Taming Concurrency using Representatives.

Blaise Genest

▶ **To cite this version:**

Blaise Genest. Taming Concurrency using Representatives.. Other [cs.OH]. Rennes 1, 2016. ⟨tel-01939514⟩

HAL Id: tel-01939514

https://hal.science/tel-01939514v1

Submitted on 29 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Université Rennes I Ecole Doctorale Matisse – Année 2016

Thèse
pour l'obtention du diplôme de l'
Habilitation à diriger des recherches
présentée par

Blaise GENEST

le 8 Mars 2016

Taming Concurrency using Representatives.

soutenue publiquement devant le jury suivant:

Ahmed Bouajjani, Professeur, Université Paris Diderot	*Président du Jury*
Patricia Bouyer, Directrice de Recherche CNRS	*Rapporteure*
Javier Esparza, Professeur, Technische Universität München	*Rapporteur*
Eric Fabre, Directeur de Recherche INRIA	*Examinateur*
Madhavan Mukund, Professeur, Chennai Mathematical Institute	*Rapporteur*
K. Narayan Kumar, Professeur, Chennai Mathematical Institute	*Examinateur*
C.-H. Luke Ong, Professeur, University of Oxford	*Examinateur*
Sophie Pinchinat, Professeure, Université Rennes 1	*Examinatrice*

Contents

Chapter 1

Introduction

Forework: *The introduction will be split into two chapters. This first chapter will introduce the overall setting of my research, as well as give the big picture of what this thesis will discuss in depth and what it will just sketch.*

Chapter 2 on page 9 will give an overview of the main technique used in this thesis, go through other competing techniques, as well as introduce the chapters and describe their content.

Further, the bibliography is partitioned into two parts: first the external bibliography after Chapter 7 on page 71 will list all the papers cited in this thesis, at the exclusion of those I am a coauthor of. The self-citations will be part of the second bibliography after Chapter 7 on page 78, and the citations themselves will carry the names of the venue (conference or journal) in small letters together with the year, e.g. [jdeds14b], to differentiate them from the "normal" citations, e.g. [MW01].

Internet of Things, Cloud, Cloud computing, DDOS, (massively) multicore, GPGPU ... Most buzzwords of this decade have a dimension synonymous with distributed systems. The issue with distributed systems is twofold. First, it is hard for a human to understand all the possible behaviors of a distributed system. E.g. it is easy to understand what happens when the different components of a distributed system evolve somehow at a comparable speed. But when one component is much faster than another, it can create situations which were not foreseen by the designers, and it is a potential source of bugs. Even more than for sequential systems, this calls for powerful tools to help the design of distributed systems.

Formal methods propose many tools to find bugs in a design, be it based

on theorem prover, abstract interpretation, tests, or model-checking. In this thesis, we will discuss the latter to analyze distributed systems. In model-checking, a formal model of the system is used (usually in the form of some transition system), and a specification of what the design should do or should avoid is given (usually in the form of some logical formula). The shortcoming with these approaches is that in case bugs are found, it is the task of the designer to modify the design in order to correct the bugs, potentially introducing other bugs, etc.

Another approach is to automatically produce a system fulfilling by construction a given specification. Formal methods provide at least two such methods. The first one is certified program produced using a proof assistant (COQ...). The second one, which we will discuss in this thesis, is implementation and control of a model given a specification (*a la* Pnueli Rosner [PR90]). Here, an architecture describing the peers and the possible communications between them is given, together with potentially non-controllable actions which have to be allowed by the system. The aim is to produce a system with this architecture and satisfying a specification (equality or inclusion of language).

The second challenge which needs to be tackled when dealing with distributed system is the combinatorial explosion. Indeed, it is easy to describe with few small components (hence in a very compact way) a very large global system. Analyzing the components one by one is not sufficient to understand their composition, and advanced techniques need to be deployed in order to have techniques which are sound and complete for the global system, without having to consider every single interleaved behavior of this global system. In other word, it is necessary to have techniques to *tame the concurrency*. One such technique will be described in length in this thesis, namely *set of representatives* (see Chapter 2 on page 9). The potential of this technique will be demonstrated in several contexts, from model checking to implementation of distributed systems. Comparisons with other important techniques will also be performed.

