that can fire from $RS(\sigma)$. Then ${}^\bullet t \subseteq RS(\sigma) = Place(\uparrow(\Pi(\sigma)))$ and $place_{\uparrow(\Pi(\sigma))}$ is injective since $Place(\uparrow(\Pi(\sigma)))$ equals $RS(\sigma)$, which is a set because we have assumed that N is safe. Then

$\Pi(\sigma \cdot t)$ is well defined. It remains to show that $RS(\sigma \cdot t) = Place(\uparrow(\Pi(\sigma \cdot t)))$. For this we first notice that $e \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(\Pi(\sigma))}({}^\bullet t), t) \notin \Pi(\sigma)$. The reason is that ${}^\bullet e \subseteq \uparrow(\Pi(\sigma))$ whereas for all event $f \in \Pi(\sigma)$, ${}^\bullet f \cap \uparrow(\Pi(\sigma)) = \emptyset$ (by definition of $\uparrow$) and ${}^\bullet f \neq \emptyset$ (because ${}^\bullet \tau(f) \neq \emptyset$). As a consequence $\uparrow(\Pi(\sigma \cdot t)) = (\uparrow(\Pi(\sigma)) \setminus {}^\bullet e) \cup e^\bullet$, and $Place(\uparrow(\Pi(\sigma \cdot t))) = (Place(\uparrow(\Pi(\sigma))) - Place({}^\bullet e)) + Place(e^\bullet) = (RS(\sigma) - {}^\bullet t) + t^\bullet = RS(\sigma \cdot t)$. □

The events of a process are only partially ordered by causality. On the other hand, a firing sequence totally orders the actions. Therefore we can say that the function Π erases from a firing sequence the ordering which is not due to causality.

1.2.3 Processes of General Petri Nets

Limitation of the Construction of the Processes from Firing Sequences via the Function Π

Although the function Π defined above to map each firing sequence of a safe Petri net to a process cannot be simply extended to the case of general nets, we can define the processes of general Petri nets directly.

The state reached after a process E is $Place(\uparrow(E))$. This formula will remain true for processes of unsafe Petri nets, provided it is understood in the sense of multisets, because it will be possible that several conditions of $\uparrow(E)$ correspond to the same place $p \in P$. In this case, if a transition that consumes a token in p occurs after the process E, there are several events that may represent this firing: we have to choose which of the available conditions is consumed by the event. In other terms $place_{|\uparrow(E)}$ is not injective any more and $place^{-1}_{|\uparrow(E)}(p)$, which was used in the definition of the mapping Π in the safe case, is not well defined any more.

However, if there are several ways to choose the consumed condition corresponding to p, we consider that all the choices are valid; they differ by the past (or the origin) of the consumed token corresponding to p, which is natural if we think of causality. As a result we can say that a process of an unsafe Petri net gives more information about causality than a firing sequence. On the other hand a process does not give any information about the ordering of concurrent events. Figure 1.6 illustrates this point.

Direct Definition of the Set of Processes of a Petri Net

We will now define the processes of a Petri net without using firing sequences, in order to avoid the problem mentioned above. This new definition will match the definition via Π in the case of safe Petri nets, but it will also

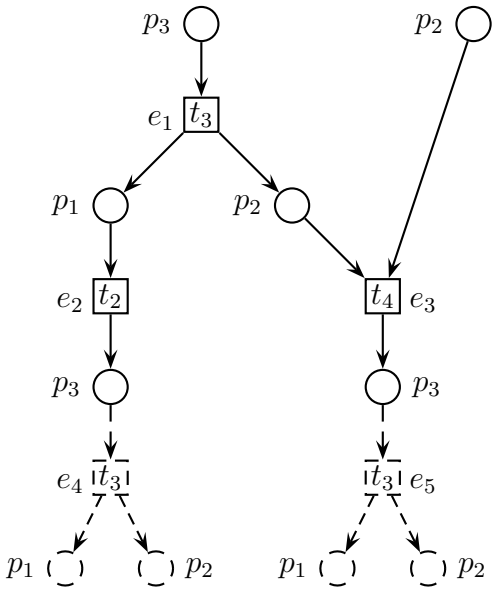


Figure 1.6: A process of the unsafe Petri net of Figure 1.1 (in solid lines). The two final conditions (after e_2 and e_3) corresponding to p_3 give two possible ways to extend the process with a firing of transition t_3. The two corresponding events are shown in dashed lines.

work for unsafe Petri nets. We reuse the same notations as for the processes defined via Π.

Definition 1.4 (processes of a Petri net (direct definition)). *The set X_N of all the processes of a Petri net N is defined inductively as follows.*

- $\emptyset \in X_N$;
- *for all process $E \in X_N$, for all transition t and for all set $B \subseteq \uparrow(E)$ of final conditions of E such that $Place(B) = {}^\bullet t$, $E \cup \{e\} \in X_N$, where the event $e \stackrel{\text{def}}{=} (B, t)$ represents the firing of t that consumes the conditions of B.*

1.3 Branching Processes and Unfoldings

Each process provides a way to represent a (finite) execution of a Petri net as a set of events, that are only partially ordered by the causality relation, so that two events that are not causally related to each other are explicitly shown as concurrent.

As well as a process represents one (finite) execution, it is possible to represent several executions in a single structure. Such a structure is called a *branching process* [76, 44] and simply consists in the superimposition of several processes. In order to avoid confusions, processes are sometimes called *non-branching processes*.

With the coding of processes as sets of events that we have chosen, a branching process can simply be defined as the union of some processes. Then the processes share their common prefixes, as shown in Figure 1.7.

Unlike in processes, two events of a branching process may be neither causally related nor concurrent. Then we say that they are in *conflict*. There are several equivalent ways to define the conflict relation.

Definition 1.5 (conflict). *Two events e_1 and e_2 of D_N are in* conflict *if the following equivalent conditions hold (the equivalence is an immediate consequence of Corollary 1.1 below):*

- *no process contains both e_1 and e_2;*
- *$\lceil e_1 \rceil \cup \lceil e_2 \rceil$ is not a process;*
- *there exists $e'_1 \in \lceil e_1 \rceil$ and $e'_2 \in \lceil e_2 \rceil$ such that $e'_1 \neq e'_2 \ \wedge \ {}^\bullet e'_1 \cap {}^\bullet e'_2 \neq \emptyset$.*

In Figure 1.7, the events e_3 and e_4 are in conflict because they consume the same condition (e_1, p_3).

But it is important that we can still retrieve the processes from their superimposition as a branching process. This theorem shows that it is easy to check if a set of events is a process. By the way, remark that the theorem holds for any subset of D_N, even if some of the events do not actually appear in a process.

Theorem 1.1. *Let $E \subseteq D_N$ be a finite set of events. E is a process iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \end{cases}$$

Proof. We first show by induction on X_N that all the processes satisfy the conditions in the curly brace. This is true for $E = \emptyset$. Let $E \in X_N$ be a process that satisfies the conditions in the curly brace, let $e \stackrel{\text{def}}{=} (B, t)$ with $B \subseteq \uparrow(E)$ and $t \in T$ such that $Place(B) = {}^\bullet t$. Then we will show that the process $E' \stackrel{\text{def}}{=} E \cup \{e\}$ also satisfies the conditions in the curly brace. By construction E' is causally closed. Moreover for each condition $b \in {}^\bullet e$ that is consumed by e, $b \in \uparrow(E)$, which implies that b has not been consumed by any event of E. Thus for all $f \in E$, ${}^\bullet e \cap {}^\bullet f = \emptyset$. So E' is conflict free.

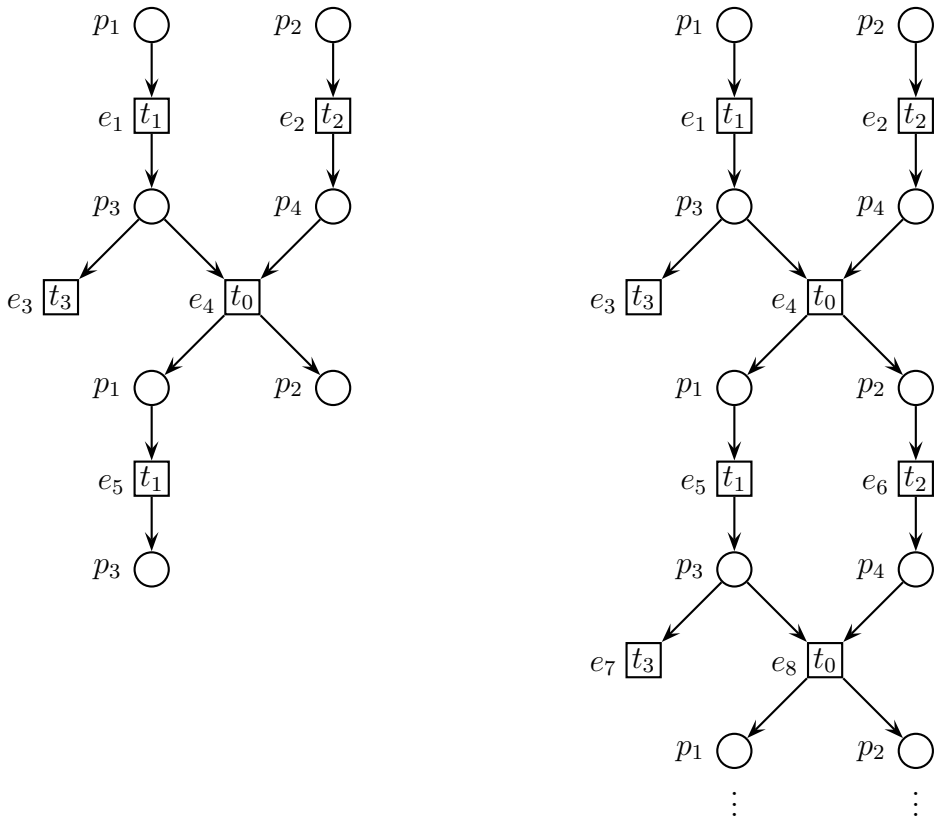


Figure 1.7: On the left, the branching process obtained by superimposing the processes of Figures 1.4 and 1.5. The events e_1 and e_2 that appear in both processes, are not duplicated.
On the right, a larger branching process of the Petri net of Figure 1.2. The dots at the bottom show how to continue the unfolding.

Conversely let E' satisfy the conditions in the curly brace. If $E' = \emptyset$, then $E' \in X_N$. Otherwise let $e \in E'$ be an event that has no successor by $\rightarrow$ in E' (the inductive definition of D_N ensures that $\rightarrow$ is acyclic). $E \stackrel{\text{def}}{=} E' \setminus \{e\}$ satisfies the conditions in the curly brace. Assume that $E \in X_N$. As E' is conflict free, ${}^\bullet e \subseteq \uparrow(E)$. And as $e \in D_N$, $Place({}^\bullet e) = {}^\bullet\tau(e)$. Thus $E' = E \cup \{e\} \in X_N$. □

As a corollary, it is also easy to check if a set E of events is a subset of a process.

Corollary 1.1. *For all set $E \subseteq D_N$,*

$$(\exists F \in X_N \quad E \subseteq F) \quad \textit{iff} \quad \lceil E \rceil \in X_N \ .$$

Proof. If $F \in X_N$, then F is conflict free. As $E \subseteq F$, then $\lceil E \rceil \subseteq \lceil F \rceil = F$. So $\lceil E \rceil$ is conflict free. And $\lceil E \rceil$ is causally closed. Then $\lceil E \rceil$ is a process. Conversely, if $\lceil E \rceil$ is a process, then $\lceil E \rceil$ is a suitable F. □

Definition 1.6 (unfolding). *It is possible to superimpose all the processes of a Petri net N inside a single branching process. The result is called the* unfolding *of N and is denoted U_N (or U when no confusion can occur). U_N is formally defined as:*

$$U_N \stackrel{\text{def}}{=} \bigcup_{E \in X_N} E \ .$$

The branching process on the right of Figure 1.7 shows a part of the unfolding.

Now we mention an important property that allows us to check directly if an event of D_N is in the unfolding. Then the unfolding can be built simply by selecting the valid events from D_N.

Theorem 1.2. *For all event $e \in D_N$, $e \in U_N$ iff $\lceil e \rceil$ is a process.*

Proof. It suffices to apply Corollary 1.1 with $E \stackrel{\text{def}}{=} \{e\}$. □

So checking if an event e is in the unfolding amounts to checking that its causal past is a process, which is done using Theorem 1.1.

Remark About the Binary Conflict

Notice that in unfoldings of Petri nets, a causally closed set of events are incompatible (i.e. cannot happen together in the same process) iff two of them are in conflict. Therefore we can say that the conflict is binary.

We emphasize the fact that this property will not hold any more in unfoldings of higher-level models like Petri nets with read arcs or colored Petri nets. In these higher-level models we will often keep calling *conflict* the fact that two distinct events consume a single condition, which is an obvious cause of incompatibility; but this will not remain the unique cause of incompatibility between sets of events.

1.3.1 Branching Processes and Unfoldings as Particular Petri Nets: Occurrence Nets

In this document we have chosen to represent (branching) processes as sets of events, and to code the causality relation in the events. But we remark that the graphical representation of (branching) processes is very close to the graphical representation of Petri nets.

In the literature, (branching) processes are often defined using particular Petri nets, which are called *occurrence nets*. It is a fact that for each (branching) process E, we can build a Petri net N_E that shows the structure of E. The transitions of N_E are the events of E, the places of N_E are the conditions of E and the initial marking is $\bot^\bullet$.

Formally: $N_E \stackrel{\text{def}}{=} (P_E, T_E, pre_E, post_E, M_{0E})$, with $P_E \stackrel{\text{def}}{=} \bigcup_{e \in E_\bot} e^\bullet$, $T_E \stackrel{\text{def}}{=} E$, $pre_E(e) \stackrel{\text{def}}{=} {}^\bullet e$, $post_E(e) \stackrel{\text{def}}{=} e^\bullet$ and $M_{0E} \stackrel{\text{def}}{=} \bot^\bullet$.

Occurrence nets have several properties:

- For each place p, if p is marked initially, then there is no transition t such that $p \in t^\bullet$, and if p is not marked initially, then there is exactly one such transition t;
- The causality relation $\rightarrow$ (defined as $t_1 \rightarrow t_2$ iff ${t_1}^\bullet \cap {}^\bullet t_2 \neq \emptyset$) is acyclic;
- In non-branching process, there is no conflict (there exists no distinct transitions t_1 and t_2 such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$); in branching processes, there may be conflicts, but if t_1 and t_2 are in conflict, then they cannot be causally related to a single transition t (there exists no t, t_1, t_2 such that $t_1 \neq t_2$ and $t_1 \rightarrow^* t$ and $t_2 \rightarrow^* t$ and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$).

Occurrence nets provide a way to code the structure of (branching) processes. But the conditions and events must also be related to the places and transitions of the original Petri net. For this a *morphism* maps the events to transitions and the conditions to places. Under some conditions, a pair (O, h), where O is an occurrence net and h is a morphism from O to N, is called a branching process of N.

Finally one can show that there exists a maximal branching process up to isomorphism. This object is called the unfolding.

In our approach both the structure and the morphism are coded in the events. Then we manipulate sets of events rather than occurrence nets and morphisms. And the unfolding is simply the union of all the processes.

1.3.2 Finite Complete Prefixes

Although the unfolding is in general an infinite structure, McMillan [72] showed that there exists a finite prefix of the unfolding that contains enough information to check many properties of the model. This technique was improved by Esparza, Römer and Vogler [46] and used for model-checking [47, 58, 65]. Its efficiency is due to the fact that it avoids the computation of the interleavings and the explosion of the state space. In [67] the finite complete prefix was defined in a canonical way. In [1], finite complete prefixes were even used to check safety properties in unbounded nets.

1.4 Extension to Petri Nets with Read Arcs

Many distributed systems allow read-only access to some data. Thus these non-destructive accesses can be done concurrently by several components of the system. In order to model these read-only accesses with Petri nets, a classical method is to consume and rewrite a token. Nevertheless this technique is not satisfactory when one is dealing with causal semantics because the consumption of the token forces to order the events that access the same data.

In order to solve this problem, read arcs were added to Petri nets [37, 75]. This extension is now quite commonly used and partial order semantics were proposed for this new model [5, 26, 94, 91]. In this connection inhibitor arcs were also introduced [60, 37]. Their expressive power is similar to the one of read arcs in the case of bounded nets. Finite complete prefixes of Petri nets with read arcs (also called contextual Petri nets) were first defined in the restricted case of *read-persistent nets* [92], and later in the general case [96].

We introduce now Petri nets with read arcs. We will use read arcs later for unfoldings of dynamic nets in Section 2.2, and in the framework of timed models in Chapter 3.

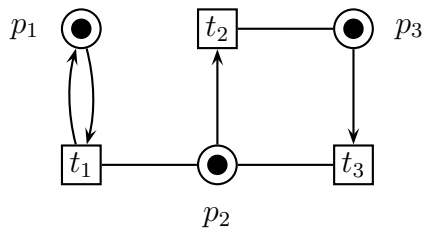


Figure 1.8: A Petri net with read arcs. Transitions t_1 and t_3 can fire concurrently because they only read the token in p_2.

1.4.1 Petri Nets with Read Arcs: Definition

A *Petri net with read arcs* (or *contextual Petri net*) is a 6-tuple $(P, T, pre, cont, post, M_0)$ where $(P, T, pre, post, M_0)$ is a Petri net and *cont* defines a set of *read arcs* by assigning a *context* $cont(t) \in 2^P$ (often denoted $\underline{t}$) to each transition $t \in T$. We denote ${}^\bullet\underline{t} \stackrel{\text{def}}{=} {}^\bullet t + \underline{t}$.

Notice that the context of a transition is defined as a *set* of places rather than as a multiset. Indeed as read arcs aim at allowing concurrent access to a single token, it is natural to allow even a single transition to read several times the same token. In this case having several read arcs from one place to one transition would be strictly equivalent to having only one read arc.

The read arcs are represented as follows: for each transition $t \in T$, for each place $p \in \underline{t}$, a line without arrows is drawn between p and t.

Figure 1.8 shows an example of Petri net with read arcs.

1.4.2 Interleaving Semantics

In the literature, one can find several slightly different definitions of the semantics of Petri nets with read arcs. For instance in [92], a transition can read and consume the same token, while this is forbidden in [5]. We adopt the latter semantics.

It is also worth noticing that, when the nets are bounded, the use of complementary places allows us to translate Petri nets with inhibitor arcs into Petri nets with read arcs, like in [60].

Sometimes the semantics allow to fire several transitions simultaneously, in what is called a *step* [60]. Even if the relevance of these steps in an asynchronous framework can be debated, we notice that such semantics may yield more reachable states than a semantics without steps. This subtlety does not exist in the context of Petri nets without read arcs.

Anyway, whatever semantics we choose, we need to refine the causality relation with a conditional or weak causality when we deal with partial order semantics of Petri nets with read arcs. This new type of causality will be defined in Section 1.4.3.

A transition $t \in T$ of a Petri net with read arcs is *enabled* in a marking M if all its input places and context places contain enough tokens: ${}^\bullet\underline{t} \leq M$. Then t can *fire* from M, which leads to the state $M' \stackrel{\text{def}}{=} (M - {}^\bullet t) + t^\bullet$. We notice that the tokens that are only read are not taken into account in the definition of the new marking. They are only a constraint on the enabling of the transition.

The notion of firing sequence of Petri nets with read arcs is identical to the case without read arcs. Nevertheless the notion of process has to be revisited.

1.4.3 Processes

The restrictions that were imposed in Section 1.2 on the structure of the Petri nets without read arcs in order to deal with processes, still hold when we deal with processes of Petri nets with read arcs.

In processes of Petri nets without read arcs, an event $e \stackrel{\text{def}}{=} ({}^\bullet e, \tau(e))$ is identified by the set of conditions that it consumes, and the transition that fires. When dealing with read arcs, the events will also contain information about the tokens that are read when the transition fires. Thus an event becomes a triple[1] $({}^\bullet e, \underline{e}, \tau(e))$ where $\underline{e}$ is the set of conditions that are read. We denote ${}^\bullet\underline{e} \stackrel{\text{def}}{=} {}^\bullet e \cup \underline{e}$.

The set $e^\bullet \stackrel{\text{def}}{=} \{(e,p) \mid p \in \tau(e)^\bullet\}$ of conditions that are *created* by the event e remains like in Petri nets without read arcs.

Like the case of Petri nets without read arcs, we can define a set D_N, such that all the events appearing in the processes of a Petri net with read arcs N, will be elements of D_N.

Definition 1.7 (D_N). *We define D_N as the smallest set such that for all $C, R \subseteq \bigcup_{e \in D_N \perp} e^\bullet$ such that $C \cap R = \emptyset$, and for all $t \in T$,*

$$\left.\begin{array}{l} Place(C) = {}^\bullet t \\ Place(R) = \underline{t} \end{array}\right\} \implies (C, R, t) \in D_N \ .$$

[1] A pair (B,t) would be sufficient for safe nets because the sets of read and consumed conditions could be defined as ${}^\bullet e \stackrel{\text{def}}{=} Place_{|B}^{-1}({}^\bullet t)$ and $\underline{e} \stackrel{\text{def}}{=} Place_{|B}^{-1}(\underline{t})$. But this would not be suitable for unsafe nets.

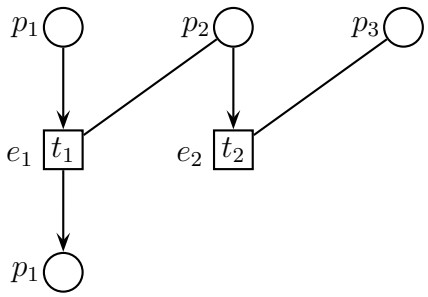


Figure 1.9: A process of the Petri net with read arcs of Figure 1.8. In this process, e_1 must fire before e_2 because e_2 consumes the condition $(\perp, p_2)$ that e_1 reads. Nevertheless e_1 is not in the causal past of e_2, thus $\{e_2\}$ is also a valid process. This phenomenon is called *conditional* or *weak* causality, and we denote $e_1 \nearrow e_2$.

The set of conditions that remain at the end of the process E is not changed by the addition of read arcs:

$$\uparrow(E) \stackrel{\text{def}}{=} \bigcup_{e \in E_\perp} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e \ .$$

Unconditional (or Strong) and Conditional (or Weak) Causality.

The most important change in processes when we deal with read arcs concerns the notion of causality. When there are no read arcs, if two events e and f are causally related ($e \rightarrow^+ f$), then:

- e occurs in every process where f occurs, and
- if a process contains e and f, then e occurs before f.

In processes with read arcs, these two notions have to be distinguished, as shown by Figure 1.9. The first one is called *unconditional* or *strong* causality and denoted $\rightarrow$ and the second one is called *conditional* or *weak* causality and denoted $\nearrow$. They are formally defined as:

- $e \rightarrow f$ iff $e^\bullet \cap {}^\bullet\underline{f} \neq \emptyset$ and
- $e \nearrow f$ iff $(e \rightarrow f) \vee (\underline{e} \cap {}^\bullet f \neq \emptyset)$.

In the context of processes of Petri nets with read arcs, the causal past of an event e is defined as the set of events that unconditionally occur before e: $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$. We extend this notation to conditions and to sets of events or conditions like we did for processes without read arcs.

Definition 1.8. *The set X_N of all the processes of a Petri net with read arcs N is defined inductively as follows.*

- $\emptyset \in X_N$;
- *for all process E, for all transition t, for all set $C \subseteq \uparrow(E)$ of final conditions of E such that $Place(C) = {}^\bullet t$ and for all set $R \subseteq \uparrow(E) \setminus C$ of final conditions of E such that $Place(R) = \underline{t}$, $E \cup \{e\} \in X_N$, where the event $e \stackrel{\text{def}}{=} (C, R, t)$ represents the firing of t that consumes the conditions of C and reads the conditions of R.*

1.4.4 Unfoldings

As well as in the case of Petri nets without read arcs, the superimposition of all the processes of a Petri net N with read arcs is called the *unfolding* of N and is denoted U_N (or U when no confusion can occur).

Definition 1.9 (unfolding of a Petri net with read arcs). *The formal definition of U_N is exactly the same as for Petri nets without read arcs: $U_N \stackrel{\text{def}}{=} \bigcup_{E \in X_N} E$.*

Figure 1.10 shows a prefix of the unfolding of the Petri net with read arcs of Figure 1.8.

In the context of Petri nets without read arcs, Theorem 1.1 allows us to check easily if a set of events is a process. When dealing with read arcs, this theorem has to be rewritten in order to deal with the notion of conditional causality. As a matter of fact the conditional causality can cause a kind of conflict between several events that are not in conflict in the sense of Petri nets without read arcs. This phenomenon is often called *asymmetric conflict* in the literature and happens when several events $e_0, e_1, \ldots, e_n$ are such that $e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0$, i.e. the conditional causality creates a cycle on these events.

We see an example in Figure 1.10: the two events e_2 and e_3 cannot be in the same process because e_2 should read the token in p_3 before it is consumed by e_3 and e_3 should read the token in p_2 before it is consumed by e_2. This leads to the cycle $e_2 \nearrow e_3 \nearrow e_2$.

Theorem 1.3. *Let $E \subseteq D_N$ be a finite set of events. E is a process iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \\ \nexists e_0, e_1, \ldots, e_n \in E \quad e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0 & \textit{($\nearrow$ is acyclic on E)} \end{cases}$$

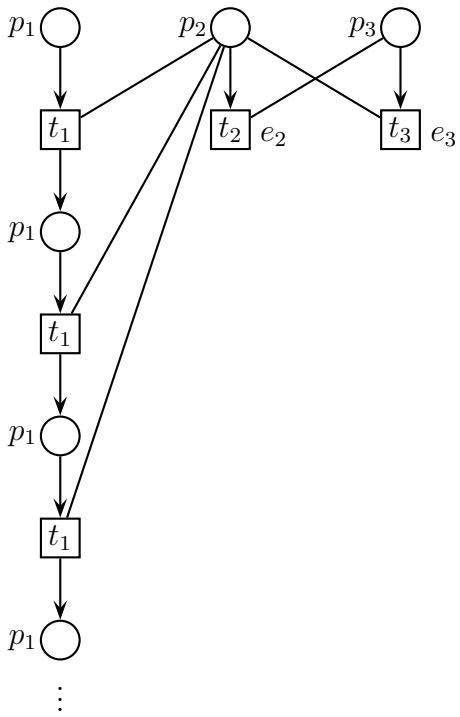


Figure 1.10: A branching process of the Petri net with read arcs of Figure 1.8. The dots at the bottom show how to continue the unfolding.

Proof. We first show by induction on X_N that all the processes satisfy the conditions in the curly brace. This is true for $E = \emptyset$. Let $E \in X_N$ be a process that satisfies the conditions in the curly brace, let $e \stackrel{\text{def}}{=} (C, R, t)$ with $t \in T$, $R \subseteq \uparrow(E)$ and $C \subseteq \uparrow(E) \setminus R$ such that $Place(C) = {}^\bullet t$ and $Place(R) = \underline{e}$. Then we will show that the process $E' \stackrel{\text{def}}{=} E \cup \{e\}$ also satisfies the conditions in the curly brace. By construction E' is causally closed. Moreover for each condition $b \in {}^\bullet e$ that is consumed by e, $b \in \uparrow(E)$, which implies that b has not been consumed by any event of E. Thus for all $f \in E$, ${}^\bullet e \cap {}^\bullet f = \emptyset$ and $\neg(e \nearrow f)$. So E' is conflict free and $\nearrow$ is acyclic on E'.

Conversely let E' satisfy the conditions in the curly brace. If $E' = \emptyset$, then $E' \in X_N$. Otherwise let $e \in E'$ be an event that has no successor by $\nearrow$ in E' (such an event exists since $\nearrow$ is acyclic on E'). $E \stackrel{\text{def}}{=} E' \setminus \{e\}$ satisfies the conditions in the curly brace. Assume that $E \in X_N$. As E' is conflict free, ${}^\bullet e \subseteq \uparrow(E)$. And as e has no successor by $\nearrow$ in E', $\underline{e} \subseteq \uparrow(E)$. Furthermore $e \in D_N$, so $C \cap R = \emptyset$ and $Place({}^\bullet e) = {}^\bullet \tau(e)$ and $Place(\underline{e}) = \underline{\tau(e)}$. Thus $E' = E \cup \{e\} \in X_N$. □

Corollary 1.1 and Theorem 1.2 can be rewritten exactly like in the case of Petri nets without read arcs (see Section 1.3).

Chapter 2

Higher-Level Models

Although they offer a satisfactory representation of concurrency, Petri nets are often difficult to use to model even small real systems. Their drawback is that the state of the system is only represented by the position of the tokens in the places. Consequently, in order to distinguish between the different values that a variable of the system can take, the simplest way is often to use one place per value. Even if more subtle codings are possible, the number of necessary places and transitions becomes very large, or even infinite, which makes the system very hard to comprehend. For this reason several extensions of Petri nets have been proposed, like the well-known colored Petri nets [63].

In this section we present two contributions of this thesis. The first contribution is the definition of symbolic unfoldings of colored Petri nets. In [66], Khomenko and Koutny, developed a notion of unfoldings for high-level Petri nets, which is based on a transformation of the high-level model into a low-level model, in order to reuse the unfolding technique that was developed for low-level models. We call this method *expanded unfolding*. Our method yields a much more compact structure, where the executions are grouped into symbolic processes that reflect the generic aspect of the model. An earlier version of this work was published in [32].

The second contribution deals with a very general version of high-level Petri nets, where the goal is to take into account dynamic structural changes inside the distributed system. Many distributed systems can now be dynamically reconfigured. In particular we think about large telecommunications systems. *Web services* also seem to give examples of such behaviors. The originality of this contribution is showing that the symbolic unfolding approach can be applied to very general models.

However it seems that it is difficult to extend the notion of finite complete prefixes to high-level Petri nets when there are infinitely many colors.

2.1 Colored Petri nets

Colored Petri nets were defined by Jensen in [63]. In these nets, each token carries some information, traditionally called the *color* of the token. Of course it is possible for the transitions to test the color of the tokens they consume, and the color of the created tokens may depend on the color of the consumed tokens. These constraints on the values of the tokens are called *guards*. This extension of Petri nets is widely used because it simplifies a lot the modeling of real systems.

We do not include read arcs in the definition of colored Petri nets in order to simplify the reading of this section; but one can very easily deal with read arcs and symbolic unfoldings together. This will be used in Section 2.2 and in Chapter 3.

2.1.1 Definition

A *colored Petri net* is a tuple $(P, T, V, PAR, pre, post, \alpha, \beta, \mathcal{S}_0)$ where

- P is a set of *places*;
- T is a set of *transitions*;
- V is a set of *values* (or *colors*);
- PAR is a set of *parameters*;
- *pre* and *post* map each transition $t \in T$ to its *preset* often denoted ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \in P^\oplus$ and its *postset* often denoted $t^\bullet \stackrel{\text{def}}{=} post(t) \in P^\oplus$.
- α maps each transition $t \in T$ to a *guard* $\alpha(t) : PAR \times ({}^\bullet t \longrightarrow V) \longrightarrow \mathbf{bool}$.
- β maps each transition $t \in T$ to a mapping $\beta(t) : PAR \times ({}^\bullet t \longrightarrow V) \longrightarrow (t^\bullet \longrightarrow V)$.
- $\mathcal{S}_0$ is the *initial state*; a *state* is a multiset $\mathcal{S}$ of pairs $(p, v) \in P \times V$ called *tokens*. Sometimes it is convenient to define the *marking* as the multiset $M_\mathcal{S} \stackrel{\text{def}}{=} \{\!|p \mid (p, v) \in \mathcal{S}|\!\}$; it says only how many tokens are in each place, but does not give any information about their values. Remark that $\mathcal{S}$ can be seen as a mapping from $M_\mathcal{S}$ to V. $M_\mathcal{S}$ is simply denoted M when no confusion can occur. Sometimes also we write for example M_1 instead of $M_{\mathcal{S}_1}$ or M' instead of $M_{\mathcal{S}'}$.

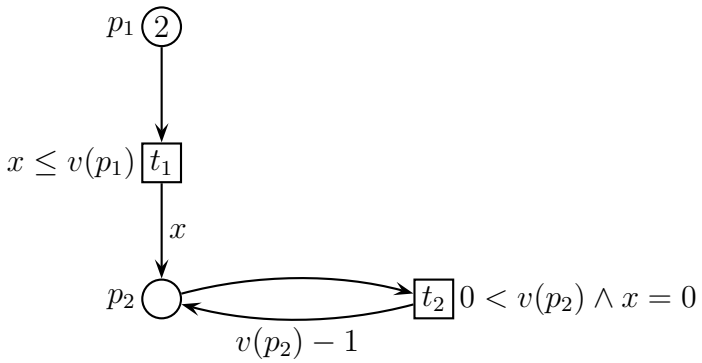


Figure 2.1: A colored Petri net. The sets of values and parameters are $V = PAR \stackrel{\text{def}}{=} \{0, 1, 2\}$. The transition t_1 puts a token in p_2 with any value x smaller than the value $v(p_1)$ of the token it consumes in p_1. The transition t_2 decreases the value of the token in p_2.

Example. Figure 2.1 shows a colored Petri net. The values of the tokens are written in the places instead of the black dots that we use in low-level Petri nets. The guard $\alpha(t)(x, v)$ is written near each transition t, and the value $\beta(t)(x, v)(p)$ is written on the arc that connects a transition t to a place $p \in t^\bullet$.

Discussion About the Parameters and Comparison with Other Definitions of High-Level Petri Nets in the Literature

In our model, the parameters are used to model local non-determinism: even with a fixed set of tokens as input, a transition may produce different outputs, depending on the parameter that is assigned to this action.

For example, the transition t_1 of the Petri net of Figure 2.1 has a non-deterministic behavior: if the consumed token carries the value 2, the created token may carry either 0, 1 or 2. On the other hand, the fact that the parameter x is not used in the output of t_2 means that t_2 has a deterministic behavior. We have even imposed in the guard of t_1 that the parameter x always take the value 0 when t_1 fires.

If local non-determinism does not occur, the parameters are not useful any more, and we can say that the unfoldings that we define are not symbolic any more, since the value of the token in each condition is known.

In the standard notations for high-level nets, this non-determinism is not highlighted, but the guards of the transitions define directly the set of all the combinations of input colors and output colors that are possible. This is the

case for colored nets in [63] or M-nets in [19] for example. The advantage of the latter definition is the symmetry of the guards, which we lose deliberately in order to highlight the non-determinism.

2.1.2 Interleaving Semantics

A transition $t \in T$ can fire with parameter $x \in PAR$ from a state $\mathcal{S}$ consuming the tokens $C \leq \mathcal{S}$ if

- The consumed tokens are in the input places of the transition; using the definition of maps as (multi)sets this can be written: $C \in {}^\bullet t \longrightarrow V$;
- The parameter x and the values of the consumed tokens satisfy the guard of t: $\alpha(t)(x, C)$.

The firing of t from $\mathcal{S}$ with the parameter x consuming the tokens C leads to the state $\mathcal{S}' \stackrel{\text{def}}{=} (\mathcal{S} - C) + \beta(t)(x, C)$.

Firing Sequences

If one wants to represent an execution by a firing sequence, the sequence of transitions is of course not precise enough; one needs to give also the parameter x with which each transition fires. If the Petri net is safe, this gives enough information. But otherwise, apart from the problem related to the notion of causality that was pointed out in Section 1.2.3, another problem arises.

It may occur that a transition consumes one token in a place that contains two tokens. If the tokens do not carry the same value, but both values satisfy the guard of the transition together with the parameter x, then one needs to know which token is consumed. Thus in general a firing sequence has to be defined as a sequence $\sigma \stackrel{\text{def}}{=} ((t_1, x_1, C_1), \ldots, (t_n, x_n, C_n))$, even if the sequence $((t_1, x_1), \ldots, (t_n, x_n))$ would be sufficient if the Petri net is safe.

2.1.3 Expansion into Low-Level Petri Net

There exists a classical method to expand colored Petri nets into low-level Petri nets. It allows even to use high-level Petri nets as simple generators of low-level models. This operation is sometimes called *unfolding* in the literature [19], but we call it *expansion* in order to avoid confusion with the unfoldings that we use throughout this document.

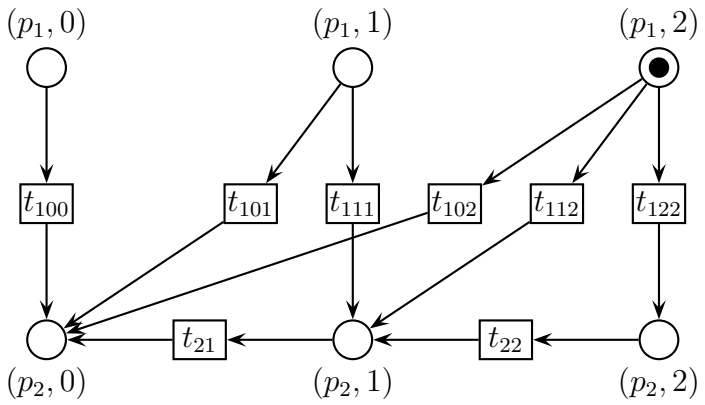


Figure 2.2: The expansion of the colored Petri net of Figure 2.1. The expanded transitions are denoted $t_{1ij} \stackrel{\text{def}}{=} (t_1, i, \{(p_1, j)\})$ and $t_{2i} \stackrel{\text{def}}{=} (t_2, 0, \{(p_2, i)\})$.

Definition 2.1 (expansion $Exp(N)$). *One can translate any colored Petri net*

$$N \stackrel{\text{def}}{=} (P, T, V, PAR, pre, post, \alpha, \beta, \mathcal{S}_0)$$

into a low-level Petri net $Exp(N)$ called the expansion *of N and defined as $Exp(N) \stackrel{\text{def}}{=} (P', T', pre', post', M_0')$ with:*

- $P' \stackrel{\text{def}}{=} P \times V$
- $T' \stackrel{\text{def}}{=} \{(t, x, L) \in T \times PAR \times ({}^\bullet t \longrightarrow V) \mid \alpha(t)(x, L)\}$
- $pre'((t, x, L)) \stackrel{\text{def}}{=} L$
- $post'((t, x, L)) \stackrel{\text{def}}{=} \beta(t)(x, L)$
- $M_0' \stackrel{\text{def}}{=} \mathcal{S}_0$.

Of course we expect finite sets of P' and T' only if the sets of colors and parameters are finite.

Figure 2.2 shows the expansion of the colored Petri net of Figure 2.1.

2.1.4 Expanded Unfolding

It is possible to define the unfolding of a colored Petri net N simply as the unfolding $U_{Exp(N)}$ of the low-level Petri net $Exp(N)$. This is the approach

of [66, 65] for a model of high-level Petri nets called *M-nets*. We call the result the *expanded unfolding* of N.

2.1.5 Processes of Colored Petri Nets and Symbolic Unfoldings

We will now present our definition of symbolic unfoldings of colored Petri nets. It was published in [32], where we also advocated their interest for diagnosis of distributed systems. We will describe the application to diagnosis in Chapter 4. Processes of high-level Petri nets were studied independently by Ehrig, Hoffmann, Padberg, Baldan and Heckel in [42] and by Fleischhack and Pelz in [53].

Instead of dealing with the processes of the expansion $Exp(N)$ of a colored Petri net N, we can define directly the processes of N. The latter will sometimes be called high-level processes in contrast to low-level processes of low-level Petri nets. High-level processes aim at being used to define the symbolic unfolding of N, and thus they are designed so that they can be efficiently superimposed.

In Theorem 2.1, we exhibit a bijection between the processes of N and those of $Exp(N)$, which can be seen as a way to cross-validate the definitions.

Generic Executions for a Generic Model

In high-level processes we definitely want to benefit from the generic aspects of the colored Petri net that we are considering. Indeed in a colored Petri net, if several states share the same marking (that is the tokens are in the same places but do not carry the same values), we can view these states as instances of a generic family of states. Similarly each transition is a generic representation of a family of actions, that differ only by the values of the tokens that are consumed and created. And we consider that grouping several states into a generic state or several actions into a high-level transition, results from a choice that was done when the system was modeled.

With respect to this choice, we can identify families of executions of a colored Petri net N based on the generic aspects related to its places and transitions. To do this, our approach is based on the fact that each execution of N can be mapped into an execution of the underlying low-level Petri net $Sup(N)$. We give the formal definition of $Sup(N)$.

Definition 2.2 ($Sup(N)$). *We define the low-level Petri net $Sup(N)$ obtained by forgetting the values, guards and parameters in a colored Petri net N as: $Sup(N) \stackrel{\text{def}}{=} (P, T, pre, post, M_{\mathcal{S}_0})$.*

If we deal with the interleaving semantics, it is clear that for all firing sequence $((t_1, x_1, C_1), \ldots, (t_n, x_n, C_n))$ of N, $(t_1, \ldots, t_n)$ is a firing sequence of $Sup(N)$. For each firing sequence of $Sup(N)$, there may be zero, one or several corresponding firing sequences of N. In the latter case, we would like to consider these sequences of N as several instances of a single generic firing sequence.

Keeping the same idea in mind, we will define the high-level processes of a colored Petri net N so that the underlying low-level process of $Sup(N)$ is highlighted.

From this point of view, the drawback of the processes of $Exp(N)$ is that the generic aspect of the model is hidden, because it has been destroyed by the expansion.

Definition of the Processes

A process of a colored Petri net N is built over a process E of the underlying low-level Petri net $Sup(N)$. Thus $Sup(N)$ is supposed to satisfy the conditions imposed in Section 1.2. This implies that M_0 is a set and allows us to denote $\mathcal{S}_0(p)$ the value of the unique token that stands in a place $p \in M_0$ in the initial marking. The low-level process E gives only the structure of the high-level process, i.e. the information about the transitions that fire and the places that contain tokens.

To complete this information, we need to assign a parameter to each event of E. Thus a process of N will be given as a set $\mathcal{E}$ of pairs $(e, x) \in E \times PAR$ such that $\mathcal{E} \in E \longrightarrow PAR$. So, the parameter that corresponds to an event $e \in E$ is $\mathcal{E}(e)$.

Remark that the low-level process over which a high-level process $\mathcal{E}$ is built can be retrieved as $E_{\mathcal{E}} \stackrel{\text{def}}{=} \{e \mid (e, x) \in \mathcal{E}\}$. It is simply denoted E when no confusion can occur. Sometimes also we write for example E_1 instead of $E_{\mathcal{E}_1}$ or E' instead of $E_{\mathcal{E}'}$.

One can compute the value $val_{\mathcal{E}}(b)$ of the token corresponding to a condition $b \stackrel{\text{def}}{=} (e, p) \in \bigcup_{e \in E_\perp} e^\bullet$ as follows:

$$val_{\mathcal{E}}(b) \stackrel{\text{def}}{=} \begin{cases} \mathcal{S}_0(p) & \text{if } e = \perp \\ \beta(\tau(e))(\mathcal{E}(e), tok_{\mathcal{E}}({}^\bullet e)) & \text{if } e \in E \end{cases}$$

where for all condition $b \in \bigcup_{e \in E_\perp} e^\bullet$, $tok_{\mathcal{E}}(b) \stackrel{\text{def}}{=} (place(b), val_{\mathcal{E}}(b))$, which makes $tok_{\mathcal{E}}({}^\bullet e)$ be a mapping from ${}^\bullet\tau(e) = Place({}^\bullet e)$ to V.

This allows us to define the state that is reached after a process $\mathcal{E}$ as $RS(\mathcal{E}) \stackrel{\text{def}}{=} tok_{\mathcal{E}}(\uparrow(E))$.

Definition 2.3. *The set X_N of all the processes of a colored Petri net N is defined inductively as follows.*

- $\emptyset \in X_N$;
- *for all process $\mathcal{E} \in X_N$, for all transition t, for all set $C \subseteq \uparrow(E)$ of final conditions of E such that $Place(C) = {}^\bullet t$, for all parameter $x \in PAR$ such that $\alpha(t)(x, tok_{\mathcal{E}}(C))$, $\mathcal{E} \cup \{(e, x)\} \in X_N$, where the event $e \stackrel{\text{def}}{=} (C, t)$ represents the firing of t that consumes the tokens $tok_{\mathcal{E}}(C)$ (which correspond to the conditions of C).*

Figure 2.3 shows the representation of an execution of the colored Petri net of Figure 2.1, both as an expanded process and as a high-level process. We remark that the high-level process is made of a process of $Sup(N)$, plus information about the parameters, that is shown in round brackets near the events. Near each condition b of the high-level process $\mathcal{E}$, we have written the value of $val_{\mathcal{E}}(b)$ in round brackets, but this information is not part of the coding of the high-level process, since by definition it can be computed from the values of the parameters.

It is clear from the definitions and from Figure 2.3 that there is a bijection between the expanded processes and the high-level processes. We define this bijection formally now.

Theorem 2.1. *There is a bijection $\Phi : X_N \longrightarrow X_{Exp(N)}$ from the processes of a colored Petri net N and the processes of its expansion $Exp(N)$. Φ and its inverse Ψ can be defined as follows.*

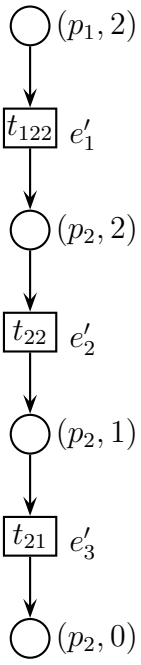
- *The process $\Phi(\mathcal{E})$ of $Exp(N)$ which corresponds to a process $\mathcal{E}$ of N is defined as $\Phi(\mathcal{E}) \stackrel{\text{def}}{=} \bigcup_{e \in E} \phi_{\mathcal{E}}(e)$, where $\phi_{\mathcal{E}}$ maps each event of E to the corresponding event in $\Phi(\mathcal{E})$:*

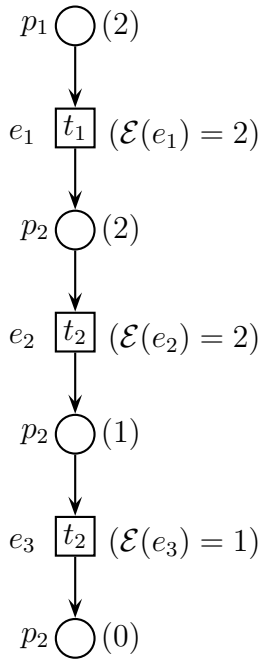
$$\phi_{\mathcal{E}}(e) \stackrel{\text{def}}{=} (C', t'), \quad \text{with} \quad \begin{cases} C' \stackrel{\text{def}}{=} \{(\phi_{\mathcal{E}}({}^\bullet b), tok_{\mathcal{E}}(b)) \mid b \in {}^\bullet e\} \\ t' \stackrel{\text{def}}{=} (\tau(e), \mathcal{E}(e), tok_{\mathcal{E}}({}^\bullet e)). \end{cases}$$

- *The process $\Psi(E')$ of N which corresponds to a process E' of $Exp(N)$ is $\Psi(E') \stackrel{\text{def}}{=} \bigcup_{e' \in E'} \psi(e')$, with ψ defined as: for all $e' \stackrel{\text{def}}{=} (C', (t, x, L)) \in E'$,*

$$\psi(e') \stackrel{\text{def}}{=} ((C, t), x), \quad \text{with} \quad C \stackrel{\text{def}}{=} \{(\psi(f'), p) \mid (f', (p, v)) \in C'\} .$$

It is important to remark that Φ and Ψ preserve the causal structure of the processes.