I have chosen to describe in this thesis the work using set of representatives to tame the concurrency, which sums up a main part of my research activity performed after my PhD (defended in 2004). The associated publications are [i&c06, jcss06, fi07, am&ai09, tcs09, i&c10, ipl12, forte06, icalp06, icalp07, concur07, fossacs08, concur08, icalp10, ictac12, icalp13, fsttcs13, fossacs15, icatpn16]. Some important contributions are however *not* de-

scribed in this thesis; in particular the following two research directions:

The first line of work concerns the application of formal method ideas, in particular methods to handle distributed systems, to the analysis of data centric workflows. This line of work took place in the ANR Docflow project (2006-2009). I was responsible for one central workpackage of this ANR project. In this context, I supervised with Albert Benveniste the PhD of Debmalya Biswas [seke08, hase08, xsym09, jdeds14b]. He defended successfully in January 2009, and he is now a research fellow at Swisscom. Some of this work [hase08] has been done together with Thomas Gazagnaire, a PhD student at that time (he defended in 2008). Also, [xsym09] has been performed following the internship of Ashwine Jiwane, supervised by Debmalya and myself. In the ANR Docflow context, a technique to model check Active XML [ABGM09] documents have also been developed [atva08, fsttcs10], applying well-structured transition system techniques [FS01]. In particular, with the help of Zhilin Wu, a postdoc at that time, we explained [fsttcs10] how infinite data can be handled, using bounded treewidth graphs.

Last, since 2008, I have been involved in the verification of stochastic systems [lics09, lics12, jacm15, jdeds14b, cmsb11, tcbb12, qest14, cmsb15]. This will be described in more length in the conclusive chapter 7 on page 65. In this context, I am leading ANR Stoch-MC (2014-2018). I am co-supervising with Wieslaw Zielonka the PhD of Burno Karelovic (the defense is planned in early 2016), as well as supervising (with a *derogation*) the PhD of Matthieu Pichené, which started in December 2014.

Chapter 2

Overview of the Thesis

2.1 Introduction

This thesis tackles several classes of systems: sequential automata, networks of automata (mainly in chapter 3 on page 19), Asynchronous Automata (chapter 4 on page 31), communicating automata (chapters 5 on page 45), and (Time-Constrained) Message Sequence Graphs (chapter 5 on page 45 and chapter 6 on page 53).

We will not describe in this preliminary chapter every class of systems. Instead they will be introduced in the first chapter using each one of them.

We will give the formal definition for the usual class of sequential automata. This class can be used as a common semantics to all the systems considered. We will also introduce in this chapter *Mazurkiewicz traces*, describing a fixed commutation architecture (that is, the fixed set of processes, the set of actions, and the distribution of actions on the processes). This will allow us to settle the notations which will be used throughout the thesis. Last but not least, we will explain in this chapter the main technique we advocate in this thesis, applied on the commutation structure we introduce with Mazurkiewicz traces.

2.2 Automata

A (finite) alphabet Σ is a (finite) set of letters denoted $a, b \ldots$. We denote by Σ^* the set of words, that is of finite sequences of letters of Σ. The empty word will be denoted ϵ. An automaton over the finite alphabet Σ is a structure

$(S, \to, I, F)$ where

- S is the (possibly infinite) set of states,
- $\to \subseteq S \times \Sigma \times S$ is the transition relation. For each $(s, a, s') \in \to$, we also write $s \xrightarrow{a} s'$. It expresses that action a can be played from state s, in which case the new state is s',
- $I \subseteq S$ is the set of initial states and
- $F \subseteq S$ is the set of final states.

An accepting path (usually denoted by ρ, σ) of an automaton is a finite sequence $s_0 a_1 s_1 \cdots a_n s_n$ such that $s_0 \in I$, $s_n \in F$ and for all $i < n$, $s_i \xrightarrow{a_{i+1}} s_{i+1}$. An automata $A = (S, \to, I, F)$ defines a set of words $\mathcal{L}(A)$, called the language of A such that $w \in \mathcal{L}(A)$ iff there is an accepting path $\rho = s_0 a_1 s_1 \cdots a_n s_n$ of A with $w = a_1 \cdots a_n$. One such automaton can be found on Figure 3.1 on page 23, with states $s_0, \ldots, s_6$, with transition labeled by letters on alphabet $\Sigma = \{a, b, c, z\}$. The set of languages of finite automata is the class of regular languages. We will not recall the usual equivalence between regular, rational and recognizable languages for finite words, nor the equivalence with monadic second order logic.

2.3 Mazurkiewicz Traces

Words express executions of sequential systems. In order to represent executions of distributed systems, one may use Mazurkiewicz traces. Namely, a Mazurkiewicz traces [Maz86, DR95] is a pair (Σ, I) with a finite alphabet Σ and an (independence) relation $I \subseteq \Sigma \times \Sigma$ such that I is symmetric and $(a, a) \notin I$ for all $a \in \Sigma$. We will denote by D the relation $\Sigma \setminus I$, called the dependence relation. We can represent (Σ, D) by its dependency graph, with a link between two letters a, b whenever $(a, b) \in D$. The relation I induces an equivalence relation $\sim_I \subseteq \Sigma^* \times \Sigma^*$ between two words: u, v are equivalent iff there exists a sequence of words $u_0 \cdots u_n$ such that $u_0 = u$, $u_n = v$ and for all $i \in [1, n]$, there exist $v, w \in \Sigma^*$ and $(a, b) \in I$ with $u_{i-1} = vabw$ and $u_i = vbaw$. For all $u \in \Sigma^*$, we denote by $[u]_I$ the equivalence class containing u. Usually, I will be clear from context. In this case, we will avoid mentioning it to avoid clutter. For instance, consider alphabet $\Sigma = \{a, b, c, z\}$

with the independence relation $I = \{(a,b),(b,a),(c,b),(b,c)\}$, that is b is independent with both a and c. We have $bcab \sim bbca$.

With each equivalence class $[u]_I$, one can associate a *labeled partial order* $(E, \leq, \lambda)$ with:

- let $E = e_1, \ldots, e_n$ such that n is the number of letters of u,
- $\lambda : E \to \Sigma$ such that $\lambda(e_i)$ is the i-th letter of u,
- We define $\leq$ the transitive closure of $\prec$, where $\prec$ is defined with $e_i \prec e_j$ iff $i < j$ and $(e_i, e_j) \in D$ (ie. $(e_i, e_j) \notin I$).

It is easy to check that no matter what v we pick with $u \sim_I v$, the labeled partial order is isomorphic to the one obtained with $[u]_I$. As from the labeled partial order, one can obtain easily the trace $[u]_I$, the labeled partial order and the trace are in bijection, and we interchangeably use the labeled partial order or the equivalence class representation to define a trace.

For instance, for $[bbca]_I$, the labeled partial order has four events e_1, e_2, e_3, e_4 with $\lambda(e_1) = \lambda(e_2) = b$, $\lambda(e_3) = c$, and $\lambda(e_4) = a$. We have $e_1 < e_2$ and $e_3 < e_4$ and e_1, e_2 is in parallel with e_3, e_4.

2.4 Network of Automata

The simplest class of systems exhibiting Mazurkiewicz trace behavior is a *network of automata* $(A_p)_{p \in \mathcal{P}}$ (intuitively one for each process $p \in \mathcal{P}$), over (non-disjoint) alphabet Σ_p. The automata synchronize over common actions: for all $a \in \Sigma$, we denote by $\mathrm{dom}(a) = \{p \mid a \in \Sigma_p\}$ the fact that a is a *synchronization* between processes in $\mathrm{dom}(a)$. The associated trace alphabet is defined as (Σ, I), with $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$ and $I = \Sigma^2 \setminus D$ for the dependency D defined as $(a,b) \in D$ iff $\mathrm{dom}(a) \cap \mathrm{dom}(b) \neq \emptyset$. The local states are just the union over all $p \in \mathcal{P}$ of the set of states of every automaton A_p.

We now give the semantics of $(A_p)_{p \in \mathcal{P}}$ in term of a global automaton A. We denote $A_p = (S_p, \to_p, I_p, F_p)$ for all $p \in \mathcal{P}$. The set of states of A is $S = \prod_{p \in \mathcal{P}} S_p$. A state $s \in S$ is a global state $s = (s_p)_{p \in \mathcal{P}}$ with s_p a state of A_p for each $p \in \mathcal{P}$. We have $(s_p)_{p \in \mathcal{P}} \xrightarrow{a} (s'_p)_{p \in \mathcal{P}}$ iff $s_p \xrightarrow{a_i} s'_p$ for $p \in \mathrm{dom}(a)$, and $s_p = s_p$ otherwise. We have $I = \prod_{p \in \mathcal{P}} I_p$ and $F = \prod_{p \in \mathcal{P}} F_p$. A run of A is called a global run of $(A_p)_{p \in \mathcal{P}}$. A word labeling a global run is called an *interleaving* or a *linearization*. The language of a network of automata is the language of A.

Consider a network of two automata A_p and A_q. The automaton A_p associated with process p is made of states $s_1, \ldots, s_n$, and for all $1 \leq i < n$, $s_i \xrightarrow{a} s_{i+1}$ and $s_{i+1} \xrightarrow{b} s_i$, Automata A_q is very similar to A_p, with letter a' replacing a and letter b' replacing b. That is A_p, A_q have no synchronizing letter. The automaton A associated with (A_p, A_q) is depicted on the left of Figure 3.2 on page 26.