$e'_1 = (\{(\bot, (p_1, 2))\}, t_{122})$
$e'_2 = (\{(e'_1, (p_2, 2))\}, t_{22})$
$e'_3 = (\{(e'_2, (p_2, 1))\}, t_{21})$

$E' = \{e'_1, e'_2, e'_3\}$

$e_1 = (\{(\bot, p_1)\}, t_1)$
$e_2 = (\{(e_1, p_2)\}, t_2)$
$e_3 = (\{(e_2, p_2)\}, t_2)$

$\mathcal{E} = \{(e_1, 2), (e_2, 2), (e_3, 1)\}$

Figure 2.3: An execution of the colored Petri net of Figure 2.1, represented as an expanded process on the left and as a high-level process on the right. Near each condition b of the high-level process $\mathcal{E}$, we have written the value of $val_{\mathcal{E}}(b)$ in round brackets, and near each event e, we have written in round brackets the value $\mathcal{E}(e)$ that is assigned to the parameter x when e fires. At the bottom, we give the coding of the events and of the processes.

Proof. The proof is based on the relation between the final conditions of the processes. Thus we show by induction on X_N that for all $\mathcal{E} \in X_N$, $\Psi(\Phi(\mathcal{E})) = \mathcal{E}$ and $\uparrow(\Phi(\mathcal{E})) = \{(\phi_{\mathcal{E}}(^\bullet b), tok_{\mathcal{E}}(b)) \mid b \in \uparrow(E_{\mathcal{E}})\}$. And conversely we show by induction on $X_{Exp(N)}$ that for all $E' \in X_{Exp(N)}$, $E' = \Phi(\Psi(E'))$ and $\uparrow(E_{\Psi(E')}) = \{(\psi(f'), p) \mid (f', (p, v)) \in \uparrow(E')\}$ and for all $(f', (p, v)) \in \uparrow(E')$, $val_{\Psi(E')}((\psi(f'), p)) = v$. The inductive steps are straightforward. □

2.1.6 Symbolic Unfoldings

The aim of symbolic unfoldings is to give a compact representation of all the processes of a colored Petri net by merging the executions that differ only by the value of some parameters.

Definition 2.4. *We define the symbolic unfolding U_N of the colored Petri net N by collecting all the events that appear in its processes: $U_N \stackrel{\text{def}}{=} \bigcup_{\mathcal{E} \in X_N} E_{\mathcal{E}}$.*

Remark that U_N is a branching process of $Sup(N)$ because the processes of N are built over processes of $Sup(N)$.

We insist on the fact that the symbolic unfolding U_N is an abstraction of the unfolding $U_{Exp(N)}$, and thus U_N is much more compact in general than $U_{Exp(N)}$. We notice even that, when the set of values V or the set of parameters PAR is infinite, $Exp(N)$ may contain infinitely many places or transitions, even if the high-level Petri net N contains finitely many places and transitions. As a result, at a given depth in the unfoldings, there are in general much fewer events in U_N than in $U_{Exp(N)}$.

Figure 2.4 shows the expanded unfolding and the symbolic unfolding of the colored Petri net of Figure 2.1. Near each condition b of the symbolic unfolding we have written in round brackets the value of $val_{\mathcal{E}}(b)$ that is taken by the token in condition b in a process $\mathcal{E} : \lceil b \rceil \longrightarrow PAR$. Near each event of the symbolic unfolding, we have written in round brackets the condition under which the guard of the corresponding transition is satisfied.

As in the unfoldings of low-level Petri nets, we would like to be able to retrieve the processes from the symbolic unfolding. This is what Theorem 2.2 allows to do. It is also important to give a way to construct the symbolic unfolding directly, without constructing all the processes; Theorem 2.3 gives a way to do this.

Theorem 2.2. *Let $E \in X_{Sup(N)}$ be a process of $Sup(N)$ and $\mathcal{E} : E \longrightarrow PAR$ be a mapping that assigns a parameter to each event of E.*

$$\mathcal{E} \in X_N \quad \textit{iff} \quad \forall e \in E \quad \alpha(\tau(e))(\mathcal{E}(e), tok_{\mathcal{E}}(^\bullet e))$$

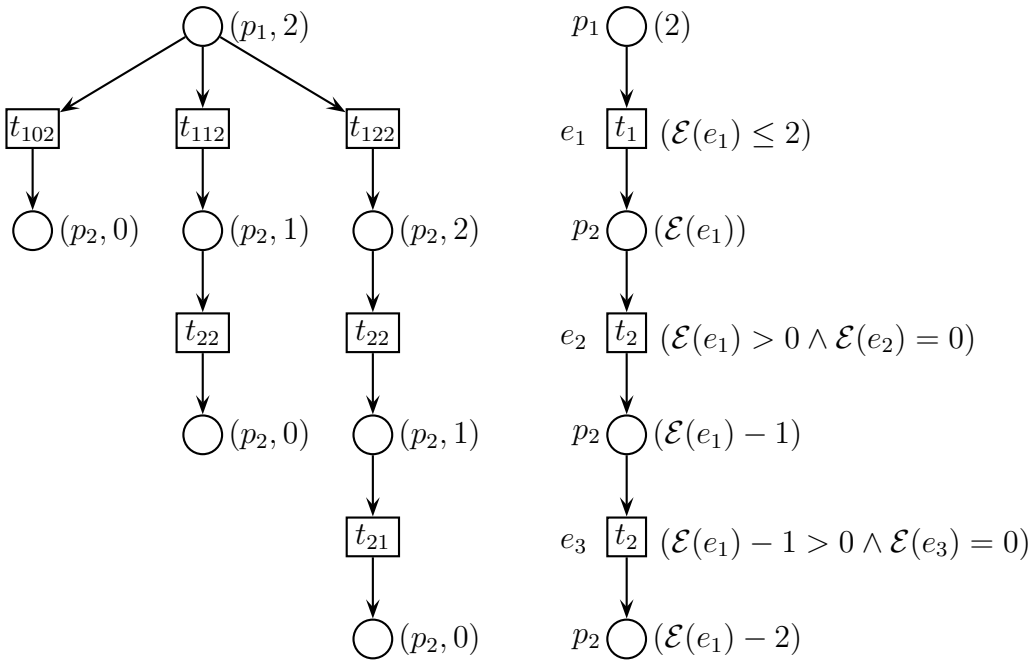


Figure 2.4: The expanded unfolding (on the left) and the symbolic unfolding (on the right) of the colored Petri net of Figure 2.1.

Proof. We first show by induction on X_N that every $\mathcal{E} \in X_N$ satisfies $(\forall e \in E \quad \alpha(\tau(e))(\mathcal{E}(e), tok_{\mathcal{E}}(^\bullet e)))$. This is true for $\mathcal{E} = \emptyset$. Let $\mathcal{E}$ be a process that satisfies the property, and let $t \in T$, $C \subseteq \uparrow(E_{\mathcal{E}})$ and $x \in PAR$ such that $Place(C) = {}^\bullet t$ and $\alpha(t)(x, tok_{\mathcal{E}}(C))$ holds. Denote $e' \stackrel{\text{def}}{=} (C, t)$ and $\mathcal{E}' = \mathcal{E} \cup \{(e', x)\}$. $\alpha(\tau(e))(\mathcal{E}'(e), tok_{\mathcal{E}'}(^\bullet e))$ holds for all event e in E because $\mathcal{E}'_E = \mathcal{E}$. And it holds also for e' since $\alpha(t)(x, tok_{\mathcal{E}'}(C))$ holds and $t = \tau(e')$, $x = \mathcal{E}'(e')$ and $C = {}^\bullet e$.

Now we show by induction on $X_{Sup(N)}$ that every $E \in X_{Sup(N)}$ satisfies:

$$\forall \mathcal{E} : E \longrightarrow PAR \quad \big(\forall e \in E \quad \alpha(\tau(e))(\mathcal{E}(e), tok_{\mathcal{E}}(^\bullet e))\big) \implies \mathcal{E} \in X_N \ .$$

This is true for $E = \emptyset$. Let $E \in X_{Sup(N)}$ that satisfies the property, let $t \in T$ and $C \subseteq \uparrow(E)$ such that $Place(C) = {}^\bullet t$. Denote $e' \stackrel{\text{def}}{=} (C, t)$ and $E' = E \cup \{e'\}$. Let $\mathcal{E}' : E' \longrightarrow PAR$ that satisfies the condition. Its restriction to E satisfies it too, so $\mathcal{E}'_{|E} \in X_N$. Moreover $\alpha(t)(\mathcal{E}'(e'), tok_{\mathcal{E}'}(C))$ holds since $\alpha(\tau(e'))(\mathcal{E}'(e'), tok_{\mathcal{E}'}(^\bullet e'))$ holds. Thus $\mathcal{E}' = \mathcal{E}'_{|E} \cup \{(e', \mathcal{E}'(e'))\}$ is in X_N. □

And, as well as for unfoldings of low-level Petri nets, we can decide if an event is in the symbolic unfolding of N by checking a constraint on its causal past.

Theorem 2.3. *For all $e \in U_{Sup(N)}$,*

$$e \in U_N \quad \textit{iff} \quad \exists \mathcal{E} : \lceil e \rceil \longrightarrow PAR \quad \mathcal{E} \in X_N \ .$$

Proof. Let $e \in U_N$. By definition of U_N, there exists $\mathcal{E}' \in X_N$ such that $e \in E_{\mathcal{E}'}$. Theorem 2.2 states that for all $f \in E_{\mathcal{E}'}$, $\alpha(\tau(f))(\mathcal{E}'(f), tok_{\mathcal{E}'}(^\bullet f))$ holds. As $\lceil e \rceil \subseteq E_{\mathcal{E}'}$, $\alpha(\tau(f))(\mathcal{E}'(f), tok_{\mathcal{E}'}(^\bullet f))$ holds for all $f \in \lceil e \rceil$. Moreover $\lceil e \rceil \in X_{Sup(N)}$. So, according to Theorem 2.2, $\mathcal{E}'_{|\lceil e \rceil}$ is in X_N and can play the role of the $\mathcal{E}$ of the theorem.

Conversely, if there exists $\mathcal{E} : \lceil e \rceil \longrightarrow PAR$ such that $\mathcal{E} \in X_N$, then $e \in \lceil e \rceil = E_{\mathcal{E}} \subseteq U_N$. □

Non-Structural Conflict

We see in Theorem 2.2 that the processes of a colored Petri net have to satisfy both structural conditions and conditions imposed by the guards on the possible values for the parameters. The structural conditions only depend on the underlying low-level process $E \in X_{Sup(N)}$. But these structural conditions are not sufficient when we deal with symbolic unfoldings: a set of events can be made incompatible by the fact that there exists no suitable value for the parameters of the events in their past.

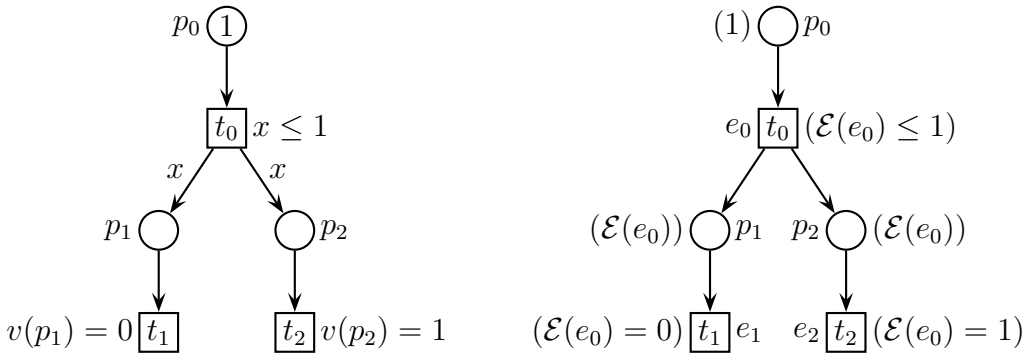


Figure 2.5: A colored Petri net (on the left) and its symbolic unfolding (on the right). The events e_1 and e_2 are not in structural conflict, but the conditions that they impose on the value of $\mathcal{E}(e_0)$ are incompatible. So no process contains both e_1 and e_2. We say that e_1 and e_2 are in non-structural conflict.

We can say that a set E of events of U are in *non-structural conflict* if they are not in conflict, but the constraints on the values of the parameters, coming from the guards of the transitions, prevent the events of E to appear in the same process of N.

Figure 2.5 shows a very simple example of a non-structural conflict.

Unlike the structural conflict due to the consumption of a single condition by several events, non-structural conflicts are not binary in general. That is, the minimal sets of events that are in conflict may have more than two elements. This is not so surprising, as the same phenomenon already appears in unfoldings of Petri nets with read arcs.

2.2 Dynamic nets

After defining symbolic unfoldings of colored Petri nets, we are now concerned with an even more powerful model of high-level Petri nets, which takes into account dynamic structural changes inside the distributed system. Many distributed systems can now be dynamically reconfigured. In particular we think about large telecommunications systems. *Web services* also seem to give examples of such behaviors.

Until now, the services provided on the Internet were based on the interaction between a client (browser) and a server through protocols like HTTP. More recently, the need to provide services that are usable by programs has launched the new model of *Web services*. A Web service is a self-describing,

self-contained modular application that can be described, published, located, and invoked over a network, e.g. the World Wide Web. A Web service performs an encapsulated function ranging from a simple request-reply to a full business process. It appears that the description of Web services and their orchestration is very similar to the description of distributed workflows in the context of business processes. A reference in this domain is the language WS-BPEL [38] (Business Process Language for Web Services).

We model services with high-level Petri nets. We model the steps in the procedure as transitions and precedence as arcs in the net. Data are treated using the symbolic aspects of high-level Petri nets. Those are also the basis for the representation of dynamic changes. The use of such kind of formal model has already been advocated in [43] and [64] in the context of workflow specifications.

We are concerned with dynamic changes in the structure of services; we are not concerned here with changes to the value of an application data variable. Dynamic means that we are required to make the change "on the fly" in the midst of continuous execution of the changing procedure. Web services must make structural changes such as adding a new service, removing an old service, or changing a given service behavior depending on the current state of the system (self-adaptation with respect to intermittent resources, reconfiguration in case of problem of quality of service).

In Section 2.1, we have defined the notion of symbolic unfolding of colored Petri nets. This model allows to deal with data aspects. It is relevant for static systems, but must be extended to deal with dynamic systems whose structure evolves. The proposed extensions are in the same vein as those described in [25]. In addition, they include read arcs to model non-destructive accesses to data. They also play a role to represent dynamicity. We propose a notion of unfolding for this model.

2.2.1 Introduction to the Model of Dynamic Nets

In our dynamic nets, the system state is represented as a multiset of tokens, called marking. Each token carries a value. Transitions are used to change the current state. When it fires, a transition consumes tokens from the current marking and creates new tokens. It is also possible to only test the presence of tokens without consuming them using read arcs.

In standard high-level Petri nets, the set of transitions is static. In contrast, the main originality in our dynamic model is that *transitions are particular tokens* included in the current marking. As any token, they can be

created or consumed. This allows us to model dynamic systems, in which the set of system components can evolve during execution.

Furthermore, our dynamic model does not use the notion of place: the presence of a token of value c in the place p in a standard high-level Petri net can be simulated by the presence of a token of value (p, c) in the marking. This avoids the difficulty to consider the dynamicity of places.

2.2.2 Definition

A *dynamic net* is a tuple $N \stackrel{\text{def}}{=} (V, PAR, \alpha, \beta, M_0)$, where V is a set of tokens, PAR is a set of parameters, α and β are maps such that:

- $\alpha : V \longrightarrow (PAR \times V^{\oplus} \times 2^V) \longrightarrow \mathbf{bool}$
- $\beta : V \longrightarrow (PAR \times V^{\oplus} \times 2^V) \longrightarrow V^{\oplus}$

and $M_0 \in V^{\oplus}$ is the *initial marking*. A *marking* is a multiset of tokens $M \in V^{\oplus}$.

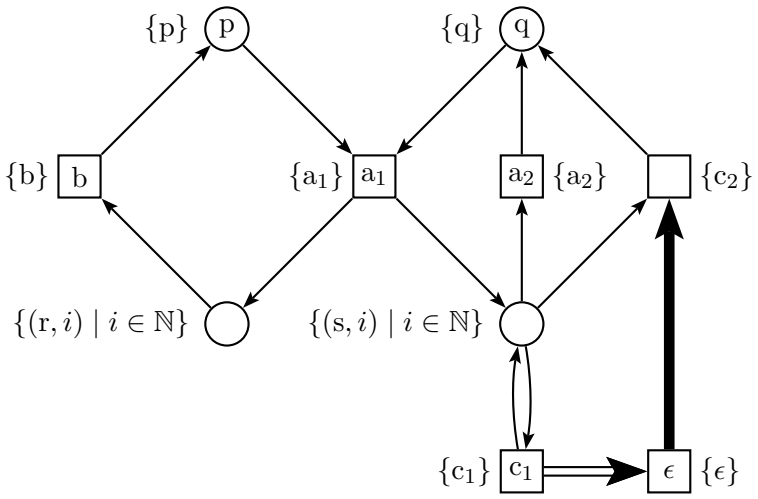
2.2.3 Interleaving Semantics

A token $t \in V$ is called *transition* if there exists $x \in PAR$, $C \in V^{\oplus}$ and $R \in 2^V$, such that $\alpha(t)(x, C, R)$. Furthermore, if $(\{t\} \cup R) + C$ is included in a marking M, then transition t can fire with the parameter x, consuming the tokens in C, and reading the tokens in R. This yields a new marking $M' \stackrel{\text{def}}{=} (M - C) + \beta(t)(x, C, R)$.

2.2.4 Examples

We give two example of dynamic nets.

For each graphical representation, we have split the set of tokens in two categories: the transitions and the others. Each square box represents a set of transitions and each circle represents a set of other tokens. Tokens currently present in the net are depicted inside the boxes and the circles. Arrows indicate consumption and creation of tokens by the transitions. An arrow is labelled by a star if it may represent the consumption or creation of several tokens. An arrow, linking a transition t to a transition t' can represent either the consumption of t by t', or the creation of t' by t. We remove the ambiguity by drawing a fat empty arrow in the first case and a solid one in the second case.



$\alpha(\mathrm{a}_1)(x, C) \stackrel{\text{def}}{=} C = \{\!|\mathrm{p}, \mathrm{q}|\!\} \wedge x \in \mathbb{N}$
$\alpha(\mathrm{a}_2)(x, C) \stackrel{\text{def}}{=} C = \{\!|(\mathrm{s}, 0)|\!\}$
$\alpha(\,\mathrm{b}\,)(x, C) \stackrel{\text{def}}{=} C = \{\!|(\mathrm{r}, i)|\!\} \quad (\text{with } i \leq 10)$
$\alpha(\mathrm{c}_1)(x, C) \stackrel{\text{def}}{=} C = \{\!|(\mathrm{s}, i)|\!\} \quad (\text{with } i > 0)$
$\alpha(\mathrm{c}_2)(x, C) \stackrel{\text{def}}{=} C = \{\!|(\mathrm{s}, i)|\!\} \quad (\text{with } i \in \mathbb{N})$
$\alpha(\,\epsilon\,)(x, C) \stackrel{\text{def}}{=} C = \{\!|\mathrm{c}_1|\!\}$

$\beta(\mathrm{a}_1)(x, C) \stackrel{\text{def}}{=} \{\!|(\mathrm{r}, x), (\mathrm{s}, x)|\!\}$
$\beta(\mathrm{a}_2)(x, C) \stackrel{\text{def}}{=} \{\!|\mathrm{q}|\!\}$
$\beta(\,\mathrm{b}\,)(x, C) \stackrel{\text{def}}{=} \{\!|\mathrm{p}|\!\}$
$\beta(\mathrm{c}_1)(x, \{\!|(\mathrm{s}, i)|\!\}) \stackrel{\text{def}}{=} \{\!|(\mathrm{s}, i-1)|\!\}$
$\beta(\mathrm{c}_2)(x, C) \stackrel{\text{def}}{=} \{\!|\mathrm{q}|\!\}$
$\beta(\,\epsilon\,)(x, C) \stackrel{\text{def}}{=} \{\!|\mathrm{c}_2|\!\}$

Figure 2.6: An example of dynamic distributed system. As this example does not use read arcs we write $\alpha(t)(x, C)$ and $\beta(t)(x, C)$ instead of $\alpha(t)(x, C, R)$ and $\beta(t)(x, C, R)$ and we always assume implicitly $R = \emptyset$. The symbol $=$ sometimes means "matches": for instance, "$C = \{\!|(\mathrm{r}, i)|\!\} \quad (\text{with } i \leq 10)$" means "$\exists i \leq 10 \quad C = \{\!|(\mathrm{r}, i)|\!\}$".

Failure Propagation

Figure 2.6 shows a dynamic distributed system. Its formal description is written at the bottom of the figure.

The system is made of three components: A (transitions a_1 and a_2), B (transition b) and C (transitions c_1, c_2 and ϵ). The main component (A) may break down (transition a_1) with a variable severity level $x \in \mathbb{N}$.

In order to repair the failure, components B and C must participate. The participation of component B is modeled by transition b and is only possible if the severity is lower than 10. Component C must also participate: Initially, it has to perform x actions modeled by the transition c_1, and this must be completed by an action a_2 from component A.

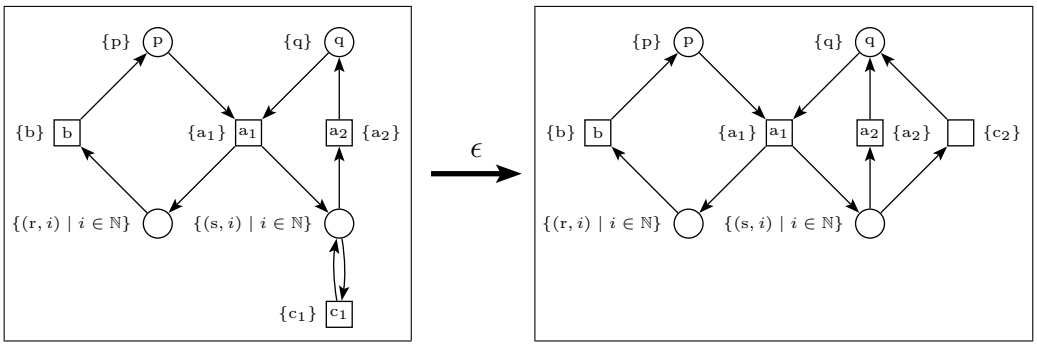


Figure 2.7: The transitions (except ϵ) that are present in the net before and after the improvement. Firing ϵ deletes transition c_1 and creates c_2.

But the component C is likely to be improved, so that it can repair any failure in a single action c_2, which does not need any more to be completed by A. The improvement of the component C is modeled by the transition ϵ, which replaces the transition c_1 by c_2. We notice that transition c_2 is not in the state of Figure 2.6.

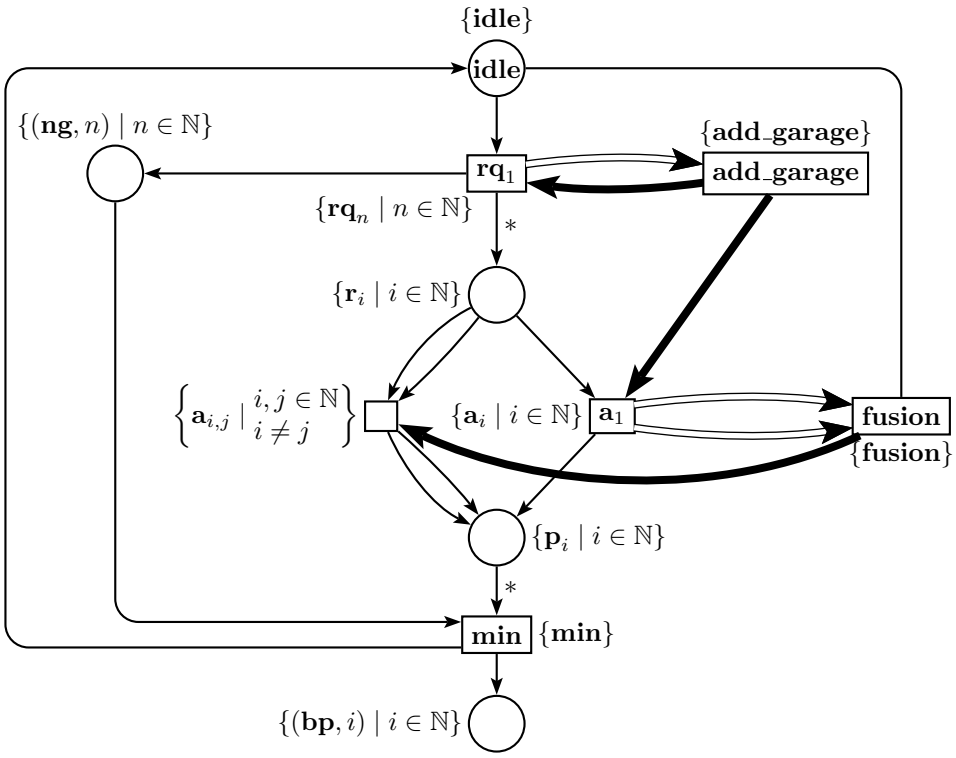
Figure 2.7 shows the transitions (except ϵ) that are present in the net before and after the improvement.

Web Services Orchestration

Our second example is shown in Figure 2.8. Its formal description is written at the bottom of the figure. It represents a Web Services orchestration where a server selects the best price for a car rent from several garages each time a client asks for it. There is initially one garage, but new garages can be added dynamically at any time. We detail this aspect later.

For the moment consider we are in a situation where there are n garages. When a request arrives, the server sends a request to each of the n garages. This action is represented by the transition $\mathbf{rq}_n$. Then all the garages are supposed to answer concurrently and give their price. The answer of the i^{th} garage is represented by the transition $\mathbf{a}_i$, that creates a token $\mathbf{p}_i$. When all the garages have answered, the server selects the best price (transition **min**, that creates a token $(\mathbf{bp}, i)$ where i is the number of one of the garages that gave the best price).

At any time a new garage may be added. The transition **add_garage** models this: it creates a transition $\mathbf{a}_{n+1}$ that represents the answer of the new garage (number $(n+1)$). Additionally, the transition $\mathbf{rq}_n$ is replaced by $\mathbf{rq}_{n+1}$, so that the next time a client asks for the price of a car, all the garages (including the new one) will be requested. Because a garage can



$\alpha(\mathbf{fusion})(x, C, R) \stackrel{\text{def}}{=} R = \{\mathbf{idle}\} \wedge C = \{\!|\mathbf{a}_i, \mathbf{a}_j|\!\} \wedge x = \bullet \quad (\text{with } i, j \in \mathbb{N} \text{ and } i \neq j)$
$\beta(\mathbf{fusion})(x, \{\!|\mathbf{a}_i, \mathbf{a}_j|\!\}, \{\mathbf{idle}\}) \stackrel{\text{def}}{=} \{\!|\mathbf{a}_{i,j}|\!\}$

$\alpha(\mathbf{add\_garage})(x, C, R) \stackrel{\text{def}}{=} R = \emptyset \wedge C = \{\!|\mathbf{rq}_n|\!\} \wedge x = \bullet \quad (\text{with } n \in \mathbb{N})$
$\beta(\mathbf{add\_garage})(x, \{\!|\mathbf{rq}_n|\!\}, \emptyset) \stackrel{\text{def}}{=} \{\!|\mathbf{rq}_{n+1}, \mathbf{a}_{n+1}|\!\}$

$\alpha(\mathbf{rq}_n)(x, C, R) \stackrel{\text{def}}{=} R = \emptyset \wedge C = \{\!|\mathbf{idle}|\!\} \wedge x = \bullet$
$\beta(\mathbf{rq}_n)(x, \{\!|\mathbf{idle}|\!\}, \emptyset) \stackrel{\text{def}}{=} \{\!|(\mathbf{ng}, n), \mathbf{r}_1, \ldots, \mathbf{r}_n|\!\}$

$\alpha(\mathbf{a}_i)(x, C, R) \stackrel{\text{def}}{=} R = \emptyset \wedge C = \{\!|\mathbf{r}_i|\!\} \wedge x = \bullet$
$\beta(\mathbf{a}_i)(x, C, R) \stackrel{\text{def}}{=} \{\!|\mathbf{p}_i|\!\}$

$\alpha(\mathbf{a}_{i,j})(x, C, R) \stackrel{\text{def}}{=} R = \emptyset \wedge C = \{\!|\mathbf{r}_i, \mathbf{r}_j|\!\} \wedge x = \bullet$
$\beta(\mathbf{a}_{i,j})(x, C, R) \stackrel{\text{def}}{=} \{\!|\mathbf{p}_i, \mathbf{p}_j|\!\}$

$\alpha(\mathbf{min})(i, C, R) \stackrel{\text{def}}{=} R = \emptyset \wedge C = \{\!|(\mathbf{ng}, n), \mathbf{p}_1, \ldots, \mathbf{p}_n|\!\} \wedge i \in \{1, \ldots, n\} \quad (\text{with } n \in \mathbb{N})$
$\beta(\mathbf{min})(i, \{\!|(\mathbf{ng}, n), \mathbf{p}_1, \ldots, \mathbf{p}_n|\!\}, \emptyset) \stackrel{\text{def}}{=} \{\!|(\mathbf{bp}, i)|\!\}$

Figure 2.8: A dynamic Web Services orchestration where a server selects the best price for a car rent from several garages.

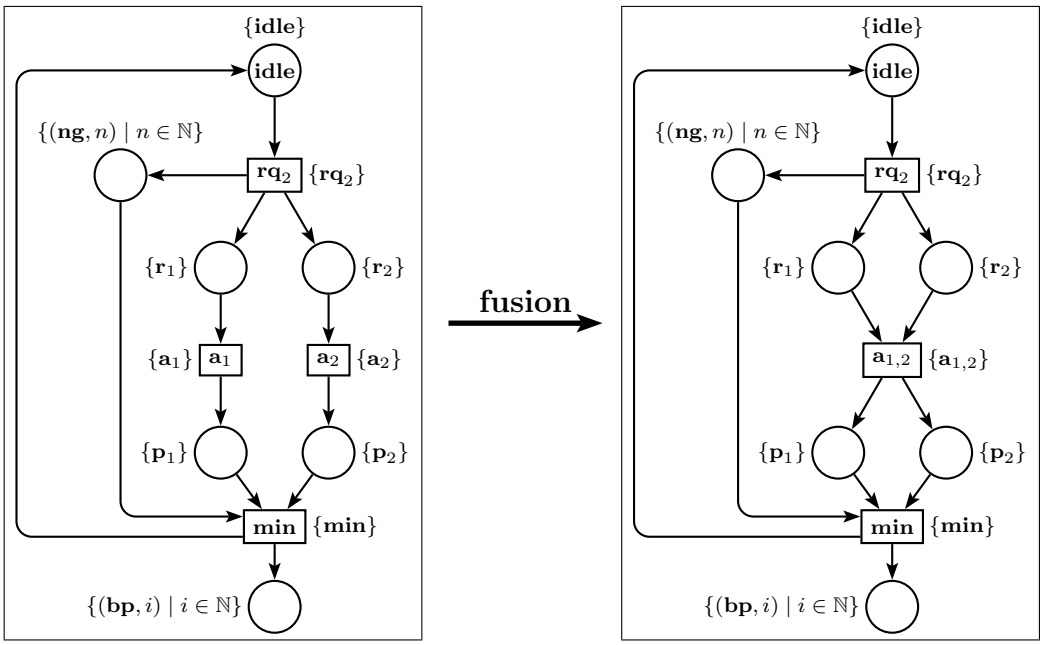


Figure 2.9: Two independent garages (on the left) and two garages that have agreed on the price (on the right).

be added even while some garages are answering a request, the number of requested garages is stored (token $(\mathbf{ng}, n)$ in the model) when the requests are sent. Thus before selecting the best price, the server can check that all the requested garages have answered, and it will not wait for answers of new garages.

In addition to the behavior that we have described, any two garages, say the i^{th} and the j^{th}, may choose to agree on the price. In this case they synchronize to give their answer together. In our model, the synchronization is represented by the transition **fusion**, which deletes the transitions $\mathbf{a}_i$ and $\mathbf{a}_j$ and replaces them by a transition $\mathbf{a}_{i,j}$ which gives the two answers simultaneously with the same price. The read arc from the **idle** token from the **fusion** transition models the fact that the fusion cannot happen when the server is waiting for answers from the garages. Figure 2.9 shows the structure of the system with two garages in two different situations where they have not or have respectively agreed on the price.

2.2.5 Unfoldings

We propose both expanded and symbolic unfoldings for our model of dynamic nets. Expanded unfoldings were published in [33] and symbolic unfoldings in [31]. Each time they were introduced in the framework of an application to supervision in dynamic distributed systems.

In both cases, the problem that arises in unfoldings of dynamic nets is to know which transitions exist in the net at a given time of the execution. By considering transitions as particular tokens we obtain a simple way to solve this problem: checking the presence of a transition amounts to checking the presence of a token in a marking. Then when we extend a process with an event, we check that the corresponding transition is in the state that is reached at the end of the process. The final condition of the process that points out the presence of the transition is coded in the event. For this we use a read arc because firing the transition does not prevent it from firing again.

Expanded Unfoldings

Like in Section 1.2, we have to make a minor restriction on the model before dealing with processes. Here we require that M_0 is a set and that for all transition $t \in V$, for all $C \in V^{\oplus}$ and $R \in 2^V$ and for all $x \in PAR$ such that $\alpha(t)(x, C, R)$ holds, C is nonempty and $\beta(t)(x, C, R)$ is a set rather than a multiset.

For expanded processes of dynamic nets, the processes are defined as sets of events. And the events are tuples $e \stackrel{\text{def}}{=} (C, R, b, x)$ taken from the set D_N defined below. b is the condition that contains the transition that fires, and x is the parameter of the firing. C, also denoted ${}^\bullet e$, codes the sets of consumed conditions. R codes the set of conditions that are read by the firing of the transition, but the context of e contains also the condition b that contains the transition that fires: $\underline{e} \stackrel{\text{def}}{=} R \cup \{b\}$.

Each of the conditions that are created by e is a pair $b \stackrel{\text{def}}{=} (e, v)$, where $v \in V$ is the created token; e is denoted ${}^\bullet b$ and v is denoted $tok(b)$. The output conditions of an event $e \stackrel{\text{def}}{=} (C, R, b, x)$ are computed using β: $e^\bullet \stackrel{\text{def}}{=} \beta(tok(b))(x, tok(C), tok(R))$. The initial conditions are $\bot^\bullet \stackrel{\text{def}}{=} \{\bot\} \times M_0$.

Definition 2.5 (D_N). *We define D_N as the smallest set such that*

$$\forall C, R \subseteq \bigcup_{e \in D_{N\bot}} e^\bullet \quad \forall b \in \bigcup_{e \in D_{N\bot}} e^\bullet \quad \forall x \in PAR \quad (C, R, b, x) \in D_N \ .$$

The final conditions of a process are defined as usual:

$$\uparrow(E) \stackrel{\text{def}}{=} \bigcup_{e \in E_\perp} e^\bullet \setminus \bigcup_{e \in E} {}^\bullet e \ .$$

As we use read arcs, we need a conditional causality relation, defined as usual: $e \nearrow f$ iff $(e \to f) \vee (\underline{e} \cap {}^\bullet f \neq \emptyset)$.

Definition 2.6. *The set X_N of* expanded processes *of a dynamic net N is defined inductively as follows.*

- $\emptyset \in X_N$;
- *for all process $E \in X_N$, for all condition $b \in \uparrow(E)$ such that $t \stackrel{\text{def}}{=} tok(b)$ is a transition, for all $C, R \subseteq \uparrow(E)$, for all parameter $x \in PAR$ such that $\alpha(t)(x, tok(C), tok(R))$, $E \cup \{e\} \in X_N$, where the event $e \stackrel{\text{def}}{=} (C, R, b, x)$ represents the firing of t with parameter x, that consumes the tokens $tok(C)$ and reads the tokens $tok(R)$.*

Definition 2.7 (expanded unfolding). *We define the expanded unfolding U_N of the dynamic net N by collecting all the events that appear in its expanded processes: $U_N \stackrel{\text{def}}{=} \bigcup_{E \in X_N} E$.*

Theorem 2.4. *Let $E \subseteq D_N$ be a finite set of events. E is a process iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \\ \nexists e_0, e_1, \ldots, e_n \in E \quad e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0 & \textit{($\nearrow$ is acyclic on E)} \\ \forall e \stackrel{\text{def}}{=} (C, R, b, x) \in E \quad \alpha(tok(b))(x, tok(C), tok(R)) & \\ \qquad \textit{(the guard of the transition $tok(b)$ is satisfied)} & \end{cases}$$

Proof. Let $E \in X$ be a process that satisfies the conditions in the curly brace, let $e \stackrel{\text{def}}{=} (C, R, b, x)$ as in Definition 2.6. Then we will show that the process $E' \stackrel{\text{def}}{=} E \cup \{e\}$ also satisfies the conditions in the curly brace. By construction E' is causally closed. Moreover for each condition $b' \in {}^\bullet e$ that is consumed by e, $b' \in \uparrow(E)$, which implies that b' has not been consumed by any event of E. Thus for all $f \in E$, ${}^\bullet e \cap {}^\bullet f = \emptyset$ and $\neg(e \nearrow f)$. So E' is conflict free and $\nearrow$ is acyclic on E'. And by construction of e the guard of the transition $tok(b)$ is satisfied when e fires; and we already know that it is true for the events of E.

Conversely let E' satisfy the conditions in the curly brace. If $E' = \emptyset$, then $E' \in X$. Otherwise let $e \stackrel{\text{def}}{=} (C, R, b, x) \in E'$ be an event that has no successor by $\nearrow$ in E'. $E \stackrel{\text{def}}{=} E' \setminus \{e\}$ satisfies the conditions in the curly brace. Assume that $E \in X$. As E' is conflict free, ${}^\bullet e \subseteq \uparrow(E)$. And as e has no successor by $\nearrow$ in E', $\underline{e} \subseteq \uparrow(E)$. Furthermore the guard of the transition $tok(b)$ is satisfied when e fires, so $E' = E \cup \{e\} \in X$. □

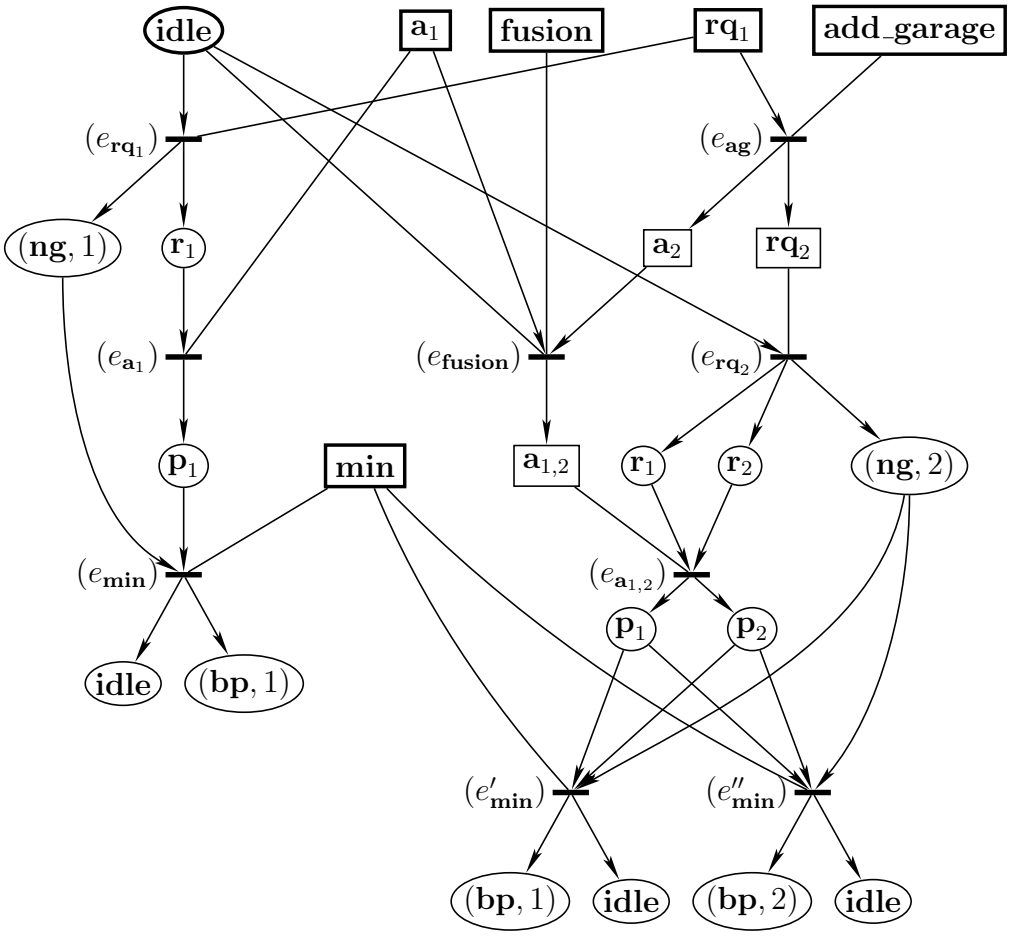


Figure 2.10: A prefix of the expanded unfolding of the model of Figure 2.8.

And here again, it is easy to check if an event is in the unfolding: for all event $e \in D_N$, $e \in U_N$ iff $\lceil e \rceil \in X_N$.

Example. As an example, Figure 2.10, shows a prefix of the expanded unfolding of the model of Figure 2.8.

As some conditions correspond to transitions, they are drawn as boxes. And the events are represented by black bars. The token **idle**, which indicates that no request is en route, and the transitions $\mathbf{rq}_1$, $\mathbf{a}_1$, **min**, **add_garage** and **fusion** are in the initial marking. In the unfolding, the corresponding initial conditions are emphasized by thicker lines. The value of $tok(b)$ is written in each condition b.

In the initial state, a client may ask for the price of a car (event $e_{\mathbf{rq}_1}$). The unique initial garage is queried and responds (event $e_{\mathbf{a}_1}$). The system also allows the addition of a new garage (event $e_{\mathbf{ag}}$). If the event $e_{\mathbf{rq}_1}$ occurs in the same history, then it must occur before $e_{\mathbf{ag}}$. The asymmetric conflict that appears in the unfolding between $e_{\mathbf{rq}_1}$ and $e_{\mathbf{ag}}$ forces this order.

After the occurrence of $e_{\mathbf{ag}}$, the transition $\mathbf{rq}_1$ does not exist any more but is replaced by the transition $\mathbf{rq}_2$, which sends a request to two garages. Moreover, the transition $\mathbf{a}_2$ is added. If $\mathbf{rq}_2$ fires (event $e_{\mathbf{rq}_2}$), it will send a request to both the first and the second garage. They may answer concurrently (this is not represented on Figure 2.10) or answer simultaneously (event $e_{\mathbf{a}_{1,2}}$) provided that they have agreed on the price (event $e_{\mathbf{fusion}}$).

Symbolic Unfoldings

For symbolic unfoldings we constrain slightly the definition of dynamic nets: we require that a transition always generates the same number of tokens. The reason for this is that we want to know how many conditions are created by an event, independently of the value of the tokens in its input conditions. We can express our constraint simply by moving the closing bracket in the type of β:

$$\beta : V \longrightarrow (PAR \times V^{\oplus} \times 2^{V} \longrightarrow V)^{\oplus} .$$

More precisely, $\beta(t)$ should always be a set when we deal with processes.

The set of tokens that are created when t fires should now be denoted $\{\!|\gamma(x, C, R) \mid \gamma \in \beta(t)|\!\}$, but we keep denoting it $\beta(t)(x, C, R)$ for simplicity.

As well as for symbolic unfoldings of colored Petri nets, a high-level processes will be coded as a mapping $\mathcal{E}$ from a set $E_{\mathcal{E}}$ of events to the parameters. The information about the parameters is not coded in the events. So we would expect events of the type (C, R, b). But as the token that corresponds to a condition depends now on the values of the parameters in its causal past (remind the definition of $val_{\mathcal{E}}(b)$ in Section 2.1.5), the transition that corresponds to condition b may depend on the parameters. It may even not be a transition. As the number of output conditions of the event is defined by the number of output tokens of the corresponding transition, we have to precise explicitly the transition in the coding of the event. That is, the events will be tuples (C, R, b, t). The consequence is that if several transitions may correspond to the condition t, then their firing will be represented by distinct events. Of course the condition b is still necessary in the coding of the event, because in a dynamic framework we have to check that the transition exists before firing it.

The set of input conditions of $e \stackrel{\text{def}}{=} (C, R, b, t)$ is ${}^\bullet e \stackrel{\text{def}}{=} C$, its context is $R \cup \{b\}$, and its set of output conditions is $e^\bullet \stackrel{\text{def}}{=} \{e\} \times \beta(t)$. The initial conditions remain like in expanded unfoldings: $\bot^\bullet \stackrel{\text{def}}{=} \{\bot\} \times M_0$.[1]

Definition 2.8 (D_N). *The events that appear in the high-level processes of a dynamic net N are taken from the set D_N defined as the smallest set such that*

$$\forall C, R \subseteq \bigcup_{e \in D_N \bot} e^\bullet \quad \forall b \in \bigcup_{e \in D_N \bot} e^\bullet \quad \forall t \in T \quad (C, R, b, t) \in D_N \ .$$

The final conditions $\uparrow(E)$ and the conditional causality relation $\nearrow$ are defined like above.

But in a high-level process $\mathcal{E}$ we need to define the token $tok_{\mathcal{E}}(b)$ that corresponds to a condition b according to the parameters:

$$tok_{\mathcal{E}}(b) \stackrel{\text{def}}{=} \begin{cases} v & \text{if } b \stackrel{\text{def}}{=} (\bot, v) \\ \beta(t)(\mathcal{E}(e), tok_{\mathcal{E}}(C), tok_{\mathcal{E}}(R)) & \text{if } b \stackrel{\text{def}}{=} (e, \gamma) \text{ with } e \stackrel{\text{def}}{=} (C, R, b, t) \end{cases}$$

Definition 2.9. *The set X_N of* processes *of a dynamic net N is defined inductively as follows.*

- $\emptyset \in X_N$;
- *for all process $\mathcal{E} \in X_N$, for all condition $b \in \uparrow(E)$ such that $t \stackrel{\text{def}}{=} tok_{\mathcal{E}}(b)$ is a transition, for all $C, R \subseteq \uparrow(E)$, for all parameter $x \in PAR$ such that $\alpha(t)(x, tok_{\mathcal{E}}(C), tok_{\mathcal{E}}(R))$, $\mathcal{E} \cup \{(e, x)\} \in X_N$, where the event $e \stackrel{\text{def}}{=} (C, R, b, t)$ represents the firing of t.*

Definition 2.10 (symbolic unfolding). *We define the symbolic unfolding U_N of the dynamic net N by collecting all the events that appear in its processes: $U_N \stackrel{\text{def}}{=} \bigcup_{\mathcal{E} \in X_N} E_{\mathcal{E}}$.*

Theorem 2.5. *Let $E \subseteq D_N$ be a finite set of events, and $\mathcal{E} : E \longrightarrow PAR$. $\mathcal{E}$ is a process iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \\ \nexists e_0, e_1, \ldots, e_n \in E \quad e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0 & (\nearrow \textit{ is acyclic on } E) \\ \forall e \stackrel{\text{def}}{=} (C, R, b, t) \in E \quad \begin{cases} t = tok_{\mathcal{E}}(b) \\ \alpha(t)(\mathcal{E}(e), tok_{\mathcal{E}}(C), tok_{\mathcal{E}}(R)) \end{cases} & \\ \qquad\qquad \textit{(the guard of the transition t is satisfied)} & \end{cases}$$

[1] With respect to the typing, it would be better to code the initial conditions as pairs $(\bot, \gamma)$, where γ would be a constant function. But we prefer to simplify the notations, and this typing inconsistency will be fixed by the function $tok_{\mathcal{E}}$ below.

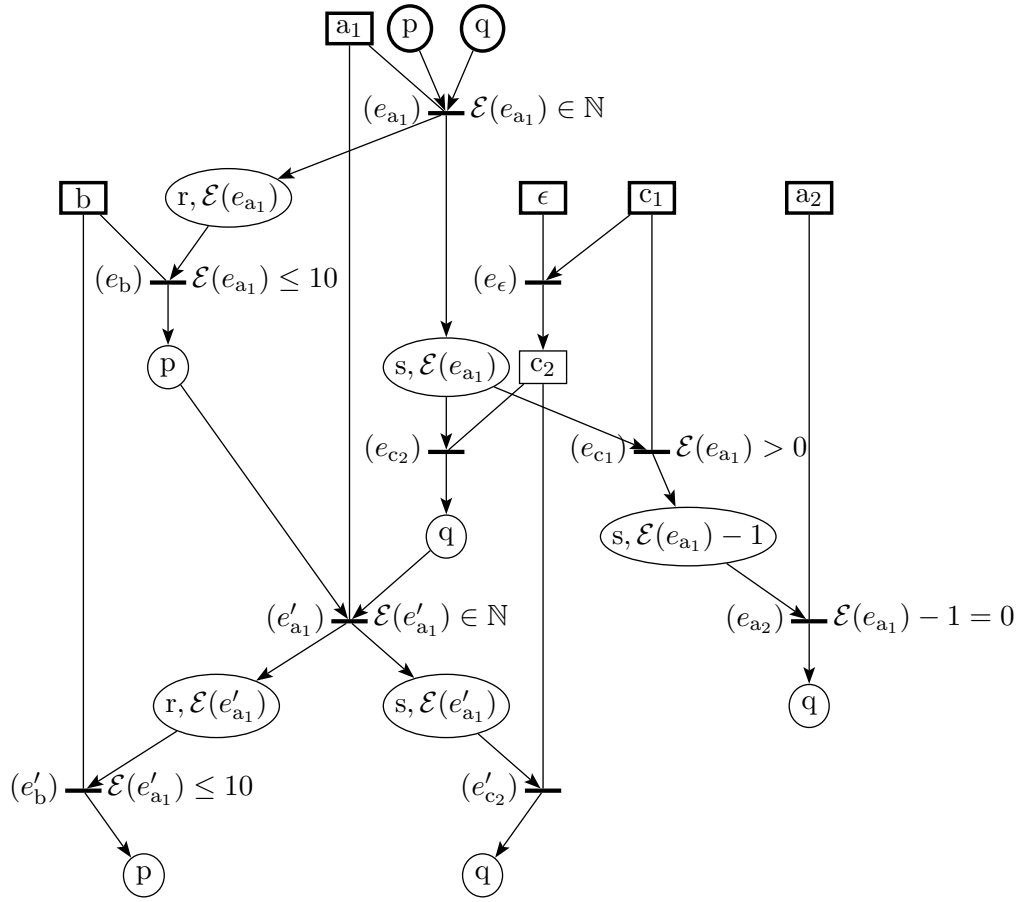


Figure 2.11: A prefix of the symbolic unfolding of the model of Figure 2.6. The initial conditions are emphasized by thicker lines.

Proof. The proof of this theorem is a combination of the proofs of Theorems 2.4 and 2.2. □

And here again, it is easy to check if an event is in the unfolding: for all event $e \in D_N$,

$$e \in U_N \quad \text{iff} \quad \exists \mathcal{E} : \lceil e \rceil \longrightarrow PAR \quad \mathcal{E} \in X_N \ .$$

Example. Figure 2.11 shows a prefix of the symbolic unfolding of the model of Figure 2.6. Notice that, unlike the expanded unfolding of Figure 2.10, the value in each condition b is a symbolic expression of $tok_{\mathcal{E}}(b)$ that depends on the values of the parameters of the events in the past of the condition.

The event e_{a_1} represents an initial failure. The read arc from the condition labeled a_1 indicates that e_{a_1} is an occurrence of the transition a_1. If the severity level of the failure is lower than 10, then transition b can fire, which is coded by the event e_{b}. The constraint $\mathcal{E}(e_{\mathrm{a}_1}) \leq 10$ is written near e_{b} in order to remind that e_{b} is not possible in all the cases.

The dynamic aspect of the example appears in event e_ϵ, which consumes the condition corresponding to transition c_1, and creates one corresponding to c_2. The arrow from c_1 to e_ϵ prevents from firing c_1 in the future of e_ϵ.

2.3 Computability Problems

In Sections 2.1 and 2.2, we have defined symbolic unfoldings of high-level models. Now we are addressing the problem of actually computing these objects.

As the unfoldings are in general infinite structures, we are more precisely looking for a way to enumerate their events. We already thought of this problem when defining the coding of the events: each time we have defined processes, we have emphasized the fact that the events are taken from a set D_N. Thus the problem of enumerating the events of an unfolding can be reduced to enumerating the events in D_N and checking for each event of D_N, if it is in the unfolding. For this also we gave each time a formula.

2.3.1 Enumeration of D_N

All our definitions of processes are based on an inductive definition of D_N. Provided a finite set of events appear at each inductive step, the definition of D_N gives directly a way to enumerate the events. We detail algorithms to enumerate D_N in Chapter 5; here we simply discuss the feasibility of the enumeration of the events.

In the case of colored Petri nets, the events of the symbolic unfolding are those of the underlying low-level Petri net $Sup(N)$, and, if the sets of places and transitions are finite, then a finite set of events can be added at each inductive step in the definition of $D_{Sup(N)}$. Remark that it is even sufficient that each inductive step generates an enumerable set of events; this allows us to require simply that the sets of places and transitions are enumerable.

Concerning the definition of D_N for symbolic unfoldings of dynamic nets, we point out a problem that arises from the coding of the transition in the event. We recall that in this case the events are tuples (C, R, b, t), where t is the transition that fires. But in our definition of dynamic nets we could easily conceive of having an infinite set of transitions. Here again if the set of

transitions remains enumerable, then we can enumerate D_N. But if the token in a condition b can take an infinite set of values, and a non enumerable part of these values are transitions, then it might be that D_N is not enumerable. In practice it is likely that the set of transitions is not very large, and D_N will be enumerable.

Finally we make a simple remark about *expanded* unfoldings: not only the sets of events that appear in these unfoldings are in general much larger than the sets of events in the symbolic unfoldings, but they may even become non-enumerable as soon as the sets of values and parameters of the net are not enumerable.

2.3.2 Checking if an Event is in the Unfolding

Once we have a way to enumerate the events of D_N, we can build the unfolding by selecting those that are in the unfolding. That is, we simply need a way to decide if an event of D_N is in U_N. The aim of Theorem 2.3 was precisely to give a simple formula to decide this problem in the case of symbolic unfoldings of colored Petri nets. We have also given similar formulas for expanded and symbolic unfoldings of dynamic nets.

To summarize these formulas, deciding if an event e is in the unfolding amounts to checking

- a structural property on the causal past of e and
- a property that says that the guards of the transitions are satisfied when the events fire.

The first property is decidable. We detail algorithms for this in Section 5.2.1.

Concerning the second property we have to distinguish symbolic unfoldings from expanded unfoldings.

If we assume that the functions $\beta(t)(\dots)$ are computable, then the function *tok* is computable. Therefore, if we assume that the guards $\alpha(t)(\dots)$ are also computable, then checking if an event is in an expanded unfolding is decidable. We believe that most models of real systems satisfy our assumptions about the computability of the guards and of the functions $\beta(t)(\dots)$.

But in the framework of symbolic unfoldings, we have to check the satisfiability of the guards. The existential quantification $\exists\mathcal{E}$ (see Theorem 2.3) prevents us from using the simple method that we described for expanded unfolding. Instead we obtain a constraint satisfaction problem. And depending on the language that is used to write the functions $\alpha(t)(\dots)$ and $\beta(t)(\dots)$, the satisfiability of the constraints that we obtain may be undecidable. And

we know that even elementary arithmetics with addition and multiplication of real numbers may make our problem be undecidable.

Nevertheless, a large number of non trivial real systems can be modeled using decidable frameworks like linear arithmetics or even only finite sets of values and parameters.

To conclude this section, as there is no general algorithm to check any type of constraints, we consider that the algorithms that compute symbolic unfoldings can be seen as meta-algorithms, where a parameter is an algorithm that solves the constraint satisfaction problems that arise from the unfolding.

2.4 Finite Complete Prefixes for Symbolic Unfoldings of High-Level Models?

We have mentioned in Section 1.3.2 that the unfoldings of low-level Petri nets can be represented by a finite complete prefix that contains enough information to check many properties of the model.

Unfortunately we believe that there is no simple way to extend the notion of complete finite prefix to symbolic unfoldings. But here again this problem depends on the form of the expressions that are used to describe the guards of the model. In some restricted cases, there will be possible to define finite complete prefixes. In Section 3.5 we show that finite complete prefixes can be defined in the framework of symbolic unfoldings of time Petri nets.

Finally we want to point out an example that makes it clear that even the geometry of the unfolding may be such that no simple geometric pattern is repeated. We chose a simple colored Petri net with two places which behave like counters. It is depicted in Figure 2.12. Transition t_2 decrements the value of the token in p_2 until it reaches 0. And then transition t_1 resets the counter in p_2 according to the value of the counter in p_1, which increases each time t_1 fires.

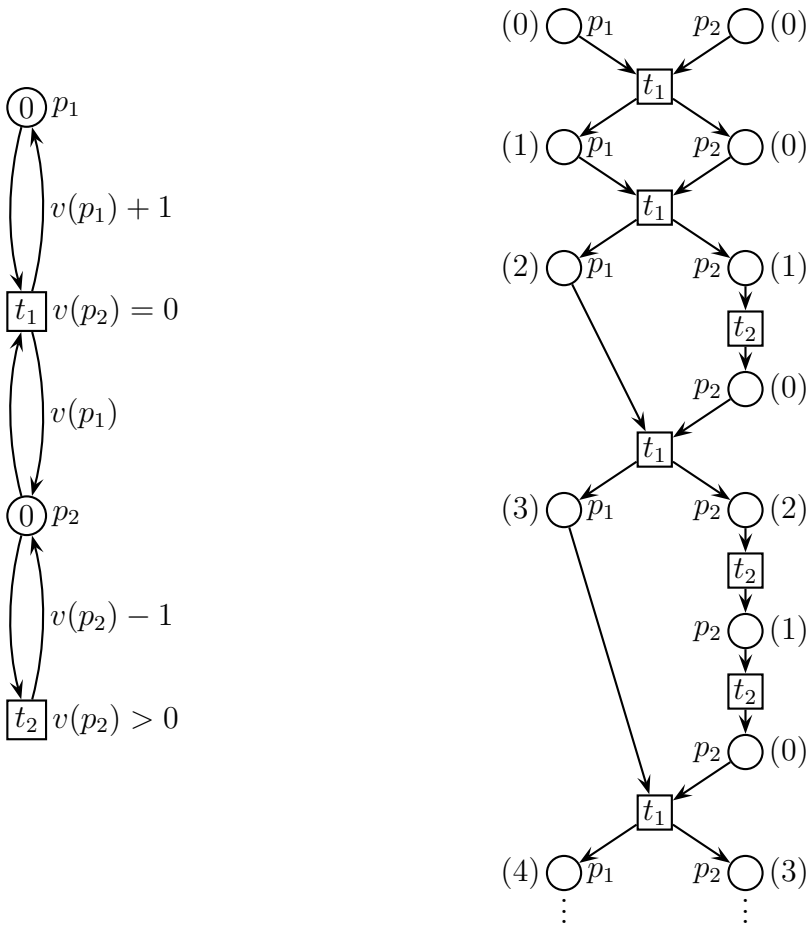


Figure 2.12: A colored Petri net (on the left) and a part of its unfolding (on the right). We see that no simple geometric pattern is repeated.

Chapter 3

Timed True Concurrency Models

All the extensions of Petri nets that we have presented so far are adapted to model completely asynchronous systems and no assumption is made about the duration of the actions or the delays between two consecutive actions. This remains true for the high-level extensions that we have described above.

This means that it is really difficult to model real-time systems with these extensions. For instance many communications protocols include the retransmission of messages that are not acknowledged after a given delay. Sometimes also a program is executed at regular intervals. Ignoring these time constraints prevents us from modeling the system correctly. Therefore numerous timed true concurrency models have been introduced. Most of them are extensions of the untimed true concurrency models.

Timed models are particularly adapted for a symbolic framework, because it allows to group some executions that differ only by the dates of the actions. The studies of the sequential semantics of timed models already use symbolic state classes (see [14, 13] for time Petri nets or [2] for timed automata).

But very little work was done about unfoldings of timed true concurrency models. We describe some preceding works in Section 3.2.3. The reason that makes it rather difficult is that time yields a kind of synchronization even between parts of the net that do not communicate. This clashes with the partial order semantics that are based on the fully asynchronous nature of the models.

In this chapter we describe our contribution to this problem. The main originality of our approach is in defining a concurrent operational semantics for time Petri nets, where it will be possible to fire a transition without looking at the global state of the net. Only a few tokens will give enough

information to be sure that the transition can fire. Then we define symbolic unfoldings of time Petri nets using our concurrent operational semantics. This work was published in [34, 35] where we also advocated its interest for supervision of real-time concurrent systems (see also [61, 62]).

Then we show the existence of finite complete prefixes for symbolic unfoldings of time Petri nets. This result was published in [36].

And finally we extend our approach to a more general model that combines colored tokens and timed aspects. The interest is to show that our idea of using a concurrent operational semantics seems to give a general method to deal with unfoldings of timed true concurrency models.

3.1 Safe Time Petri Nets

Time Petri nets are one of the most popular timed extensions of Petri nets. They were introduced by Merlin and Farber in [73] and have proved their interest in modeling real-time concurrent systems.

In time Petri nets a time interval is assigned to each transition in order to restrict the possible delays between the time when it is enabled and the time when it fires.

The analysis of unbounded behaviors of time Petri nets leads to several undecidability results. For instance, boundedness is undecidable. Under the assumption that the net is bounded, reachability becomes decidable [80].

In our work we consider only safe time Petri nets. Their usual semantics is defined in terms of firing sequences, which can be coded in a graph of symbolic global states. This graph is finite if the net is bounded and if the bounds on the delays are rational numbers [14, 13, 15].

Partial-order reduction methods have been developed for time Petri nets [68, 79, 15], but very little work was done about unfoldings of time Petri nets.

3.1.1 Definition

A *time Petri net* is a tuple $(P, T, pre, post, efd, lfd, M_0, \theta_0)$ where $(P, T, pre, post, M_0)$ is a low-level Petri net, θ_0 is the initial date and $efd : T \longrightarrow \mathbb{R}$ and $lfd : T \longrightarrow \mathbb{R} \cup \{\infty\}$ associate an *earliest firing delay* $efd(t)$ and a *latest firing delay* $lfd(t)$ with each transition t. In graphical representations of time Petri nets, the closed interval $[efd(t), lfd(t)]$ (or the half-open interval $[efd(t), \infty)$ when $lfd(t) = \infty$) is written near each transition (see Figure 3.1).

In time Petri nets we require that the preset of each transition $t \in T$ is nonempty (${}^\bullet t \neq \emptyset$). This is not only required for the definition of processes, like in untimed Petri nets, but even for the interleaving semantics of time Petri nets. Indeed one needs to define the time when a transition was enabled; and it would be needlessly tricky to define it for a transition whose preset is empty.

3.1.2 Interleaving Semantics

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

A transition $t \in T$ is *enabled* in the state (M, dob, θ) if all of its input places are marked: ${}^\bullet t \subseteq M$. Its *date of enabling* $doe(t)$ is the date of birth of the youngest token in its input places: $doe(t) \stackrel{\text{def}}{=} \max_{p \in {}^\bullet t} dob(p)$.

All the time Petri nets we consider are *safe*, i.e. in each reachable state (M, dob, θ), if a transition t is enabled in (M, dob, θ), then $t^\bullet \cap (M \setminus {}^\bullet t) = \emptyset$. In practice it is convenient to ensure this property by requiring that the underlying untimed Petri net is safe. However this condition is not necessary: a safe Petri net can be built over an unsafe untimed Petri net.

A time Petri net starts in its *initial state* (M_0, dob_0, θ_0), which is given by the *initial marking* M_0 and the initial date θ_0. Initially, all the tokens carry the date θ_0 as date of birth: for all $p \in M_0$, $dob_0(p) \stackrel{\text{def}}{=} \theta_0$.

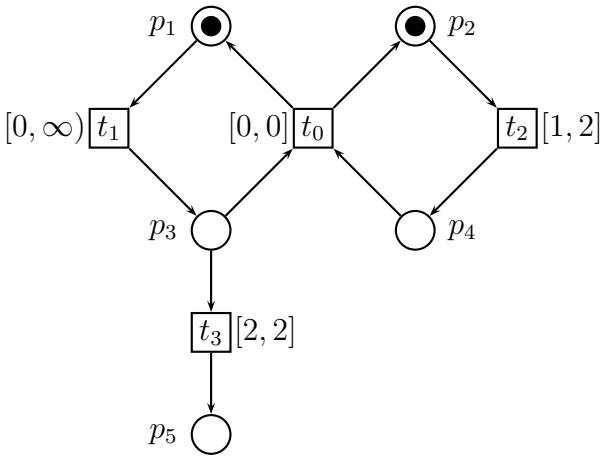
The transition t can fire at date $\theta' \geq \theta$ from state (M, dob, θ), if:

- t is enabled: ${}^\bullet t \subseteq M$;
- the minimum delay is reached: $\theta' \geq doe(t) + efd(t)$;
- the enabled transitions do not overtake the maximum delays: $\forall t' \in T \quad {}^\bullet t' \subseteq M \implies \theta' \leq doe(t') + lfd(t')$.

The firing of t at date θ' leads to the state $((M \setminus {}^\bullet t) \cup t^\bullet, dob', \theta')$, where $dob'(p) \stackrel{\text{def}}{=} dob(p)$ if $p \in M \setminus {}^\bullet t$ and $dob'(p) \stackrel{\text{def}}{=} \theta'$ if $p \in t^\bullet$.

We call *firing sequence* starting from the initial state S_0 any sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ where there exist states $S_1, \ldots, S_n$ such that for all $i \in \{1, \ldots, n\}$, firing t_i from S_{i-1} at date θ_i is possible and leads to S_i. The empty firing sequence is denoted ϵ.

Finally we assume that time *diverges*: when infinitely many transitions fire, time necessarily diverges to infinity.



$\sigma_1 \stackrel{\text{def}}{=} ((t_1,1),(t_2,2),(t_0,2))$
$\sigma_2 \stackrel{\text{def}}{=} ((t_2,1.3),(t_1,3),(t_0,3),(t_1,3),(t_2,5),(t_3,5))$
$\sigma_3 \stackrel{\text{def}}{=} ((t_2,1.3),(t_1,3),(t_0,3),(t_1,3),(t_3,5),(t_2,5))$

Figure 3.1: A safe time Petri net and some of its firing sequences starting at date 0 from the marking $M_0 \stackrel{\text{def}}{=} \{p_1, p_2\}$.

Example

In the initial state of the net of Figure 3.1, p_1 and p_2 are marked and their date of birth is 0. t_1 and t_2 are enabled and their date of enabling is the initial date 0. t_2 can fire in the initial state at any time between 1 and 2, say time 1.3. After this firing, p_1 and p_4 are marked, t_1 is the only enabled transition and it has already waited 1.3 time units. t_1 can fire at any time θ, provided it is greater than 1.3. Let t_1 fire at time 3. p_3 and p_4 are marked in the new state, and transitions t_3 and t_0 are enabled, and their date of enabling is 3 because they have just been enabled by the firing of t_1. To fire, t_3 would have to wait 2 time units. But transition t_0 cannot wait at all. So t_0 will necessarily fire (at time 3), and t_3 cannot fire.

Remark

The semantics of time Petri nets is often defined in a slightly different way: the state of the net is given as a pair (M, I), where M is the marking, and I maps each enabled transition t to the delay that has elapsed since it was enabled, that is $\theta - \mathit{doe}(t)$ with our notations. It is more convenient for us to attach time information to the tokens of the marking than to the enabled

transitions. We have chosen the date of birth of the tokens rather than their age, because we want to make the impact of the firing of transitions as local as possible. And the age of each token in the marking must be updated each time a transition t fires, whereas the date of birth has to be set only for the tokens that are created by t. Furthermore, usual semantics often deal with the delay between the firing of two consecutive transitions. On the other hand, we use the absolute firing date of the transitions instead. This fits better our approach in which we are not interested in the total ordering of the events.

3.1.3 Comparison with Other Timed True Concurrency Models

A plethora of timed extensions of Petri nets exist in the literature. Timed Petri nets [82] were often used for performance evaluation [88]. In these nets the firing of a transition takes time. More recently, time stream Petri nets [41] and timed-arc Petri nets [39] were introduced. Here time information is attached to the arcs, which allows more flexible definitions of the minimal and maximal firing delay of a transition than in time Petri nets. In [78, 77, 7], an attempt is done to define timed-arc Petri nets with more local timing constraints.

Timed extensions of networks of automata [2] are also very popular. The region graph can be compared to the state class graph of [14, 13].

Recently several results were proved concerning the comparison of the expressiveness of all these models [24, 10, 11, 12, 21, 22, 30, 89]. But these results are rather based on the sequential semantics of these models and do not take into account the preservation of concurrency.

3.2 Introduction to Unfoldings of Timed True Concurrency Models

The main problem with unfoldings of timed true concurrency models is to take *urgency* into account. Urgency refers to any specification that prevents the system from staying in a given state. Numerous real-time systems cannot be modeled correctly without urgency.

In time Petri nets and variants of this model, urgency is coded by the maximum delay between the time when a transition is enabled and the time when it fires. In timed automata invariants are assigned to some locations. Each invariant is a constraint on the values of the clocks. The invariant of a

state must be satisfied as long as the system stays in the location. Thus the semantics forces any transition to fire before the invariant becomes false.

Other models, like timed Petri nets (time and timed Petri nets are two different models), do not allow to represent urgency. Concerning unfoldings, it is much simpler to deal with these models. We will give an example in Section 3.2.5.

We recall in Sections 3.2.2 and 3.2.3 preceding work about causal semantics of time Petri nets. We found few works about causal semantics of other timed true concurrency models. Processes of timed Petri nets (without urgency) were defined in [84]. [59], deals with processes of an extension of Petri nets with bounds on the time a token may stay in a place. Finally, the very recent works [23, 28] tackle the problem of unfoldings of timed automata, using techniques that arise from Petri nets.

In [81], an approach based on linear logic aims at avoiding the computation of interleavings in the study of timed nets. Concerning partial order reductions, [71] deals with networks of timed automata without invariants, so without urgency. [6] studies the interleavings of the executions of networks of timed automata and proves a convexity property that allows to factorize the exploration of several nodes in the graph of zones.

3.2.1 Processes of Safe Time Petri Nets

The first definition of a causal semantics for time Petri nets was given by Aura and Lilius [3, 4]. In [70] an efficient state space search for time Petri nets is based on this causal semantics.

We carry on with this approach. As usual we study non-branching processes before defining unfoldings. A process of the time Petri net of Figure 3.1 is presented in Figure 3.2.

Following the example of processes of colored Petri nets (see Section 2.1.5), we remark that each execution of a time Petri net $N \stackrel{\text{def}}{=} (P, T, \mathit{pre}, \mathit{post}, \mathit{efd}, \mathit{lfd}, M_0, \theta_0)$ can be mapped into an execution of the underlying untimed Petri net $\mathit{Sup}(N) \stackrel{\text{def}}{=} (P, T, \mathit{pre}, \mathit{post}, M_0)$. If we deal with the interleaving semantics, we get: for all firing sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ of N, $(t_1, \ldots, t_n)$ is a firing sequence of $\mathit{Sup}(N)$.

In the context of processes, we will keep this remark in mind and define a process of a time Petri net N so that the corresponding process of the underlying untimed Petri net $\mathit{Sup}(N)$ is clearly highlighted.

Thus each process of N built over a process E of $\mathit{Sup}(N)$, will be given as a set $\mathcal{E}$ of dated events $(e, x) \in E \times \mathbb{R}$ such that $\mathcal{E} \in E \longrightarrow \mathbb{R}$. So, the time when an event $e \in E$ fired is $\mathcal{E}(e)$. By convention we also define $\mathcal{E}(\perp) \stackrel{\text{def}}{=} \theta_0$.

$e_1 \stackrel{\text{def}}{=} (\{(\bot, p_1)\}, t_1)$
$e_2 \stackrel{\text{def}}{=} (\{(\bot, p_2)\}, t_2)$
$e_3 \stackrel{\text{def}}{=} (\{(e_1, p_3), (e_2, p_4)\}, t_0)$
$e_4 \stackrel{\text{def}}{=} (\{(e_3, p_1)\}, t_1)$
$e_5 \stackrel{\text{def}}{=} (\{(e_3, p_2)\}, t_2)$
$e_6 \stackrel{\text{def}}{=} (\{(e_4, p_3)\}, t_3)$

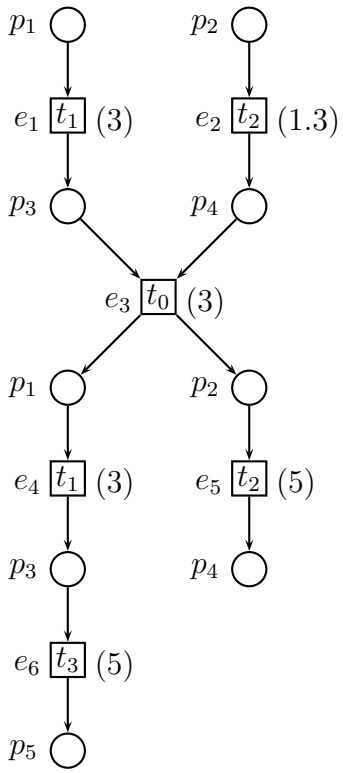


Figure 3.2: A process of the time Petri net of Figure 3.1. It corresponds to both firing sequences σ_2 and σ_3 (see Figure 3.1). The dates of the events are written in round brackets.

We define the date of birth of the token which stands in a place $p \in \mathit{Place}(\uparrow(E))$ after a process $\mathcal{E}$ as $\mathit{dob}_{\mathcal{E}}(p) \stackrel{\text{def}}{=} \mathcal{E}(^\bullet(\mathit{place}^{-1}_{|\uparrow(E)}(p)))$.

This allows us to define the state that is reached after a process $\mathcal{E}$ of N as: $(\mathit{Place}(\uparrow(E)), \mathit{dob}_{\mathcal{E}}, \max_{e \in E_\bot} \mathcal{E}(e))$.

As the time Petri nets that we consider are safe, we can take the comfort of defining their processes through a function Π that maps each firing sequence $((t_1, \theta_1), \ldots, (t_n, \theta_n))$ to a process, as we did for processes of safe untimed Petri nets in Section 1.2.2. Π is defined as follows:

- $\Pi(\epsilon) \stackrel{\text{def}}{=} \emptyset$
- $\Pi(((t_1, \theta_1), \ldots, (t_{n+1}, \theta_{n+1}))) \stackrel{\text{def}}{=} \mathcal{E} \cup \{(e, \theta')\}$, where
  - $\mathcal{E} \stackrel{\text{def}}{=} \Pi(((t_1, \theta_1), \ldots, (t_n, \theta_n)))$ and
  - the event $e \stackrel{\text{def}}{=} (\mathit{Place}^{-1}_{|\uparrow(E)}(^\bullet t_{n+1}), t_{n+1})$ represents the last firing of the sequence.

The set of all the processes obtained as the image by Π of the firing sequences of the Petri net N, is denoted by X_N.

3.2.2 Characterization of Timed Processes

In [3, 4], Aura and Lilius are concerned with the following problem: given an untimed process $E \in X_{Sup(N)}$, how can we assign a date to each event so that the timed process respects the time constraints of the time Petri net N? It may be that no dating function is possible. In order to answer this question, we have to solve a system of linear constraints on the dates of the events. But the difficulty consists in constructing these constraints. For this we need of course to check that each time a transition fired, the bounds on its firing delay were respected. *But we must also take into account all the transitions that were enabled during the execution and did not actually fire.* Indeed the semantics of time Petri nets forbids that some transitions remain enabled for a too long time. This forces us to attach a constraint to each of the events that would have coded the firing of one of the transitions that were enabled during the execution and did not actually fire. Yet these events are not in the process.

We recall now the main result of [3, 4]. The proof can be found in these articles. Moreover we will give a prove of an extended case in Theorem 3.12.

Theorem 3.1. *Let $E \in X_{Sup}(N)$ be a process of the untimed Petri net $Sup(N)$, and $\mathcal{E} : E \longrightarrow \mathbb{R}$ assign a date to each event of E. $\mathcal{E}$ is a process of N iff:*

$$\begin{cases} \forall e \in E \quad \Theta(e) \geq doe(e) + efd(\tau(e)) & \textit{(the minimum delay is respected)} \\ \forall e \in D_N \quad {}^\bullet e \subseteq \bigcup_{f \in E_\perp} f^\bullet \implies \min\{\theta_{end}, dod(e)\} \leq doe(e) + lfd(\tau(e)) \\ \qquad \textit{(the events that were enabled did not overtake their firing delay)} \end{cases}$$

where $doe(e) \stackrel{\text{def}}{=} \max_{b \in {}^\bullet e} \mathcal{E}({}^\bullet b)$ is the date when the event e was enabled, $\theta_{end} \stackrel{\text{def}}{=} \max_{f \in E_\perp} \mathcal{E}(f)$ is the date that is reached at the end of the process, and $dod(e) \stackrel{\text{def}}{=} \min\{\mathcal{E}(f) \mid f \in E \ \wedge \ {}^\bullet f \cap {}^\bullet e \neq \emptyset\}$ is the date when e was disabled (because an event f consumed one condition in ${}^\bullet e$).

In Figure 3.3, we give two examples where $\mathcal{E}$ is not a process.

It is essential to remark that the problem that we have just pointed out is characteristic of timed true concurrency models. In untimed models, nothing forces a transition to fire, and the constraints that allow us to build high-level processes concern only the parameters of the events of the process.

Because of the peculiarity that we have mentioned, defining timed unfoldings is much more difficult than to characterize timed processes.

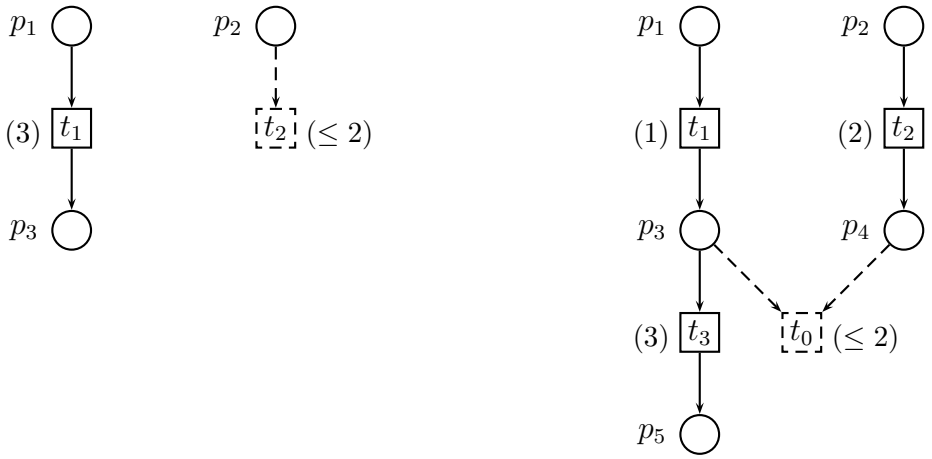


Figure 3.3: Two examples of incorrect timed processes for the model of Figure 3.1. In the process on the left, t_2 should have fired before the end of the process. In the process on the right, t_0 should have fired before t_3. In both cases, we show the corresponding events in dashed lines and we indicate the date that they cannot overtake.

3.2.3 Preceding Work on Timed Unfoldings

Few works tackle the problem of unfoldings of time Petri nets. We present the two approaches that we found in the literature.

Transformation Into Low-Level Petri Nets

A transformation from time Petri nets to low-level Petri nets was proposed in [20, 54]. For this one has to consider only discrete time elapsing. A place is created for each possible value of the age of each token. In actual fact a place is used for all the tokens whose age is beyond a bound after which the ageing of the token has no effect on the behavior of the net. Each transition of the time Petri net yields also several transitions in the low-level model. Each of them corresponds to a given age of the input tokens. Finally the progress of time is modeled by transitions called *ticks*, which move the tokens by increasing their age. Besides the problem of discrete time, the major drawback of this approach is that all the components of the net participate in the *ticks*, because time progresses for all the components. This breaks a lot of concurrency in the processes. Consequently this method does not resolve the problem of state space explosion.

In [52] this approach was extended to high-level timed nets but the transitions that model the progress of time remain global.

But even with small time intervals, the unfolding is huge, because the ticks synchronize all the parts of the net and prevent the benefit that we expect when we deal with partial order methods. As an example, a time Petri net made of n concurrent transitions that all carry the same interval $[0, 1]$ gives $n + 2^n + \sum_{k=0}^{n} \binom{n}{k} (n-k) = n + 2^n + n2^{n-1}$ events: n events are needed to code the firing of the transitions at time 0. Then, when time progresses from 0 to 1, each transition may or may not have fired; this yields 2^n *tick* events. Finally, the transitions that did not fire at time 0, fire at time 1, after a *tick*. For each of the $\binom{n}{k}$ *tick* events that represent a progress of time after k transitions have fired, the *tick* event is followed by $n - k$ events that model the firing of the remaining transitions. Figure 3.4 shows the unfolding for $n = 2$. For $n = 4$, we would have 52 events.

By comparison, the symbolic unfolding that we define in Section 3.4 has only n events.

Case Where Urgency and Confusion Do Not Appear Together

We have mentioned that the difficulty with timed unfoldings comes from urgency. More precisely the problem comes from the combination of urgency with *confusion*. Intuitively, confusion appears when the choices are not resolved locally. When these two phenomena appear together, a transition may be enabled and forced to fire quickly and a second transition, in conflict with the first one, cannot fire because of the urgency of the firing of the first transition.

Unfoldings have been introduced for timed models where this difficulty does not arise. [93, 95] deal with processes of timed Petri nets; this models does not allow to represent urgency. In [87], unfoldings were introduced for a class of time Petri nets, called *time independent choice nets*, in which urgency and confusion never appear together.

In Section 3.2.5, we explain that in some simple cases, the unfolding of a time Petri net can be defined as a branching process of the underlying low-level Petri net.

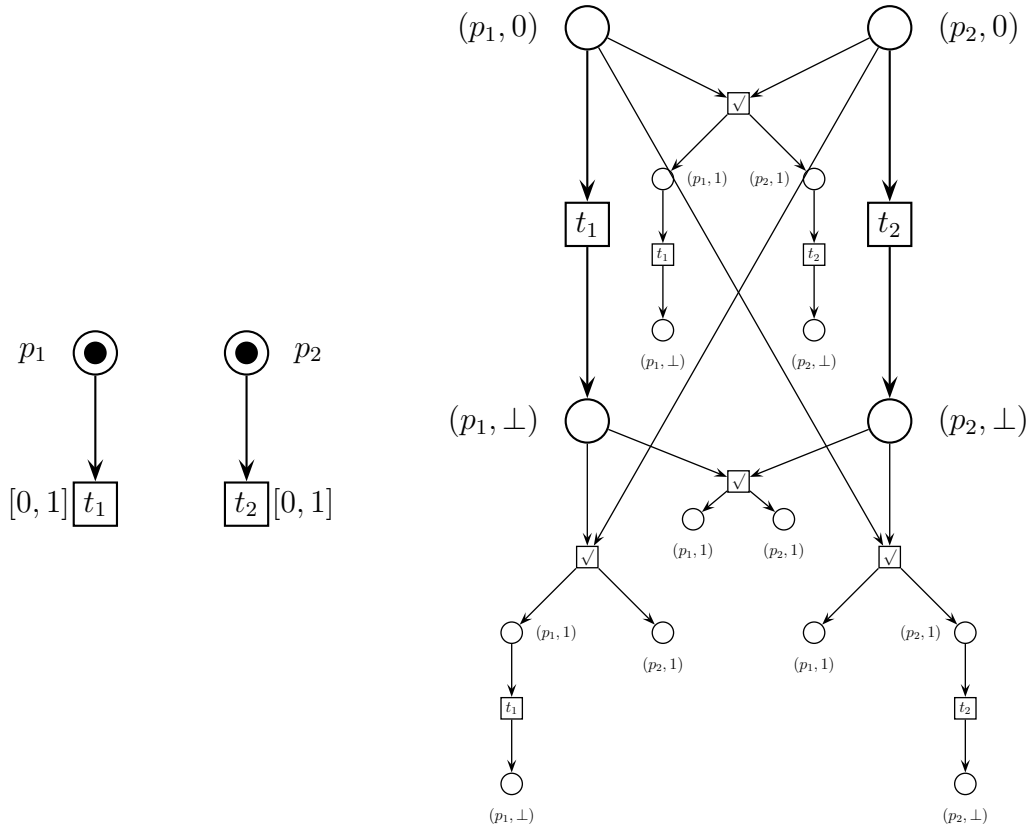


Figure 3.4: A time Petri net made of 2 concurrent transitions, and the unfolding that we obtain if we transform it into a low-level Petri net. The *tick* events are labelled by $\surd$. A condition corresponding to a token of age θ in place p is labeled (p, θ). When a place p is empty, a condition is labeled $(p, \bot)$ to represent it.

3.2.4 Pre-Processes

When we build unfoldings, we would like to be able to unfold separately two parts of the system when these two parts do not communicate, like the left part and the right part of the net of Figure 3.1 when t_1 and t_2 fire.

Unlike untimed Petri nets, in a timed context we accept that this may not yield proper processes but only what we will call *pre-processes*, in which the different parts of the system may not have reached the same date, provided the events that have been built are contained in a real execution. Let us define formally these *pre-process* as prefixes of the processes. Notice that in an untimed context pre-processes would simply be processes as any prefix of

a process is a process. But this is not true for time Petri nets, which explains why we have to expect only pre-processes in the unfolding. This is already what happens in [87] in the case of *time independent choice nets*. Let us define formally these *pre-process* as prefixes of the processes.

Definition 3.1 (pre-processes). *For all process $\mathcal{E}$, and for all causally closed set of events $E' \subseteq E_{\mathcal{E}}$ ($\lceil E' \rceil = E'$), $\mathcal{E}_{|E'}$ is called a* pre-process.

The set inclusion between pre-processes is called the *prefix* relation. For example, in the previous definition, $\mathcal{E}_{|E'}$ is a prefix of $\mathcal{E}$.

3.2.5 Symbolic Unfoldings of Timed True Concurrency Models: Simple Cases

As we explained above, the difficulties with timed unfoldings arise when urgency and confusion appear together. In this section we define symbolic unfoldings in a subclass of time Petri nets where these difficulties are not met.

Definition 3.2 (extended free choice [16, 40]). *An* extended free choice *(timed or untimed) Petri net is a Petri net such that:*

$$\forall t, t' \in T \quad {}^\bullet t \cap {}^\bullet t' \neq \emptyset \implies {}^\bullet t = {}^\bullet t' \ .$$

Another class of timed models for which an unfolding can be defined simply, consists in the models with no urgency. In time Petri nets the urgency comes from the latest firing delays attached to the transitions. If all these delays are infinite, then the time Petri net has no urgency and can be unfolded easily.

Actually, it is even possible to define a class of time Petri nets that encompasses both extended free choice time Petri nets and time Petri nets with no urgency. This new class is defined below and is called time extended free choice nets. Roughly, it allows a transition to have a finite latest firing delay provided all transition that is in conflict with it have a larger preset.

Definition 3.3 (time extended free choice). *A time Petri net is* time extended free choice *if:*

$$\forall t, t' \in T \quad \left\{ \begin{matrix} lfd(t) < \infty \\ {}^\bullet t \cap {}^\bullet t' \neq \emptyset \end{matrix} \right\} \implies {}^\bullet t \subseteq {}^\bullet t' \ .$$

In the same spirit, in [87], Semenov and Yakovlev define a class of *time independent choice nets* that takes not only structural, but also semantic aspects into account. Here we use only structural constraints for simplicity.

Definition 3.4 (symbolic unfolding of time extended free choice time Petri nets). *We define the* symbolic unfolding U_N *of a time extended free choice time Petri net by collecting all the events that appear in its processes:* $U_N \stackrel{\text{def}}{=} \bigcup_{\mathcal{E} \in X_N} E_{\mathcal{E}}$.

This unfolding has two important properties in the case of time extended free choice time Petri nets:

Theorem 3.2. *Let* $E \in X_{Sup(N)}$ *be a process of* $Sup(N)$ *and* $\mathcal{E} : E \longrightarrow \mathbb{R}$ *associate a firing date with each event of* E. $\mathcal{E}$ *is a pre-process of* N *iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \\ \forall e \in E \quad LP(e, \mathcal{E}) & \textit{(all the events respect the firing delays)} \end{cases}$$

where

$$LP(e, \mathcal{E}) \stackrel{\text{def}}{=} \begin{cases} \mathcal{E}(e) \geq \max_{b \in {}^\bullet e} \mathcal{E}({}^\bullet b) & \textit{(t is enabled when e fires)} \\ \mathcal{E}(e) \geq doe(t) + efd(t) & \textit{(the earliest firing delay is reached)} \\ \forall t' \in T \quad {}^\bullet t' \subseteq {}^\bullet t \implies \mathcal{E}(e) \leq doe(t') + lfd(t') & \\ & \textit{(the latest firing delays are respected)} \end{cases}$$

with $t \stackrel{\text{def}}{=} \tau(e)$ *and*
for all $t' \in T$ *such that* ${}^\bullet t' \subseteq {}^\bullet t$, $doe(t') \stackrel{\text{def}}{=} \max_{b \in Place^{-1}_{|{}^\bullet e}({}^\bullet t')} \mathcal{E}({}^\bullet b)$.

And, as well as for unfoldings of untimed Petri nets, we can decide if an event is in the symbolic unfolding of N by checking a condition on its causal past.

Theorem 3.3. *For all event* $e \in U_{Sup(N)}$, $e \in U_N$ *iff there exists a mapping* $\mathcal{E} : \lceil e \rceil \longrightarrow \mathbb{R}$ *which is a pre-process of* N.

We do not give proofs for the theorems 3.2 and 3.3 as they are particular cases of the theorems 3.6 and 3.7: the symbolic unfolding of time extended free choice time Petri nets as defined in this section is the same as the symbolic unfolding we obtain if we use the general definition of Section 3.3.5.

3.2.6 Difficulties that Arise in the General Case

The previous section shows that one can easily define the symbolic unfoldings of some classes of timed true concurrency models using similar tools as for symbolic unfoldings of colored Petri nets. However handling the general case is much more complicated.

In the general case, superimposing all the processes as we did for the simple cases would not be satisfactory because we would lose Theorems 3.2 and 3.3. Theorem 3.2 aims at characterizing pre-processes; it could be replaced by a characterization of processes using the technique of [3, 4]. But we would not be able to find an equivalent for Theorem 3.3, either for processes or pre-processes. The consequence is that we do not know any efficient way to build this object in general without constructing explicitly all the processes. Moreover, even if we build it, extracting the processes from their superimposition would be very hard.

The problem that we have is that it is very difficult to check if a set of dated events is a pre-process. It is easy only in simple cases like those described in Section 3.2.5, where urgency and confusion do not appear together. In the general case what we can do is to check if a set of dated events is a process, using the technique of [3, 4]. But as we explained this technique consists in solving a constraint on the firing dates of the events, and to construct this constraint one needs to take into account the events that have been enabled at some moment but did not fire, and are not in the process. But in order to determine this set of events, we need full information about the places that were marked during the execution. By nature the pre-processes do not give this information.