2.5 Set of Representatives as main technique.

2.5.1 Closure by commutation

For a set $S \subseteq \Sigma^*$ of words, we will denote by $[S]$ the closure by commutation of S, that is $[S] = \{w \mid \exists w' \in S, w' \sim_I w\}$. This notion is crucial in our work. Indeed, for a system distributed over the alphabet (Σ, I), its language L is closed by commutation, that is $[L]_I = L$. Moreover, two words $w \sim_I w'$, w, w' have the same distributed action on the system: each process sees the same sequence of local states and local actions when either of those words are executed. The language of a network of automaton is closed by commutation, for the commutation defined by its associated trace alphabet.

In particular, the final global state after each word is the same, although intermediate global states may differ: this can be noticed only with a global supervisor with exact global time, and cannot be noticed in a distributed way. That is, as long as the property one wants to check is closed by commutation as well (if w satisfies the property, then any $w' \sim w$ will also satisfy the property), two equivalent words $w \sim_I w'$ will be *indistinguishable*, that is either both will satisfy the property, or neither of them will. Several logics generate formulae which are always closed by commutation: LTrL [TW02], local LTL [DG04], MSO on traces [MM01], Template MSCs [fossacs04]. Notice that it is not always the case. For instance, some (sequential) LTL property can distinguish between $w \sim_I w'$ (there is an execution with letter a followed by letter b, while $(a, b) \in I$), as well as some property over atomic proposition on states of the system (can we reach a global state with property p_1 holding on process 1 and p_2 on process 2?).

These observations are the basis of *Partial Order Reduction* (*POR* for short) techniques. In POR, the idea is to reduce the number of runs to consider. Instead of considering all the global runs of the systems, one either reduces the number of states (Persistent set [God94], ample set [Pel94], stub-

born set [Val92]), or the number of transitions (sleep set [God94], edgelean reduction [am&ai09]) to explore.

The POR techniques contrast with other techniques handling Partial Order directly without exploring (part of) the global runs of the system. *Unfoldings* [McM95], [ERV02] are one of these techniques, considering the (infinite) event structure produced by distributed systems, and giving ways to present it in a finite way by cutting the branches. Lately, a new technique based on representing the partial order as a graph and using tree automata techniques to handle the graph [CG14] (for the numerous cases where the graph has bounded treewidth) has been used, allowing to scale decidability to more complex distributed systems (using stacks, data, and possibly time).

Using any of these Partial Order techniques allows handling the high complexity of distributed systems. We will develop the presentation of some of the Partial Order Reduction techniques in the following.

2.5.2 Set of Representatives

The first idea is that when considering model checking a bounded system, made of a finite set S of words closed by commutation ($[S] = S$), one can avoid some of them, as long as a set of representatives R is checked. We call R a *set of representatives* for S whenever $[R] = [S]$. Notice that as $R \subseteq [R]$ and $S = [S]$, it means that $R \subseteq S$. This strategy to model check a bounded system has been used in several works. Among them, we can mention Learning Grey box systems [forte06] and Bounded model checking [WYKG08, KWG09].

2.5.3 Lexical Normal Form

In order to chose a minimal set of representatives, our strategy is to consider only words in some *normal form* nf. The normal form of a word w is a word $nf(w) \sim w$, such that for all $w' \sim w$, $nf(w) = nf(w')$. Considering only those words w with $w = nf(w)$ allows to consider a set of representatives $R = nf(S)$ such that there does not exists $w, w' \in R$ with $w \sim w'$.

The normal form we will mainly consider in this Thesis is the *Lexical Normal Form* (*LNF* for short), used in particular in the central theorem of Ochmanski [Och95] characterizing a class of rational trace languages equivalent with regular trace languages.

Let $\ll$ be an order on the letters of Σ. The word $u = a_0 \cdots a_n$ is said to be in *Lexical Normal form (LNF)* if for all $v = b_0 \cdots b_n$ with $u \sim v$, we have $b_0 \cdots b_{i-1} = a_0 \cdots a_{i-1}$ and $a_i \ll b_i$ for some i (i is unique). We define $LNF(u)$ to be the (unique) word $v \sim u$ in Lexical Normal Form. That is, $LNF(u)$ is the first word w for the lexicographic order such that $w \sim u$.

We describe now different framework where representatives can be used to reduce the complexity of distributed systems.

2.6 Using Representatives

We introduce here the 4 next chapters of this thesis, one for each application, each focused on one class of systems.