To be convinced that things are much more complicated than in the untimed case, it may be interesting to remark that in the timed case, the union of two pre-processes $\mathcal{E}$ and $\mathcal{E}'$ is not necessarily a pre-process, even if $E \cup E'$ is conflict free and $\mathcal{E}_{|E\cap E'} = \mathcal{E}'_{|E\cap E'}$. In the example of Figure 3.1, we observe this if $\mathcal{E}$ is the process which contains a firing of t_1 at time 0 and a firing of t_2 at time 1, and $\mathcal{E}'$ is the pre-process that we obtain by removing the firing of t_2 from the process made of t_1 at time 0, t_2 at time 2 and t_3 at time 2.

These difficulties come from the fact that in the semantics of time Petri nets, the condition that allows us to fire a transition concerns all the current state because it is necessary to check that no enabled transition in the net overtakes its maximum firing delay. And however not all the conditions that correspond to this global state become input conditions of the event: only the consumed ones do.

3.2.7 Outline of Our Method for the General Case

The main originality of our approach is to define a concurrent operational semantics for time Petri nets, where it will be possible to fire a transition without looking at the global state of the net. Only a few tokens will give enough information to be sure that the transition can fire.

Then processes will be replaced by extended processes, made of extended events instead of events; and the coding of each extended event will take into account all the tokens that were necessary to check that the corresponding transition could fire. With this additional information, it will become possible to check that an extended process corresponds to a pre-process by solving a constraint on the firing dates of the extended events. These constraints are of the same kind as those used in [3, 4] but they are adapted to pre-processes.

Finally we can define the symbolic unfolding of a time Petri net simply by superimposing extended processes instead of processes.

3.3 Concurrent Operational Semantics for Safe Time Petri Nets

Although the semantics of time Petri nets requires to check time conditions for all the enabled transitions in the net before firing a transition, there are cases when we know that a transition can fire at a given date θ, even if other transitions will fire before θ in other parts of the net. As an example consider the net of Figure 3.1 starting at time 0 in the marking $\{p_1, p_2\}$. The semantics forbids to fire t_1 at time 10 from the initial state because t_2 is enabled and must fire before time 2. However we are allowed to run the net until time 10 without firing t_1 (because its latest firing delay is infinite). Then, whatever has occurred until time 10, nothing can prevent t_1 from firing at date 10, because only t_1 can remove the token in place p_1. On the contrary, the firing of t_3 highly depends on the firing date of t_2 because when t_0 is enabled it fires immediately and disables t_3. So if we want to fire t_3 we have to check whether p_2 or p_4 is marked. This intuition leads us to define a *concurrent operational semantics* where it is possible to fire a transition without knowing the entire marking of the net, but only a partial marking made of the consumed tokens plus possibly some tokens which are only read (not consumed) in order to get enough information. Theorems 3.4 and 3.5 will validate our concurrent partial order semantics by establishing connections with the processes of Section 3.2.1.

Assumption

From now on we assume that we know a partition of the set P of places of the net into sets $P_i \subseteq P$ of mutually exclusive places[1], such that for all reachable

[1]If we do not know any such partition, a solution is to extend the structure of the net with one complementary place for each place of the net and to add these new places in the preset and in the postset of the transitions such that in any reachable marking each

marking M, $P_i \cap M$ is a singleton. For all place $p \in P_i$, we denote $\bar{p} \stackrel{\text{def}}{=} P_i \setminus \{p\}$. In the example of Figure 3.1, we will use the partition $\{p_1, p_3, p_5\}$, $\{p_2, p_4\}$. In fact this partition will be used to test the absence of a token in a marking. For instance if we want to fire t_3, we have to check that t_0 will not fire before t_3 and remove the token in place p_3; if we know that p_2 is marked then we can deduce that p_4 is not, and that t_0 is disabled.

Definition 3.5 (partial state). *A* partial state *of a time Petri net is a triple* (L, dob, lrd) *where* $L \subseteq P$ *is a partial marking and* $dob, lrd : L \longrightarrow \mathbb{R}$ *associate a* date of birth $dob(p)$ *and a* latest reading date $lrd(p)$ *with each token* $p \in L$.

As opposed to global states, partial states may give only partial information on the state of the net since the partial marking L may not contain one place per set of mutually exclusive places. Notice also that the date θ that appears in global states is replaced by a function that gives the latest reading date of each token of the partial marking, since the global time of the system is not relevant any more in a concurrent semantics.

Definition 3.6 (maximal partial state). *A partial state* (L, dob, lrd) *is* maximal *if* L *contains one place per set of mutually exclusive places (see the assumption before). From now on the notion of* maximal partial state *or* maximal state *will replace the notion of global state.*

Only in a maximal state it makes sense to define the date that is reached by the system, and consequently the age of a token in this state.

Definition 3.7 (age of a token in a maximal state). *Let* $S \stackrel{\text{def}}{=} (M, dob, lrd)$ *be a maximal state and let* $p \in M$ *be a token (marked place). The date that is reached by the system can be defined as* $\max_{p' \in M} lrd(p')$. *We define the* age $I_S(p)$ *of* p *in the state* S *as the difference:* $I_S(p) \stackrel{\text{def}}{=} \max_{p' \in M} lrd(p') - dob(p)$.

As explained in Section 3.2, when we deal with our concurrent operational semantics for safe time Petri nets, we accept that the different parts of the system may not have reached the same date. We give now conditions that allow to distinguish these states from those that may actually be reached after a proper process.

---

place $p \in P$ is marked iff its complementary place is not. This operation does not change the behavior of the time Petri net, provided the underlying untimed Petri net $Sup(N)$ is safe. If this is not true, a more tricky construction could be envisaged.

Definition 3.8 (temporally consistent maximal state (or consistent state)). *A maximal state* $S \stackrel{\text{def}}{=} (M, dob, lrd)$ *is* temporally consistent *if for each transition* $t \in T$ *which is enabled in* M *(*$^\bullet t \subseteq M$*),* $\min_{p \in {}^\bullet t} I_S(p) \leq lfd(t)$*. A temporally consistent maximal state is also called a* consistent state *for short.*

3.3.1 Local Firing Condition

In order to define a concurrent operational semantics for time Petri nets, we will construct a predicate that applies to tuples (L, dob, t, θ), where $L \subseteq P$ is a partial marking, $dob : L \longrightarrow \mathbb{R}$ associates a *date of birth* $dob(p)$ with each token (marked place) $p \in L$, t is a transition and $\theta \geq \max_{p \in L} dob(p)$ is a date.

Such a predicate is called a *local firing condition* and is supposed to tell if knowing that the net is in any state that contains a partial state (L, dob, lrd) with $lrd(p) \leq \theta$ for all $p \in L$, is enough to be sure that t can fire at date θ. Given a local firing condition LFC, we can define a concurrent operational semantics of the net where the transitions fire according to LFC.

Several local firing conditions are possible. In Section 3.3.4 we discuss what we expect for a good choice, and we give a trivial local firing condition that is correct but not satisfactory. In Section 3.3.5, we define a non trivial local firing condition, that we expect to be of interest.

But a local firing condition is considered correct only if the induced semantics reflects the classical interleaving semantics of safe time Petri nets. This notion of correctness has to be defined formally. This is the purpose of Section 3.3.3. As the correctness is defined in terms of processes, we introduce in Section 3.3.2 the notion of *extended processes*, which is the equivalent of processes for the concurrent operational semantics.

Semantics of Local Firings

We will now define formally the concurrent operational semantics that we obtain when we allow transitions to fire using a partial state if the local firing condition LFC is satisfied.

The time Petri net starts in an *initial maximal state* (M_0, dob_0, lrd_0), which is given by the *initial marking* M_0 and the initial date θ_0. Initially, all the tokens carry the date θ_0 as date of birth and latest reading date: for all $p \in M_0$, $dob_0(p) \stackrel{\text{def}}{=} lrd_0(p) \stackrel{\text{def}}{=} \theta_0$.

The transition t can fire at date θ using the partial marking $L \subseteq M$, from the maximal state (M, dob, lrd) if $(L, dob_{|L}, t, \theta)$ satisfies LFC and for all $p \in L$, $\theta \geq lrd(p)$.

This action leads to the maximal state $((M \setminus {}^\bullet t) \cup t^\bullet, dob', lrd')$ with

$$dob'(p) \stackrel{\text{def}}{=} \begin{cases} dob(p) & \text{if } p \in M \setminus {}^\bullet t \\ \theta & \text{if } p \in t^\bullet \end{cases}$$

and

$$lrd'(p) \stackrel{\text{def}}{=} \begin{cases} lrd(p) & \text{if } p \in M \setminus L \\ \theta & \text{if } p \in (L \setminus {}^\bullet t) \cup t^\bullet \end{cases}$$

We call *sequence of local firings* starting from the initial state S_0 any sequence $((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n))$ where there exist states $S_1, \ldots, S_n$ such that for all $i \in \{1, \ldots, n\}$, firing t_i from S_{i-1} at date θ_i using the partial marking L_i is possible and leads to S_i. The empty sequence of local firings is denoted ϵ.

3.3.2 Extended Processes

We will define a notion of *extended process*, which is close to the notion of process, but the events are replaced by *extended events* which represent firings from partial states and keep track of all the conditions corresponding to the partial state, not only those that are consumed by the transition: the other conditions will be treated as context of the event. It uses classical techniques of *contextual nets* or nets with *read arcs* (see Section 1.4). It would also be possible to consume and rewrite the conditions in the context of an event, but we feel that the notion of read arc or contextual net is a good way to capture the idea that we develop here.

An extended process is a set $\dot{\mathcal{E}}$ of pairs $(\dot{e}, \theta)$, where $\dot{e}$ is an extended event and θ is its date. The set of extended events in $\dot{\mathcal{E}}$ is denoted $\dot{E}_{\dot{\mathcal{E}}}$ or simply $\dot{E}$. Thus the extended process can be seen as a mapping $\dot{\mathcal{E}} : \dot{E} \longrightarrow \mathbb{R}$.

Extended events will be pairs $\dot{e} \stackrel{\text{def}}{=} (B, t)$, and we denote $\tau(\dot{e}) \stackrel{\text{def}}{=} t$ and $\dot{e}^\bullet \stackrel{\text{def}}{=} \{(\dot{e}, p) \mid p \in t^\bullet\}$. In an extended event, not all the conditions of B are consumed, but only ${}^\bullet\dot{e} \stackrel{\text{def}}{=} Place^{-1}_{|B}({}^\bullet t)$; the conditions in $\underline{\dot{e}} \stackrel{\text{def}}{=} B \setminus {}^\bullet\dot{e}$ are only *read* by $\dot{e}$, which is represented by read arcs.

Definition 3.9 ($\dot{D}_N$). *In the context of extended processes of a safe time Petri net N, the extended events will be elements of the set $\dot{D}_N$ defined as the smallest set such that:*

$$\forall B \subseteq \bigcup_{\dot{e} \in \dot{D}_{N\perp}} \dot{e}^\bullet \quad \forall t \in T \quad {}^\bullet t \subseteq Place(B) \implies (B, t) \in \dot{D}_N \ .$$

Like for processes, we define the set of conditions that remain at the end of the extended process $\dot{\mathcal{E}}$ built over the set $\dot{E}$ of extended events as $\uparrow(\dot{E}) \stackrel{\text{def}}{=} \bigcup_{\dot{e}\in\dot{E}} \dot{e}^\bullet \setminus \bigcup_{\dot{e}\in\dot{E}} {}^\bullet\dot{e}$.

The function $\dot{\Pi}$ that maps each sequence of local firings $((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n))$ to an extended process is defined as follows:

- like for processes, $\dot{\Pi}(\epsilon) \stackrel{\text{def}}{=} \emptyset$;
- $\dot{\Pi}(((t_1, L_1, \theta_1), \ldots, (t_{n+1}, L_{n+1}, \theta_{n+1}))) \stackrel{\text{def}}{=} \dot{\mathcal{E}} \cup \{(\dot{e}, \theta_{n+1})\}$, where $\dot{\mathcal{E}} \stackrel{\text{def}}{=} \dot{\Pi}(((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n)))$ and the extended event $\dot{e} \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(\dot{E})}(L_{n+1}), t_{n+1})$ represents the last local firing of the sequence.

The set of all the extended processes obtained as the image by $\dot{\Pi}$ of the sequences of local firings induced by LFC, is denoted $\dot{X}_{LFC}$. We sometimes write simply $\dot{X}$.

Figure 3.6 shows several extended processes.

Causality. As we use read arcs like in Section 1.4.3, we recall the definition of the unconditional or strong causality $\rightarrow$ and of the conditional or weak causality $\nearrow$ between extended events:

- $\dot{e} \rightarrow \dot{f}$ iff $\dot{e}^\bullet \cap ({}^\bullet\dot{f} \cup \underline{\dot{f}}) \neq \emptyset$ and
- $\dot{e} \nearrow \dot{f}$ iff $(\dot{e} \rightarrow \dot{f}) \vee (\underline{\dot{e}} \cap {}^\bullet\dot{f} \neq \emptyset)$.

The causal past of an extended event is defined like the causal past of an event: $\lceil \dot{e} \rceil \stackrel{\text{def}}{=} \{\dot{f} \in \dot{E} \mid \dot{f} \rightarrow^* \dot{e}\}$.

Definition 3.10 ($dob_{\dot{\mathcal{E}}}$ and $lrd_{\dot{\mathcal{E}}}$). *For extended processes we use the same definition as for processes of time Petri nets (see Section 3.2.1): the date of birth of the token which stands in place p after the extended process $\dot{\mathcal{E}}$ is:*

$$dob_{\dot{\mathcal{E}}}(p) \stackrel{\text{def}}{=} \dot{\mathcal{E}}({}^\bullet(place^{-1}_{|\uparrow(\dot{E})}(p))) \ .$$

But additionally we define the latest date when the token which stands in place p was read, as:

$$lrd_{\dot{\mathcal{E}}}(p) \stackrel{\text{def}}{=} \max\{dob_{\dot{\mathcal{E}}}(p), \max_{\dot{e}\in\dot{E},\, b\in\underline{\dot{e}}} \dot{\mathcal{E}}(\dot{e})\} \ .$$

Remark that $lrd_{\dot{\mathcal{E}}}(p)$ is set to $dob_{\dot{\mathcal{E}}}(p)$ when no event reads the token.

Definition 3.11 ($RS(\dot{\mathcal{E}})$). *The maximal state that is reached after an extended process $\dot{\mathcal{E}}$ is defined as $RS(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} (Place(\uparrow(\dot{E})), dob_{\dot{\mathcal{E}}}, lrd_{\dot{\mathcal{E}}})$.*

Definition 3.12 (temporally complete extended process, $\dot{Y}$). *We say that $\dot{\mathcal{E}}$ is* temporally complete *if $RS(\dot{\mathcal{E}})$ is temporally consistent. The set of all temporally complete extended processes is denoted $\dot{Y}$.*

3.3.3 Correctness of a Local Firing Condition

Each extended event $\dot{e} \in \dot{D}_N$ can be mapped to the corresponding event $h(\dot{e}) \in D_N$ defined as:

$$h(\dot{e}) \stackrel{\text{def}}{=} \left(\left\{ (h(\dot{f}), p) \mid (\dot{f}, p) \in {}^\bullet\dot{e} \right\}, \tau(\dot{e}) \right) .$$

This definition is also valid for the initial extended event $\perp$ and gives $h(\perp) = \perp$.

We denote also $h(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} \{(h(\dot{e}), \theta) \mid (\dot{e}, \theta) \in \dot{\mathcal{E}}\}$ for all extended process $\dot{\mathcal{E}}$. Intuitively, $h(\dot{\mathcal{E}})$ is what we obtain if we remove the read arcs from $\dot{\mathcal{E}}$. We want that this yields a pre-process. For example the extended process $\dot{\mathcal{E}}$ of Figure 3.6 is mapped to the process of Figure 3.2.

Definition 3.13 (correctness). *We say that LFC is a* valid local firing condition *iff for all extended process $\dot{\mathcal{E}} \in \dot{X}_{LFC}$, $h(\dot{\mathcal{E}})$ is a pre-process. In other terms there exists a process $\mathcal{E}' \in X$ such that $h(\dot{\mathcal{E}}) \subseteq \mathcal{E}'$.*

3.3.4 Selecting Local Firing Conditions

The definition of extended processes is parameterized by a local firing condition LFC: each extended event must correspond to a local firing that satisfies LFC, the others are forbidden. A good choice for LFC takes three notions into account: completeness, redundancy and preservation of concurrency.

Definition 3.14 (completeness). *A local firing condition LFC is* complete *if for all process $\mathcal{E} \in X$, there exists an extended process $\dot{\mathcal{E}}' \in \dot{X}_{LFC}$ such that $h(\dot{\mathcal{E}}') = \mathcal{E}$.*

Redundancy. Given a local firing condition LFC and a process $\mathcal{E} \in X$, there may exist *several* extended processes $\dot{\mathcal{E}}' \in \dot{X}_{LFC}$ such that $h(\dot{\mathcal{E}}') = \mathcal{E}$. We call this *redundancy*.

In particular, if LFC is satisfied by two tuples (L, dob, t, θ) and (L', dob', t, θ) with $L' \subsetneq L$ and $dob' = dob_{|L'}$, then all the extended processes involving (L, dob, t, θ) are redundant.

But redundancy can also appear in other situations. Figure 3.5 illustrates this fact. It uses the definition of LFC that we propose in Section 3.3.5 above. The sets of mutually exclusive places are $\{p_1, p_4\}$, $\{p_2, p_5\}$ and $\{p_3, p_6\}$. In order to let t_1 fire at time 2, we have to check that t_2 is disabled, or has not been enabled for a too long time. In $\dot{\mathcal{E}}_1$, this is ensured by checking via the read arc that p_2 is still marked when t_1 fires. In $\dot{\mathcal{E}}_2$ this is done by checking that p_6 is marked, which implies that p_3 is empty. Nevertheless, if we erase

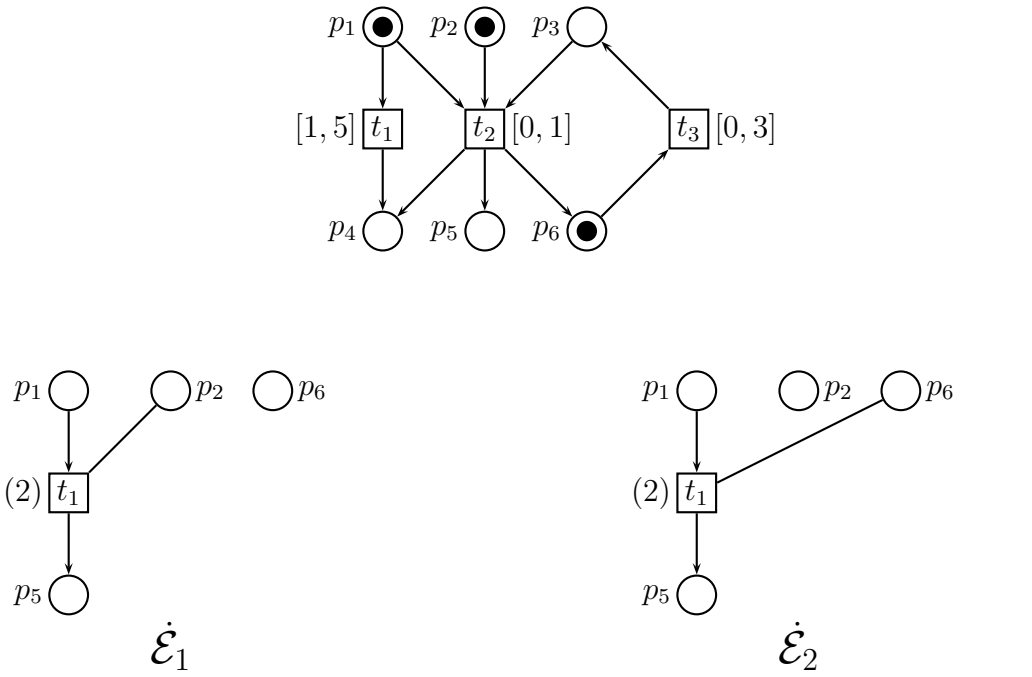


Figure 3.5: A time Petri net (on top) and two of its extended processes $\dot{\mathcal{E}}_1$ and $\dot{\mathcal{E}}_2$. If we erase their read arcs, the two pre-processes $h(\dot{\mathcal{E}}_1)$ and $h(\dot{\mathcal{E}}_2)$ that we obtain are identical. This is what we call redundancy.

the read arcs from $\dot{\mathcal{E}}_1$ and $\dot{\mathcal{E}}_2$, the two pre-processes $h(\dot{\mathcal{E}}_1)$ and $h(\dot{\mathcal{E}}_2)$ that we obtain are identical.

A Trivial Choice for *LFC* Which does not Preserve any Concurrency

A trivial complete predicate *LFC* is the predicate that demands that the state S is a global state, and then check that t can fire at date θ' from S according to the interleaving semantics or time Petri nets. In addition, this choice gives little redundancy. But the extended events of the extended processes that we obtain in this case are totally ordered by causality. In other words, these extended processes do not exhibit any concurrency at all. As a result, the unfoldings that we obtain if we use this definition of *LFC* are comparable to the one of Figure 3.4.

3.3.5 A Proposition for *LFC*

What we want is a complete local firing condition that generates as little redundancy as possible and that exhibits as much concurrency as possible. In the local firing condition that we propose, we check that the partial marking that is used gives enough information to be sure that all its tokens can stay until the time when we want to fire the condition.

Definition 3.15 (local firing condition *LFC′*). *We first define a predicate LFC' as follows: $LFC'(L, dob, t, \theta)$ holds iff*

- *t is enabled: ${}^\bullet t \subseteq L$;*
- *the minimum delay is reached: $\theta \geq doe(t) + efd(t)$;*
- *the transitions that might consume tokens of L are disabled or do not overtake the maximum delays:*

$$\forall t' \in T \quad {}^\bullet t' \cap L \neq \emptyset \implies \begin{cases} & \exists p \in {}^\bullet t' \quad \bar{p} \cap L \neq \emptyset \\ \vee & \theta \leq \max\limits_{p \in {}^\bullet t' \cap L} dob(p) + lfd(t') \end{cases}$$

The predicate LFC' guarantees that a partial state (L, dob, lrd) with $lrd(p) \leq \theta$ for all $p \in L$ gives enough information to be sure that t can fire at date θ. We define a new local firing condition LFC by demanding that the partial marking L is minimal.

Definition 3.16 (local firing condition *LFC*). *LFC is defined as:*

$$LFC(L, dob, t, \theta) \quad \textit{iff} \quad \begin{cases} LFC'(L, dob, t, \theta) \\ \nexists L' \subsetneq L \quad LFC'(L', dob_{|L'}, t, \theta) \ . \end{cases}$$

It is important that the constraints (see Theorems 3.6 and 3.7) can be solved automatically: with the definition of LFC we have proposed here, the quantifiers ($\forall$ and $\exists$) on places and transitions expand into disjunctions and conjunctions. The result is a disjunction of conjunctions of linear inequalities on the $\dot{\mathcal{E}}(\dot{e})$. When a "max" appears in an inequality, this inequality can be rewritten into the desired form. These systems are shown near the events in Figure 3.7.

Correctness and Completeness of *LFC*

Here we prove that the proposition of local firing condition that we defined is correct and complete.

Lemma 3.1. *For all* $\dot{\mathcal{E}} \in \dot{X}$, $h(\dot{\mathcal{E}}) \in X$ *iff* $\dot{\mathcal{E}} \in \dot{Y}$.

Proof. Let $\dot{\mathcal{E}} \in \dot{X}$ be an extended process and denote $(M, dob, lrd) \stackrel{\text{def}}{=} RS(\dot{\mathcal{E}})$ and $\theta \stackrel{\text{def}}{=} \max_{p\in M} lrd(p) = \max_{\dot{e}\in\dot{E}} \dot{\mathcal{E}}(\dot{e})$.

It follows from the definition of the processes that if $h(\dot{\mathcal{E}}) \in X$, then $\dot{\mathcal{E}}$ is temporally complete.

Conversely, assume that $\dot{\mathcal{E}}$ is temporally complete. Choose $\dot{e} \in \dot{E}$ such that $\dot{\mathcal{E}}(\dot{e}) = \theta$ and $\nexists \dot{f} \in \dot{E}$ such that $\dot{e} \nearrow \dot{f}$. Then denote $(M', dob', lrd') \stackrel{\text{def}}{=} RS(\dot{\mathcal{E}}_{|\dot{E}\setminus\{\dot{e}\}})$ and $\theta' \stackrel{\text{def}}{=} \max_{p\in M'} lrd'(p)$, and let $t \in T$ such that ${}^\bullet t \subseteq M'$. If ${}^\bullet t \cap {}^\bullet\tau(\dot{e}) = \emptyset$, then $doe'(t) = doe(t) \geq \theta - lfd(t) \geq \theta' - lfd(t)$. Otherwise let $L \stackrel{\text{def}}{=} {}^\bullet\dot{e} \cup \underline{\dot{e}}$. As $LFC(L, dob, \tau(\dot{e}), \dot{\mathcal{E}}(\dot{e}))$ holds, then

$$\begin{cases} & \exists p \in {}^\bullet t \quad \bar{p} \cap L \neq \emptyset \\ \vee & \theta \leq \max\limits_{p\in{}^\bullet t\cap L} dob'(p) + lfd(t) \end{cases}$$

As ${}^\bullet t \subseteq M'$, then $\nexists p \in {}^\bullet t$ such that $\bar{p} \cap L \neq \emptyset$; thus $\theta \leq \max\limits_{p\in{}^\bullet t\cap L} dob'(p) + lfd(t)$. Hence $doe'(t) = \max\limits_{p\in{}^\bullet t} dob'(p) \geq \max\limits_{p\in{}^\bullet t\cap L} dob'(p) \geq \theta - lfd(t) \geq \theta' - lfd(t)$. As a result $\dot{\mathcal{E}}_{|\dot{E}\setminus\{\dot{e}\}} \in \dot{Y}$.

Assume now that $\mathcal{E}' \stackrel{\text{def}}{=} h(\dot{\mathcal{E}}_{|\dot{E}\setminus\{\dot{e}\}}) \in X$. It leads to the global state (M', dob', θ'). As ${}^\bullet\tau(\dot{e}) \subseteq M'$ and $\theta \geq \theta'$ and $\theta \geq doe'(\tau(\dot{e})) + efd(\tau(\dot{e}))$ and for all $t \in T$, ${}^\bullet t \subseteq M' \implies \theta \leq doe'(t) + lfd(t)$, then $\tau(\dot{e})$ can fire at date θ from (M', dob', θ'), which is coded by the event $(Place^{-1}_{|\uparrow(E_{\mathcal{E}'})}(\tau(\dot{e})), \tau(\dot{e})) = h(\dot{e})$. Thus $h(\dot{\mathcal{E}}) \in X$. □

Theorem 3.4 (correctness of *LFC*). *For all extended process* $\dot{\mathcal{E}} \in \dot{X}$, $h(\dot{\mathcal{E}})$ *is a pre-process. In other terms there exists a process* $\mathcal{E}' \in X$ *such that* $h(\dot{\mathcal{E}}) \subseteq \mathcal{E}'$.

Proof. To prove that *LFC* is correct, we will prove that for all $\dot{\mathcal{E}} \in \dot{X}$, there exists $\dot{\mathcal{E}}' \in \dot{Y}$ such that $\dot{\mathcal{E}} \subseteq \dot{\mathcal{E}}'$; as a consequence $h(\dot{\mathcal{E}}) \subseteq h(\dot{E}') \in X$.

Let $\dot{\mathcal{E}} \in \dot{X}$. If $\dot{\mathcal{E}}$ is temporally complete, then it is sufficient to take $\dot{\mathcal{E}}' \stackrel{\text{def}}{=} \dot{\mathcal{E}}$.

Otherwise, denote $(M, dob, lrd) \stackrel{\text{def}}{=} RS(\dot{\mathcal{E}})$ and $\theta \stackrel{\text{def}}{=} \max_{p\in M} lrd(p)$, choose $t \in T$ such that ${}^\bullet t \subseteq M \wedge \theta > doe(t) + lfd(t)$ and such that t minimizes $\theta_t \stackrel{\text{def}}{=} doe(t) + lfd(t)$. Let $\dot{F} \stackrel{\text{def}}{=} \{\dot{f} \in \dot{E} \mid \dot{\mathcal{E}}(\dot{f}) \leq \theta_t\}$. $\dot{\mathcal{E}}_{|\dot{F}}$ is a temporally complete extended process. Denote $(M', dob', lrd') \stackrel{\text{def}}{=} RS(\dot{\mathcal{E}}_{|\dot{F}})$. $LFC'(M', dob', t, \theta_t)$ holds. Thus there exists $L \subseteq M'$ such that $LFC(L, dob'_{|L}, t, \theta_t)$ holds. Let $\dot{e} \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(\dot{F})}(L), t)$. We will show that $\dot{\mathcal{E}} \cup \{(\dot{e}, \theta_t)\} \in \dot{X}$. $\dot{\mathcal{E}} \cup \{(\dot{e}, \theta_t)\}$ is

compatible with $\nearrow$: if an extended event $\dot{f} \in \dot{E}$ is such that $\underline{\dot{f}} \cap {}^\bullet\dot{e} \neq \emptyset$, then $\dot{\mathcal{E}}(\dot{f}) \leq \theta_t$ and if ${}^\bullet\dot{f} \cap \underline{\dot{e}} \neq \emptyset$, then $\dot{\mathcal{E}}(\dot{f}) > \theta_t$. The strict inequality in the second case also guarantees that $\nearrow$ is acyclic on $\dot{E} \cup \{\dot{e}\}$. As a result, we have built an extended process $\dot{\mathcal{E}} \cup \{(\dot{e}, \theta_t)\} \in \dot{X}$ by adding the event to $\dot{\mathcal{E}}$.

Iterating this until $\dot{\mathcal{E}}$ is temporally complete, terminates if we assume that time diverges: at each step $\dot{\mathcal{E}}_{|\dot{F}}$ is temporally complete, so $h(\dot{\mathcal{E}}_{|\dot{F}}) \in X$; moreover this process has strictly more events at each step and the dates remain below θ, which does not increase. □

Theorem 3.5 (completeness of *LFC*). *For all process $\mathcal{E} \in X$, there exists an extended process $\dot{\mathcal{E}} \in \dot{X}$ such that $h(\dot{\mathcal{E}}) = \mathcal{E}$.*

Proof. Let $\mathcal{E} \in X$ leading to the global state (M, dob, θ), let $t \in T$ be a transition that can fire at date $\theta' \geq \theta$ from (M, dob, θ), and assume that there exists an extended process $\dot{\mathcal{E}}' \in \dot{X}$ such that $h(\dot{\mathcal{E}}') = \mathcal{E}$. $LFC'(M, dob, t, \theta')$ holds. Thus there exists $L \subseteq M$ such that $LFC(L, dob_{|L}, t, \theta')$ holds. Define $\dot{e} \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow \dot{E}'}(L), t)$. $\dot{\mathcal{E}}' \cup \{(\dot{e}, \theta')\} \in \dot{X}$ and the event $h(\dot{e})$ codes the firing of t at date θ' after $\mathcal{E}$. □

Discussion About the Relevance of *LFC*

In Section 3.3.1 we have given a general definition of firing conditions. Then in Definition 3.13 we have expressed constraints about the local firing conditions, in order to guarantee that the extended processes that we obtain are mapped to valid pre-processes. In Section 3.3.4, we have other guidelines for the construction of a relevant *LFC*.

It is a fact that there exists firing conditions that are correct w.r.t. Definition 3.13, but are not very relevant because the unfolding that they generate is not likely to be compact. This is the case of the trivial choice described at the end of Section 3.3.4.

Here we give some arguments in order to convince the reader that the non trivial definition of *LFC* that we have proposed in Section 3.3.5 is relevant. First we have demanded that the partial state that is used in *LFC* is minimal; this prevents from using needlessly large partial states, that would destroy the concurrency.

As a consequence the unfolding contains no read arc at all if we deal with a time extended free choice Petri net (see Definition 3.3). As an even simpler example, consider a time Petri net N where all the time intervals are $[0, \infty)$; it behaves like the underlying untimed Petri net $Sup(N)$. And its unfolding will precisely be equal to the unfolding of $Sup(N)$.

Finally consider a time Petri net where two components do not communicate at all (that is the corresponding graph is disconnected). Then our

local firing condition will never try to connect these parts. As a result, the unfolding of the whole net will be exactly the juxtaposition (or formally, the union) of the unfoldings of the disconnected parts.

3.4 Symbolic Unfoldings of Safe Time Petri Nets

We have explained in Section 3.2 that the definition of the unfolding has to be based on a concurrent operational semantics. Now we will show how the extended processes induced by a local firing condition, can be superimposed to build a symbolic unfolding, and that it is easy to retrieve the extended processes from their superimposition and to build the unfolding.

Definition 3.17 (symbolic unfolding). *We define the* symbolic unfolding *U_{LFC} (or simply U) of a time Petri net by collecting all the extended events that appear in its extended processes induced by a valid local firing condition LFC: $U_{LFC} \stackrel{\text{def}}{=} \bigcup_{\dot{\mathcal{E}} \in \dot{X}_{LFC}} \dot{E}_{\dot{\mathcal{E}}}$.*

We write a theorem that gives an efficient way to check if a set of extended event of $\dot{D}_N$ can be used to build an extended process.

Theorem 3.6. *Let $\dot{E} \subseteq \dot{D}_N$ be a finite set of extended events and $\dot{\mathcal{E}} : \dot{E} \longrightarrow \mathbb{R}$ associate a firing date with each extended event of $\dot{E}$. $\dot{\mathcal{E}}$ is an extended process iff:*

$$\begin{cases} \lceil \dot{E} \rceil = \dot{E} & \text{(}\dot{E}\text{ is causally closed)} \\ \nexists \dot{e}, \dot{e}' \in \dot{E} \quad \dot{e} \neq \dot{e}' \ \wedge \ {}^\bullet\dot{e} \cap {}^\bullet\dot{e}' \neq \emptyset & \text{(}\dot{E}\text{ is conflict free)} \\ \nexists \dot{e}_0, \dot{e}_1, \ldots, \dot{e}_n \in \dot{E} \quad \dot{e}_0 \nearrow \dot{e}_1 \nearrow \cdots \nearrow \dot{e}_n \nearrow \dot{e}_0 & \text{(}\nearrow\text{ is acyclic on }\dot{E}\text{)} \\ \forall \dot{e}, \dot{e}' \in \dot{E} \quad \dot{e} \nearrow \dot{e}' \implies \dot{\mathcal{E}}(\dot{e}) \leq \dot{\mathcal{E}}(\dot{e}') & \text{(}\dot{\mathcal{E}}\text{ is compatible with }\nearrow\text{)} \\ \forall \dot{e} = (B, t) \in \dot{E} \quad LFC\left(Place(B), dob_{\dot{\mathcal{E}}_{\lfloor \lceil B \rceil}}, t, \dot{\mathcal{E}}(\dot{e})\right) & \\ & \text{(}\dot{e}\text{ satisfies the local firing condition)} \end{cases}$$

Proof. Let $\dot{\mathcal{E}} \in \dot{X}$ be an extended process that satisfies the conditions in the curly brace, let $\dot{e} \stackrel{\text{def}}{=} (B, t)$ with $B \subseteq \uparrow(\dot{E})$ and $t \in T$ and $\theta' \geq \max_{\dot{f} \in \dot{E},\, \dot{f} \nearrow \dot{e}} \dot{\mathcal{E}}(\dot{f})$ such that $LFC(Place(B), dob_{\dot{\mathcal{E}}}, t, \theta')$ holds. Then we will show that the extended process $\dot{\mathcal{E}}' \stackrel{\text{def}}{=} \dot{\mathcal{E}} \cup \{(\dot{e}, \theta')\}$ also satisfies the conditions in the curly brace. By construction $\dot{E}'$ is causally closed. Moreover for each condition $b \in {}^\bullet\dot{e}$ that is consumed by $\dot{e}$, $b \in \uparrow(\dot{E})$, which implies that b has not been consumed by any event of $\dot{E}$. Thus for all $\dot{f} \in \dot{E}$, ${}^\bullet\dot{e} \cap {}^\bullet\dot{f} = \emptyset$ and $\neg(\dot{e} \nearrow \dot{f})$. So $\dot{E}'$ is conflict free and $\nearrow$ is acyclic on $\dot{E}'$. $\dot{\mathcal{E}}'$ is compatible with $\nearrow$ because $\dot{\mathcal{E}}$ is compatible with $\nearrow$ and $\dot{\mathcal{E}}'(\dot{e}) = \theta' \geq \max_{\dot{f} \in \dot{E},\, \dot{f} \nearrow \dot{e}} \dot{\mathcal{E}}(\dot{f})$.

Conversely let $\dot{\mathcal{E}}'$ satisfy the conditions in the curly brace. If $\dot{E}' = \emptyset$, then $\dot{\mathcal{E}}' = \emptyset \in \dot{X}$. Otherwise let $\dot{e} \in \dot{E}'$ be an extended event that has no successor by $\nearrow$ in $\dot{E}'$ (such an extended event exists since $\nearrow$ is acyclic on $\dot{E}'$). $\dot{\mathcal{E}} \stackrel{\text{def}}{=} \dot{\mathcal{E}}'_{|\dot{E}'\setminus\{\dot{e}\}}$ satisfies the conditions in the curly brace. Assume that $\dot{\mathcal{E}} \in \dot{X}$. As $\dot{E}'$ is conflict free, ${}^\bullet\dot{e} \subseteq \uparrow(\dot{E})$. And as $\dot{e}$ has no successor by $\nearrow$ in $\dot{E}'$, $\underline{\dot{e}} \subseteq \uparrow(\dot{E})$. Furthermore $\dot{\mathcal{E}}'(\dot{e}) \geq \max_{\dot{f}\in\dot{E},\, \dot{f}\nearrow\dot{e}} \dot{\mathcal{E}}(\dot{f})$ and $LFC\left(Place({}^\bullet\underline{\dot{e}}), dob_{\dot{\mathcal{E}}_{|\lceil {}^\bullet\underline{\dot{e}}\rceil}}, \tau(\dot{e}), \dot{\mathcal{E}}'(\dot{e})\right)$ holds. Thus $\dot{\mathcal{E}}' = \dot{\mathcal{E}} \cup \{(\dot{e}, \dot{\mathcal{E}}'(\dot{e}))\} \in \dot{X}$. □

As well as in unfoldings of untimed models, we can now check easily if an extended event $\dot{e} \in \dot{D}_N$ is in the unfolding of N, by checking only if its causal past can be the support of an extended process.

Theorem 3.7. *For all $\dot{e} \in \dot{D}_N$,*

$$\dot{e} \in U \quad \textit{iff} \quad \exists \dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \quad \dot{\mathcal{E}} \in \dot{X} \ .$$

Proof. This proof is similar to the proof of Theorem 2.3. □

This theorem allows us to simply build the unfolding starting from the empty set and adding the extended events that satisfy the condition.

3.5 Complete Finite Prefixes

We have defined the symbolic unfolding of a time Petri net. In general this structure is infinite, as well as the unfoldings of untimed Petri nets.

However in the untimed case it is possible to define a finite prefix of the unfolding, which contains complete information about the unfolding [72, 46]. To construct this complete finite prefix one remarks that each untimed safe Petri net has finitely many markings, and that if two processes reach the same marking, then they have the same possible futures.

With time Petri nets, the same is true with two temporally complete extended processes that reach the same consistent state. But in general there are infinitely many possible maximal states. This is why we will try to group them as much as possible. The problem of the density of time has already been solved by the use of a symbolic representation of the dates. Another problem is that time keeps progressing and never loops; this is why the age of the tokens will now be used instead of their date of birth. Recall that the date of birth was first preferred in order to define a concurrent semantics where the system is allowed to reach temporally inconsistent states, that is states where the different parts of the net have not reached the same date; for

the construction of the complete finite prefix we will work with temporally consistent states.

A last problem arises: even the age of a token may progress forever. But we will define a *reduced age* for each token in a marking, which gives enough information to know which actions are possible, and remains bounded.

Throughout this section, we use the non trivial local firing condition defined in Section 3.3.5. The existence of the complete finite prefix is shown when all the $efd(t)$ and the $lfd(t)$ in the Petri net are rational numbers. Even the state class graph of [14, 13] is not finite in general when there are irrational numbers.

3.5.1 Equivalence of Two Maximal States

It was already shown in [54] that the age of the tokens can be reduced to bounded values without losing information about the possible future actions.

Definition 3.18 (reduced age of a token). *The* reduced age $J_S(p)$ *of the token* $p \in M$ *in the maximal state* $S \stackrel{\text{def}}{=} (M, dob, lrd)$ *as:*

$$J_S(p) \stackrel{\text{def}}{=} \min\{I_S(p), \max\{bound(t) \mid t \in T \wedge p \in {}^\bullet t\}\}$$

where $bound(t) \stackrel{\text{def}}{=} \begin{cases} efd(t) & \text{if} \quad lfd(t) = \infty \\ lfd(t) & \text{otherwise.} \end{cases}$

Definition 3.19 (equivalence of two maximal states). *Two maximal states* $S_1 \stackrel{\text{def}}{=} (M_1, dob_1, lrd_1)$ *and* $S_2 \stackrel{\text{def}}{=} (M_2, dob_2, lrd_2)$ *are* equivalent *(denoted* $S_1 \sim S_2$*) iff* $M_1 = M_2$ *and* $J_{S_1} = J_{S_2}$*.*

Theorem 3.8 (firing a transition from two equivalent consistent states). *Let* S_1 *and* S_2 *be two equivalent consistent states. Let* M *be their marking. A transition* t *can fire from* S_1 *at date* $\theta_1 \geq \max_{p \in M} lrd_1(p)$ *using the partial marking* $L \subseteq M$ *iff it can fire from* S_2 *at date* $\theta_1 - \max_{p \in M} lrd_1(p) + \max_{p \in M} lrd_2(p)$ *using the same partial marking* L*.*

Proof. Let $(M, dob_i, lrd_i) \stackrel{\text{def}}{=} S_i$ and $\theta'_i \stackrel{\text{def}}{=} \max_{p \in M} lrd_i(p)$ for $i \in \{1, 2\}$. Assume that t can fire from S_1 at date $\theta_1 \geq \max_{p \in M} lrd_1(p)$ using the partial marking $L \subseteq M$. To prove that t can fire from S_2 at date $\theta_2 \stackrel{\text{def}}{=} \theta_1 - \theta'_1 + \theta'_2$ using the same partial marking L, we will show that:

1. $\theta_2 \geq doe_2(t) + efd(t)$, with $doe_2(t) \stackrel{\text{def}}{=} \max_{p \in {}^\bullet t} dob_2(p)$,
2. $\forall t' \in T \quad \left\{ \begin{array}{l} {}^\bullet t' \cap L \neq \emptyset \\ \nexists p \in {}^\bullet t' \quad \bar{p} \cap L \neq \emptyset \end{array} \right\} \implies \theta_2 \leq \max\limits_{p \in {}^\bullet t' \cap L} dob_2(p) + lfd(t')$.

Here are the proofs for these two points:

1. If $\min_{p\in{}^\bullet t} J_{S_1}(p) \geq efd(t)$, then $\min_{p\in{}^\bullet t} I_{S_2}(p) \geq \min_{p\in{}^\bullet t} J_{S_2}(p) = \min_{p\in{}^\bullet t} J_{S_1}(p) \geq efd(t)$ and $\theta_1 \geq \theta'_1$ implies that $\theta_2 \geq \theta'_2$. Furthermore $\theta'_2 = doe_2(t) + \min_{p\in{}^\bullet t} I_{S_2}(p) \geq doe_2(t) + efd(t)$.
Otherwise, (if $\min_{p\in{}^\bullet t} J_{S_1}(p) < efd(t)$), then $\theta_1 \geq doe_1(t) + efd(t) = \theta'_1 - \min_{p\in{}^\bullet t} I_{S_1}(p) + efd(t)$. And $\min_{p\in{}^\bullet t} I_{S_1}(p) = \min_{p\in{}^\bullet t} I_{S_2}(p)$ because for all p such that $I_{S_1}(p) < efd(t)$, $I_{S_1}(p) = J_{S_1}(p) = J_{S_2}(p) = I_{S_2}(p)$ (the equalities between $I_{S_i}(p)$ and $J_{S_i}(p)$ hold since one of them is strictly smaller than $efd(t)$), and for all p such that $I_{S_1}(p) \geq efd(t)$, $I_{S_2}(p) \geq J_{S_2}(p) = J_{S_1}(p) \geq efd(t)$. Thus $\theta_2 \geq \theta'_2 - \min_{p\in{}^\bullet t} I_{S_2}(p) + efd(t) = doe_2(t) + efd(t)$.

2. Let $t' \in T$ such that ${}^\bullet t' \cap L \neq \emptyset$ and $(\nexists p \in {}^\bullet t' \quad \bar{p} \cap L \neq \emptyset)$. Then $\theta_1 \leq \max_{p\in{}^\bullet t'\cap L} dob_1(p) + lfd(t')$.
If $lfd(t') = \infty$ then $\max_{p\in{}^\bullet t'\cap L} dob_2(p) + lfd(t') = \infty \geq \theta_2$.
Otherwise for $i \in \{1, 2\}$, $\max_{p\in{}^\bullet t'\cap L} dob_i(p) = \theta'_i - \min_{p\in{}^\bullet t'\cap L} I_{S_i}(p)$. Then $\theta'_1 \leq \theta_1 \leq \theta'_1 - \min_{p\in{}^\bullet t'\cap L} I_{S_1}(p) + lfd(t')$, and consequently $\min_{p\in{}^\bullet t'\cap L} I_{S_1}(p) \leq lfd(t')$. So $\min_{p\in{}^\bullet t'\cap L} I_{S_2}(p) \geq \min_{p\in{}^\bullet t'\cap L} I_{S_1}(p)$ because for all p such that $I_{S_1}(p) \leq lfd(t')$, $I_{S_1}(p) = J_{S_1}(p) = J_{S_2}(p) \leq I_{S_2}(p)$, and for all p such that $I_{S_1}(p) > lfd(t')$, $I_{S_2}(p) \geq J_{S_2}(p) = J_{S_1}(p) \geq lfd(t')$. Thus $\theta_2 \leq \theta'_2 - \min_{p\in{}^\bullet t'\cap L} I_{S_2}(p) + lfd(t') = \max_{p\in{}^\bullet t'\cap L} dob_2(p) + lfd(t')$.

□

3.5.2 Substitution of Prefixes in Extended Processes

Knowing that the same actions are possible from equivalent consistent states, if we have two temporally complete extended processes that reach equivalent states, we can translate any continuation of one extended process to the other, providing we also translate the firing dates of the events. This operation is illustrated in Figure 3.6. It corresponds intuitively to merging the final conditions of the first extended process with the conditions from which the extension starts in the second process.

First we need to define the relation $\sqsubseteq$ between temporally complete extended processes. Intuitively we expect that $\dot{\mathcal{E}} \sqsubseteq \dot{\mathcal{E}}'$ means that $\dot{\mathcal{E}}$ can be continued to $\dot{\mathcal{E}}'$. In the framework of Petri nets, $\sqsubseteq$ would simply be the set inclusion. This remains true if we deal with symbolic processes of colored Petri nets. But the set inclusion is not satisfactory any more as soon as we deal either with read arcs or with time.

With time the problem is simple: an event of $\dot{\mathcal{E}}'$ may have a smaller firing date than an event of $\dot{\mathcal{E}}$.

But this problem arises even with processes of low-level Petri nets with read arcs (see Section 1.4.3), the reason is that the conditional causality does not respect the set inclusion. Even if $E \subseteq E'$, there may be two events $e \in E$ and $e' \in E'$ such that $e' \nearrow e$. This occurs if e' reads a condition that is consumed by e. As a consequence, when E has been executed, it is too late to add e'.

Definition 3.20. *Let $\dot{\mathcal{E}}$ and $\dot{\mathcal{E}}'$ be two temporally complete extended processes.*

$$\dot{\mathcal{E}} \sqsubseteq \dot{\mathcal{E}}' \quad \textit{iff} \quad \begin{cases} \dot{\mathcal{E}} \subseteq \dot{\mathcal{E}}' \\ \forall \dot{e} \in \dot{E} \quad \forall \dot{e}' \in \dot{E}' \setminus \dot{E} \quad \begin{cases} \dot{\mathcal{E}}(\dot{e}) \leq \dot{\mathcal{E}}'(\dot{e}') \\ \neg(\dot{e}' \nearrow \dot{e}) \end{cases} \end{cases}$$

The relation $\sqsubseteq$ could also be defined using sequences of local firings as: $\dot{\mathcal{E}} \sqsubseteq \dot{\mathcal{E}}'$ iff there exists $\dot{\sigma}$ and $\dot{\sigma}'$ such that $\dot{\Pi}(\dot{\sigma}) = \dot{\mathcal{E}}$ and $\dot{\Pi}(\dot{\sigma}') = \dot{\mathcal{E}}'$ and $\dot{\sigma}$ is a prefix of $\dot{\sigma}'$.

Actually we notice that if $\dot{\mathcal{E}}'$ is temporally complete, and $\dot{\mathcal{E}}$ satisfies the formula, then $\dot{\mathcal{E}}$ is necessarily temporally complete.

Definition 3.21 (substitution of prefixes in temporally complete extended processes). *Let $\dot{\mathcal{E}}_1$ and $\dot{\mathcal{E}}_2$ be two temporally complete extended processes, and $\dot{E}'_2 \subseteq \dot{E}_2$ such that $\dot{\mathcal{E}}_{2|\dot{E}'_2} \sqsubseteq \dot{\mathcal{E}}_2$ and $RS(\dot{\mathcal{E}}_{2|\dot{E}'_2}) \sim RS(\dot{\mathcal{E}}_1)$.*

We define the substitution *which replaces $\dot{\mathcal{E}}_{2|\dot{E}'_2}$ by $\dot{\mathcal{E}}_1$ in $\dot{\mathcal{E}}_2$ as:*

$$subst(\dot{\mathcal{E}}_1, \dot{E}'_2, \dot{\mathcal{E}}_2) \stackrel{\text{def}}{=} \dot{\mathcal{E}}_1 \cup \{(\phi(\dot{e}), \dot{\mathcal{E}}_2(\dot{e}) - \theta'_2 + \theta'_1) \mid \dot{e} \in \dot{E}_2 \setminus \dot{E}'_2\}$$

where

$$\begin{array}{l} \theta'_1 \stackrel{\text{def}}{=} \max_{\dot{f} \in \dot{E}_1} \dot{\mathcal{E}}_1(\dot{f}) \\ \theta'_2 \stackrel{\text{def}}{=} \max_{\dot{f} \in \dot{E}'_2} \dot{\mathcal{E}}_2(\dot{f}) \\ \forall \dot{e} \stackrel{\text{def}}{=} (B, t) \in \dot{E}_2 \setminus \dot{E}'_2 \quad \phi(\dot{e}) \stackrel{\text{def}}{=} (\psi(B), t) \\ \forall b \stackrel{\text{def}}{=} (\dot{e}, p) \in \bigcup_{\dot{e} \in \dot{E}_2 \setminus \dot{E}'_2} {}^\bullet\dot{e} \cup \underline{\dot{e}} \quad \psi(b) \stackrel{\text{def}}{=} \begin{cases} (\phi(\dot{e}), p) & \textit{if } \dot{e} \notin \dot{E}'_2 \\ place^{-1}_{|\uparrow(\dot{E}_1)}(p) & \textit{if } \dot{e} \in \dot{E}'_2 \end{cases} \end{array}$$

We generalize this notation to more than two extended processes as:

$$subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_n, \dot{\mathcal{E}}_n) \stackrel{\text{def}}{=} subst(subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_{n-1}, \dot{\mathcal{E}}_{n-1}), \dot{E}'_n, \dot{\mathcal{E}}_n).$$

Theorem 3.9 ($\dot{Y}$ is closed under substitution of prefixes). *Let $\dot{\mathcal{E}}_0, \ldots, \dot{\mathcal{E}}_m \in \dot{Y}$ and $\dot{\mathcal{E}}'_1 \subseteq \dot{\mathcal{E}}_1, \ldots, \dot{\mathcal{E}}'_m \subseteq \dot{\mathcal{E}}_m$ such that for all $i \in \{1, \ldots, m\}$, $RS(\dot{\mathcal{E}}_{i-1}) \sim RS(\dot{\mathcal{E}}_{i|\dot{E}'_i})$. Then $subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_m, \dot{\mathcal{E}}_m) \in \dot{Y}$.*

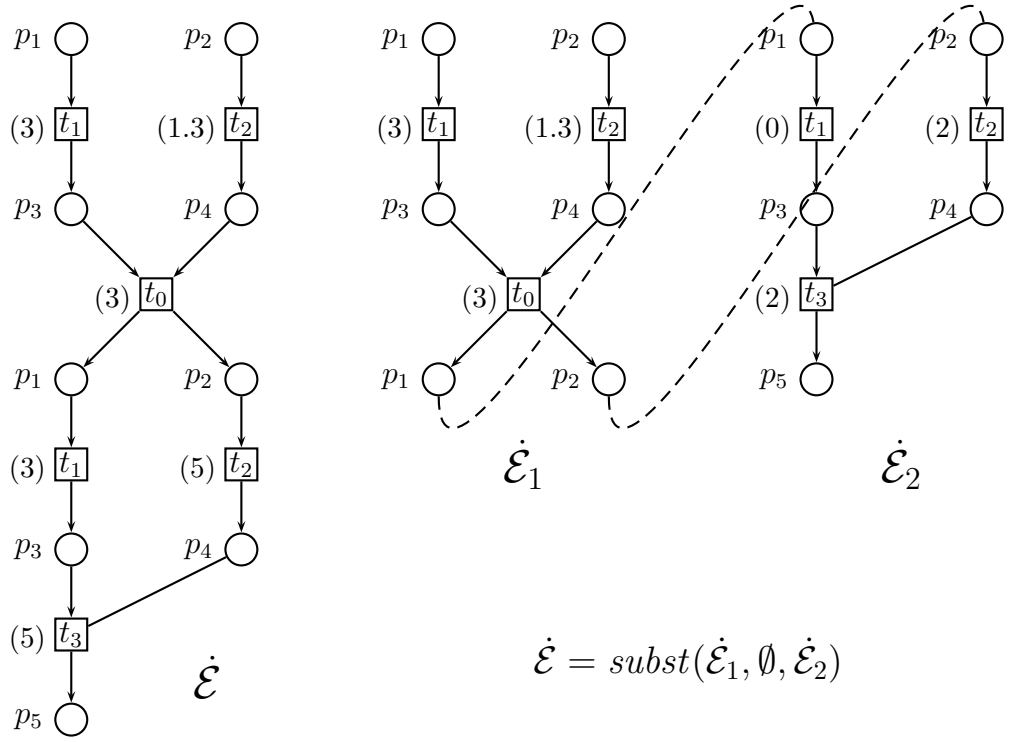


Figure 3.6: Substitution of prefixes in extended processes. The dashed curves show how the final conditions of $\dot{\mathcal{E}}_1$ are merged with the conditions of $\uparrow(\emptyset)$ in $\dot{\mathcal{E}}_2$.

Proof. We detail the proof for the substitution $subst(\dot{\mathcal{E}}_1, \dot{E}'_2, \dot{\mathcal{E}}_2)$; the case of more than two extended processes follows immediately.
If $\dot{E}'_2 = \dot{E}_2$ then $subst(\dot{\mathcal{E}}_1, \dot{E}'_2, \dot{\mathcal{E}}_2) = \dot{\mathcal{E}}_1 \in \dot{Y}$. Now assume that the theorem is true when $\dot{E}_2 \setminus \dot{E}'_2$ has n elements and consider the case where there are $n+1$ elements. Choose an extended event $\dot{e}_2 \stackrel{\text{def}}{=} (B, t) \in \dot{E}_2 \setminus \dot{E}'_2$ that is minimal in $\dot{E}_2 \setminus \dot{E}'_2$ w.r.t. causality ($\nearrow$) and temporal ordering by $\dot{\mathcal{E}}_2$. Theorem 3.8 says that t can fire from $RS(\dot{\mathcal{E}}_1)$ at date $\theta_1 \stackrel{\text{def}}{=} \dot{\mathcal{E}}_2(\dot{e}_2) - \max_{\dot{e} \in \dot{E}'_2} \dot{\mathcal{E}}_2(\dot{e}) + \max_{\dot{e} \in \dot{E}'_1} \dot{\mathcal{E}}_1(\dot{e})$ using the same partial marking $L \stackrel{\text{def}}{=} Place(B)$. This firing can be coded by the extended event $\dot{e}_1 \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(\dot{E}_1)}(L), t)$, and $\dot{\mathcal{E}}'_1 \stackrel{\text{def}}{=} \dot{\mathcal{E}}_1 \cup \{(\dot{e}_1, \theta_1)\} \in \dot{Y}$. Moreover $subst(\dot{\mathcal{E}}_1, \dot{E}'_2, \dot{\mathcal{E}}_2)$ equals $subst(\dot{\mathcal{E}}'_1, \dot{E}'_2 \cup \{\dot{e}_2\}, \dot{\mathcal{E}}_2)$, which belongs to $\dot{Y}$ because $\dot{E}_2 \setminus (\dot{E}'_2 \cup \{\dot{e}_2\})$ has n elements. □

3.5.3 Study of the Form of the Constraints

Now we have to deal with the fact that the unfolding we have defined is symbolic, and thus each event represents an action that may occur at several dates. We will show how to check that all the actions that are possible after a symbolic extended process are possible after another one. For this we have to take into account all the possible values for the date of the events of the symbolic extended processes. As well as Berthomieu defined a finite graph of symbolic state classes in [13] using the interleaving semantics, we show that the set of possible reduced ages after a symbolic extended process ranges over a finite set, which allows us to define a complete finite prefix of the symbolic unfolding of a time Petri net.

Definition 3.22 (constraints $predJ(\dot{E})$). *Let $\dot{E}$ be a finite, causally closed, conflict free set of extended events such that $\nearrow$ is acyclic on $\dot{E}$. Such a set $\dot{E}$ is called a* configuration*, and according to Theorem 3.6, it satisfies the structural properties that would allow a mapping $\dot{\mathcal{E}} : \dot{E} \longrightarrow \mathbb{R}$ to be an extended process, provided the additional constraints on the dates of the events are also satisfied. Denote $M \stackrel{\text{def}}{=} Place(\uparrow(\dot{E}))$. We define the predicate $predJ(\dot{E})$ as follows: for all $J : M \longrightarrow \mathbb{R}$, $predJ(\dot{E})(J)$ holds iff there exists $\dot{\mathcal{E}} : \dot{E} \to \mathbb{R}$ such that $\dot{\mathcal{E}} \in \dot{Y}$ and $J = J_{RS(\dot{\mathcal{E}})}$.*

Theorem 3.10. *If all the $efd(t)$ and the $lfd(t)$ in the Petri net are rational numbers, then for each maximal marking M, the set of the $predJ(\dot{E})$ with $M \stackrel{\text{def}}{=} Place(\uparrow(\dot{E}))$ is finite.*

Proof. Recall that $predJ(\dot{E})$ is a predicate on the $J(p)$, $p \in M$. If we denote $\dot{e}_1, \ldots, \dot{e}_n$ the events of $\dot{E}$ and introduce a variable θ to represent $\max_{\dot{e} \in \dot{E}} \dot{\mathcal{E}}(\dot{e})$, which plays a role in the definition of $J_{RS(\dot{\mathcal{E}})}$ and also in checking that $\dot{\mathcal{E}}$ is temporally complete, we can write $predJ(\dot{E})(J)$ as:

$$\exists \dot{\mathcal{E}}(\dot{e}_1), \ldots, \dot{\mathcal{E}}(\dot{e}_n), \theta \in \mathbb{R} \text{ such that}$$
$$\begin{cases} \dot{\mathcal{E}} \in \dot{X} \\ \theta = \max_{\dot{e} \in \dot{E}} \dot{\mathcal{E}}(\dot{e}) \\ \forall t \in T \quad {}^\bullet t \subseteq M \implies \theta \leq \max_{p \in {}^\bullet t} \dot{\mathcal{E}}({}^\bullet(place^{-1}_{|\uparrow(\dot{E})}(p))) + lfd(t) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{(to check that } \dot{\mathcal{E}} \text{ is temporally complete)} \\ \forall p \in M \quad J(p) = \min\{I(p), \max\{bound(t) \mid t \in T \wedge p \in {}^\bullet t\}\} \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{(with } I(p) \stackrel{\text{def}}{=} \theta - \dot{\mathcal{E}}({}^\bullet(place^{-1}_{|\uparrow(\dot{E})}(p)))) \end{cases}$$

Consider the system in the curly brace and rewrite all the quantifiers that concern information about the structure of the time Petri net or the structure of $\dot{E}$ (including those coming from $(\dot{\mathcal{E}} \in \dot{X})$) as disjunctions or conjunctions.

For instance $(\exists p \in {}^\bullet t_0 \quad f(p))$ becomes $(f(p_3) \vee f(p_4))$. The result is a system of inequalities, some of which containing one "min" or one "max". These inequalities can be rewritten without "min" and "max", so that we obtain a Boolean combination of inequalities of the following types:

$$\dot{\mathcal{E}}(\dot{e}) \# \dot{\mathcal{E}}(\dot{e}') + c \qquad \dot{\mathcal{E}}(\dot{e}) \# \theta + c \qquad \dot{\mathcal{E}}(\dot{e}) \# \theta - J(p)$$

where $\dot{e}$ and $\dot{e}'$ are events of $\dot{E}$, p is a place, c is a constant taken among the $efd(t)$ and $lfd(t)$ and $\#$ is an operator in $\{<, \leq, \geq, >\}$ ($=$ is not necessary).

Rewrite now this Boolean combination of inequalities in normal disjunctive form. The quantifiers $(\exists \dot{\mathcal{E}}(\dot{e}_1), \ldots, \dot{\mathcal{E}}(\dot{e}_n), \theta \in \mathbb{R})$ can be distributed in each term of the disjunction. $predJ(\dot{E})(J)$ becomes a disjunction of quantified conjunctions of inequalities of the types we have described before. We will now eliminate one by one the quantifiers $\exists \dot{\mathcal{E}}(\dot{e}_i)$ in one quantified conjunction of inequalities: we show that there remains a quantified conjunction of inequalities of a slightly more general type than before:

$$\begin{array}{ll} \dot{\mathcal{E}}(\dot{e}) \# \dot{\mathcal{E}}(\dot{e}') + c & \\ \dot{\mathcal{E}}(\dot{e}) \# \theta + c & J(p) \# c \\ \dot{\mathcal{E}}(\dot{e}) \# \theta - J(p) + c & J(p) \# J(p') + c \end{array}$$

where the constants c may now be linear combinations of the $efd(t)$ and $lfd(t)$. To eliminate $\dot{\mathcal{E}}(\dot{e}_i)$, we isolate it in each inequality when it appears, which leads to a conjunction C of $a < \dot{\mathcal{E}}(\dot{e}_i)$, $b \leq \dot{\mathcal{E}}(\dot{e}_i)$, $c \geq \dot{\mathcal{E}}(\dot{e}_i)$ and $d > \dot{\mathcal{E}}(\dot{e}_i)$, where a, b, c and d are terms of the form $(\dot{\mathcal{E}}(\dot{e}) + c)$, $(\theta + c)$ or $(\theta - J(p) + c)$. Then $(\exists \dot{\mathcal{E}}(\dot{e}_i) \quad C)$ is equivalent to the conjunction of all the inequalities $(a < c)$, $(a < d)$, $(b \leq c)$ and $(b < d)$, which all have one of the expected forms.

Once all the $\dot{\mathcal{E}}(\dot{e}_i)$ have been eliminated, the remaining inequalities can be only of the form $J(p) \# c$ or $J(p) \# J(p') + c$. That is, θ does not appear any more. So the quantifier $\exists \theta$ can be removed.

Notice now that by definition $0 \leq J(p) \leq \max_p \stackrel{\text{def}}{=} \max\{bound(t) \mid t \in T \wedge p \in {}^\bullet t\}$ for all $p \in M$. Therefore all the inequalities of type $(J(p) \# c)$ with $|c| > \max_p$ can be immediately evaluated to **true** or **false**. The same happens for the inequalities of the form $(J(p) \# J(p') + c)$ with $|c| > \max\{\max_p, \max_{p'}\}$. The constants c that remain are bounded. Recall also that they are linear combinations of the $efd(t)$ and $lfd(t)$. Since the $efd(t)$ and $lfd(t)$ are rationals, there are finitely many acceptable values for the constants c.

As a consequence, there are finitely many relevant inequalities, and then finitely many conjunctions of such inequalities, and then finitely many disjunctions of such conjunctions. Finally there is a finite number of $predJ(\dot{E})$. □

Let us take the example of $predJ(\{\bot, \dot{e}_1\})(J)$ using the extended events that are represented in Figure 3.7. It can be written, after some simplifications, as:

$$\exists \dot{\mathcal{E}}(\bot), \dot{\mathcal{E}}(\dot{e}_1), \theta \quad \begin{cases} \dot{\mathcal{E}}(\dot{e}_1) \geq \dot{\mathcal{E}}(\bot) \\ \theta = \max\{\dot{\mathcal{E}}(\bot), \dot{\mathcal{E}}(\dot{e}_1)\} \\ \theta - \dot{\mathcal{E}}(\dot{e}_1) \leq 2 \\ \theta - \dot{\mathcal{E}}(\bot) \leq 2 \\ J(p_2) = \min\{\theta - \dot{\mathcal{E}}(\bot), 2\} \\ J(p_3) = \min\{\theta - \dot{\mathcal{E}}(\dot{e}_1), 2\} \end{cases}$$

These inequalities can be rewritten without "min" and "max", so that we obtain a Boolean combination of inequalities, which can be written in normal disjunctive form. Then the quantified variables can be eliminated one by one, and we obtain: $J(p_3) = 0 \ \wedge \ 0 \leq J(p_2) \leq 2$.

3.5.4 Complete Finite Prefix

Now we can define the complete finite prefix of the symbolic unfolding of a time Petri net, by keeping only a finite number of extended events, that contain all the information about the unfolding. More precisely, we show that every temporally complete extended process can be obtained by substitution of prefixes in extended processes that belong to the prefix. Figure 3.6 shows an example of such a decomposition. The complete finite prefix is represented in Figure 3.7.

The idea behind the construction of the prefix is that a temporally complete extended process $\dot{\mathcal{E}}$ will not be continued if the predicate $predJ(\dot{E})$ is equal to a $predJ(\dot{E}')$ with $|\dot{E}'| < |\dot{E}|$.

In fact we can improve this definition by requiring only that $predJ(\dot{E}) \implies predJ(\dot{E}')$, that is for all J that satisfies $predJ(\dot{E})$, J satisfies $predJ(\dot{E}')$.

As another possible improvement, the idea of parameterizing the order between the configurations, like in [46] with adequate orders, seems to be usable in our framework.

Definition 3.23 ($\bar{Y}$ and complete finite prefix $\bar{U}$). *We define the subset $\bar{Y}$ of $\dot{Y}$ as:*

$$\dot{\mathcal{E}} \in \bar{Y} \quad \textit{iff} \quad \exists \sigma \quad \begin{cases} \dot{\Pi}(\sigma) = \dot{\mathcal{E}} \\ \nexists n', \sigma'' \quad |\sigma''| < n' < |\sigma| \ \wedge \ RS(\dot{\Pi}(\sigma'')) \sim RS(\dot{\Pi}(\sigma')) \ . \end{cases}$$

We define the finite complete prefix $\bar{U}$ *of the symbolic unfolding U of a time Petri net by collecting all the extended events that appear in the temporally complete extended processes of $\bar{Y}$: $\bar{U} \stackrel{\text{def}}{=} \bigcup_{\dot{\mathcal{E}} \in \bar{Y}} \dot{E}$.*

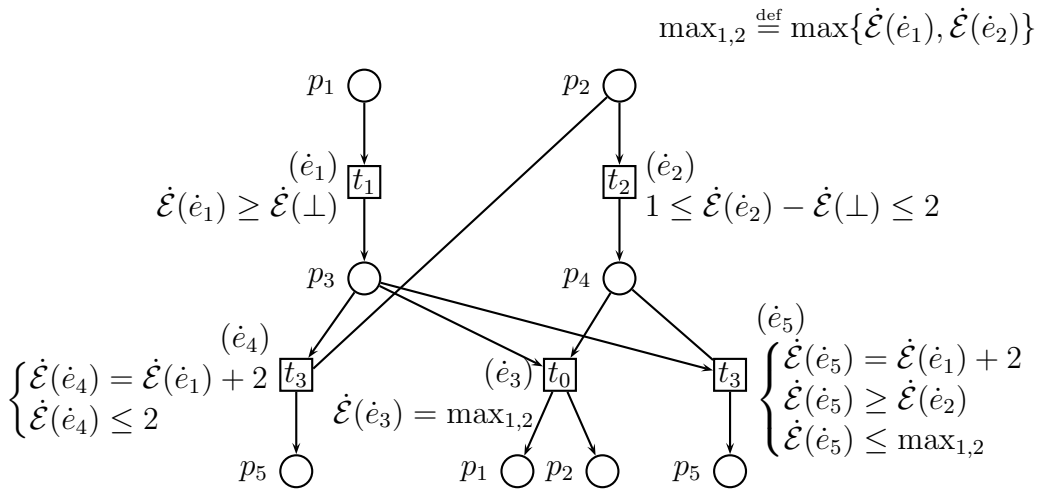


Figure 3.7: The complete finite prefix of the symbolic unfolding of the time Petri net of Figure 3.1. The predicate $LFC(Place(B), dob_{\dot{\mathcal{E}}_{\lfloor\lceil B\rceil}}, t, \dot{\mathcal{E}}(\dot{e}))$ is written near each extended event $\dot{e} \stackrel{\text{def}}{=} (B, t)$.

Denote N the cardinality of $\{predJ(\dot{E}) \mid \dot{\mathcal{E}} \in \bar{Y}\}$, which was proved finite in Section 3.5.3. The extended process $\dot{\Pi}(\sigma)$ may belong to $\bar{Y}$ only if $|\sigma| < N$. Since there is a finite number of sequences of local firings that are shorter than N, there is a finite number of extended processes in $\bar{Y}$ and each of them has less than N events (without $\perp$). Thus $\bar{U}$ is finite.

Theorem 3.11 (decomposition of a temporally complete extended process in $\bar{U}$). *For every temporally complete extended process $\dot{\mathcal{E}} \in \dot{Y}$, there exist temporally complete extended processes $\dot{\mathcal{E}}_0, \ldots, \dot{\mathcal{E}}_n \in \bar{Y}$ and $\dot{\mathcal{E}}'_1 \subseteq \dot{\mathcal{E}}_1, \ldots, \dot{\mathcal{E}}'_m \subseteq \dot{\mathcal{E}}_m$ such that $\dot{\mathcal{E}} = subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_m, \dot{\mathcal{E}}_m)$.*

Proof. To build the substitution we take the extended events in a total order that respects causality and temporal ordering. Each time we add an event, we try to add it at the end of the last extended process ($\dot{\mathcal{E}}_m$) if it remains in $\bar{Y}$. Otherwise we add an extended process $\dot{\mathcal{E}}_{m+1}$ in the substitution. More precisely, let $\dot{\mathcal{E}} \in \dot{Y}$, $\dot{e} \stackrel{\text{def}}{=} (B, t) \in \dot{E}$ a maximal extended event in $\dot{E}$ w.r.t. causality ($\nearrow$) and temporal ordering by $\dot{\mathcal{E}}$, and assume that there exist $\dot{\mathcal{E}}_0, \ldots, \dot{\mathcal{E}}_m \in \bar{Y}$ and $\dot{E}'_1, \ldots, \dot{E}'_m$ such that $\dot{\mathcal{E}}_{|\dot{E}\setminus\{\dot{e}\}} = subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_m, \dot{\mathcal{E}}_m)$. Let $\dot{\mathcal{E}}'_m \stackrel{\text{def}}{=} subst(\dot{\mathcal{E}}_m, \dot{E} \setminus \{\dot{e}\}, \dot{\mathcal{E}})$ and $\dot{e}' \stackrel{\text{def}}{=} \phi(\dot{e})$ in the substitution, so that $\dot{\mathcal{E}}'_m = \dot{\mathcal{E}}_m \cup \{(\dot{e}', \dot{\mathcal{E}}'_m(\dot{e}'))\}$. If $\dot{\mathcal{E}}'_m \in \bar{Y}$ then $subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_m, \dot{\mathcal{E}}'_m)$ fits. Otherwise let $\dot{F} \stackrel{\text{def}}{=} \dot{E}_m \cup \{\dot{e}'\}$ and find a sequence of local firings $\sigma_n \stackrel{\text{def}}{=} ((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n))$ such

that $\dot{\Pi}(\sigma_n) = \dot{\mathcal{E}}_m$. Let $t_{n+1} \stackrel{\text{def}}{=} t$, $L_{n+1} \stackrel{\text{def}}{=} B$, and $\theta_{n+1} \stackrel{\text{def}}{=} \dot{\mathcal{E}}'_m(\dot{e}')$ so that $\dot{\mathcal{E}}'_m = \dot{\Pi}((t_1, L_1, \theta_1), \ldots, (t_{n+1}, L_{n+1}, \theta_{n+1}))$. There exist σ'' and $n' < n+1$ such that $|\sigma''| < n'$ and $RS(\dot{\Pi}(\sigma'')) \sim RS(\dot{\Pi}(\sigma'))$. Let $\dot{F}'$ (respectively $\dot{F}''$) be the set of events that appear in $\dot{\Pi}((t_1, L_1, \theta_1), \ldots, (t_{n'}, L_{n'}, \theta_{n'}))$ (respectively $\dot{\Pi}(\sigma'')$). It holds that $n' = n$ since $\dot{\mathcal{E}}_m \in \bar{Y}$. Choose σ'' of minimal length so that $\dot{\mathcal{E}}_{m+1} \stackrel{\text{def}}{=} \dot{\Pi}(\sigma'' \cdot (t_{n+1}, L_{n+1}, \theta_{n+1})) \in \bar{Y}$. And $\dot{\mathcal{E}} = subst(\dot{\mathcal{E}}_0, \dot{E}'_1, \dot{\mathcal{E}}_1, \ldots, \dot{E}'_m, \dot{\mathcal{E}}_m, \dot{F}'', \dot{\mathcal{E}}_{m+1})$. □

3.6 Colored Time Petri Nets

The idea of combining temporal aspects and high-level features like colors arises naturally and can be found for example in [52]. We have already addressed symbolic unfoldings of high-level Petri nets in Chapter 2. The purpose of this section is to show that our method based on the definition of a concurrent operational semantics for time Petri nets, can be applied to more general timed true concurrency models and that it is compatible with symbolic unfoldings of high-level models. It seems to give a general method to deal with the difficulties that arise when we try to define unfoldings of timed true concurrency models with urgency and confusion.

3.6.1 Definition

We define a model of colored time Petri nets which is convenient for the definition of a concurrent operational semantics and of timed unfoldings.

We have deliberately chosen to highlight the cause of urgency by coding it in the function ω. By contrast, in time Petri nets, the earliest and latest firing delays seem to play a symmetrical role, but only the latest firing delays are responsible for the urgency and then for the difficulties in the unfoldings.

In the states of a colored time Petri net, we do not code explicitly the date of birth of each token. But when a transition fires, the value of the tokens that are created may depend on the firing date. Thus it is possible to code the date of birth of a token in its value. We did not choose to attach this information to all the tokens because in practice it is common that only a few tokens are involved in the time constraints.

A *colored time Petri net* is a tuple

$$(P, T, V, PAR, pre, cont, post, \alpha, \beta, \omega, S_0)$$

where

- P is a set of *places*;
- T is a set of *transitions*;
- V is a set of *values* (or *colors*);
- PAR is a set of *parameters*;
- *pre*, *cont* and *post* map each transition $t \in T$ to the sets of places ${}^\bullet t \stackrel{\text{def}}{=} pre(t) \in 2^P \setminus \{\emptyset\}$, $\underline{t} \stackrel{\text{def}}{=} cont(t) \in 2^P$ and $t^\bullet \stackrel{\text{def}}{=} post(t) \in 2^P$; we denote ${}^\bullet\underline{t} \stackrel{\text{def}}{=} {}^\bullet t \cup \underline{t}$;
- α maps each transition $t \in T$ to a *guard* $\alpha(t) : \mathbb{R} \times PAR \times ({}^\bullet\underline{t} \longrightarrow V) \longrightarrow \mathbf{bool}$;
- β assigns to each transition $t \in T$ a mapping $\beta(t) : \mathbb{R} \times PAR \times ({}^\bullet\underline{t} \longrightarrow V) \longrightarrow (t^\bullet \longrightarrow V)$ that defines the values of the tokens that are created when t fires;
- ω assigns to each transition $t \in T$ a mapping $\omega(t) : ({}^\bullet\underline{t} \longrightarrow V) \longrightarrow \mathbb{R}$ which gives the latest firing date of t;
- $S_0 = (M_0, v_0, \theta_0)$ is the *initial state*; a *state* is a triple (M, v, θ), where $M \subseteq P$ is the *marking*, $\theta \in \mathbb{R}$ the date and $v : M \longrightarrow V$ maps each place $p \in M$ to the value $v(p)$ of the token which stands in it.

3.6.2 Interleaving Semantics

A transition $t \in T$ is *enabled* in a state (M, v, θ) if ${}^\bullet\underline{t} \subseteq M$. The colored time Petri nets that we consider are *safe*, i.e. for all reachable state $S \stackrel{\text{def}}{=} (M, v, \theta)$, if a transition t is enabled in S, then $t^\bullet \cap (M \setminus {}^\bullet t) = \emptyset$.

A transition t can *fire* at time $\theta' \geq \theta$ with parameter $x \in PAR$ from a state (M, v, θ) if:

- t is enabled: ${}^\bullet\underline{t} \subseteq M$;
- the parameter x, the values of the tokens that are consumed or read, and the date θ satisfy the guard of t: $\alpha(t)(\theta', x, v)$;
- the enabled transitions do not overtake the maximum delays: $\forall t' \in T \quad {}^\bullet t' \subseteq M \implies \theta' \leq \omega(t')(v)$.

Firing t at time θ' with parameter x leads to the state $((M \setminus {}^\bullet t) \cup t^\bullet, v', \theta')$, where

$$v'(p) \stackrel{\text{def}}{=} \begin{cases} v(p) & \text{if } p \in M \setminus {}^\bullet t \\ \beta(t)(\theta', x, v)(p) & \text{if } p \in t^\bullet. \end{cases}$$

In *firing sequences* we need the information about the parameters and the dates. Thus the firing sequences are coded as sequences $((t_1, \theta_1, x_1), \ldots, (t_n, \theta_n, x_n))$.

Like in time Petri nets, we assume that time *diverges*. Moreover, in colored time Petri nets, we demand that for all reachable state $S \stackrel{\text{def}}{=} (M, v, \theta)$, if a transition t is enabled in S, then $\omega(t)(v) \geq \theta$.

3.6.3 Example

Figure 3.8 shows a colored time Petri net. We adopt similar graphical conventions as for colored Petri nets. The guard $\alpha(t)(\theta, x, v)$ is written near each transition t, and the value $\beta(t)(\theta, x, v)(p)$ is written on the arc that connects a transition t to a place $p \in t^\bullet$. Moreover the latest firing date is written near each event after "ω :".

Transition t_0 models a failure. In the state of the figure, it has just occurred. The tokens in p_1 and p_2 indicate the date of the last failure. Three steps are needed to repair the failure. They are modeled by the transitions t_1, t_2 and t_3. t_1 takes no more than 5 time units and t_2 takes no more than 8 time units. As t_2 prepares the work of t_3, the duration of t_3 depends on the quality of the preparation. This quality is proportional to the duration of t_2, and it is written on the token in p_5 when t_2 fires. If the quality of the preparation is x, then t_3 takes $4 - x$ time units.

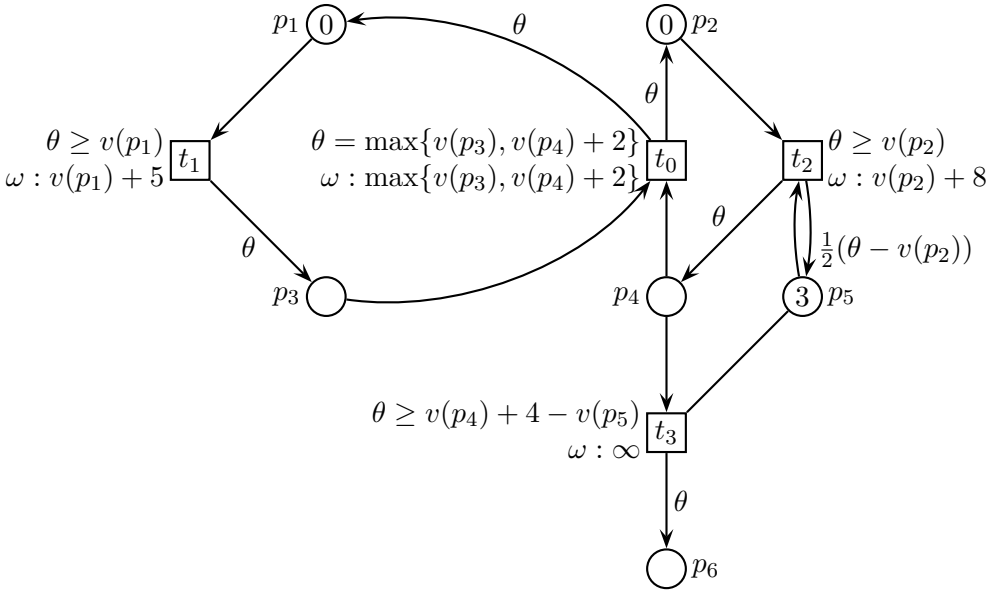


Figure 3.8: A colored time Petri net.

But, as soon as t_2 has fired, the failure (t_0) repeats 2 time units later than the firing of t_2, unless t_3 has already fired. So the repair is complete if t_3 has enough time to fire before the failure repeats. There are two possible choices: either t_1 takes less than 2 time units; then t_3 needs at least 3 time units, but can fire before t_1 fires. Otherwise, t_1 must last at least 4 time units, so that t_3 fires quickly (in less than 2 time units).

3.6.4 Processes

A process $\mathcal{E}$ of a colored time Petri net N is built over a process E of the underlying Petri net with read arcs $Sup(N) \stackrel{\text{def}}{=} (P, T, pre, cont, post, M_0)$.

In the process of N, we need also information about the firing dates of the events and about the parameters. Thus a process $\mathcal{E}$ of N built over the process E of $Sup(N)$, is a set of pairs $(e, (\theta, x))$ such that $\mathcal{E} \in E \longrightarrow (\mathbb{R} \times PAR)$. For all event $e \in E$, it is convenient to define its firing date $\Theta(e)$ and its parameter $\Xi(e)$ such that $\mathcal{E}(e) = (\Theta(e), \Xi(e))$. Each event $e \in E$ is a pair[2] $({}^\bullet\underline{e}, \tau(e))$ which codes an occurrence of the transition $\tau(e)$. The

[2]As explained in Section 1.4.3 in the footnote on page 29, a pair is sufficient since the processes we are dealing with represent safe behaviors.

conditions in ${}^\bullet\underline{e}$ are either read or consumed by e. The set of read conditions is $\underline{e} \stackrel{\text{def}}{=} \{b \in {}^\bullet\underline{e} \mid place(b) \in \underline{\tau(e)}\}$ and the set of consumed conditions is ${}^\bullet e \stackrel{\text{def}}{=} \{b \in {}^\bullet\underline{e} \mid place(b) \in {}^\bullet(\tau(e))\}$. As usual the event e creates the conditions $e^\bullet \stackrel{\text{def}}{=} \{(e,p) \mid p \in (\tau(e))^\bullet\}$. Finally we set $\bot^\bullet \stackrel{\text{def}}{=} \{(\bot,p) \mid p \in M_0\}$ and $\Theta(\bot) \stackrel{\text{def}}{=} \theta_0$.

Definition 3.24 (D_N). *As a result all the extended events of the processes of a colored Petri net N will be members of the set D_N, defined as the smallest set such that*

$$\forall B \subseteq \bigcup_{e \in D_{N\bot}} e^\bullet \quad \forall t \in T \quad Place(B) = {}^\bullet\underline{t} \implies (B,t) \in D_N \ .$$

As the colored time Petri nets that we consider are safe, we define the set X_N (or X) of processes of N by mapping each firing sequence $((t_1,\theta_1,x_1),\ldots,(t_n,\theta_n,x_n))$ to a process:

- $\Pi(\epsilon) \stackrel{\text{def}}{=} \emptyset$
- $\Pi(((t_1,\theta_1,x_1),\ldots,(t_{n+1},\theta_{n+1},x_{n+1}))) \stackrel{\text{def}}{=} \mathcal{E} \cup \{(e,(\theta_n,x_n))\}$, where
  - $\mathcal{E} \stackrel{\text{def}}{=} \Pi(((t_1,\theta_1,x_1),\ldots,(t_n,\theta_n,x_n)))$ and
  - the event $e \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(E)}({}^\bullet\underline{t_{n+1}}), t_{n+1})$ represents the last firing of the sequence,

The causality relations $\rightarrow$ and $\nearrow$ between the events are defined like in all the other cases of processes with read arcs (see Sections 1.4.3, 2.2 and 3.2.1). We reuse also the definition of causal past.

Definition 3.25 ($val_{\mathcal{E}}$ and $tok_{\Theta,\Xi}$). *Like for process of colored Petri nets in Section 2.1.5, we define the value $val_{\mathcal{E}}(b)$ of a condition b as:*

$$val_{\mathcal{E}}((e,p)) \stackrel{\text{def}}{=} \begin{cases} v_0(p) & \text{if } e = \bot \\ \beta(\tau(e))(\Theta(e),\Xi(e),tok_{\Theta,\Xi}({}^\bullet\underline{e})) & \text{otherwise} \end{cases}$$

where for all condition $b \in \bigcup_{e \in E_\bot} e^\bullet$, $tok_{\Theta,\Xi}(b) \stackrel{\text{def}}{=} (place(b), val_{\mathcal{E}}(b))$.

3.6.5 Extension of the Result of [4, 3] to Colored Time Petri Nets

As well as Aura and Lilius gave a way to characterize the processes of a time Petri net [4, 3], we can characterize the processes of a colored time Petri

net. For this we have to check that all the events that were enabled during the process (*even those that did not actually fire*), did not overtake their maximum firing delays.

Theorem 3.12. *Let $E \subseteq D_N$ be a finite set of events and $\mathcal{E} : E \longrightarrow \mathbb{R} \times PAR$. $\mathcal{E}$ is a process iff:*

$$\begin{cases} \lceil E \rceil = E & \textit{(E is causally closed)} \\ \nexists e, e' \in E \quad e \neq e' \ \wedge \ {}^\bullet e \cap {}^\bullet e' \neq \emptyset & \textit{(E is conflict free)} \\ \nexists e_0, e_1, \ldots, e_n \in E \quad e_0 \nearrow e_1 \nearrow \cdots \nearrow e_n \nearrow e_0 & \textit{($\nearrow$ is acyclic on E)} \\ \forall e, e' \in E_\perp \quad e \nearrow e' \implies \Theta(e) \leq \Theta(e') & \textit{(Θ is compatible with $\nearrow$)} \\ \forall e \in E \quad \alpha(\tau(e))(\Theta(e), \Xi(e), tok_{\Theta,\Xi}({}^\bullet \underline{b})) & \textit{(the guard of e is satisfied)} \\ \forall e \in D_N \quad {}^\bullet \underline{e} \subseteq \bigcup_{f \in E_\perp} f^\bullet \implies & \\ \qquad \min\{\theta_{end}, dod(e)\} \leq \omega(\tau(e))(\Theta(e), \Xi(e), tok_{\Theta,\Xi}({}^\bullet \underline{b})) & \\ \textit{(the events that were enabled did not overtake their latest firing dates)} & \end{cases}$$

where $\theta_{end} \stackrel{\text{def}}{=} \max_{f \in E_\perp} \Theta(f)$ is the date the is reached at the end of the process, and $dod(e) \stackrel{\text{def}}{=} \min\{\Theta(f) \mid f \in E \ \wedge \ {}^\bullet f \cap {}^\bullet \underline{e} \neq \emptyset\}$ is the date when e was disabled (because an event f consumed one condition in ${}^\bullet \underline{e}$).

Proof. We first show by induction on X_N that all the processes of N satisfy the condition in the curly brace. For $\mathcal{E} = \emptyset$, only the last line needs some explanation: it holds since $\theta_{\text{end}} = \Theta(\perp) = 0$. Let $\mathcal{E}$ be a process of N that satisfies the condition in the curly brace. Denote (M, v, θ) the state reached after $\mathcal{E}$. Let $t \in T$, $\theta' \geq \theta$ and $x \in PAR$ that satisfy the firing condition. Denote $e' \stackrel{\text{def}}{=} (Place^{-1}_{\uparrow(E)}({}^\bullet \underline{t}), t)$ the event that codes the firing of t, and $\mathcal{E}' \stackrel{\text{def}}{=} \mathcal{E} \cup \{(e', (\theta', x))\}$. The first 5 lines are satisfied by E' (see the proofs of Theorems 1.3 and 2.2). But the condition on the latest firing delays has to be checked even for the events e such that ${}^\bullet \underline{e} \subseteq \bigcup_{f \in E_\perp} f^\bullet$, because θ_{end} is larger after $\mathcal{E}'$ than after $\mathcal{E}$. Let $e \in D_N$ such that ${}^\bullet \underline{e} \subseteq \bigcup_{f \in E'_\perp} f^\bullet$. If ${}^\bullet \underline{e} \cap ({}^\bullet e') \neq \emptyset$, then e was enabled at time $\theta' = \theta_{\text{end}}$, so e does not overtake its latest firing date. Otherwise, if ${}^\bullet \underline{e} \not\subseteq \uparrow(E)$, then e was disabled by an event of E before its latest firing date; this remains true in $\mathcal{E}'$. And if ${}^\bullet \underline{e} \subseteq \uparrow(E)$ then ${}^\bullet \underline{\tau(e)} \subseteq Place(\uparrow(E))$. So $\tau(e)$ is enabled when e' fires, and the firing condition of t guarantees that $\tau(e)$ does not overtake its latest firing date. Then $\theta_{\text{end}} \leq \omega(\tau(e))(\Theta(e), \Xi(e), tok_{\Theta,\Xi}({}^\bullet \underline{b}))$.

Now we show the converse, that is, if the condition in the curly brace is satisfied, then $\mathcal{E}$ is a process. We proceed by induction on E. If $E = \emptyset$, then the only $\mathcal{E} : E \longrightarrow \mathbb{R} \times PAR$ is $\emptyset$, which is a process. If $E \neq \emptyset$, and $\mathcal{E} : E \longrightarrow \mathbb{R} \times PAR$ satisfies the curly brace, then choose an event e of E that is maximal w.r.t. weak causality and temporal ordering ($\nexists e' \in$

$E \quad (e \nearrow e') \vee (\Theta(e) < \Theta(e')))$. $E' \stackrel{\text{def}}{=} E \setminus \{e\}$ satisfies the first 5 lines in the curly brace. $\mathcal{E}_{|E'}$ also satisfies the last line because the $\min\{\theta_{\text{end}}, dod(e)\}$ can only be smaller in $\mathcal{E}_{E'}$ than in $\mathcal{E}$, and fewer events are concerned. Thus $\mathcal{E}_{E'} \in X_N$. Furthermore, the last line but one of the curly brace applied to e, and the last line applied to the e' such that ${}^\bullet\underline{e'} \subseteq \uparrow(E')$, allow $\tau(e)$ to fire at date $\Theta(e)$ with parameter $\Xi(e)$ from the state reached after $\mathcal{E}_{E'}$. That is $\mathcal{E} = \mathcal{E}_{E'} \cup \{(e, (\Theta(e), \Xi(e)))\}$ is a process of N. □

3.6.6 Concurrent Operational Semantics for Colored Time Petri Nets

We define the pre-processes and the prefix relation exactly like in Section 3.2 for time Petri nets, and we make the same assumption as in Section 3.3 about the partition of the set of places in sets of mutually exclusive places.