- Chapter 3 on page 19 describes how to reduce the complexity of model checking of *regular distributed systems* (network of automata), using partial order reduction of transitions.
- Chapter 4 on page 31 explains how to *reduce the complexity* for *synthesis of distributed system (Deterministic Asynchronous Automata).*
- Chapter 5 on page 45 describes how to obtain *decidability* for model checking of *non regular message passing systems* (Communicating Finite State Machines, MSC-Graphs).
- Chapter 6 on page 53 extends the *decidability* to some *non-regular timed* distributed systems (Timed Constrained MSC-Graphs).

2.6.1 Partial Order Reduction of Transitions

It seems reasonable to extend the method with set of representatives from model checking of a finite set of runs [forte06, WYKG08, KWG09] to the model checking of systems with finite states (that is of a regular language closed by commutation). Model checking a regular system boils down to checking the existence of a loop around one state in a set in a finite automaton. This in turn can be reduced for simplicity to the accessibility of some state. Hence, the question is basically to explore all the reachable states from a given state. What we need is to explore a set of words $R \subseteq S$, such that every reachable state of the system is reached using R, but some transitions

may not be explored. This has been for instance used in the so called *Partial Order Reduction* (of transitions), either with SleepSets [God94] or with EdgeLean reduction [icalp07, am&ai09].

Model checking a regular set, even against a formula closed by commutation, is not as straightforward as on a finite set of words. Indeed, one needs to combine the partial order reduction with a finite search, which avoids loops, e.g. Breadth First Search (BFS), Depth First Search (DFS). So far, no partial order reduction considering at most one representative per trace is known to work with Depth First Search. In Chapter 3 on page 19, we will discuss more on this issue. We will provide examples where LNF fails with DFS. This is similar to what happens with unfoldings [EKS08], where no adequate order (which is used to truncate the infinite unfolding) is known to be compatible with DFS (unlike BFS). We will propose a non-minimal reduction compatible with DFS. On the other hand, BFS and LNF are compatible, because they consider somehow the same lexicographic ordering of paths, unlike DFS and LNF.

2.6.2 Set of Representatives as Model

We explained how using set of representatives can help reducing the number of transitions considered. One can also use set of representatives to represent in a regular way a non-regular language. A language L which is closed by commutation can be represented by a regular set R of representatives, such that $[R] = L$. This is actually a variant definition of *rational trace languages*. For instance, with $\Sigma = \{a, b\}; I = \{(a, b), (b, a)\}$, the regular set $R = (ab)^*$ represents the language $[R] = \{w \mid |w|_a = |w|_b\}$, that is the set of words with as many a's as b's, which is non-regular. That's also the way MSC-graphs (see chapter 5 on page 45) can be presented, although it is not the original definition.

Now, model checking can be directly applied to the set R of representative representing the language $L = [R]$, granted that the property to verify is expressed in a closed by commutation form (formulas of e.g. MSO on traces [MM01], LTrL [TW02], Template MSCs [fossacs04]). The answer of the model checking procedure for R will hold for the full language $L = [R]$ as well. Last, intersections of two system represented by a set of representatives was also considered [jcss06]. The two sets of representatives need not be compatible, and the emptiness of intersection is undecidable in general. Decidability is obtained for a large subclass of systems (namely globally co-

operative, also called weakly star connected), and the complexity becomes polynomial time for the more restricted class of local choices systems [jcss06].

2.6.3 Distributed Timed systems

The situation becomes more complicated when (global) time is involved. Indeed, time defines a total order which breaks the partial order defined by the concurrent model. In presence of time, most decidability results hold on regular systems. The technique is to generate a global automaton accepting the language, before using the usual region abstraction [AD94] to handle time. Because of the explicit generation of every state, this is computationally very intensive in practice.

In Chapter 6 on page 53, we will explain how model checking can be applied for many *non-regular timed systems* represented by a regular set of representatives R, namely timed constrained MSC-graphs [GMN09]. In general, model-checking is undecidable[GMN09]. For the reasonably large subclass of non-drifting system, we show that the emptiness problem is however decidable. We will provide two different techniques to do so. First, we show that as for the untimed case, it suffices to model check a regular set of representatives R', which is built from R. This demonstrates the flexibility of this technique to accommodate different generalization, as time. We will also show an *ad-hoc* algorithm based on Fourier-Motzkin constraint elimination, preserving the partial order. This technique uses a symbolic representation, and allows to check whether a system is non b-drifting.

2.6.4 Synthesis of Distributed Systems

Designing distributed systems is notoriously difficult, as it is hard to grasp all the possible behaviors of a distributed system. One way to facilitate the design is to a posteriori verify automatically the design with e.g. some model checking algorithms we describe in this thesis. In case a bug is discovered, the designer could change its design. Another possibility is to automatically produce a distributed system satisfying a specification. The specification can take many forms: logical formula (PDL [BKM10], EMSO [BL06]), or a global (regular) specification describing the set of interleavings.