Definition 3.26 (partial state). *A* partial state *of a colored time Petri net is a triple (L, v, lrd) where $L \subseteq P$ is a* partial marking, *$v : L \longrightarrow V$ and $lrd : L \longrightarrow \mathbb{R}$ assign a value $v(p)$ and a* latest reading date *$lrd(p)$ to each token $p \in L$.*

As well as in Section 3.3 for time Petri nets, a partial state (L, v, lrd) is *maximal* if L contains one place per set of mutually exclusive places.

Definition 3.27 (temporally consistent maximal state (or consistent state)). *A maximal state (M, v, lrd) is* temporally consistent *if for all transition $t \in T$ enabled in M (${}^\bullet\underline{t} \subseteq M$), $\omega(t)(v) \leq \theta$, where $\theta \stackrel{\text{def}}{=} \max_{p \in M} lrd(p)$ is the date that is reached by the system.*

Local Firing Condition

Now we will define a local firing condition. As well as for safe time Petri nets, several local firing conditions are possible. But here we will give directly a non trivial one.

In the context of colored time Petri nets, this condition will be a predicate on 5-tuples (L, v, t, θ, x), where $L \subseteq P$ is a partial marking, $v : L \longrightarrow V$ assigns a value to each token $p \in L$, $t \in T$ is a transition, $\theta \in \mathbb{R}$ is a date and $x \in PAR$ is a parameter.

Definition 3.28 (local firing condition *LFC′*). *We first define a predicate LFC' as follows: $LFC'(L, v, t, \theta, x)$ holds iff:*

- *t is enabled: ${}^\bullet\underline{t} \subseteq L$;*

- *the guard of t is satisfied at time θ with the parameter x: $\alpha(t)(\theta, x, v)$;*
- *the transitions that might consume tokens of L are disabled or can wait until θ:*

$$\forall t' \in T \quad {}^\bullet t' \cap L \neq \emptyset \implies \begin{cases} & \exists p \in {}^\bullet \underline{t'} \quad \bar{p} \cap L \neq \emptyset \\ \vee & \forall v' : {}^\bullet \underline{t'} \longrightarrow V \\ & \qquad v'_{|L} = v_{|{}^\bullet \underline{t'}} \implies \theta \leq \omega(t')(v') \end{cases}$$

The predicate LFC' guarantees that a partial state (L, v, lrd) with $lrd(p) \leq \theta$ for all $p \in L$ gives enough information to be sure that t can fire at date θ with the parameter x. Like for time Petri nets, we define a new local firing condition LFC by demanding that the partial marking L is minimal.

Definition 3.29 (local firing condition *LFC*). *We define LFC by:*

$$LFC(L, v, t, \theta, x) \quad \textit{iff} \quad \begin{cases} (LFC'(L, v, t, \theta, x) \\ \nexists L' \subsetneq L \quad LFC'(L', v_{|L'}, t, \theta, x) \ . \end{cases}$$

Semantics of Local Firings

In the maximal state (M, v, lrd), a transition t can fire at date θ with the parameter x, using the partial marking $L \subseteq M$, if $(L, v_{|L}, t, \theta, x)$ satisfies LFC and for all $p \in L$, $\theta \geq lrd(p)$.

This action leads to the maximal state $((M \setminus {}^\bullet t) \cup t^\bullet, v', lrd')$ with

$$v'(p) \stackrel{\text{def}}{=} \begin{cases} v(p) & \text{if } p \in M \setminus {}^\bullet t \\ \beta(t)(\theta, x, v_{|{}^\bullet \underline{t}}) & \text{if } p \in t^\bullet \end{cases}$$

and

$$lrd'(p) \stackrel{\text{def}}{=} \begin{cases} lrd(p) & \text{if } p \in M \setminus L \\ \theta & \text{if } p \in (L \setminus {}^\bullet t) \cup t^\bullet. \end{cases}$$

In *sequences of local firings* we need the information about the partial marking that is used, the parameters and the dates. Thus the sequences of local firings are coded as sequences $((t_1, L_1, \theta_1, x_1), \ldots, (t_n, L_n, \theta_n, x_n))$.

Extended Processes

Like the processes of colored time Petri nets defined in Section 3.6.4, an extended process $\dot{\mathcal{E}}$ of N is a set of pairs $(\dot{e}, (\theta, x))$ such that $\dot{\mathcal{E}} \in \dot{E} \longrightarrow (\mathbb{R} \times PAR)$, where $\dot{E}$ is the set of extended events of the process. And for all extended event $\dot{e} \in \dot{E}$, we define its firing date $\Theta(\dot{e})$ and its parameter $\Xi(\dot{e})$ such that $\dot{\mathcal{E}}(\dot{e}) = (\Theta(\dot{e}), \Xi(\dot{e}))$.

Like in extended processes of time Petri nets in Section 3.3.2, each extended event $\dot{e} \stackrel{\text{def}}{=} (B, t)$ represents the firing of the transition t from a partial state and keeps track of all the conditions corresponding corresponding to the partial state, not only those corresponding to ${}^\bullet\underline{t}$. We denote $\tau(\dot{e}) \stackrel{\text{def}}{=} t$, ${}^\bullet\underline{e} \stackrel{\text{def}}{=} B$, ${}^\bullet\dot{e} \stackrel{\text{def}}{=} Place^{-1}_{|B}({}^\bullet t)$, $\underline{\dot{e}} \stackrel{\text{def}}{=} B \setminus {}^\bullet e$ and $\dot{e}^\bullet \stackrel{\text{def}}{=} \{(\dot{e}, p) \mid p \in t^\bullet\}$.

Definition 3.30 ($\dot{D}_N$). *In the context of extended processes of a colored time Petri net N, the extended events will be elements of the set $\dot{D}_N$ defined as the smallest set such that:*

$$\forall B \subseteq \bigcup_{\dot{e} \in \dot{D}_{N\perp}} \dot{e}^\bullet \quad \forall t \in T \quad {}^\bullet\underline{t} \subseteq Place(B) \implies (B, t) \in \dot{D}_N \ .$$

The set $\dot{X}$ of extended processes is defined by mapping each sequence of local firings $((t_1, L_1, \theta_1, x_1), \ldots, (t_n, L_n, \theta_n, x_n))$ to an extended process as follows:

- $\dot{\Pi}(\epsilon) \stackrel{\text{def}}{=} \emptyset$,
- $\dot{\Pi}(((t_1, L_1, \theta_1, x_1), \ldots, (t_{n+1}, L_{n+1}, \theta_{n+1}, x_{n+1}))) \stackrel{\text{def}}{=} \dot{\mathcal{E}} \cup \{(\dot{e}, \theta_{n+1}, x_{n+1})\}$, where $\dot{\mathcal{E}} \stackrel{\text{def}}{=} \dot{\Pi}(((t_1, L_1, \theta_1, x_1), \ldots, (t_n, L_n, \theta_n, x_n)))$ and the extended event $\dot{e} \stackrel{\text{def}}{=} (Place^{-1}_{|\uparrow(\dot{E})}(L_{n+1}), t_{n+1})$ represents the last local firing of the sequence.

Definition 3.31 ($val_{\dot{\mathcal{E}}}$, $tok_{\Theta,\Xi}$, $v_{\dot{\mathcal{E}}}$ and $lrd_{\dot{\mathcal{E}}}$). *The value $val_{\dot{\mathcal{E}}}(b)$ of a condition b is defined like for processes (see Definition 3.25), as well as $tok_{\Theta,\Xi}(b) \stackrel{\text{def}}{=} (place(b), val_{\dot{\mathcal{E}}}(b))$.*

Now the value of the token which stands in place p in the state reached after the extended process $\dot{\mathcal{E}}$ is simply

$$v_{\dot{\mathcal{E}}}(p) \stackrel{\text{def}}{=} val_{\dot{\mathcal{E}}}(place^{-1}_{|\uparrow(\dot{E})}(p))) \ .$$

Remark that $v_{\mathcal{E}}$ can be seen as a simple shortcut for $tok_{\Theta,\Xi}(\uparrow(\dot{E}))$.

And, as well as for extended processes of time Petri nets, we define the latest date when the token which stands in place p was read, according to the extended process $\dot{\mathcal{E}}$, as:

$$lrd_{\dot{\mathcal{E}}}(p) \stackrel{\text{def}}{=} \max\{dob_{\dot{\mathcal{E}}}(p), \max_{\dot{e} \in \dot{E},\, b \in \underline{\dot{e}}} \dot{\mathcal{E}}(\dot{e})\} \ .$$

Definition 3.32 ($RS(\dot{\mathcal{E}})$). *The maximal state that is reached after an extended process $\dot{\mathcal{E}}$ is defined as $RS(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} (Place(\uparrow(\dot{E})), v_{\dot{\mathcal{E}}}, lrd_{\dot{\mathcal{E}}})$.*

Definition 3.33 (temporally complete extended process, $\dot{Y}$). *We say that $\dot{\mathcal{E}}$ is* temporally complete *if $RS(\dot{\mathcal{E}})$ is temporally consistent. The set of all temporally complete extended processes is denoted $\dot{Y}$.*

Correctness and Completeness of *LFC*

Each extended event $\dot{e} = (B, t)$ can be mapped to the corresponding event

$$h(\dot{e}) \stackrel{\text{def}}{=} \left(\left\{ (h(\dot{f}), p) \mid (\dot{f}, p) \in B \wedge p \in {}^\bullet\underline{t} \right\}, t \right) .$$

For all extended process $\dot{\mathcal{E}}$, $h(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} \{(h(\dot{e}), \theta) \mid (\dot{e}, \theta) \in \dot{\mathcal{E}}\}$ is the process that we obtain when we remove the read arcs that are induced by the local firing conditions; only those that come from the $\underline{t}$ are kept.

With these notations, the lemma and the two theorems that express the correctness and completeness of our concurrent operational semantics are exactly identical to Lemma 3.1 and Theorems 3.4 and 3.5 for time Petri nets.

Lemma 3.2. *For all $\dot{\mathcal{E}} \in \dot{X}$, $h(\dot{\mathcal{E}}) \in X$ iff $\dot{\mathcal{E}} \in \dot{Y}$.*

Proof. The proof is similar to the proof of Lemma 3.1. □

Theorem 3.13 (correctness of *LFC*). *For all extended process $\dot{\mathcal{E}} \in \dot{X}$, $h(\dot{\mathcal{E}})$ is a pre-process. In other terms there exists a process $\mathcal{E}' \in X$ such that $h(\dot{\mathcal{E}}) \subseteq \mathcal{E}'$.*

Proof. The proof is similar to the proof of Theorem 3.4. □

Theorem 3.14 (completeness of *LFC*). *For all process $\mathcal{E} \in X$, there exists an extended process $\dot{\mathcal{E}} \in \dot{X}$ such that $h(\dot{\mathcal{E}}) = \mathcal{E}$.*

Proof. The proof is similar to the proof of Theorem 3.5. □

3.6.7 Symbolic Unfoldings of Colored Time Petri Nets

Definition 3.34 (symbolic unfolding). *We define the* symbolic unfolding *U of a colored time Petri net by collecting all the extended events that appear in its extended processes: $U \stackrel{\text{def}}{=} \bigcup_{\dot{\mathcal{E}} \in \dot{X}} \dot{E}_{\dot{\mathcal{E}}}$.*

Theorem 3.15. *Let $\dot{E} \subseteq \dot{D}_N$ be a finite set of extended events and $\dot{\mathcal{E}} : \dot{E} \longrightarrow \mathbb{R} \times PAR$. $\dot{\mathcal{E}}$ is an extended process iff:*

$$\begin{cases} \lceil \dot{E} \rceil = \dot{E} & (\dot{E} \text{ is causally closed}) \\ \nexists \dot{e}, \dot{e}' \in \dot{E} \quad \dot{e} \neq \dot{e}' \ \wedge \ {}^\bullet\dot{e} \cap {}^\bullet\dot{e}' \neq \emptyset & (\dot{E} \text{ is conflict free}) \\ \nexists \dot{e}_0, \dot{e}_1, \ldots, \dot{e}_n \in \dot{E} \quad \dot{e}_0 \nearrow \dot{e}_1 \nearrow \cdots \nearrow \dot{e}_n \nearrow \dot{e}_0 & (\nearrow \text{ is acyclic on } \dot{E}) \\ \forall \dot{e}, \dot{e}' \in \dot{E}_\perp \quad \dot{e} \nearrow \dot{e}' \implies \Theta(\dot{e}) \leq \Theta(\dot{e}') & (\Theta \text{ is compatible with } \nearrow) \\ \forall \dot{e} = (B, t) \in \dot{E} \quad LFC\left(Place(B), v_{\dot{\mathcal{E}}_{\lfloor \lceil B \rceil}}, t, \Theta(\dot{e}), \Xi(\dot{e}) \right) & \\ & (\dot{e} \text{ satisfies the local firing condition}) \end{cases}$$

Proof. The proof of this theorem is a combination of the proofs of Theorem 2.2 and 3.6. □

Exactly like in unfoldings of time Petri nets, we can check easily if an extended event $\dot{e} \in \dot{D}_N$ is in the symbolic unfolding of N, by checking only if its causal past can be the support of an extended process:

$$\dot{e} \in U \quad \text{iff} \quad \exists \dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \times PAR \quad \dot{\mathcal{E}} \in \dot{X} \ ,$$

which gives an efficient way to build U.

As a consequence of the negative results about the existence of a complete finite prefix for unfoldings of high-level Petri nets, there is no chance to define one for the model of colored time Petri nets, which is more general. Nevertheless the positive results about some restricted classes of colored Petri nets may be reused in order to define finite complete prefixes of colored time Petri nets, for example in the case where the sets of colors and parameters are finite. We have seen in Section 3.5 that the constraints that arise from the timed aspects do not prevent from defining a complete finite prefix.

Chapter 4

Supervision of Causality

We have explained how to define unfoldings for some high-level extensions of Petri nets, like colored Petri nets [63] and dynamic nets (see Section 2.2). Finally we have dealt with the problem of time Petri nets [73] and tackled the difficulties linked to the combination of urgency and confusion by defining a concurrent operational semantics for time Petri nets.

We now apply these methods to the problem of supervision in distributed systems. The idea of fault diagnosis in discrete event systems can be found for example in [27, 86]. Moreover this approach was recently extended to distributed systems [55]. Nevertheless supervision based on true concurrency models remains a challenging problem.

Our work carries on with the unfolding-based approach of [8, 9, 49]. But the use of high-level models is original, as well as our method based on the idea of embedding the supervision architecture in the model. For instance the sensors will become places where a token carries the current observation. This will allow us to define the diagnosis of an observation as the set of executions that lead to a state where these tokens carry the expected values.

In this framework we expect to take advantage of the clear representation of causality and concurrency, and to obtain a compact representation of the explanations. First, like in unfoldings of low-level models, symbolic unfoldings avoid the computation of the interleavings and the state space explosion due to concurrency. But the other reason why symbolic unfoldings give compact representations of the runs is that families of executions are grouped in symbolic processes.

Now, both giving a compact representation of the explanations and highlighting causality and concurrency are assets to the problem of supervision. Indeed our main goal is to find the causes of the failures. From this point of view, in a sequential representation of an explanation, the causes of a failure appear among the set of actions that occurred before the failure, some of

which are not causally related to it. In our approach based on unfoldings, we deliberately filter out some irrelevant information about the total order in which the events occurred, in order to keep only the causal partial order between the events.

Remark About the Different Models

We try to make this chapter as independent as possible from the exact model that is used. For this reason we often use the word "net" to refer to either "Petri net", "colored Petri net", "dynamic net", "time Petri net"... Sometimes, however, we need to choose the notations of one of these models. In such a case we often choose the model of colored time Petri nets presented in Section 3.6, because it is very general. When we think that it is useful to the reader, we give several notations according to several models.

And What About Low-Level Models? In this chapter we will have to build some models, and for these constructions we will often use some features of high-level models. We made this choice because we think that these features allow us to significantly simplify the constructions and to make them easier to understand. However one may prefer to avoid building a high-level model if one is starting from a low-level model of the supervised system. In many cases it is possible to use only low-level models; for this one could imagine a variety of tricks that are very dependent on the exact model that one uses. However a more general way to move from a high-level model to a low-level one is the expansion as described in Section 2.1.3. In some cases the finite high-level models that we build in this chapter can be expanded into finite (or at least enumerable) low-level models.

4.1 Method of Off-line Diagnosis

In this section we are considering the problem of off-line diagnosis in a distributed system. We are given a distributed system and a model of its behavior, under the form of one of the models that were introduced in Chapters 1, 2 and 3. Additionally, some actions emit alarms. Both observable and silent actions are represented by transitions in the net. Moreover, the system is equipped with sensors. Each sensor records some of the alarms, in sequence. The different sensors perform independently and asynchronously. We assume that the transmission of the alarms to the sensors is instantaneous.

Finally, at some moment a single diagnoser (sometimes called supervisor), collects the alarm sequences from the sensors. We assume that no alarm is

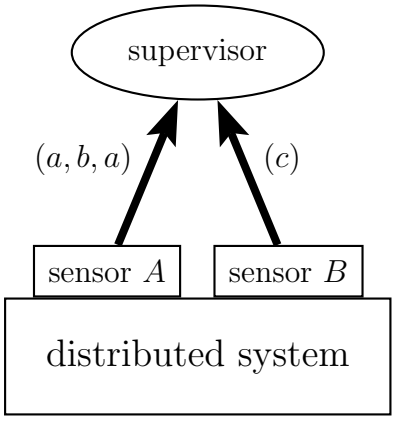


Figure 4.1: The supervision architecture. Here sensor A has collected the alarm sequence (a, b, a) and sensor B has collected the sequence (c).

lost at any moment and that the communication between the diagnoser and the sensors takes very little time, so that the diagnoser gets a snapshot of all the sensors at the same time. This architecture is illustrated in Figure 4.1.

The aim of the diagnoser is to compute the set of all the executions of the system, that are compatible with the observations. The executions are given as processes, according to the model of the system. Due to the concurrency between the sensors, any interleaving of the records from different sensors is possible, and causalities and temporal ordering among alarms from different sensors are lost.

Notice that our goal is not to check that the system conforms to the specification given by the model. Instead we assume that it does, and we compute all the processes of the model that explain the observations. This means that the faulty behaviors are also represented in the model. Only if no explanation is found, one can deduce that some of our assumptions are not satisfied: either the system does not conform to the model, or some alarms were lost...; but this is not really what we want to do here.

4.1.1 Definition of the Off-line Diagnosis

In order to compute efficiently the diagnosis, we propose a method based on the construction of a new model where not only the system, but also the sensors are represented.

Definition 4.1 (sensor (general definition)). *In a net that represents a distributed system together with its observation architecture, a* sensor *is a place of the net, which always contains one token, whose value is the current observation.*

Definition 4.2 (observation). *An* observation *is a set W of pairs (s, w) where s is a sensor and w is the observation that was recorded by s. In general we consider that w is a sequence of alarms.*

We can now define the diagnosis directly in terms of processes of the complete model that includes the sensors.

Definition 4.3 (diagnosis). *The diagnosis of an observation W in a net N is the set of processes of N defined as:*

$$Diag_N(W) = \{\mathcal{E} \in X_N \mid \forall (s, w) \in W \quad v_{\mathcal{E}}(s) = w\} .$$

We recall that $v_{\mathcal{E}}(s)$ is the value of the token that stands in place s in the state $RS(\mathcal{E})$.

In Section 4.1.2, we describe a possible setup for off-line diagnosis and give a definition of a model of the system plus its sensors, from a model of the system and a description of the sensors. We give examples of diagnoses using this setup.

Then in Section 4.1.3, we give several examples that are not in the scope of Section 4.1.2. So we have to give another construction of the augmented model. The interest is to show that if one has a precise knowledge of the behavior of the sensors, one can define a more accurate model of the observation architecture and use it to compute the diagnosis.

4.1.2 An Example of Representation of the Sensors

Presentation of the Setup

In this section we start from a model N of a net, which does not include the sensors and a description of the sensors is given separately. From this information we explain how to build a net N' where the sensors are represented by places.

The setup we consider here assumes a sequential behavior of the sensors and instantaneous observations. Each time a transition t fires, it is observed by a set of sensors, denoted $Sensors(t)$. If $Sensors(t) = \emptyset$, then t is a silent transition. And if two transitions t_1 and t_2 are observed by the same sensor $s \in Sensors(t_1) \cap Sensors(t_2)$, then for all firing sequence where t_1 fires before t_2, s records the alarm emitted by t_1 before the one emitted by t_2.

A sensor can be formally defined by the alarms that it observes when the transitions fire. In high-level models, the alarm may depend also on the values of the tokens that are consumed or read, and on the value of the parameter. It may even depend on the time if we are dealing with a timed model.

Definition 4.4 (sensor). *Let Λ be a (possibly infinite) set of alarms. With the notations of colored time Petri nets, a* sensor *is formally described by a function s that assigns to each transition $t \in T$ observed by s ($s \in Sensors(t)$), a mapping*

$$s(t) : \mathbb{R} \times PAR \times ({}^{\bullet}\underline{t} \longrightarrow V) \longrightarrow \Lambda$$

which defines the alarm which is observed by sensor s when transition t fires.

We remark that the value of the alarm that is observed by a sensor is defined in a very similar way as the value of the token that is created in a place of the postset of the transition. This justifies our idea of constructing a model where the sensors are represented by places.

Definition 4.5 (observed alarm sequence). *The sequence of alarms that is observed by sensor s during a firing sequence σ is denoted $obs^s_N(\sigma)$ and obtained as the sequence of all the observed alarms:*

$$obs^s_N(\epsilon) \stackrel{\text{def}}{=} \epsilon$$
$$obs^s_N(\sigma \cdot (t, \theta, x)) \stackrel{\text{def}}{=} \begin{cases} obs^s_N(\sigma) \cdot s(t)(\theta, x, v_{|{}^{\bullet}\underline{t}}) & \text{if } s \in Sensors(t) \\ obs^s_N(\sigma) & \text{otherwise} \end{cases}$$

where the state that is reached after σ is (M, v, θ).

We notice that this kind of observation respects the causal relations between the actions: if two observable actions are (even weakly) causally related (more precisely, if the corresponding events in $\Pi(\sigma)$ are related by the transitive closure of the restriction of $\nearrow$ to the events of $\Pi(\sigma)$), then the alarm emitted by e_1 is observed before the alarm emitted by e_2.

In order to simplify the following, we define the off-line diagnosis for safe models first. This allows us to deal with firing sequences and to map them to processes when needed. We hope that this makes the section easier to read and understand. Only at the end, we will give a final result that can also be applied to unsafe models.

Definition 4.6 (explanations). *A firing sequence σ of a net N conforms to an observation $W \stackrel{\text{def}}{=} \{(s_1, w_1), \ldots, (s_n, w_n)\}$ if for all $i \in \{1, \ldots, n\}$,*

$obs_N^{s_i}(\sigma) = w_i$. *The* explanations *of an observation* W *according to a model* N *is the set of processes of* N *defined as:*

$$PDiag_N(W) \stackrel{\text{def}}{=} \{\Pi(\sigma) \mid \sigma \in \Sigma_N \text{ and } \sigma \text{ conforms to } W\} .$$

We give some examples in Section 4.1.2 below.

This definition of the explanations is based on the sequential semantics of N. We will show how to rewrite it in terms of processes. For this we use our method based on the construction of a model N' that represents both the system and the sensors. This will allow us to avoid building firing sequences and to define the explanations directly in terms of processes of N'.

Representation of the Sensors in the Model

We will now define what could be the whole system corresponding to a net N observed by a set of sensors $\{s_1, \ldots, s_n\}$. The sensors will become new places of the net, which contain the current sequence of observed alarms. Each transition which is observed by a sensor s_i will simply update the observed firing sequence.

Definition 4.7 (N'). *We propose the net* N' *as a model of the net* $N \stackrel{\text{def}}{=} (P, T, V, PAR, pre, cont, post, \alpha, \beta, \omega, S_0)$*, plus its sensors* $\{s_1, \ldots, s_n\}$*.*

$$N' \stackrel{\text{def}}{=} (P \cup \{s_1, \ldots, s_n\}, T, V \cup \Lambda^*, PAR, pre', cont, post', \alpha', \beta', \omega', S_0')$$

with

- $pre'(t) \stackrel{\text{def}}{=} pre(t) \cup Sensors(t)$
- $post'(t) \stackrel{\text{def}}{=} post(t) \cup Sensors(t)$
- $\alpha'(t)(\theta, x, v) \stackrel{\text{def}}{=} \alpha(t)(\theta, x, v_{|pre(t) \cup cont(t)})$
- $\beta'(t)(\theta, x, v)(p) \stackrel{\text{def}}{=} \begin{cases} \beta(t)(\theta, x, v_{|pre(t) \cup cont(t)})(p) & \text{if } p \in P \\ v(s_i) \cdot s_i(t)(\theta, x, v_{|pre(t) \cup cont(t)}) & \text{if } p = s_i \end{cases}$
- $\omega'(t)(v) \stackrel{\text{def}}{=} \omega(t)(v_{|pre(t) \cup cont(t)})$
- $S_0' \stackrel{\text{def}}{=} (M_0', v_0', \theta_0)$*, where* $S_0 \stackrel{\text{def}}{=} (M_0, v_0, \theta_0)$*,* $M_0' \stackrel{\text{def}}{=} M_0 \cup \{s_1, \ldots, s_n\}$ *and* $v_0' \stackrel{\text{def}}{=} v_0 \cup (\{s_1, \ldots, s_n\} \times \{\epsilon\})$*.*

Figure 4.2 illustrates this definition.

Obviously the definition of N' ensures that each place s_i contains a single token at any time. Moreover this token carries the observed alarm sequence, as stated in the following theorem.

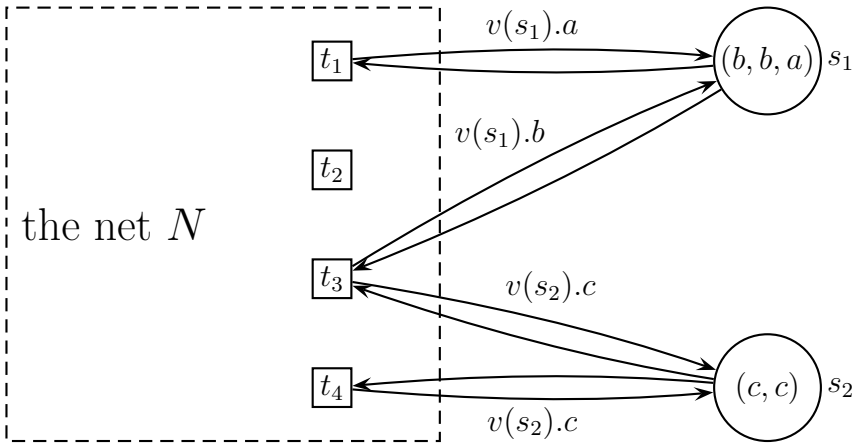


Figure 4.2: The construction of the net N' which embeds a representation of two sensors s_1 and s_2 in a model N of a distributed system. Only the transitions of N are represented. The sensor s_1 observes an alarm a when t_1 fires and b when t_3 fires. t_3 is also observed by s_2. s_2 observes c when either t_3 or t_4 fire. t_2 is silent. In the state that is represented, s_1 has observed (b, b, a) and s_2 has observed (c, c).

Theorem 4.1. *The firing sequences of N' are exactly equal to those of N. In addition, for every sensor s_i, the token in the place s_i carries the value $obs_N^{s_i}(\sigma)$ after a firing sequence σ has been executed in N'.*

Proof. Clearly all the firing sequences of N' are firing sequences of N. Let us show that the converse holds, and that for all σ, if σ leads to the state (M, v, θ) when executed in N, then, when executed in N', it leads to (M', v', θ), with $M' \stackrel{\text{def}}{=} M \cup \{s_1, \ldots, s_n\}$, $v' \stackrel{\text{def}}{=} v \cup \{(s_1, obs_N^{s_1}(\sigma)), \ldots, (s_1, obs_N^{s_1}(\sigma))\}$. This holds when $\sigma = \epsilon$. Now assume that it holds for σ and let t fire at date θ with parameter x from (M, v, θ). This firing is also possible in N' since $\alpha(t)(\theta, x, v) \implies \alpha'(t)(\theta, x, v \cup \{(s_1, obs_N^{s_1}(\sigma)), \ldots, (s_1, obs_N^{s_1}(\sigma))\})$. For all $p \in P$, the consumptions and creations of tokens in p are the same in N and in N'. For s_i, if $s_i \in Sensors(t)$, then the new token in s_i takes the value $v'(s_i) \cdot s_i(t)(\theta, x, v'_{|\bullet \underline{t}}) = obs_N^{s_i}(\sigma).s_i(t)(\theta, x, v) = obs_N^{s_i}(\sigma \cdot (t, \theta, x))$. And if $s_i \notin Sensors(t)$, then the token in s_i remains and $obs_N^{s_i}(\sigma \cdot (t, \theta, x)) = obs_N^{s_i}(\sigma)$. □

Notice that the requirements at the end of Section 3.6.2 are preserved by this transformation, since the new places do not constrain the interleaving behavior of the net and do not take part in the definition of ω'.

Finally the processes of N' can be mapped to processes of N by removing the conditions that are related to the sensors.

Definition 4.8. *The process of N that corresponds to a process $\mathcal{E}$ of N' is defined as:*

$$Proj(\mathcal{E}) \stackrel{\text{def}}{=} \{(proj(e), \mathcal{E}(e)) \mid e \in E_{\mathcal{E}}\}$$

where for all $e \stackrel{\text{def}}{=} (B, t) \in E_{\mathcal{E}} \cup \{\bot\}$,

$$proj(e) \stackrel{\text{def}}{=} \Big(\{(proj(f), p) \mid (f, p) \in B \wedge p \in P\}, t\Big) .$$

Theorem 4.2. *For all firing sequence σ, $Proj(\Pi_{N'}(\sigma)) = \Pi_N(\sigma)$.*

Proof. To prove this theorem we have to relate the final conditions of $\Pi_N(\sigma)$ with those of $Proj(\Pi_{N'}(\sigma))$. That is, we show by induction on the length of σ that for all σ, $Proj(\Pi_{N'}(\sigma)) = \Pi_N(\sigma)$ and $\uparrow(\Pi_N(\sigma)) = \{(proj(f), p) \mid (f, p) \in \uparrow(\Pi_{N'}(\sigma)) \wedge p \in P\}$. The inductive step is straightforward. □

Theorem 4.3. *The explanations $PDiag_N(W)$ of an observation W according to a net N (see Definition 4.6) can now be rewritten only in terms of processes, without using any firing sequences.*

$$PDiag_N(W) = \{Proj(\mathcal{E}') \mid \mathcal{E}' \in Diag_{N'}(W)\} .$$

Proof. According to Definition 4.6, $PDiag_N(W) \stackrel{\text{def}}{=} \{\Pi_N(\sigma) \mid \sigma \in \Sigma_N \wedge (\forall i \quad obs_N^{s_i}(\sigma) = w_i)\}$. According to Theorem 4.1, $\Sigma_N = \Sigma_{N'}$ and $obs_N^{s_i}(\sigma)$ is the value of the token that stands in place s_i in the state of N' that is reached after $\Pi_{N'}(\sigma)$. Thus $PDiag_N(W) = \{\Pi_N(\sigma) \mid \sigma \in \Sigma_{N'} \wedge (\forall i \quad v_{\Pi_{N'}(\sigma)}(s_i) = w_i)\} = \{\Pi_N(\sigma) \mid \Pi_{N'}(\sigma) \in Diag_{N'}(W)\}$. Moreover, according to Theorem 4.2, $\Pi_N(\sigma) = Proj(\Pi_{N'}(\sigma))$. This gives $PDiag_N(W) = \{Proj(\Pi_{N'}(\sigma)) \mid \Pi_{N'}(\sigma) \in Diag_{N'}(W)\}$, and finally $PDiag_N(W) = \{Proj(\mathcal{E}') \mid \mathcal{E}' \in Diag_{N'}(W)\}$. □

Examples

We give now some examples of distributed systems that are observed according to the setup of the current section and we describe the explanations $PDiag_N(W)$.

Failure Propagation. Figure 4.3 shows a colored Petri net that models a system made of two interacting components A and B. The actions of component A are represented by transitions a and b. The first component

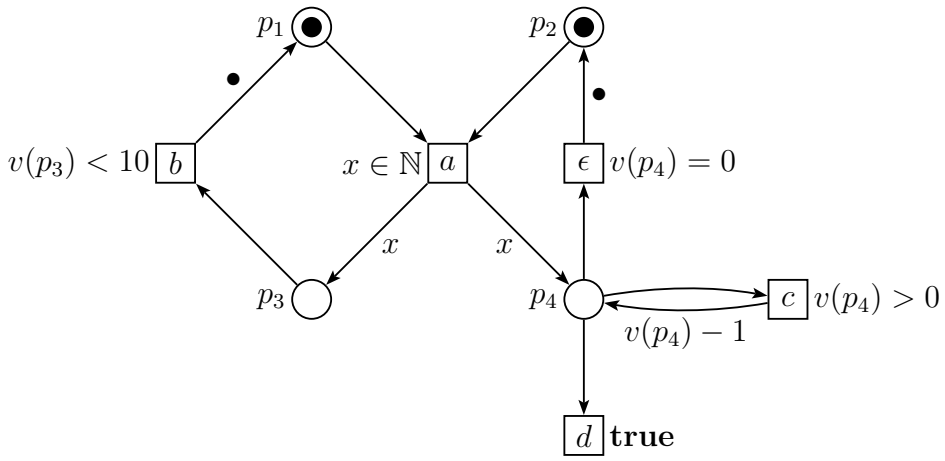


Figure 4.3: A concurrent machine with two components, which may fail with an unknown severity level and can be repaired accordingly. A sensor (not represented) collects the alarms a and b and a second one collects the alarms b and d. The transition ϵ is silent.

may fail (observed as a) with a given non observable severity level (parameter x).

To be completely repaired, component A must execute a local action (observed as b, and possible only if the severity level is lower than 10), and wait for the completion of the recovery procedure of component B, which has been informed of the failure. To recover from a failure of severity x, component B must execute x repairs, observed as c. The last action recovers the failure by putting a token again in place p_2. This action is not observable. Additionally, at any time during recovery, component B may also fail and stop (observed as d).

In the initial state p_1 and p_2 are marked, which models the fact that no failure has occurred yet.

The system is observed by two independent sensors s_1 and s_2. Sensor s_1 collects the alarms a and b and s_2 collects the alarms b and d. The transition ϵ is silent.

Figure 4.4 shows the two parameterized explanations that take part in the explanations of $PDiag_N(\{(s_1,(a,b,a)),(s_2,(c,d))\})$. We adopt the graphical notations of symbolic unfoldings of colored Petri net, like in Figure 2.4 for example.

It is interesting to notice the refinement of the constraints on the parameter of the event that models the initial failure: for instance, at the end of

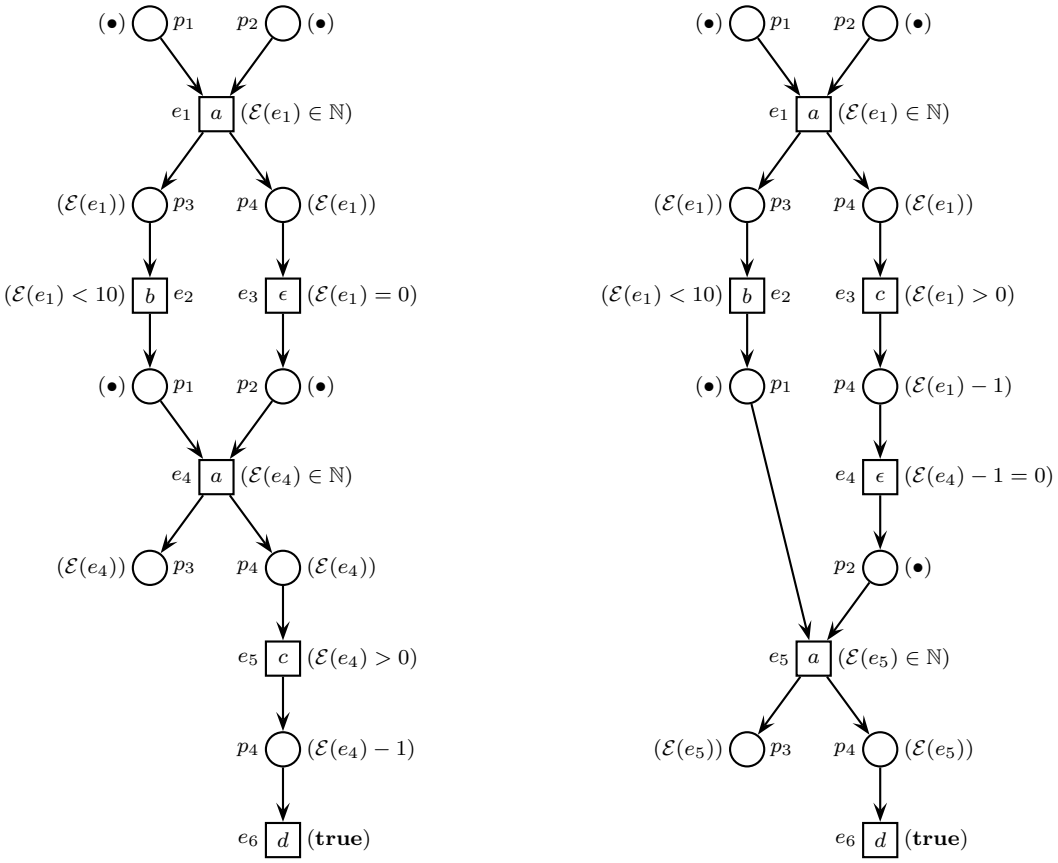


Figure 4.4: The two parameterized explanations of the off-line diagnosis of the observations (a, b, a) and (c, d). The symbolic representation of the executions avoids generating a separate explanation for each value of the severity level of the second failure a.

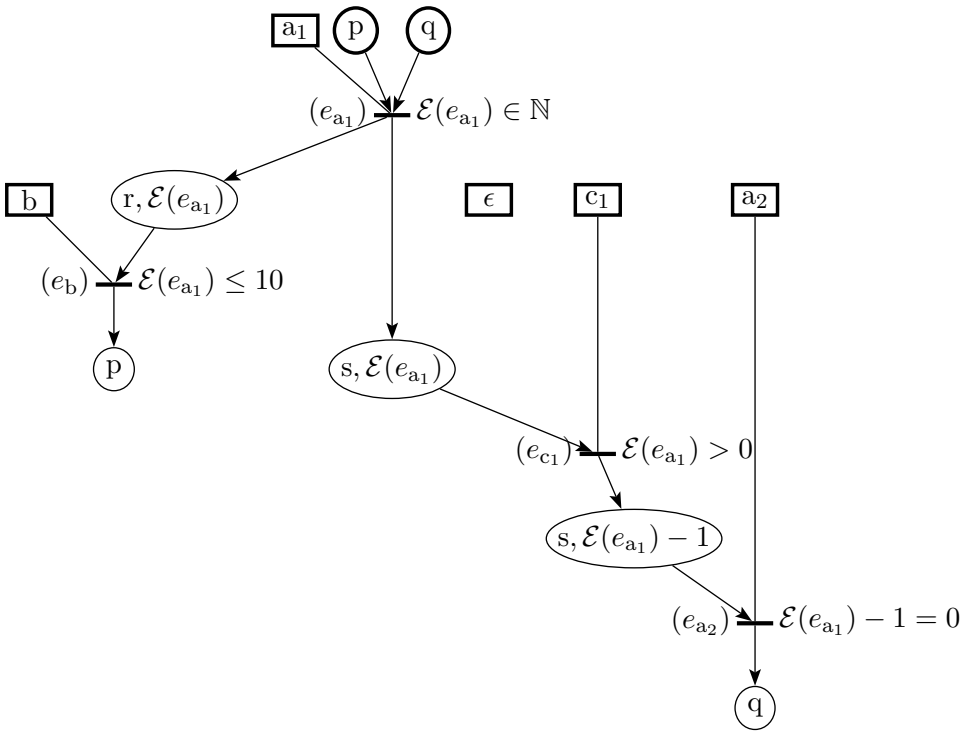


Figure 4.5: A process of the dynamic net of Figure 2.6 that explains the observation $\{(s_1, (a, b, a)), (s_2, (c))\}$. This explanation involves the structural change modeled by ϵ.

the first explanation, we can infer that its severity level was 0, because of the conjunction of the constraints that come from the guards of the transitions.

Moreover the symbolic representation of the executions avoids generating a separate explanation for each value of the severity level of the second failure a.

Failure Propagation with Dynamic Structure. In our second example we consider the dynamic net which is shown on Figure 2.6, where two sensors observe the system. Sensor s_1 observes a when either a_1 or a_2 fire, and b when b fires; s_2 observes c when either c_1 or c_2 fire. The transition ϵ which models the dynamic change of the structure is not observable. Figures 4.5 and 4.6 show the valid explanations if s_1 has observed (a, b, a) and s_2 has observed (c).

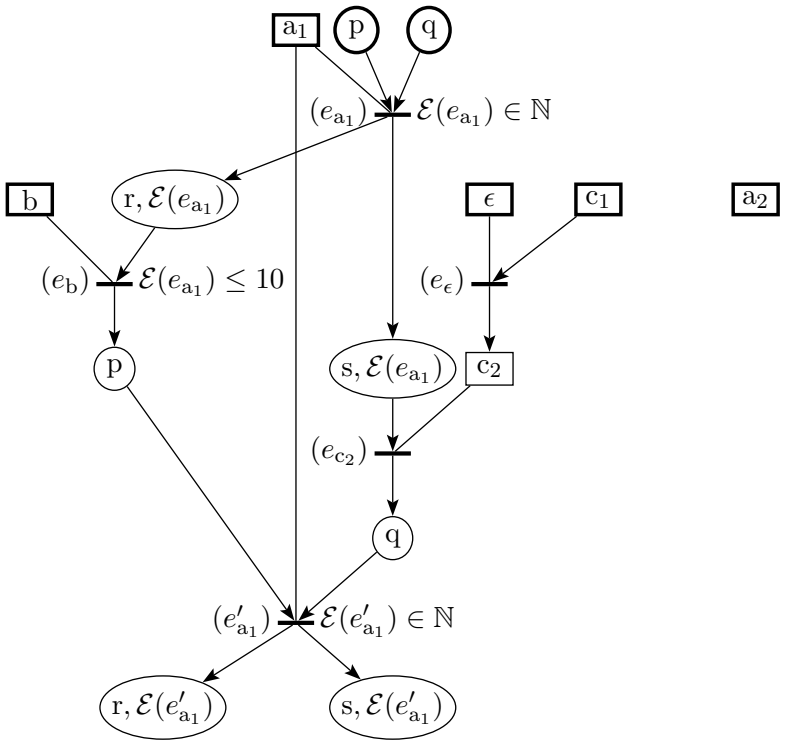


Figure 4.6: A process of the dynamic net of Figure 2.6 that explains the observation $\{(s_1, (a, b, a)), (s_2, (c))\}$. In this process the structural change modeled by transition ϵ does not occur. Nevertheless, as ϵ is silent, the same process with an occurrence of ϵ at the end, would be also a valid explanation.

$\mathcal{E}(e_1) \geq 0$
$1 \leq \mathcal{E}(e_2) \leq 2$
$\mathcal{E}(e_3) = \max\{\mathcal{E}(e_1), \mathcal{E}(e_2)\}$

$\mathcal{E}(e_1) = 0$
$\mathcal{E}(e_2) = \mathcal{E}(e_3) = 2$

Figure 4.7: The two possible explanations of the observation $\{(s_1, (a, a, a)), (s_2, (b))\}$, according to the model of Figure 3.1.

Real-Time System. In our third example we deal with diagnosis in a real-time system. The model is the one of Figure 3.1, and a sensor s_1 observes a each time either t_0, t_1 or t_3 fires, while sensor s_2 observes b when t_2 fires.

Figure 4.7 shows the two possible explanations of the observation $\{(s_1, (a, a, a)), (s_2, (b))\}$. Below each explanation we give constraints about the dates of the events. We notice that even untimed observations we can infer some information about the dates of the events that are involved in the explanations.

We will refine this example below and see how we can handle the case where the alarms contain information about the firing date of the events.

Finite Symbolic Representation of $Diag_{N'}(W)$

We notice that the set of processes $Diag_{N'}(W)$ is in general smaller than the set of firing sequences that conform to W, since two firing sequences can be mapped to the same process. But $Diag_{N'}(W)$ will often be very large anyway, and even infinite. Indeed the values that the parameters take when the transitions fire are often hidden to the sensors. The same is true for the firing dates. As a result there will often be situations where we know which transition fired, but we have little information about the value of the parameter and the firing date. This type of situation will often make $Diag_{N'}(W)$ be infinite.

However, if several processes differ only by the values of the parameters and by the firing dates, then we will choose to group them into a single "symbolic process". As well as in Section 2.1.5 when we introduced processes of colored Petri nets, we can group the processes that involve the same set E of events. The information about the possible values for the parameters and the firing dates will be given under the form of a constraint on the $\Xi(e)$ and the $\Theta(e)$, $e \in E$, like those that we solve when we want to decide if a mapping $\mathcal{E} : E \longrightarrow \mathbb{R} \times PAR$ is a process of a colored time Petri net (see Section 3.6.5) or if a mapping $\dot{\mathcal{E}} : \dot{E} \longrightarrow \mathbb{R} \times PAR$ is an extended process (see Theorem 3.15).

Hence the diagnosis can be represented as a set of symbolic processes where each symbolic process may represent even an infinite set of processes that share the same set of events. Moreover, if we make the assumption that enough firings are observed so that the ratio between silent transitions and observable transitions is bounded, then $Diag_N(W)$ can be represented as a finite set of symbolic processes.

Theorem 4.4. *Given a net N and a set $\{s_1, \ldots, s_n\}$ of sensors, if $\sup\{\frac{|\sigma|}{\sum_{i=1}^{n} |s_i(\sigma)|} \mid \sigma \in \Sigma_N\} < \infty$, then for all observation $W \stackrel{\text{def}}{=} \{(s_i, w_1), \ldots, (s_n, w_n)\}$, $\{E_{\mathcal{E}'} \mid \mathcal{E}' \in Diag_{N'}(W)\}$ and $\{E_{\mathcal{E}} \mid \mathcal{E} \in PDiag_N(W)\}$ are finite.*

As a consequence $Diag_{N'}(W)$ and $PDiag_N(W)$ can be represented as finite sets of symbolic processes.

Proof. Denote $r \stackrel{\text{def}}{=} \sup\{\frac{|\sigma|}{\sum_{i=1}^{n} |s_i(\sigma)|} \mid \sigma \in \Sigma_N\}$. Let $\mathcal{E} \in PDiag_N(W)$ and σ such that $\Pi_N(\sigma) = \mathcal{E}$ and for all $i \in \{1, \ldots, n\}$, $obs_N^{s_i}(\sigma) = w_i$. By definition of r, we have $r \geq \frac{|\sigma|}{\sum_{i=1}^{n} |obs_N^{s_i}(\sigma)|} = \frac{|\sigma|}{\sum_{i=1}^{n} |w_i|}$. So $|\sigma| \leq r \sum_{i=1}^{n} |w_i|$. And $|E_{\mathcal{E}}| = |\sigma|$. Let $e \in E_{\mathcal{E}}$. As $\lceil e \rceil \subseteq E_{\mathcal{E}}$, $|\lceil e \rceil|$ is smaller than $r \sum_{i=1}^{n} |w_i|$, which is finite if r is finite. Then we remark that for all $n \in \mathbb{N}$, there is only a finite set of events e in D_N such that $|\lceil e \rceil| \leq n$ (see Section 2.3.1 for more details and a discussion about the case of dynamic nets). This proves that $\{E_{\mathcal{E}} \mid \mathcal{E} \in PDiag_N(W)\}$ is finite.

For $Diag_{N'}(W)$, we proceed in the same way after noticing that for all $\mathcal{E}' \in Diag_{N'}(W)$, and for all σ such that $\Pi_{N'}(\sigma) = \mathcal{E}'$, $|\sigma| \leq r \sum_{i=1}^{n} |w_i|$ because $\Pi_N(\sigma) = Proj(\Pi_{N'}(\sigma)) = Proj(\mathcal{E}') \in PDiag_N(W)$. □

Remark About Diagnosis in Unsafe Nets

The new expression of the diagnosis of a safe net, that appears in Theorem 4.3, can be used as a definition in the case of unsafe nets. It is worth mentioning that with unsafe nets, the number of processes that explain an

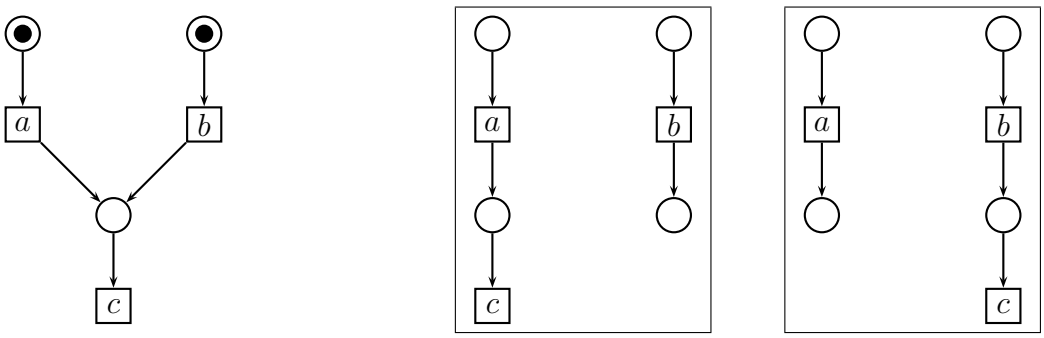


Figure 4.8: A Petri net (on the left) where the transitions emit the alarms a, b or c. The unique sensor has recorded the sequence (a, b, c). The two processes (on the right) that explain the observation correspond to the single firing sequence (a, b, c).

observation may be larger than the number of firing sequences, as shown in Figure 4.8. But in our approach we consider that these two different processes represent actually two different explanations, since in the first explanation, a is the cause of c, whereas b is the cause of c in the second explanation.

Remark About Concurrency in N'.

We notice that the net N' that we have built has in general less concurrency than the net N. This is the consequence of the sequential nature of the observed alarm sequences, which suggested to us that the accesses to each sensor should be done in sequence.

However it is likely that in a real architecture, a sensor is responsible for the observation of only few components of the distributed system. As a matter of fact, it will often be the case that all the transitions that send alarms to a single sensor, represent actions of the same sequential machine. In this case the net N' does not break any concurrency in the system.

Finally we emphasize the fact that the use of unfoldings is not very profitable if there is only one sensor, because the processes of N' do not contain any concurrency between the observable events. However this is not true any more if there are many silent events, which may not be totally ordered by causality. We can also argue that the graphical aspect of unfoldings is helpful; for example, even if a process $\mathcal{E}$ of N' does not contain any concurrency, maybe its projection $Proj(\mathcal{E})$ does, and it is obtained simply be removing the conditions that correspond to the sensors.

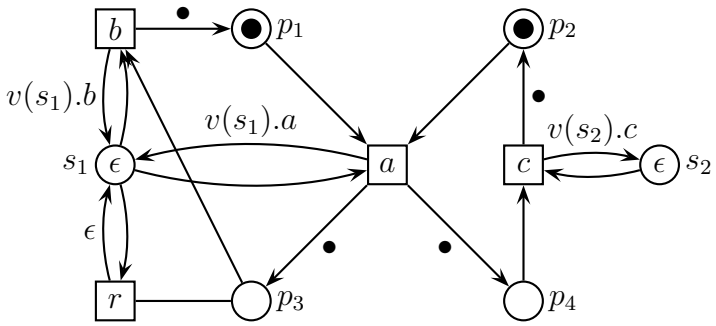


Figure 4.9: A model of the observation architecture of a distributed system where sensor s_1 may be affected by a failure.

4.1.3 Other Examples of Representation of the Sensors

In Definition 4.7, we try to build a model of a system plus its sensors, from a model of the system and a description of the sensors. Maybe in some situations the net N' does not give a very satisfactory representation of the observation architecture.

If one has a precise knowledge of the behavior of the sensors, one can give a more accurate model of the observation architecture. Then the diagnosis can be defined using the expression that appears in Theorem 4.3, provided the sensors are defined as follows.

Sensor Affected by a Failure

Here we present an example of a distributed system that is observed by two sensors s_1 and s_2. But when a failure occurs in the system, it may happen that s_1 is affected. Then it simply forgets its previous observations and restarts from the empty alarm sequence. Figure 4.9 shows a possible model of this observation architecture. The reset of s_1 is modeled by transition r. The read arc from the faulty state p_3 to r represents the fact that the reset may happen only after a failure. The rest of the model is a simplified version of the model of Figure 4.3.

If the diagnoser gets the alarm sequence (b) from s_1 and (c) from s_2, we can compute the two explanations that are shown in Figure 4.10. For this observation, both explanations involve a reset of the sensor s_1.

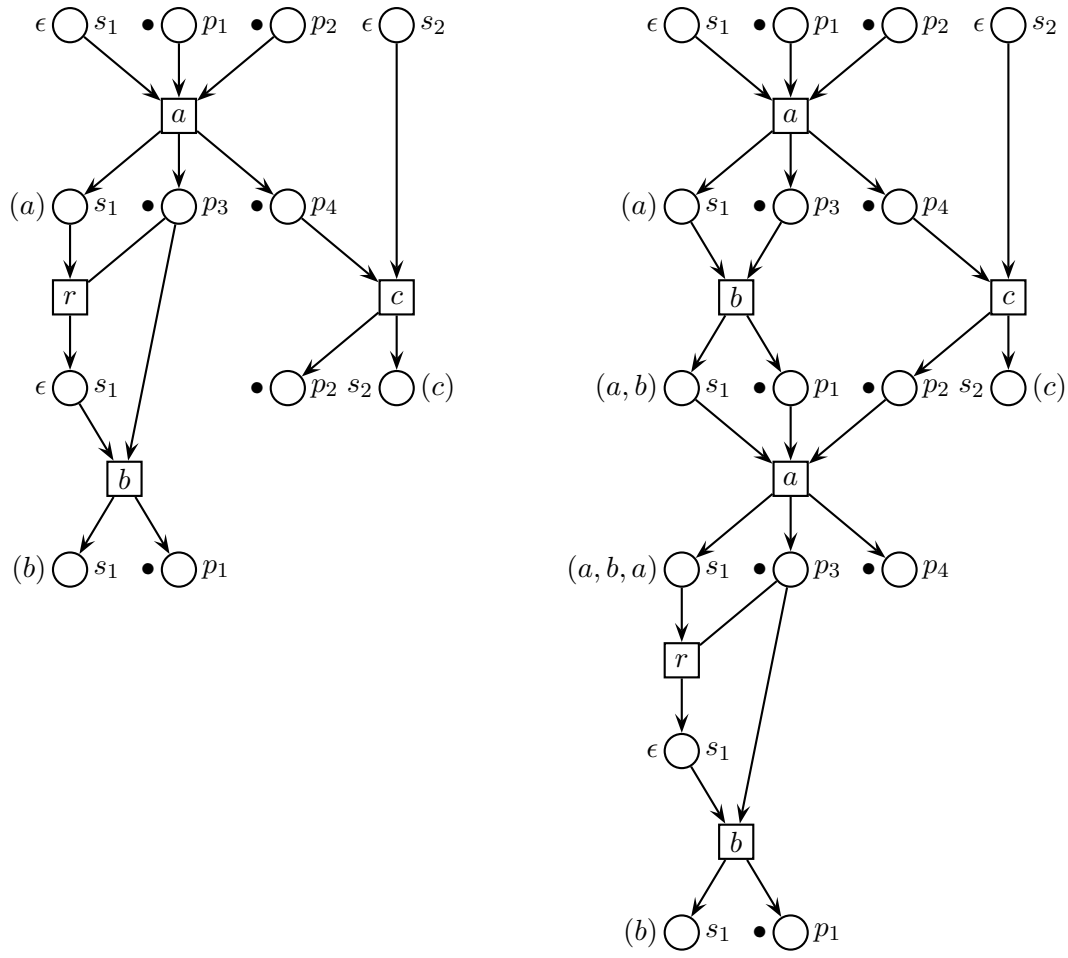


Figure 4.10: The two explanations of the observation $\{(s_1, (b)), (s_2, (c))\}$. In both of them the sensor s_1 was affected by a failure.

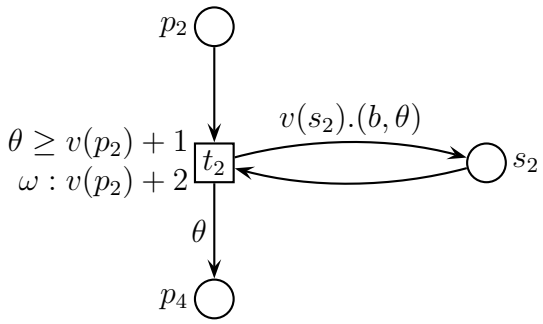


Figure 4.11: The representation of a transition and a sensor in a model of a real-time system where the transitions emit alarms that contain the exact firing date.

Dated Alarms

In the context of real-time systems, it is very likely that the alarms contain some information about the dates of the events. We can handle this setup by representing in the model the way the alarms are created and how the dates are assigned. In Section 4.1.2 we have given an example of diagnosis in a real-time system where the dates were not observed.

First we can assume that each alarm carries the exact date of the event that emitted it. Then the observation architecture can be modeled by a colored time Petri net. Figure 4.11 shows the way transition t_2 of the model of Figure 3.1 can be connected to the sensor. Each alarm becomes a pair (λ, θ), where λ is a label that depends on the transition that emits the alarm, and θ is the time when the alarm is emitted. With this setup, the first process of Figure 4.7 can explain the observation $W_1 \stackrel{\text{def}}{=} \{(s_1, ((a, 1.3), (a, 1.7))), (s_2, ((b, 1.7)))\}$. But the observation $W_2 \stackrel{\text{def}}{=} \{(s_1, ((a, 1.3), (a, 1.8))), (s_2, ((b, 1.7)))\}$ has no explanation.

Nevertheless it is very common in such real-time systems that the precision of the clocks is not perfect. This is particularly relevant for distributed systems where the clocks may not be synchronized very often. Consequently we expect that the dates that are given by the alarms may be slightly erroneous. We propose to take this into account in our model of the observation architecture. Our first construction requires to know a bound δ on the gap. The idea is to use the parameters of the transitions in order to model the errors by non-determinism in the assignment of the dates, so that all the dates

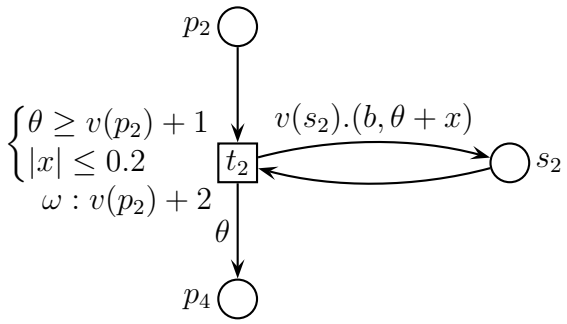


Figure 4.12: The representation of a transition and a sensor in a model of a real-time system where the transitions emit alarms that contain an approximation of the firing date. The gap is assumed to be smaller than 0.2 time units.

in $[\theta - \delta, \theta + \delta]$ may appear on the alarm that is emitted when a transition fires at time θ.

Figure 4.12 shows the way the transition t_2 of the model of Figure 3.1 can be connected to the sensor, assuming that the gap is smaller than 0.2 time units. Under the assumption that the same bound holds for the other transitions of the model, the observation $\{(s_1, ((a, 1.3), (a, 1.8))), (s_2, ((b, 1.7)))\}$ has an explanation: the first process of Figure 4.7 becomes possible with $1.1 \leq \mathcal{E}(e_1) \leq 1.5$ and $1.6 \leq \mathcal{E}(e_2) = \mathcal{E}(e_3) \leq 1.9$.

4.2 An On-line Diagnosis Method

In off-line diagnosis we assume that the alarm sequences are sent to the diagnoser only before the diagnosis. On the contrary, in on-line diagnosis, the diagnoser performs while the system keeps running, and receives regularly new information about the observations.

4.2.1 A Naïve Solution

A naïve solution to this problem is to compute a new off-line diagnosis each time the observations are updated. The successive diagnoses can be computed using the technique of off-line diagnosis. It is even possible to reuse large parts of the previous diagnosis and to update it according to the new observations. Hence this technique can be made rather efficient.

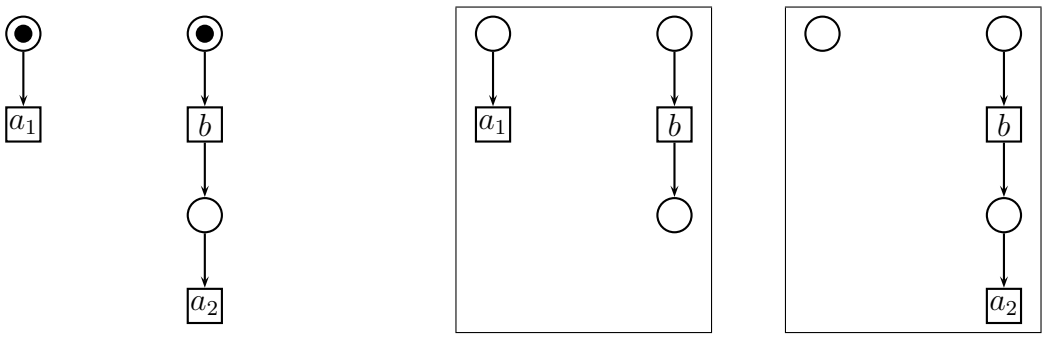


Figure 4.13: The Petri net on the left models a system which is observed by two sensors: s_a observes an alarm a when either transition a_1 or a_2 fires; s_b observes b when transition b fires. In the off-line diagnosis of the observation $\{(s_a,(a)),(s_b,(b))\}$, the two explanations on the right are correct. But in the context of on-line diagnosis, only the first one remains if the diagnoser was aware of the observation $\{(s_a,(a)),(s_b,())\}$ first, and then $\{(s_a,()),(s_b,(b))\}$.

But the diagnosis which is computed using this technique is not selective enough. The problem is that the diagnosis does not take into account the fact that, at some moment, the earlier observations were done. Figure 4.13 shows an example where neglecting the earlier observations leads to a false diagnosis.

4.2.2 Transmission of the Observations from the Sensors to the Diagnoser

Moreover on-line diagnosis requires to know precisely when and how the observations are sent to the diagnoser. In the context of off-line diagnosis, we made the assumption that all the sensors send their observations to the diagnoser at the same time. When we deal with on-line diagnosis, the system is observed while it is running. For this reason, it is likely that the sensors do not necessarily send their observations together. And several setups can be envisaged. We give some examples.

General requests from the diagnoser. This case is the continuation of the assumption that we made for the off-line diagnosis: all the sensors send their observations to the diagnoser at the same time. But in on-line diagnosis, there are several successive requests, which can be either planned or left to the discretion of the diagnoser.

Requests from the diagnoser to a subset of the sensors. Maybe it is not realistic to assume that all the sensors are requested simultaneously like before, so we can assume that the requests are sent separately to the sensors. As a consequence, the diagnoser does not get a snapshot any more. The diagnoser has to take into account the fact that the sensors that were requested first may have received new alarms before the end of the requests.

Immediate transmission to the diagnoser. If the sensors inform the diagnoser immediately each time a new alarm is observed, then the diagnoser has full information about the total temporal ordering of the events. As a matter of fact this setup is very similar to the case where there would be only a single sensor.

Buffered transmission. Each sensor sends its observation when it has reached a given length.

Triggered transmission. An event triggers the transmission of the observation of a sensor. This event could be for instance a timeout or the observation of a particular alarm.

To take into account this variety of setups, we suggest to represent the diagnoser in the model. That is, we will build a model where not only the sensors are represented by places, like we did in the framework of off-line diagnosis, but also the diagnoser is represented by a place. This place is denoted $\mathcal{D}$ and always contains one token, whose value gives full information about all the observations that were transmitted to the diagnoser since it was connected to the system.

Before giving a general definition of off-line diagnosis, we study an example.

Representation of the Diagnoser in the Case of General Requests from the Diagnoser

In the case of general requests from the diagnoser, the value of the token in $\mathcal{D}$ could be the list of all the answers to the past requests. Each answer is coded as a mapping from the set of sensors $\{s_1, \ldots, s_n\}$, to Λ^*.

We can build a net N'' by adding a place $\mathcal{D}$ to a net $N' \stackrel{\text{def}}{=} (P', T', V', PAR', pre', cont', post', \alpha', \beta', \omega', S'_0)$ where sensors are represented by places $\{s_1, \ldots, s_n\}$. The transition **request** models the requests from the diagnoser.

Definition 4.9. *The net N'' is defined as:*

$$N'' \stackrel{\text{def}}{=} (P'', T'', V'', PAR', pre'', cont'', post'', \alpha'', \beta'', \omega'', S_0'')$$

where

- $P'' \stackrel{\text{def}}{=} P' \cup \{\mathcal{D}\}$;
- $T'' \stackrel{\text{def}}{=} T' \cup \{\textbf{request}\}$;
- $V'' \stackrel{\text{def}}{=} V' \cup (\{s_1, \ldots, s_n\} \longrightarrow \Lambda^*)^*$;
- $pre''(\textbf{request}) = post''(\textbf{request}) \stackrel{\text{def}}{=} \{\mathcal{D}, s_1, \ldots, s_n\}$;
  $cont''(\textbf{request}) \stackrel{\text{def}}{=} \emptyset$;
  $\alpha''(\textbf{request})(\theta, x, v) \stackrel{\text{def}}{=} \textbf{true}$;
  $\beta''(\textbf{request})(\theta, x, v)(\mathcal{D}) \stackrel{\text{def}}{=} v(\mathcal{D}) \cdot v_{|\{s_1,\ldots,s_n\}}$;
  $\beta''(\textbf{request})(\theta, x, v)(s_i) \stackrel{\text{def}}{=} \epsilon$;
  $\omega''(\textbf{request})(v) \stackrel{\text{def}}{=} \infty$;
- *for all* $t \in T'$, $pre''(t) \stackrel{\text{def}}{=} pre'(t)$, $cont''(t) \stackrel{\text{def}}{=} cont'(t)$, $post''(t) \stackrel{\text{def}}{=} post'(t)$, $\alpha''(t) \stackrel{\text{def}}{=} \alpha'(t)$, $\beta''(t) \stackrel{\text{def}}{=} \beta'(t)$, $\omega''(t) \stackrel{\text{def}}{=} \omega'(t)$;
- $S_0'' \stackrel{\text{def}}{=} (M_0'', v_0'', \theta_0')$, *where* $S_0' \stackrel{\text{def}}{=} (M_0', v_0', \theta_0')$, $M_0'' \stackrel{\text{def}}{=} M_0' \cup \{\mathcal{D}\}$ *and* $v_0'' \stackrel{\text{def}}{=} v_0' \cup \{(\mathcal{D}, \epsilon)\}$.

Representation of the Diagnoser in the General Case

In the general framework of off-line diagnosis, we have to express formally the fact that the information that the diagnoser received in the past, is retained. For this we use a partial order $\leq$ on the possible values for the token in $\mathcal{D}$. With the construction that we proposed in Definition 4.9, $\leq$ would be the prefix relation on the words of $(\{s_1, \ldots, s_n\} \longrightarrow \Lambda^*)^*$.

Intuitively $\Omega_1 \leq \Omega_2$ means that Ω_1 gives a part of the information in Ω_2. We impose that the value of the token in $\mathcal{D}$ keeps growing during the execution. Moreover we need a way to retrieve all the values that were reached in the past. For this we require that, if a value Ω is reached after an execution, then all the values $\Omega' \leq \Omega$ were reached in the past.

Definition 4.10 (on-line diagnoser). *In the context of on-line diagnosis, a diagnoser is a place $\mathcal{D}$ of a net N, together with an order $\leq$ on the possible values of the tokens in $\mathcal{D}$, such that:*

- *$\mathcal{D}$ always contains one token;*

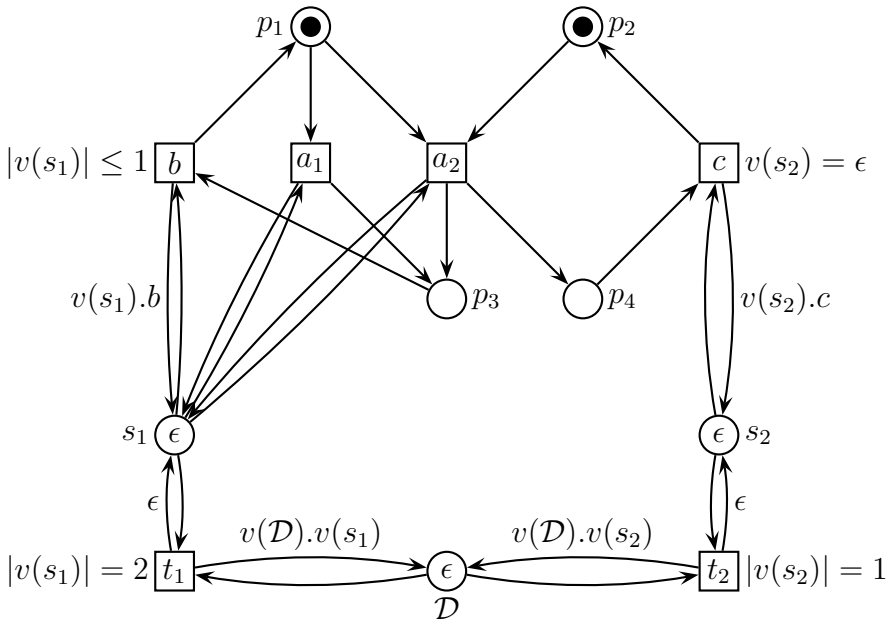


Figure 4.14: A model of a system with a sensor s_2 that sends each alarm as soon as it observes it, and a sensor s_1 that waits until it has two alarms. Sensor s_1 observes b when b fires and a when either a_1 or a_2 fires. s_2 observes c when c fires. In order to keep the figure readable, some guards are omitted.

- *For all processes $\mathcal{E}_1$ and $\mathcal{E}_2$ of N such that $\mathcal{E}_1 \subseteq \mathcal{E}_2$, $v_{\mathcal{E}_1}(\mathcal{D}) \leq v_{\mathcal{E}_2}(\mathcal{D})$;*

- *For all process $\mathcal{E}$ of N and for all $\Omega \leq v_{\mathcal{E}}(\mathcal{D})$, there exists a process $\mathcal{E}' \subseteq \mathcal{E}$ such that $v_{\mathcal{E}'}(\mathcal{D}) = \Omega$.*

Example of Representation of the Diagnoser

Our example is made of a distributed system that is observed by two sensors s_1 and s_2. The originality of this example is the way the sensors transmit the observations to the diagnoser: s_2 sends each alarm as soon as it observes it, while s_1 waits until it has two alarms. The value of the token in $\mathcal{D}$ is a word in $((a+b)^2+c)^*$. Figure 4.14 shows a model of this system with the sensors and the diagnoser.

Figure 4.15 shows the explanation of the observation (c, a, b). We notice that although e_1 is in the causal past of e_3, the diagnoser is informed of the alarm c before the alarm a.

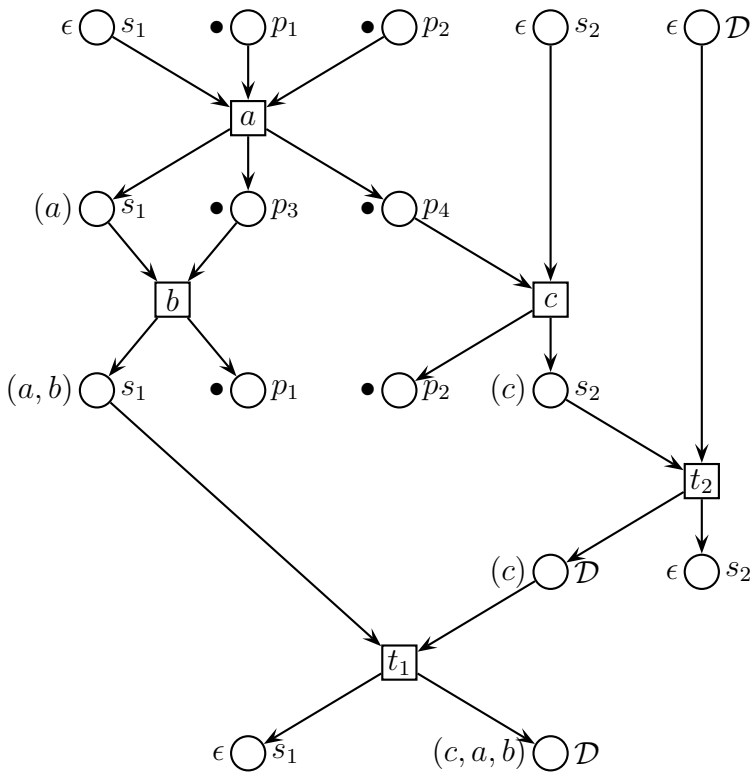


Figure 4.15: The explanation of the observation (c, a, b) according to the the model of Figure 4.14.

4.2.3 Definition of the On-Line Diagnosis

After a series of observations Ω, we can define the diagnosis as the set of processes that lead to a state where the place $\mathcal{D}$ contains the expected value, that is:

$$\{\mathcal{E} \in X_N \mid v_{\mathcal{E}}(\mathcal{D}) = \Omega\} .$$

In fact this definition can be slightly improved: a process is accepted if the diagnoser has received Ω, and the system is allowed to keep running, provided nothing more is received by the diagnoser. Probably what we want is rather the set of processes where the diagnoser has *just* received Ω, and nothing more happened after this. For this we require that the latest event e that sent information to the diagnoser, be maximal in E w.r.t. (weak) causality and temporal ordering by Θ.

Definition 4.11. *Let N be a net, where a place $\mathcal{D}$ represents the diagnoser, and let Ω be a series of observations. The diagnosis of this series of observations, according to the net N, is the set of processes defined as:*

$$Diag_N^{\mathcal{D}}(\Omega) \stackrel{\text{def}}{=} \{\mathcal{E} \in X_N \mid v_{\mathcal{E}}(\mathcal{D}) = \Omega \wedge (\forall f \in E \quad \Theta(f) \leq \Theta(e) \wedge \neg(e \nearrow f))\}$$

where $e \stackrel{\text{def}}{=} {}^\bullet(place^{-1}_{|\uparrow(E)}(\mathcal{D}))$ is the latest event that sent information to the diagnoser.

The order $\leq$ on the possible values in the diagnoser, guarantees that every explanation in the on-line diagnosis of a long observation, is a continuation of an explanation in the on-line diagnosis of the earlier versions of the observation.

Theorem 4.5. *If $\Omega_1 \leq \Omega_2$, then for all process $\mathcal{E}_2$ in $Diag_N^{\mathcal{D}}(\Omega_2)$, there exists a prefix $\mathcal{E}_1$ of $\mathcal{E}_2$ that is in $Diag_N^{\mathcal{D}}(\Omega_1)$.*

Proof. According to the last point of Definition 4.10, there exists a process $\mathcal{E}'_1 \subseteq \mathcal{E}_2$ such that $v_{\mathcal{E}'_1}(\mathcal{D}) = \Omega_1$. Denote $e \stackrel{\text{def}}{=} {}^\bullet(place^{-1}_{|\uparrow(E)}(\mathcal{D}))$. The suitable $\mathcal{E}_1$ is made of the events f of $\mathcal{E}'_1$ such that $f \in \lceil e \rceil$ or $\Theta(f) < \Theta(e)$. □

With the naïve solution to on-line diagnosis presented in Section 4.2.1, a similar theorem would not hold.

4.2.4 Use of Extended Processes in the Diagnosis

If we are dealing with a timed model, we explained in Chapter 3 that extended processes are more convenient than processes for superimposing several executions. Precisely in the context of diagnosis we want to represent sets of

explanations, which suggests to use extended processes. Then apparently we encounter a problem due to the fact that in general extended processes correspond to pre-processes rather than to processes (see Definition 3.13 and Theorem 3.13). Nevertheless the diagnosis is expressed more in terms of processes than in terms of pre-processes. Then for example the formula for the off-line diagnosis in Theorem 4.3 would not be very convenient in practice.

On the contrary, in Definition 4.11 the condition for being an explanation of Ω is made local: it suffices that one value is obtained in place $\mathcal{D}$. As a result it can be applied easily to pre-processes: even if we have no information about what occurs in a part of the net, knowing that the expected value was reached locally is sufficient to ensure that there is an explanation that is an extension of a given pre-process. Thus Definition 4.11 can be applied directly to extended processes.

In next section we explain how to compute a diagnosis by selecting events from the unfolding of the net. We will deal with extended processes in order to avoid the kind of problem that we have just mentioned.

4.3 Efficient Computation of the Diagnoses Using Guided Unfoldings

In this section, we will focus on the computation of the on-line diagnosis. We consider off-line diagnosis as a particular case of on-line diagnosis.

We remark that the solution to on-line diagnosis is given as a set of (extended) processes of a net N. This net contains a place $\mathcal{D}$ which contains the information that was collected by the diagnoser, and the explanations are the extended processes which put the system in a state where $\mathcal{D}$ contains the value that was actually observed.

As branching (extended) processes and unfoldings are intended to represent sets of (extended) processes, we would like to use the unfolding technique to compute the diagnosis. But as soon as a net N contains infinite runs, the unfolding of N is infinite. This problem was the purpose of studies about the construction of finite complete prefixes [72, 46, 67, 65], that provide finite representations of the unfolding. But as we discussed in Section 2.4, these techniques seem difficult to extend to general high-level models.

However, in the context of diagnosis, we make a very particular use of unfoldings: in many cases, the whole set of relevant extended processes involves only a finite number of extended events. Therefore these extended processes hold in a finite branching extended process, as explained in Section 4.1.2.

We will explain now how we can build this finite branching extended process without dealing with the infinite unfolding of the net. For this we have to select the extended events that appear in the diagnosis.

4.3.1 Examples of Criteria to Select the Promising Extended Events

As a first criterion, we point out a property of the extended events that appear in the on-line diagnosis. This property uses the order $\leq$ on the possible values in the place $\mathcal{D}$, and takes advantage of the fact that this value keeps growing when the system runs.

Property 4.1. *For all $\dot{\mathcal{E}} \in Diag_N^{\mathcal{D}}(\Omega)$ and for all $\dot{e} \in \dot{E}_{\dot{\mathcal{E}}}$, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq \Omega$.*

Proof. This property is a direct consequence of Definition 4.10: as $\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil} \subseteq \dot{\mathcal{E}}$, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq v_{\dot{\mathcal{E}}}(\mathcal{D}) = \Omega$. □

This remark allows us to discard the extended events of the unfolding U_N that do not satisfy this property. More precisely an extended event $\dot{e}$ can be discarded if there is no mapping $\dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \times PAR$ such that $\dot{\mathcal{E}}$ is an extended process and $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq \Omega$.

Unfortunately, this criterion is not selective enough. For example, in the case of on-line diagnosis with general requests from the diagnoser (Definition 4.9) and sensors defined as in Definition 4.7, arbitrary long extended processes can be executed without informing the diagnoser. All the extended events that appear in these extended processes satisfy Property 4.1. Nevertheless we have the feeling that it is not worth considering all of them when we build a diagnosis. For this reason we propose a more subtle criterion, that takes into account the sequence of alarms that has been recorded by each sensor.

Property 4.2. *Consider the case of on-line diagnosis with general requests from the diagnoser (Definition 4.9) and sensors $\{s_1, \ldots, s_n\}$ defined as in Definition 4.7. For all $\dot{\mathcal{E}} \in Diag_N^{\mathcal{D}}(\Omega)$, every extended event $\dot{e} \in \dot{E}_{\dot{\mathcal{E}}}$ satisfies one of the following:*

- *$v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) = \Omega$ and for all $i \in \{1, \ldots, n\}$, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(s_i) = \epsilon$.*
- *$v_{\mathcal{E}_{|\lceil \dot{e} \rceil}}(\mathcal{D})$ is a strict prefix[1] of Ω, and for all $i \in \{1, \ldots, n\}$, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(s_i) = W(s_i)$, where $W \in \{s_1, \ldots, s_n\} \longrightarrow \Lambda^*$ is the first letter of the suffix Ω' of Ω defined as $\Omega = v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \cdot \Omega'$.*

[1] Here "prefix" is used in the sense of words. Recall that Ω is a word in $(\{s_1, \ldots, s_n\} \longrightarrow \Lambda^*)^*$.

Proof. According to Property 4.1, either $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) = \Omega$ or $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D})$ is a strict prefix of Ω.

In the first case, the condition $b \stackrel{\text{def}}{=} place^{-1}_{|\uparrow(\lceil \dot{e} \rceil)}(\mathcal{D})$ is a final condition of $\dot{\mathcal{E}}$ (if an extended event $\dot{f} \in \dot{E}$ consumes b, then $\tau(\dot{f}) = \textbf{request}$, and $\dot{f}$ does not satisfy Property 4.1 since the token that $\dot{f}$ creates in $\mathcal{D}$ carries the value $v_{\dot{\mathcal{E}}_{|\lceil \dot{f} \rceil}}(\mathcal{D}) > v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D})$ which is not any more a prefix of Ω). Hence ${}^\bullet b$ is the latest extended event in $\dot{E}$ that sent information to the diagnoser (see Definition 4.11), and therefore it is maximal in $\dot{E}$ w.r.t. causality. So for all i, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(s_i) = val_{\dot{\mathcal{E}}}(({}^\bullet b, s_i)) = \epsilon$.

Otherwise, the same condition b is not maximal in $\dot{\mathcal{E}}$ and we let $\dot{f}$ be the extended event of $\dot{E}$ that consumes it. As $\dot{f}$ is the next extended event that sends information to the diagnoser, the value in the diagnoser after $\dot{f}$ is $val_{\dot{\mathcal{E}}}((\dot{f}, \mathcal{D})) = val_{\dot{\mathcal{E}}}(({}^\bullet b, \mathcal{D})) \cdot (v_{\lceil \dot{f} \rceil})_{|\{s_1,\ldots,s_n\}}$. As $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) = val_{\dot{\mathcal{E}}}(({}^\bullet b, \mathcal{D}))$ and $v_{\dot{\mathcal{E}}_{|\lceil \dot{f} \rceil}}(\mathcal{D}) = val_{\dot{\mathcal{E}}}((\dot{f}, \mathcal{D}))$, we have $v_{\dot{\mathcal{E}}_{|\lceil \dot{f} \rceil}}(\mathcal{D}) = v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \cdot (v_{\lceil \dot{f} \rceil})_{|\{s_1,\ldots,s_n\}}$. Moreover, as an extended event of $\dot{E}$, $\dot{f}$ satisfies Property 4.1. So $v_{\dot{\mathcal{E}}_{|\lceil \dot{f} \rceil}}(\mathcal{D})$ is a prefix of Ω. This imposes that $(v_{\lceil \dot{f} \rceil})_{|\{s_1,\ldots,s_n\}}$ equals the letter W mentioned in Property 4.2. And for all i, $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(s_i)$ is a prefix of $val_{\dot{\mathcal{E}}}((\dot{f}, s_i)) = W(s_i)$ since no occurrence of **request** resets s_i after $\dot{f}$ in $\lceil \dot{e} \rceil$. □

Assuming that there are enough observable transitions in the system (see Theorem 4.4), only a finite number of extended events in the unfolding of N, satisfy Property 4.2. So it is possible to compute the diagnosis using this criterion. Of course this criterion can be used in the framework of off-line diagnosis.

4.3.2 Designing Other Criteria

Until now we have only given examples of criteria to select promising extended events from the unfolding of the net. These criteria work under certain assumptions on the supervision architecture. Of course if one deals with more specific supervision architectures, one will have to build more specific criteria. Here we give general guidelines about the form that these criteria could take, and which extended events they allow to discard.

Definition 4.12. *A* criterion *is defined as a tuple* $(\mathcal{V}, \leq, \mathcal{C}_1, \mathcal{C}_2)$ *where:*

- $\mathcal{V}$ *is a set of values, which is partially ordered by* $\leq$*;*
- $\mathcal{C}_1$ *maps each extended process* $\dot{\mathcal{E}}$ *to a value* $\mathcal{C}_1(\dot{\mathcal{E}}) \in \mathcal{V}$*;*

- *$\mathcal{C}_2$ maps each possible value Ω for the token in the diagnoser, to a value $\mathcal{C}_2(\Omega) \in \mathcal{V}$;*
- *for all extended processes $\dot{\mathcal{E}}$ and $\dot{\mathcal{E}}'$ of N, if $\dot{\mathcal{E}} \subseteq \dot{\mathcal{E}}'$, then $\mathcal{C}_1(\dot{\mathcal{E}}) \leq \mathcal{C}_1(\dot{\mathcal{E}}')$;*
- *for all observation Ω, and for all extended process $\dot{\mathcal{E}} \in Diag_N^{\mathcal{D}}(\Omega)$, $\mathcal{C}_1(\dot{\mathcal{E}}) = \mathcal{C}_2(\Omega)$.*

Theorem 4.6. *For all criterion $(\mathcal{V}, \leq, \mathcal{C}_1, \mathcal{C}_2)$, for all observation Ω, for all $\dot{\mathcal{E}} \in Diag_N^{\mathcal{D}}(\Omega)$ and for all extended event $\dot{e} \in \dot{E}_{\dot{\mathcal{E}}}$, $\mathcal{C}_1(\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}) \leq \mathcal{C}_2(\Omega)$.*

Proof. As $\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil} \subseteq \dot{\mathcal{E}}$, $\mathcal{C}_1(\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}) \leq \mathcal{C}_1(\dot{\mathcal{E}}) = \mathcal{C}_2(\Omega)$. □

Therefore an extended event $\dot{e}$ can be discarded from the diagnosis if there is no mapping $\dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \times PAR$ such that $\dot{\mathcal{E}}$ is an extended process and $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq \Omega$.

Notice that we get Property 4.1 by choosing $\mathcal{V}$ as the set of possible values in the diagnoser, with the associated partial order $\leq$, $\mathcal{C}_1(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} v_{\dot{\mathcal{E}}}(\mathcal{D})$ and $\mathcal{C}_2(\Omega) \stackrel{\text{def}}{=} \Omega$.

Property 4.2 can be decomposed into a conjunction of n criteria: we attach to each sensor s_i the criterion $(\mathcal{V}, \leq, \mathcal{C}_1, \mathcal{C}_2)$ which takes into account the alarm sequences that have already been sent to the diagnoser, plus the one currently in the sensor:

- $\mathcal{V} \stackrel{\text{def}}{=} (\Lambda^*)^*$;
- $v_1 \leq v_2$ iff $|v_1| \leq |v_2|$ and the first $|v_1| - 1$ letters of v_1 coincide with the first $|v_1| - 1$ letters of v_2 and the $|v_1|^{\text{th}}$ (i.e. the last) letter of v_1 is a prefix of the $|v_1|^{\text{th}}$ letter of v_2;
- $\mathcal{C}_1(\dot{\mathcal{E}}) \stackrel{\text{def}}{=} (W_1(s_i), \ldots, W_m(s_i), v_{\dot{\mathcal{E}}}(s_i))$, where $(W_1, \ldots, W_m) \stackrel{\text{def}}{=} v_{\dot{\mathcal{E}}}(\mathcal{D})$;
- $\mathcal{C}_2(\Omega) = (W_1(s_i), \ldots, W_m(s_i), \epsilon)$, where $(W_1, \ldots, W_m) \stackrel{\text{def}}{=} \Omega$.

4.3.3 Discarding the Promising Extended Events that do Not Take Part in any Explanation

As we pointed out in Theorem 4.5, every explanation of an observation $\Omega_2 \geq \Omega_1$ is a continuation of an explanation of Ω_1. A similar remark can be formulated about the criteria of Definition 4.12: for all observation $\Omega_2 \geq \Omega_1$, for all extended event $\dot{e}$, if there is a mapping $\dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \times PAR$ such that $\dot{\mathcal{E}}$ is an extended process and $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq \Omega_1$, then $v_{\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}}(\mathcal{D}) \leq \Omega_2$. This

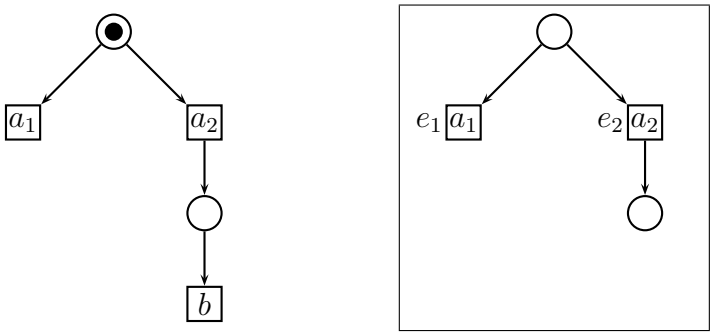


Figure 4.16: The Petri net on the left models a system which is observed by a single sensor s. The transitions a_1 and a_2 emit an alarm a; the transition b emits b. The alarms are sent to the diagnoser instantaneously. It receives a first, and then b. The branching process on the right has two maximal processes: $\{e_1\}$ or $\{e_2\}$. Both explain the first observation. But only the second one can be continued to explain the final observation. Therefore, when the alarm b is observed, the event $\{e_1\}$ can be discarded from the diagnosis.

means that if an extended event is promising for the diagnosis of Ω_1, then it is promising as well in the diagnosis of Ω_2.

However, it is not true that every extended event that takes part in an explanation of Ω_1, takes part in an explanation of Ω_2. Figure 4.16 shows an example.

It is even worth noticing that not all the extended events that are promising for Ω_1 will actually take part in an explanation of Ω_1. Intuitively we feel that such extended events should also be discarded for the diagnosis of $\Omega_2 \geq \Omega_1$.

To formalize this intuition we require an additional property of the criterion.

Definition 4.13 (strong criterion). *A criterion is* strong *if for all extended processes $\dot{\mathcal{E}}$ and and for all $v \in \mathcal{V}$ such that $\mathcal{C}_1(\emptyset) < v \leq \mathcal{C}_1(\dot{\mathcal{E}})$, there exists a unique extended event $\dot{e} \in \dot{E}$ such that $\mathcal{C}_1(\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil \setminus \{\dot{e}\}}) < \mathcal{C}_1(\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil}) = v$, and this extended event satisfies: for all Ω such that $\mathcal{C}_2(\Omega) = v$, $\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil} \in Diag^{\mathcal{D}}_N(\Omega)$.*

At the end of Section 4.3, we pointed out that we can get Properties 4.1 and 4.2 using criteria. In fact the involved criteria are strong criteria. So they can be used to discard promising extended events that do not take part in an explanation. For this we need the following theorem.

Theorem 4.7. *For all strong criterion $(\mathcal{V}, \leq, \mathcal{C}_1, \mathcal{C}_2)$, for all observations $\Omega_1 \leq \Omega_2$, and for all extended event $\dot{e}$:*

$$\left\{\begin{array}{l}\forall \dot{\mathcal{E}} : \lceil \dot{e} \rceil \longrightarrow \mathbb{R} \times PAR \\ \qquad\qquad \mathcal{C}_1(\dot{\mathcal{E}}_{|\lceil \dot{e} \rceil \setminus \{\dot{e}\}}) < \mathcal{C}_1(\dot{\mathcal{E}}) \leq \mathcal{C}_2(\Omega_1) \\ \nexists \dot{\mathcal{E}}_1 \in Diag_N^{\mathcal{D}}(\Omega_1) \quad \dot{e} \in \dot{E}_1\end{array}\right\} \implies \nexists \dot{\mathcal{E}}_2 \in Diag_N^{\mathcal{D}}(\Omega_2) \quad \dot{e} \in \dot{E}_2$$

Proof. Assume $\dot{e}$ participates in an extended process $\dot{\mathcal{E}}_2 \in Diag_N^{\mathcal{D}}(\Omega_2)$. As $\mathcal{C}_1(\emptyset) \leq \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e} \rceil \setminus \{\dot{e}\}}) < \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e} \rceil}) \leq \mathcal{C}_2(\Omega_1)$, there exists a unique extended event $\dot{e}_1 \in \dot{E}_2$ such that $\mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e}_1 \rceil \setminus \{\dot{e}_1\}}) < \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e}_1 \rceil}) = \mathcal{C}_2(\Omega_1)$ and a unique extended event $\dot{e}' \in \dot{E}_2$ such that $\mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e}' \rceil \setminus \{\dot{e}'\}}) < \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e}' \rceil}) = \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e} \rceil})$. As $\dot{e}$ is a suitable $\dot{e}'$, then $\dot{e}' = \dot{e}$. Moreover there exists also such an $\dot{e}'$ in $\lceil \dot{e}_1 \rceil$ because $\mathcal{C}_1(\emptyset) < \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e} \rceil}) \leq \mathcal{C}_2(\Omega_1) = \mathcal{C}_1(\dot{\mathcal{E}}_{2|\lceil \dot{e}_1 \rceil})$. As $\lceil \dot{e}_1 \rceil \subseteq \dot{E}_2$, $\dot{e} \in \lceil \dot{e}_1 \rceil$. Moreover, according to Definition 4.13, $\dot{\mathcal{E}}_{2|\lceil \dot{e}_1 \rceil} \in Diag_N^{\mathcal{D}}(\Omega_1)$. So $\dot{\mathcal{E}}_{2|\lceil \dot{e}_1 \rceil}$ can play the role of the $\dot{\mathcal{E}}_1$ in the theorem. □

As a conclusion, any extended event $\dot{e}$ that satisfies the condition in the curly brace in the theorem can be discarded from the diagnosis of $\Omega_2 \geq \Omega_1$. In Figure 4.16, this is the case of the event e_1.