Producing a distributed system equivalent with a global specification is not easy. Intuitively, a process p does not know what the other processes are doing concurrently. In a deterministic distributed system, p needs to

be ready for anything done in parallel. In a non-deterministic distributed system, p could guess what happens in parallel, and check whether it is what happened, creating a deadlock if it did not guess correctly. Unlike with sequential systems, deadlocks are hard to remove a posteriori, and it is usually preferred to generate a deterministic system. The complexity to determinize a system can be doubly exponential for distributed systems [KMS94].

In Chapter 4 on page 31, we tackle the synthesis of deterministic distributed systems complying to a global regular specification. Representatives are also useful to manage the complexity for synthesis in this case. Indeed, instead for a process of keeping every possible combination of behaviors for the other processes, it can keep only a small number of representatives which allows reconstructing every possible combination.

Notice that the set of representative factors are not kept explicitely, as it would potentially require unbounded memory to keep. Only a summary of the factors is kept in terms of the states of the automaton reached, and the processes involved.

In chapter 4 on page 31, we also explain how to use this construction in order to compute in an efficient way what a process knows about others. We extend this to open systems, where not all actions are controllable.

Chapter 3

Partial Order Reduction for Network of Automata

3.1 Introduction

The problem of *state space search* is fundamental to many areas of computer science, such as, e.g., AI and formal methods. Often, the state space to be searched is huge, so optimizing the search is an important issue. This will be the first application of the set of representatives technique. The setting is the one of a network of automata where each local automaton is known, and the global state space (not initially known) needs to be entirely explored. We show how to use commutativity to achieve full coverage of the states, while traversing a relatively small number of edges, defining a new Partial Order Reduction of transitions. One previously known such Partial Order Reduction of transitions is the sleep set method [GW91]. The transitions pruned are actually the same, even though the algorithms (and its overhead) differ. The overhead of our method is arguably very light.

3.2 Preliminaries

Let (Σ, I) be a trace alphabet (see section 2.3 on page 10). We recall that $\sim$ denotes the trace equivalence relation between words of (Σ, I). For example, for $\Sigma = \{a, b\}$ and $I = \{(a, b), (b, a)\}$ we have $abbab \sim ababb \sim bbbaa$.

For $\ll$ a total order on letters, we extend $\ll$ to the lexicographic order on words in a standard way, i.e., by setting $v \ll vu$ and $vau \ll vbw$ for any

v, u, $w \in \Sigma^*$ and any a, $b \in \Sigma$ such that $a \ll b$. We recall (see section 2.5.3 on page 13) that the *Lexical Normal Form* of a word $w \in \Sigma^*$ is $LNF(w)$ the least word under the relation $\ll$ that is equivalent to w [Och95].

3.3 Generating the LNF set of representatives.

Let $S \subseteq \Sigma^*$ be a finite set of words closed by commutation, that is $[S] = S$, with $[S] = \{w' \mid \exists w \in S, w \sim w'\}$. Assume that S is prefix-closed. In many applications (bounded model checking of a network of automata [WYKG08], checking compliance of an automata with a black box, important in learning an automaton [forte06]), one has to check some property over a prefix closed set S, which is also closed by commutation. When the property is closed by commutation, which is often the case, it suffices to check the property over a set R of representative, that is with $[R] = S$. We explain here how to generate such a set R in an efficient way, choosing $R = LNF(S)$ the set of words of S in lexical normal form.

We assume S is given as a tree, as it is closed by prefix. The algorithm to generate $R = LNF(S)$ uses *summaries* (also called last appearance records in other contexts), which are data structures that represent the order in which the last appearance of each letter occurred in the string. Using summaries provides a significant computational advantage, as they are much cheaper to store and analyze than the strings themselves.

Definition 1 *Given a string σ, let $\alpha(\sigma)$ denote the set of letters occurring in σ. A* summary *of σ is the total order $\prec_\sigma$ on the letters from $\alpha(\sigma)$ such that $a \prec_\sigma b$ if and only if the last occurrence of a in σ precedes the last occurrence of b in σ. That is, $a \prec_\sigma b$ if σ can be represented as $vaubw$, where $v \in \Sigma^*$, $u \in (\Sigma \setminus \{a\})^*$, $w \in (\Sigma \setminus \{a, b\})^*$.*

We will also represent a summary $\prec_\sigma$ as a word over Σ in which each letter occurs at most once, and $a \in \Sigma$ appears before $b \in \Sigma$ if and only $a \prec_\sigma b$. For instance, in this notation, the summary of the word $w = bcbabza$ is $cbza$. Clearly, the length of a summary is always at most $|\Sigma|$.

We will now formulate a criterion that allows us to check whether adding a letter to a string in LNF results in a string that is also in LNF.

Lemma 1 *Consider a string $\sigma \in \Sigma^*$ in LNF and a letter $a \in \Sigma$. The string σa is not in LNF if and only if there is a letter $b \in \alpha(\sigma)$ such that $a \ll b$*

and for each c such that $b \preceq_\sigma c$ we have $(a, c) \in I$.

Intuitively, Lemma 1 means that σa is not in LNF iff we can move a backwards past an appropriate suffix of σ to obtain a string that is lexicographically smaller than σ.

Now, in order to generate the set $R = LNF(S)$, it suffices to prune top down the tree representing S, keeping in each node n of the tree, from the root to the leaves, the summary $\prec_\sigma$ of the word σ labeling the branch of the tree leading to n. If at any point, the summary is σa with a letter $b \in \alpha(\sigma)$ such that $a \ll b$ and for each c such that $b \preceq_\sigma c$ we have aIc, then we delete the subtree rooted at node n from the tree. The result is a tree representing the set $R = LNF(S)$. Some results of this procedure can be found in [forte06]. We report them here. Different sets are tested, with a different number of letters and maximal length of words in S. We report the number of words in millions, as well as the time in minutes (or seconds if indicated by s in the case of DAS) to generate them, either by just reading the tree, or by reading and pruning the tree on the fly.

example	letters	length	Tree	LNF
simple_2	2	18	$.5M$ (9)	$.5M$ (9)
simple_4	6	9	$7.2M$ (22)	$2.3M$ (3)
DAS	12	4	$.25M$ (13s)	$.13M$ (8s)
COMA(1)	8	6	$9.8M$ (33)	$5.7M$ (16)
COMA(2)	8	7	$46M$ (190)	$25M$ (75)

These examples do not exhibit too much parallelism, having only 4 processes at most. The number of words considered can be reduced up to 3 times, with a mean reduction of 2 times. The overhead due to checking the summary is negligible. LNF is never slower than checking the whole tree without the summary overhead.

3.4 LNF for Global finite-state Systems?

We give a generic definition of global finite state system, which applies to the global state space of network of automata but is more general. A *global finite-state system* over the trace alphabet (Σ, I) is a tuple $\mathcal{A} = \langle S, s_0, T \rangle$ where:

- S is a finite set of *global states*.
- $s_0 \in S$ is the *initial global state*.
- Σ is a finite *alphabet of actions*.
- $T \subseteq S \times \Sigma \times S$ is a *labeled transition relation*. We write $s \xrightarrow{a} s'$ when $(s, a, s') \in T$.
- $I \subseteq \Sigma \times \Sigma$ is the symmetric and irreflexive independence relation on actions.

We say that the system has the *diamond property* if for every global state s and any pair of independent actions $(a, b) \in I$ it is the case that if $s \xrightarrow{a} q \xrightarrow{b} r$ then there exists a global state $q' \in S$ such that $s \xrightarrow{b} q' \xrightarrow{a} r$. For instance, the global finite-state systems associated with networks of Automata (see Section 2.4 on page 11) and Asynchronous Automata (introduced in Chapter 4 on page 31) have the diamond property. Every global finite-state system considered in this chapter is thus assumed to have the diamond property. Notice that if the system has the diamond property and $u \sim v$, then $s \xrightarrow{u} r$ if and only if $s \xrightarrow{v} r$.

However, we do not require the *forward diamond* property:

> If $s \xrightarrow{a} q$ and $s \xrightarrow{b} q'$ then there exists a state $r \in S$ such that $s \xrightarrow{a} q \xrightarrow{b} r$.

From now on, we will call a global state just a state. An action a is *enabled* from a state $s \in S$ if there exists some state $s' \in S$ such that $s \xrightarrow{a} s'$. We say that a path $\rho = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$ is *loop-free* or *simple* if $s_i \neq s_j$ for all $i \neq j$. A *labeling* $\ell(\rho)$ of a path $\rho = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$ is given by $\ell(\rho) = a_1 \cdots a_n$. We extend the arrow notation from transitions to paths, denoting the above path by $s_0 \xrightarrow{a_1 \ldots a_n} s_n$.