4.4 Interest of Symbolic Unfoldings for Supervision

In Chapters 2 and 3, we have defined symbolic unfoldings for extensions of Petri nets that allow to model generic behaviors of a distributed system in a compact way. It would sometimes be possible to expand these models into low-level models as we described in Sections 2.1.3 and 3.2.3. Nevertheless we think that it is better to take advantage of the generic aspects of the model, in order to compute generic explanations. First we expect that there will be less generic explanations than instantiated explanations. But also we hope that sorting the explanations according to the way the system was modeled, might help to finally use the diagnosis. For example it will be easier to sea which components are likely to be concerned by a failure.

4.4.1 Dealing with an Incomplete Model

Furthermore it is possible that some details of the system are not precisely known when the model is built. A way to tackle this problem is to use an over approximation of the behavior of the system. Then our diagnosis approach

can be used. Simply we take the risk that some of the explanations that are exhibited, actually do not correspond to possible runs of the system.

Sometimes also only a parameter of the system is unknown, for instance the size of a buffer. Then it is possible to represent this size as data on a particular token; the behavior of the net will depend on the value of this token. But this technique supposes that we have a way to start the diagnosis from a state where the value of some tokens is unknown. This is what we describe now.

4.4.2 Starting the Diagnosis from an Unknown State

Starting the diagnosis from an unknown sate might be helpful in some cases where we are dealing with an incomplete model. It could be interesting also to use an unknown initial state when the diagnoser has to be connected to the system without stopping or resetting it. For example it may be a real nuisance to stop a part of a large communications network.

In order to start a diagnosis from an unknown state, we have to represent in the model the knowledge that we have about this unknown initial state. However if we want to use one of the models that are defined in Chapters 1, 2 and 3, we need an initial state that is totally known. For this we can start the model from a totally known fake initial state, from which a silent parameterized transition puts the system in one of the possible initial states.

It is possible to have several such parameterized transitions if the possible initial states are grouped in several families of initial states.

It would also be possible to change slightly the definitions of the models and to replace their initial state by an initial transition that consumes and reads no tokens and creates the tokens of the initial state. Using the parameter of this transition, the initial state can be parameterized. The initial event $\perp$ that is used in the coding of the (branching) processes, would become an instance of this initial transition. Apparently this method is less flexible than the previous one because there is only one initial transition.

It is worth noticing an interesting aspect of diagnosis with an unknown initial state: after a series of observations we can expect that the system reveals to the diagnoser some information about the initial values of the tokens. When the observations progress, the number of initial states that are compatible with the observations decreases. Consequently the diagnoser is able to infer some information about the past of the execution, and especially about the initial state of the system.

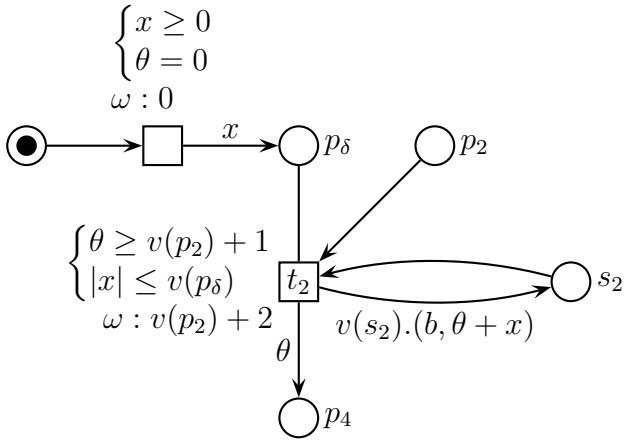


Figure 4.17: A model of a real-time system where the transitions emit alarms that contain an approximation of the firing date and the bound on the errors is not known initially.

Example

We give a short example where starting the system in an unknown state is used to deal with an incomplete model. We carry on with the problem of dated alarms with possible errors that we have tackled in Section 4.1.3. In Figure 4.12 we have proposed a construction that allows to deal with bounded errors. But we were assuming that the bound is known. Now we explain how to start the diagnosis with an unknown bound. For this we use a place p_δ which contains a token whose value is the bound on the errors. And as the initial value of the token is not known, we add a special transition, with a special initially marked place as input, so that the executions of the model start by feeding the place p_δ. Then the value of the token in p_δ can be read by any transition that emits an alarm, in order to model the possible errors on the dates of the alarms. Figure 4.17 illustrates our construction.

It is interesting to notice that if we start the diagnosis with an unknown bound δ, and if we get the observation $\{(s_1, ((a, 1.3), (a, 1.8))), (s_2, ((b, 1.7)))\}$, then we can infer from the observations that $\delta \geq 0.05$. Otherwise the observation has no explanation.

Chapter 5

Algorithms and Implementation

In this chapter we give some details concerning the way to compute some of the objects that we defined in the previous sections, that is essentially the (symbolic) unfoldings and the diagnoses.

As an introduction to this chapter we would like to make a remark: throughout this document we tried to give constructive definitions as soon as it was possible. So in most cases an algorithm can be derived directly from the definition. As an example we give an algorithm to compute the unfolding of a Petri net with read arcs using Definitions 1.9 and 1.8.

More precisely, as the unfolding is infinite in general, the procedure does not terminate, but the value of the variable U converges to the unfolding, and the value of the variable X converges to the set of processes of N.

Algorithm 5.1 (unfolding of a Petri net with read arcs (naïve version)).

procedure *unfoding*(N : *Petri net with read arcs*) :
  $U := \emptyset$; $X := \{\emptyset\}$
  for all $E \in X$
    for all $t \in T$
      for all $R \subseteq \uparrow(E)$ **for all** $C \subseteq \uparrow(E) \setminus R$
        if $Place(C) = {}^\bullet t \ \wedge \ Place(R) = \underline{t}$ **then**
          let $e = (C, R, t)$ **in**
            $X := X \cup \{E \cup \{e\}\}$
            $U := U \cup \{e\}$

The computation of $\uparrow(E)$ (or *final_conditions*(E)) can be done as follows:

function *final_conditions*(E : process)
  $B := \emptyset$
  for all $e \in E$ **do** $B := B \cup \{e^\bullet\}$
  for all $e \in E$ **do** $B := B \setminus \{{}^\bullet e\}$
  return B

Moreover the instructions of the form (**for all** $A \subseteq B$ ...) can be replaced by (**for all** $A \in 2^B$ ...), where for all finite set B, 2^B (or *subsets*(B)) is computed using the following function.

funtion *subsets*(B : set)
  if $B = \emptyset$
  then return $\{\emptyset\}$
  else
    $C := \emptyset$
    choose $b \in B$
      for all $D \in$ *subsets*$(A \setminus \{b\})$
        $C := C \cup \{D, D \cup \{b\}\}$
    return C

As this type of algorithm consists only in rewriting the definition in a way that looks a bit more like a program, we will avoid doing this for the other definitions, rather point out some properties that allow us to write more efficient algorithms.

5.1 Use of D_N

First we remark that in Algorithm 5.1, the events are generated several times. An event e is generated as many times as the number of processes E that can be continued with the event e. In fact enumerating the processes as we do in Algorithm 5.1 is a very unefficient way to build the unfolding. As a matter of fact, when the set of parameters is infinite, it is even not always possible to enumerate the processes of high-level models.

On the contrary, as the unfolding is a set of events taken from a set D_N, it would be useful to check directly whether an event of D_N is in the unfolding or not. After that we simply need to enumerate the events of D_N and add the good ones in the unfolding.

If we have a function *valid_event*? that checks whether an event $e \in D_N$ is in the unfolding or not, then we can use it to build the unfolding as follows:

Algorithm 5.2 (unfolding of a Petri net with read arcs using *valid_event?*).

procedure *unfoding*(N : *Petri net with read arcs*) :
  $U := \emptyset$
  for all $e \in D_N$
    if *valid_event?*(e) **then** $U := U \cup \{e\}$

The idea of using a function to check the validity of an event can be found for example in [85] in a context without read arcs. In his tool punf [65], Khomenko uses a different technique, which requires to code not only the past of the events, but also their possible futures, in order to detect the conflicts. In our approach only pointers to the causal predecessors are required.

We will see in Sections 5.2.1, 5.2.2 and 5.2.3 how to write a function *valid_event?*. But first it is worth giving some details about the way to enumerate the events of D_N, in order to make the instruction (**for all** $e \in D_N$ ...) precise.

Using directly Definition 1.7 for D_N, we can replace

for all $e \in D_N$ *do_something*(e)

by:

$D_N := \emptyset$; $B := \perp^\bullet$ (* the value of B is $\bigcup_{e \in D_{N\perp}} e^\bullet$ *)
do forever
  let *oldB* $= B$ **in**
    for all $t \in T$
      for all $C \subseteq$ *oldB* **for all** $R \subseteq$ *oldB*
        if *Place*(C) $= {}^\bullet t \ \wedge$ *Place*(R) $= \underline{t}$ **then**
          let $e = (C, R, t)$ **in**
            if $e \in D_N$ **then**
              $D_N := D_N \cup \{e\}$
              $B := B \cup e^\bullet$
              do_something(e)

The interest of the variable B is to avoid recomputing $\bigcup_{e \in D_{N\perp}} e^\bullet$ at each iteration of the main loop. At the beginning of each iteration of the main loop, the value of B is copied in the variable *oldB*. Otherwise the value of B would be updated inside a (**for all** $C \subseteq B$ **for all** $R \subseteq B$ ...), which we prefer to avoid.

Unfortunately this version is not efficient because all the events are computed again at each iteration.

5.1.1 Enumeration of D_N Using the Depth of the Events

An improvement consists in adding only the events that read or consume at least one "new" condition, that is one condition that was created by the previous iteration. This amounts to creating at each iteration the events of a given depth, where the depth is defined as: $depth(\bot) \stackrel{\text{def}}{=} 0$, and for $e \in D_N$, $depth(e) \stackrel{\text{def}}{=} \max_{b \in {}^\bullet\underline{e}} depth({}^\bullet b)$.

For this the sets C and R are built by adding a new condition b to one[1] of two sets C' and R' of conditions that are not necessarily new. We obtain the following:

$D_N := \emptyset$
$B_1 := \bot^\bullet$ (* the conditions created during the previous iteration *)
$B := B_1$ (* the old conditions, plus those in B_1 *)
do forever
  let $B_2 = \emptyset$ **in** (* the conditions created in the current iteration *)
  for all $t \in T$
    for all $C' \subseteq B \setminus \{b\}$ **for all** $R' \subseteq B \setminus \{b\}$
      for all $(C, R) \in \{(C' \cup \{b\}, R'), (C', R' \cup \{b\})\}$
        if $Place(C) = {}^\bullet t \ \wedge \ Place(R) = \underline{t}$ **then**
          let $e = (C, R, t)$ **in**
            if $e \in D_N$ **then**
              $D_N := D_N \cup \{e\}$
              $B_2 := B_2 \cup e^\bullet$
              $do\_something(e)$
  $B_1 := B_2$; $B := B \cup B_1$

Now an event e is created only during the iteration of **do forever** that follows immediately the creation of the newest conditions in ${}^\bullet\underline{e}$.

But still e can be created several times during this iteration, if several conditions in ${}^\bullet\underline{e}$ are new: it is created once for each new condition. Even though in general e is created only a rather small amount of times, this forces us to check if $e \in D_N$ before evaluating $do\_something(e)$. Moreover this test is required also if we want to store the sets D_N and B_2 in structures that do not contain several versions of a single element. So we prefer to be sure that each event is created only once in order to remove this test.

[1] In fact if we want to respect strictly Definition 1.7, we should also try to add b to both C' and R', but this never yields valid events.

5.1.2 Preventing Multiple Creations of a Single Event

In order to prevent multiple creations of a single event, we order totally the newly created conditions so that at any time only one condition is new. To do this we store B in a sequence (or in practice an adjustable array for instance), and we remember the index of the newest condition. We denote $B[i]$ the i^{th} element of B, and $B[1,i]$ the first i elements of B. We remark that the variable D_N is not useful any more.

$B := ()$
for all $b \in \bot^\bullet$ **do** $B := B \cdot b$
for i **from** 1 **to** ∞ **do**
  for all $t \in T$
    for all $C' \subseteq B[1,i-1]$ **for all** $R' \subseteq B[1,i-1]$
      for all $(C,R) \in \{(C' \cup \{B[i]\}, R'), (C', R' \cup \{B[i]\})\}$
        if $Place(C) = {}^\bullet t \ \wedge \ Place(R) = \underline{t}$ **then**
          let $e = (C,R,t)$ **in**
            for all $b \in e^\bullet$ **do** $B := B \cdot b$
            $do\_something(e)$

5.1.3 Final Optimizations

Finally we see that it is not worth enumerating all the C' and the R' because C and R will have to satisfy $Place(C) = {}^\bullet t \wedge Place(R) = \underline{t}$. This can be taken into account: the set of possible C' and R' are built by successive additions of conditions that correspond to the places of ${}^\bullet t$ or $\underline{t}$ that are missing. In order to optimize this, it is better to have a way to find all the conditions that correspond to a given place. We do not detail this optimized search for C' and R'.

To conclude this section, when we enumerate D_N in order to build the unfolding using Algorithm 5.2, we can avoid enumerating all D_N. For this we remark that the events of U read and consume only conditions that are created by events of U. So we do not put the other conditions in B. This amounts simply to replacing (**let** $e = (C,R,t)$ **in** ...) by

let $e = (C,R,t)$ **in**
  if $valid\_event?(e)$ **then**
    for all $b \in e^\bullet$ **do** $B := B \cdot b$
    $U := U \cup \{e\}$

Then we get a quite efficient way to compute the unfolding, provided we have a function $valid\_event?$. This is the purpose of the next sections.

5.2 Deciding if an Event is in the Unfolding

We have described algorithms to enumerate a set that includes the events that appear in the unfolding of a model. In order to build the unfolding it suffices now to have a function that decides which of these events take part in the unfolding. For this we have to check both structural properties and properties that express the fact that the guards of the transitions are satisfied when they fire. Of course in low-level models, only the structural properties have to be checked.

5.2.1 Checking structural conflicts

In the case of unfoldings without read arcs, the only structural property that has to be checked for an event e is that no condition in the causal past of e is consumed by several events. This is why we tackle this problem first.

Nets Without Read Arcs

The algorithm that we present here works only for models without read arcs. It consists in a backward depth first search traversal from e following the causal relation. For all event in $f \in \lceil e \rceil$, $status(f)$ can be **not_visited**, **visit_started** or **visit_finished** according to the progress of the traversal.

The goal of the traversal is to check that no condition in the causal past of e is consumed by two different events. For all condition $b \in \bigcup_{f \in \lceil e \rceil} (^\bullet f)$, $consumer(b)$ indicates the event of $\lceil e \rceil$ that consumes b, as soon as this event has been visited. The function $consumed(b)$ returns **true** iff an event that consumes b was already visited.

Algorithm 5.3 (structurally valid event (in processes without read arcs)).

```
function structurally_valid_event?(e : event in D_N) :
  initialize consumer
  initialize status
  let function traverse(f : event in ⌈e⌉) :
    status(f) := visit_started
    for all b ∈ •f
      if consumed(b)
      then
        if consumer(b) ≠ f
        then return false from structurally_valid_event?
      else
        consumer(b) := f
        if status(•b) = not_visited then traverse(•b)
    status(f) := visit_finished
  in traverse(e)
  return true
```

Notice that the traversal terminates when $f = \perp$ because ${}^\bullet f = \emptyset$.

We mention that this algorithm allowed us to improve dramatically the efficiency of a tool that was implemented in the framework of an industrial collaboration with Alcatel. This tool uses the unfolding technique for fault diagnosis in Submarine Line Terminal Equipment. Before our improvement, checking structural conflicts was done using another technique based on concurrency tables.

Nets With Read Arcs

When we deal with read arcs, we have to refine the previous algorithm in order to check that $\nearrow$ is acyclic on $\lceil e \rceil$. Therefore we add a backward depth first traversal (coded in the function *traverse2*) following the *conditional* causal relation $\nearrow$ over the events of $\lceil e \rceil$. For this we need to access the predecessors of an event f w.r.t. $\nearrow$. These predecessors are the events that create a condition of ${}^\bullet \underline{f}$, plus those that read a condition of ${}^\bullet f$. But the latter are not coded directly in the events, as they depend on the whole process. This means that we need a preliminary traversal (coded in the function *traverse1*) to build this hidden information. For each condition b, *traverse1* collects the set of events that read b. This information is stored in the function *readers*. At the end of the first traversal, *readers*(b) indicates

the set of events in $\lceil e \rceil$ that read b. As the preliminary traversal follows the unconditional causal relation, it is similar to the traversal of Algorithm 5.3. Therefore we use *traverse1* both to prepare *traverse2* and to check that no condition in the causal past of e is consumed by two different events.

Algorithm 5.4 (structurally valid event (in processes with read arcs)).

function *structurally_valid_event?*(e : *event in* D_N) :
  initialize *consumer* **and** *readers*
  initialize *status*
  let function *traverse1*(f : *event in* $\lceil e \rceil$) :
    $status(f) :=$ **visit_started**
    for all $b \in {}^\bullet f$
      if $consumed(b)$
      then
        if $consumer(b) \neq f$
        then return false from *structurally_valid_event?*
      else
        $consumer(b) := f$
        if $status({}^\bullet b) =$ **not_visited then** *traverse1*$({}^\bullet b)$
    for all $b \in \underline{f}$
      $readers(b) := readers(b) \cup \{f\}$
      if $status({}^\bullet b) =$ **not_visited then** *traverse1*$({}^\bullet b)$
    $status(f) :=$ **visit_finished**
  in *traverse1*(e)
  initialize *status*
  let function *traverse2*(f : *event in* $\lceil e \rceil$) :
    $status(f) :=$ **visit_started**
    for all $f' \in \left(\bigcup_{b \in {}^\bullet \underline{f}} {}^\bullet b\right) \cup \left(\bigcup_{b \in {}^\bullet f} readers(f)\right)$
      if $status(f') =$ **not_visited then** *traverse2*(f')
      else if $status(f') =$ **visit_started**
      then return false from *structurally_valid_event?*
    $status(f) :=$ **visit_finished**
  in *traverse2*(e)
  return true

5.2.2 Expanded Unfoldings: Checking that the Guards are Satisfied

In this section we adopt the notations of Section 2.2.5.

In the context of expanded unfoldings, for each condition c, the value $tok(c)$ is instantiated. And, besides the structural properties, checking that an event e is in an expanded unfolding amounts to checking that for all event $f \stackrel{\text{def}}{=} (C, R, b, x) \in \lceil e \rceil$, the values $tok(c)$ of the conditions c in ${}^\bullet\underline{f}$ are compatible with the guard of the transition $tok(b)$. This can be done by a simple traversal of the causal past of e.

Algorithm 5.5.

function *guards_satisfied?*$(e \stackrel{\text{def}}{=} (C, R, b, x) : \textit{event in } D_N)$:
  for all $b \in {}^\bullet\underline{e}$
    if ${}^\bullet b \neq \bot \wedge \neg$*guards_satisfied?*$({}^\bullet b)$ **then return false**
  return $\alpha(tok(b))(x, tok(C), tok(R))$

We notice that this algorithm is recursive. And if it is implemented naïvely, it may compute the same results several times, which is very unefficient. A good implementation stores the results for later reuse using the classical techniques of dynamic programming. Thus a single traversal of the causal past of the condition is sufficient to get the result.

But we notice also that if all the conditions in ${}^\bullet\underline{e}$ are created by events that are in the unfolding, then the loop can simply be forgotten.

5.2.3 Symbolic Unfoldings: Checking that the Guards are Satisfiable

In this section we adopt the notations of Section 2.1.

In the context of symbolic unfoldings, for each condition b, the value $val_{\mathcal{E}}(b)$ of the token in b depends on the parameters of the events in the causal past of b. Thus the guard of the transition that corresponds to an event may or may not be satisfied, depending on the values of the parameters. Therefore, in order to decide if an event e is in a symbolic unfolding we have to check that there exists a function $\mathcal{E} : \lceil e \rceil \longrightarrow PAR$ such that for all event $f \in \lceil e \rceil$, the values $val_{\mathcal{E}}(b)$ of the conditions b in ${}^\bullet f$ are compatible with the guard of the transition $\tau(f)$, as stated by Theorem 2.3.

This means that we have to decide the satisfiability of a set of constraints that comes from the guards of the transitions that fire in the past of e. The type of constraints to solve depends essentially on the language that is used to write the guards $\alpha(t)(\dots)$ and the functions $\beta(t)(\dots)$.

We will not focus on the techniques that can be used to solve such or such type of constraints. We consider that the algorithms that compute symbolic unfoldings can be seen as meta-algorithms, where a parameter is

an algorithm that solves the constraint satisfaction problems that arise from the unfolding. Instead we will explain how to build the constraint on the that have to be solved for an event e.

We denote this constraint $constr_{\mathcal{E}}(e)$. In Theorems 2.3 and 2.2 it is expressed as:

$$constr_{\mathcal{E}}(e) \stackrel{\text{def}}{=} (\forall f \in \lceil e \rceil \quad \alpha(\tau(f))(\mathcal{E}(f), tok_{\mathcal{E}}({}^\bullet f))) \ .$$

The variables of this constraint are the $\mathcal{E}(f), f \in \lceil e \rceil$. We remind that $tok_{\mathcal{E}}(b)$ is simply a shorthand for $(place(b), val_{\mathcal{E}}(b))$, which makes $tok_{\mathcal{E}}({}^\bullet e)$ be a mapping from ${}^\bullet\tau(e) = Place({}^\bullet e)$ to V.

We first explain how to construct the symbolic expression of $val_{\mathcal{E}}(b)$. We denote $\langle val_{\mathcal{E}}(b) \rangle$ the symbolic expression of $val_{\mathcal{E}}(b)$ using the variables $\mathcal{E}(f), f \in \lceil b \rceil$. Similarly we write $\langle constr_{\mathcal{E}}(e) \rangle$ for the symbolic expression of $constr_{\mathcal{E}}(e)$ using the variables $\mathcal{E}(f), f \in \lceil e \rceil$. These symbolic expressions will be computed using the symbolic expressions of the guards $\alpha(t)(x, v)$ and the functions $\beta(t)(x, v)(p)$ with $x \in PAR$, $v : {}^\bullet t \longrightarrow V$ and $p \in t^\bullet$. We denote again $\langle \alpha(t)(x, v) \rangle$ the symbolic expression of $\alpha(t)(x, v)$ using the variables x and $v(p'), p' \in {}^\bullet t$ and $\langle \beta(t)(x, v)(p) \rangle$ the symbolic expression of $\beta(t)(x, v)(p)$ using the same variables x and $v(p'), p' \in {}^\bullet t$.

For example, in the example of Figure 2.1, we have

$$\langle \alpha(t_1)(x, v) \rangle = \text{“}x \leq v(p_1)\text{”}$$
$$\langle \beta(t_1)(x, v)(p_2) \rangle = \text{“}x\text{”} \ .$$

We need also a notation for the α-conversion: we write $expr[v := expr']$ for the expression $expr$ with every free occurrence of v in $expr$ replaced with $expr'$. We sometimes do several α-conversions “in parallel”. Then we write $expr[v_1 := expr_1, \ldots, v_n := expr_n]$ or even $expr[v_i := expr_i, i \in \{1, \ldots, n\}]$. For example

$$\langle \alpha(t_1)(x, v) \rangle [x := \text{“}\mathcal{E}(e_1)\text{”}, v(p_1) := \text{“2”}] = \text{“}\mathcal{E}(e_1) \leq 2\text{”}$$
$$\langle \beta(t_1)(x, v)(p_2) \rangle [x := \text{“}\mathcal{E}(e_1)\text{”}, v(p_1) := \text{“2”}] = \text{“}\mathcal{E}(e_1)\text{”}$$

The symbolic expression $\langle val_{\mathcal{E}}(b) \rangle$ can now be computed as follows:

Algorithm 5.6 (construction of $\langle val_{\mathcal{E}}(b) \rangle$).

function *symb_expr_val*$_{\mathcal{E}}(b : condition)$:
  if ${}^\bullet b = \bot$
  then $\langle \mathcal{S}_0(place(b)) \rangle$
  else
    let $p = place(b)$ **and** $e = {}^\bullet b$ **and** $t = \tau(e)$ **in**
      $\langle \beta(t)(x, v)(p) \rangle [x :=$ “$\mathcal{E}(e)$”,
                $v(p') := symb\_expr\_val_{\mathcal{E}}(place^{-1}_{|{}^\bullet e}(p')), p' \in {}^\bullet t]$

Here again we notice that this algorithm is recursive and may compute the same results several times if it is implemented naïvely. However a single traversal of the causal past of the condition b is sufficient if we store the results for later reuse.

Now we can compute $\langle constr_{\mathcal{E}}(e)\rangle$ as follows. $expr_1$ *AND* $expr_2$ denotes the symbolic expression of the conjunction of $expr_1$ and $expr_2$.

Algorithm 5.7 (construction of $\langle constr_{\mathcal{E}}(e)\rangle$).

function $symb\_expr\_constr_{\mathcal{E}}(e : event\ in\ D_N)$:
  $constr := \langle \alpha(\tau(e))(x,v)\rangle[x := \text{“}\mathcal{E}(e)\text{”},$
    $v(p) := symb\_expr\_val_{\mathcal{E}}(place^{-1}_{|^\bullet e}(p)), p \in {}^\bullet\tau(e)]$
  for all $b \in {}^\bullet e$
    if ${}^\bullet b \neq \perp$ **then** $constr := constr\ AND\ symb\_expr\_constr_{\mathcal{E}}({}^\bullet b)$
  return $constr$

We have two remarks about this algorithm.

1. After Algorithm 5.5 we pointed out that when deciding the satisfaction of the guards in the causal past of an event of an expanded unfolding, the loop over the conditions b in ${}^\bullet e$ can be forgotten if we know that all the ${}^\bullet b$ are in the unfolding. Indeed the fact that ${}^\bullet b$ is in the unfolding ensures that the guards in $\lceil b \rceil$ are satisfied.

   But in the symbolic context, the fact that ${}^\bullet b$ is in the unfolding ensures only that $\langle constr_{\mathcal{E}}({}^\bullet b)\rangle$ is satisfiable. But it may be that the conjunction of all the $\langle constr_{\mathcal{E}}({}^\bullet b)\rangle$ for $b \in {}^\bullet e$ is not satisfiable even if each of these guards is satisfiable.

   So in the case of symbolic unfoldings, the loop over the conditions of ${}^\bullet e$ cannot be removed.

2. Once again our algorithm may compute some constraints several times. But here even storing the results will not help that much. The main problem is that some constraints will be repeated anyway. For example, if an event e consumes two conditions b_1 and b_2, and both of them are created by the same event $f \stackrel{\text{def}}{=} {}^\bullet b_1 = {}^\bullet b_2$, then the constraint $\langle constr_{\mathcal{E}}(f)\rangle$ will appear twice in $\langle constr_{\mathcal{E}}(e)\rangle$. If we want to avoid this we can rewrite the algorithm using a classical graph traversal, like in Algorithm 5.3.

Algorithm 5.8 (construction of $\langle constr_{\mathcal{E}}(e)\rangle$ (second version)).

function *symb_expr_constr*$_{\mathcal{E}}(e : event\ in\ D_N)$:
  initialize *status*
  constr := "**true**"
  let function *traverse*$(f : event\ in\ \lceil e\rceil)$:
    status(f) := **visit_started**
    let $t = \tau(f)$ **in**
      constr := *constr AND*
        $\langle\alpha(t)(x,v)\rangle[x :=$ "$\mathcal{E}(f)$",
          $v(p) := symb\_expr\_val_{\mathcal{E}}(place^{-1}_{|^\bullet f}(p)), p \in {}^\bullet t]$
    for all $b \in {}^\bullet f$
      if *status*$({}^\bullet b)$ = **not_visited then** *traverse*$({}^\bullet b)$
    status(f) := **visit_finished**
  in *traverse*(e)
  return *constr*

5.3 Implementation

In this section we want to describe briefly some implementations that we did. We deliberately chose to write very concise programs in order to make them as close as possible to the algorithms that we have described above.

This is a reason why we chose to program in Lisp. Another reason is that it is very easy to build, manipulate, evaluate and even compile expressions on the fly, which is helpful when we deal with the expressions of guards $\alpha(t)(\dots)$ and the functions $\beta(t)(\dots)$.

As many different models are presented in this document, we had to choose to implement the unfoldings for only a part of them. We wanted to choose models that include most of the features that appear in the extensions of Petri nets that we have described. We wanted to deal with read arcs (see Section 1.4), colors (see Section 2.1), dynamic structure (see Section 2.2) and time (see Chapter 3).

Moreover the association of symbolic techniques with unfoldings is one of the original points of this thesis, so we wanted to implement symbolic unfoldings. But, as we already pointed out in Sections 2.3.2 and 5.2.3, symbolic unfoldings requires to solve symbolic constraints, and this forces us to constrain the type of the symbolic expressions of the guards $\alpha(t)(\dots)$ and of the functions $\beta(t)(\dots)$.

On the other hand we have explained in Sections 2.3.2 and 5.2.2 that any computable $\alpha(t)(\dots)$ and $\beta(t)(\dots)$ can be used if we deal with expanded

unfoldings. We decided to emphasize this point by implementing expanded unfoldings of dynamic nets as they are described in Section 2.2.5. And for this we did not restrict the expression of the $\alpha(t)(\dots)$ and $\beta(t)(\dots)$ at all, since the user can define these functions using any kind of Lisp expression. This expression is simply evaluated when we unfold the net in order to compute the values of the tokens that correspond to the conditions. The only restriction that we had to do here concerns the set of parameters. As pointed out at the end of Section 2.3.1, the events in the expanded unfolding may not be enumerable if the set of parameters is not enumerable. In fact we considered only finite sets of parameters.

Once we have proved that it is possible to deal with any computable expressions in the context of expanded unfoldings, we can restrict ourselves to simpler expressions that allow to compute symbolic unfoldings. As we wanted to implement timed unfoldings we chose the model of time Petri nets to prove the feasibility of symbolic unfoldings of timed models. The symbolic unfoldings are computed according to the non trivial local firing condition that we have defined in Section 3.3.5. The timed constraints are computed like in Algorithm 5.8. Then these symbolic expressions are manipulated so that their satisfiability can finally be checked by a simplex function that we reused.

To sum up, we have two main programs for unfoldings. The first one implements expanded unfoldings of the very general model of dynamic nets (defined in Section 2.2.5), where the $\alpha(t)(\dots)$ and $\beta(t)(\dots)$ can defined using any Lisp expression. And the other program implements symbolic unfoldings of time Petri nets, using the non trivial local firing condition that we have defined in Section 3.3.5. Each program consists of about 800 lines of Lisp code (we include the code that is used for the construction of the timed constraints but not the simplex function that is called to solve these systems).

Chapter 6

Conclusion

In this thesis we have extended the unfolding technique to high-level extensions of Petri nets and we have shown the interest of applying these techniques to the problem of supervision in distributed systems. This original use of unfoldings was suffering from the difficulty of specifying real systems with low-level models like Petri nets. Although Petri nets give a satisfactory representation of concurrency, modeling real distributed systems requires to use their higher-level extensions.

6.1 Results

6.1.1 Symbolic Unfoldings

We have introduced unfoldings as an efficient way to represent a set of executions of a true concurrency model. We have extended the unfolding technique to high-level models and shown how to deal with detailed specifications of the system, where some aspects linked to the manipulation of data are also modeled.

The definition of symbolic unfoldings allows to keep in the unfoldings the generic aspects of the model. As a result we obtain a compact representation of the executions, that are grouped in symbolic processes.

We have realized that these unfolding techniques can even be applied to very high level models where the representation of the system takes into account strongly dynamic architectures. We have illustrated the use of dynamic nets to model Web services.

Finally we have tackled the problem of unfoldings of timed true concurrency models. Among the numerous instances of such models, we have identified those that resist the unfolding-based approach. The combination

of *urgency* with non locality of choices, called *confusion*, forces to refine the usual techniques. Time Petri nets are a widespread example of timed model where these difficulties arise. In order to resolve them, we have rewritten the semantics of time Petri nets. The concurrent operational semantics that we obtain reduces concurrency only when this is required by the time constraints. Thus we still take advantage of the partial order semantics, that avoid the computation of the interleavings and the explosion of the state space.

6.1.2 Application to Supervision of Distributed Systems

Concerning supervision, we have advocated the use of symbolic unfoldings of high-level models. This method allows a compact representation of the execution and highlights the causality relation between the events, which makes it easier to find the causes of the failures.

First we have tackled the problem of off-line diagnosis, where the diagnoser has to explain a set of alarm sequences recorded by several independent sensors. For this we have proposed a method based on the representation of the supervision architecture inside the model. For instance the sensors are modeled by places where a token carries the information about the observed alarms. Thus the diagnosis problem amounts to searching states where the tokens that model the sensors carry the expected information.

Then we have extended this method to the problem of on-line diagnosis, where the diagnoser receives observation continuously. Each time now observations are received, it tries to extend the previous explanations.

In both cases of off-line and on-line diagnosis, we have explained how to select among the events of the unfolding a finite set of events that are likely to take part in explanations. For this our approach relies on the fact that there are sufficiently many observable actions in the system.

6.1.3 Algorithms and Implementation

Finally we have detailed algorithms that allow to build the unfoldings and the diagnoses efficiently. The first part of these algorithms handles the construction of the structure of the unfolding, and applies to both unfoldings of low-level models and symbolic unfoldings of high-level models. The second part deals with the problems that arise because of the manipulation of symbolic expressions and constraints in the framework of symbolic unfoldings.

These algorithms were implemented, we have described the implementations that were done.

6.2 Perspectives

One of the most natural extensions of this work is the problem of distributed diagnosis. It happens that very large distributed systems may include a lot of sensors. Consequently, collecting all the observed alarms is very difficult, and inferring information about the whole system is probably hopeless. Indeed this requires to use an enormous model of the whole system and to cope with huge amounts of observations. A solution is to distribute the diagnosis over several diagnosers in the network. Then each local diagnoser infers as much information as possible from local observations. But at some point the information that was inferred by one diagnoser may help another diagnoser. For example a failure is pointed out by one diagnoser and it may have affected other parts of the net.

Distributed diagnosis was studied in [51, 50, 49] as an extension of the approach of [8, 9] based on unfoldings of low-level Petri nets. But this method involves complex operations on nets and unfoldings. For this reason the most recent developments of this approach were embedded in a categorical framework [48]. This enables concise definitions of complex notions like products of unfoldings.

Concerning the extension of our diagnosis approach based on symbolic unfoldings of high-level models, we were warned about the little categorical framework for high-level Petri nets. Nevertheless the first attempts to define a sound categorical framework for colored Petri nets and symbolic unfoldings are encouraging. We expect that both symbolic unfoldings and expansion of colored Petri nets into low-level Petri nets can be defined as coreflections. This would allow to easily express relations between expanded and symbolic unfoldings for example.

In the timed framework, we strongly believe that our unfolding technique based on the definition of a concurrent operational semantics can be applied to other true concurrency models. We have already tackled symbolic unfoldings of networks of timed automata [28, 29]. But we think that there is more to do in this direction.

Moreover the definition of causal semantics for timed models poses new interesting problem: it allows to compare the causal semantics of several models. In Section 3.1.3 we have mentioned recent work about the comparison of timed models. But the comparisons are done according to the sequential semantics of the models. We would like to have results that take into account the preservation of concurrency.

We are also convinced that the idea of using a concurrent partial order semantics is interesting beyond unfoldings. Other formal methods take ad-

vantage of concurrency; the use of these methods often encounters the same kind of problems as the ones that we described for timed unfoldings. And therefore many of these methods are restricted to timed models without urgency or confusion.

As a possible extension of our diagnosis approach we can mention the following: in our method for on-line diagnosis, the diagnoser keeps in memory all the possible explanations of the observations since the diagnosis started. Of course this may require too much memory and also decrease the time performance of the diagnosis. And although longer observations allow to refine the set of explanations of the earliest observations, one should expect that long observations lead to large diagnoses. Resetting the diagnoser at regular intervals allows to free memory, but prevents us from inferring information from the correlation of new alarms with earlier ones. Therefore it would be of practical interest to design techniques that allow to forget the parts of the early diagnosis that are not likely to help in the future.

Another long term research topic is the combination of diagnosis with active supervision of distributed systems. Then we can consider the idea of equipping distributed systems with mechanisms that make them reconfigure automatically when some failures are detected.

Finally, in Section 4.4 we have mentioned the interest of symbolic unfoldings to handle the problem of incomplete models. Further research may carry on with this idea and consider learning not only parameters of the model, but larger parts of it. Then one could think of diagnosers that investigate themselves the structure of the system they supervise in order to construct a model of the system.

Bibliography

[1] P. A. Abdulla, S. Purushothaman Iyer, and A. Nylen. Unfoldings of unbounded petri nets. In *Computer Aided Verification*, pages 495–507, 2000.

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

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

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

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

[6] R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR*, Lecture Notes in Computer Science. Springer, 2006. To appear.

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

[8] A. Benveniste, E. Fabre, C. Jard, and S. Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[9] A. Benveniste, S. Haar, E. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. In *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2003.

[10] B. Bérard, F. Cassez, S. Haddad, O. H. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *ATVA'2005*, volume 3707 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2005.

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

[12] B. Bérard, F. Cassez, S. Haddad, O. H. Roux, and D. Lime. When are timed automata weakly timed bisimilar to time Petri nets? In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3707 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2005.

[13] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.

[14] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *IFIP Congress*, pages 41–46, 1983.

[15] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS*, pages 442–457, 2003.

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

[17] E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theor. Comput. Sci.*, 55(1):87–136, 1987.

[18] E. Best and C. Fernández. *Nonsequential Processes: A Petri Net View*, volume 13 of *EATCS monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[19] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. A class of composable high level petri nets with an application to the semantics of $B(PN)^2$. In *Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 103–120. Springer, 1995.

[20] B. Bieber and H. Fleischhack. Model checking of time Petri nets based on partial order semantics. In *CONCUR*, volume 1664 of *LNCS*, pages 210–225, 1999.

[21] P. Bouyer, S. Haddad, and P.-A. Reynier. Extended timed automata and time Petri nets. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 91–100. IEEE Computer Society Press, 2006.

[22] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06) — Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.

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

[24] M. Boyer and M. Diaz. Non equivalence between time Petri nets and time stream Petri nets. In *PNPM*, pages 198–207. IEEE Computer Society, 1999.

[25] M. G. Buscemi and V. Sassone. High-level Petri nets as type theories in the join calculus. In *FoSSaCS*, volume 2030 of *LNCS*, pages 104–120, 2001.

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

[27] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[28] F. Cassez, T. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. In *ATVA*, Beijing, ROC, october 2006. To appear. Extended version available in IRCCyN/CNRS Technical Report RI-2006-4.

[29] F. Cassez, T. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. Technical Report RI-2006-4, IRCCyN/CNRS, 2006.

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

[31] T. Chatain. Diagnostic pour les systèmes distribués dynamiques partiellement observables. In *Colloque Francophone sur l'Ingénierie des Protocoles (CFIP) – papiers courts*, 2005.

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

[33] T. Chatain and C. Jard. Models for the supervision of web services orchestration with dynamic changes. In *AICT/SAPIR/ELETE*, pages 446–451. IEEE Computer Society, 2005.

[34] T. Chatain and C. Jard. Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In *FORMATS*, volume 3829 of *LNCS*, pages 193–207, 2005.

[35] T. Chatain and C. Jard. Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. Technical Report RR-1740, Institut National de Recherche en Informatique et en Automatique (INRIA), 2005.

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

[37] S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In *Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 186–205. Springer, 1993.

[38] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.1. Technical report, BEA Systems, IBM and Microsoft, May 2003.

[39] D. de Frutos-Escrig, V. V. Ruiz, and O. M. Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN*, pages 187–206, 2000.

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

[41] M. Diaz and P. Sénac. Time stream Petri nets: A model for timed multimedia information. In *Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 219–238. Springer, 1994.

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

[43] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COOCS*, pages 10–21, 1995.

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

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

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

[47] Javier Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2-3):151–195, 1994.

[48] E. Fabre. On the construction of pullbacks for safe Petri nets. In *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2006.

[49] E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Dynamic Systems*, 15(1):33–84, 2005.

[50] E. Fabre, A. Benveniste, S. Haar, C. Jard, and A. Aghasaryan. Algorithms for distributed fault management in telecommunications networks. In *ICT*, volume 3124 of *Lecture Notes in Computer Science*, pages 820–825. Springer, 2004.

[51] E. Fabre and V. Pigourier. Monitoring distributed systems with distributed algorithms. In *41st IEEE Conference on Decision and Control (CDC), Las Vegas*, 2002.

[52] H. Fleischhack and E. Pelz. Hierarchical timed high level nets and their branching processes. In *ICATPN*, volume 2679 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2003.

[53] H. Fleischhack and E. Pelz. High level branching processes for high level Petri nets. In *Proceedings of High Performances Computing Conference*, pages 246–253. SCS, 2003.

[54] H. Fleischhack and C. Stehno. Computing a finite prefix of a time Petri net. In *ICATPN*, volume 2360 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2002.

[55] S. Genc and S. Lafortune. Distributed diagnosis of discrete-event systems using Petri nets. In *ICATPN*, volume 2679 of *Lecture Notes in Computer Science*, pages 316–336. Springer, 2003.

[56] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[57] U. Goltz and W. Reisig. The non-sequential behavior of Petri nets. *Information and Control*, 57(2/3):125–147, 1983.

[58] K. Heljanko. *Combining Symbolic and Partial Order Methods for Model Checking 1-Safe Petri Nets*. PhD thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Espoo, Finland, February 2002.

[59] H. Hulgaard and S. M. Burns. Efficient timing analysis of a class of Petri nets. In *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 423–436. Springer, 1995.

[60] R. Janicki and M. Koutny. Semantics of inhibitor nets. *Inf. Comput.*, 123(1):1–16, 1995.

[61] C. Jard, T. Chatain, and P. Bourhis. Diagnostic temporel dans les systèmes répartis à l'aide de dépliages de réseaux de petri temporels. *Journal Européen des Systèmes Automatisés*, 39(1-3):351–365, 2005.

[62] C. Jard, T. Chatain, and P. Bourhis. Diagnostic temporel dans les systèmes répartis à l'aide de dépliages de réseaux de petri temporels. In *MSR*. Hermes, 2005.

[63] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer-Verlag, 1995.

[64] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *CoopIS*, pages 310–321, 1998.

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

[66] V. Khomenko and M. Koutny. Branching processes of high-level Petri nets. In *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 458–472. Springer, 2003.

[67] V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.

[68] T. Kitai, Y. Oguro, T. Yoneda, E. Mercer, and C. Myers. Partial order reduction for timed circuit verification based on a level oriented model. *IEICE Trans.*, E86-D(12):2601–2611, 2001.

[69] R. Langerak and E. Brinksma. A complete finite prefix for process algebra. In *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 1999.

[70] J. Lilius. Efficient state space search for time Petri nets. In *MFCS Workshop on Concurrency '98*, volume 18 of *ENTCS*. Elsevier, 1999.

[71] D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. In *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2004.

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

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

[74] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of place/transition Petri nets. *Mathematical Structures in Computer Science*, 7(4):359–397, 1997.

[75] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6):545–596, 1995.

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

[77] M. Nielsen, V. Sassone, and J. Srba. Properties of distributed timed-arc petri nets. In *FSTTCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 280–291. Springer, 2001.

[78] M. Nielsen, V. Sassone, and J. Srba. Towards a notion of distributed time for petri nets. In *ICATPN*, volume 2075 of *Lecture Notes in Computer Science*, pages 23–31. Springer, 2001.

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

[80] L. Popova-Zeugmann. On time Petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 27(4):227–244, 1991.

[81] B. Pradin-Chézalviel, R. Valette, and L. A. Künzle. Scenario duration characterization of t-timed Petri nets using linear logic. In *IEEE PNPM*, pages 208–217, 1999.

[82] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[83] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1985.

[84] V. V. Ruiz, D. de Frutos-Escrig, and F. Cuartero. Timed processes of timed Petri nets. In *Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 490–509. Springer, 1995.

[85] S. Römer. *Theorie und Praxis der Netzentfaltungen als Grundlage für die Verifikation nebenläufiger Systeme*. PhD thesis, Technische Universität München, 2000.

[86] M. Sampath, R. Sengupta, K. Sinnamohideen, S. Lafortune, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Trans. on Systems Technology*, 4(2):105–124, 1996.

[87] A. L. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri net unfolding. In *DAC*, pages 59–62. ACM Press, 1996.

[88] J. Sifakis. Performance evaluation of systems using nets. *Net Theory and Applications*, 84:307–319, 1980.

[89] J. Srba. Timed-arc Petri nets vs. networks of timed automata. In *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 385–402. Springer, 2005.

[90] W. Vogler. Executions: A new partial-order semantics of Petri nets. *Theor. Comput. Sci.*, 91(2):205–238, 1991.

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

[92] W. Vogler, A. L. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *CONCUR*, volume 1466 of *LNCS*, pages 501–516, 1998.

[93] J. Winkowski. Algebras of processes of timed Petri nets. In *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 1994.

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

[95] J. Winkowski. Processes of timed Petri nets. *Theoretical Computer Science*, 243(1-2):1–34, 2000.

[96] J. Winkowski. Reachability in contextual nets. *Fundamenta Informaticae*, 51(1-2):235–250, 2002.

Index