Most of the search algorithms to be presented are based on depth-first search (DFS), which provides some advantages over breadth-first search (e.g., linear time detection of loops using Tarjan's algorithm [Tar71]). It uses a hash table to check whether a state has been previously visited.

```
proc Dfs(q);
     local variable q';
     hash(q);
```

```
     forall q -a-> q' do
            if q' is not hashed then Dfs(q');
end Dfs;
```

The idea would be simply to make the DFS search, without exploring an action if it would generate a word (the word on the stack) which is not in LNF. We call `LNF_Dfs` such an algorithm.

```
proc LNF_Dfs(q,≺_w);
     local variable q';
     hash(q);
     forall q -a-> q' do
            if ≺_wa is in LNF and q' is not hashed then Dfs(q',≺_wa);
end LNF_Dfs;
```

For each state on the stack, it is sufficient to remember the current summary $\prec_w$, in order to know whether the word is in LNF or not. This simple algorithm actually fails to explore every state, because LNF and the DFS search are somehow not compatible. Consider the example on Figure 3.1.

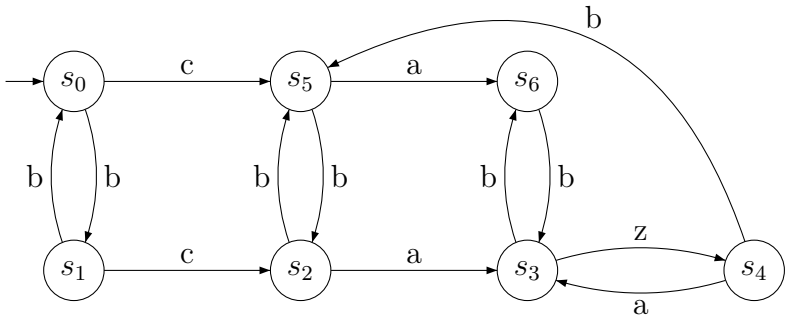


Figure 3.1: A state space for which `LNF_Dfs` does not explore every state.

Figure 3.1 provides an example of a state space that satisfies the diamond property, but is not fully explored by `LNF_Dfs`. The nodes, except s_6, are numbered in the order in which they are discovered. The node s_6 is not discovered. The alphabet is $\{a, b, c, z\}$, with the ordering $a \ll b \ll c \ll z$, which is also used to choose which state is explored first by the DFS search. The independence relation is $I = \{(a, b); (b, a); (b, c); (c, b)\}$. Letter

z is dependent on every other letter a, b, c, and a, c are dependent. The state s_6 can only be visited through s_3 and s_5. The word on the stack when s_3 is visited is bca. However, $bcab$ which would visit s_6 is not in LNF ($LNF(bcab) = bbca$). In the same way, the word on the stack when s_5 is visited is $bcazb$, but $bcazba$ which would visit s_6 is not in LNF either ($LNF(bcazba) = bcazab$). Hence s_6 is not visited. However, on graph without cycles, `LNF_Dfs` explores all the reachable states [am&ai09]. In the next section, we explain how to perform suboptimal pruning of the transitions, but which is yet compatible with DFS and explore all the reachable states, even in the presence of cycles.

In [icalp07, am&ai09], it is shown that when using breadth-first search (BFS), ignoring paths labeled by words not in LNF, every state is explored, even on graphs with cycles. The resulting algorithm is called `LNF_Bfs`.

Proposition 1 *[icalp07, am&ai09] For any state system $\mathcal{A}$, the algorithm* `LNF_Bfs`*(s_0) explores all states that are reachable from s_0.*

3.5 An Edge Lean Algorithm for Complete State Coverage

In this section, we show how to reduce the number of explored transitions in a DFS search by making use of the diamond property. In what follows, we define a smaller relation on strings in Σ^* than LNF, and prove that it suffices to explore paths whose labeling is minimum with respect to this relation.

Definition 2 *We denote $ubav \hat{\Longrightarrow} uabv$ when $(a, b) \in I$ and $a \ll b$, and let $\Longrightarrow$ be the transitive closure of $\hat{\Longrightarrow}$. We say that a word $w \in \Sigma^*$ is* irreducible *if there exists no $w' \neq w$ such that $w \Longrightarrow w'$.*

Intuitively, a word is irreducible if it cannot be transformed into a smaller word with respect to $\Longrightarrow$ by a local fix (a single permutation of adjacent independent letters). We call a path ρ irreducible if its labeling $\ell(\rho)$ is an irreducible word. Observe that a prefix of an irreducible path is also irreducible. Note that if w is in LNF, then it is irreducible. However, the converse does not necessarily hold. Indeed, consider $a \ll b \ll c$, $I = \{(a, b), (b, a), (b, c), (c, b)\}$ (that is a and c are dependent). Then $x = cab$ is irreducible, but $LNF(x) = bca \neq x$.

The algorithm `EdgeLeanDfs` [am&ai09] only explores paths whose labeling are irreducible. For this, it suffices to remember the last letter x seen along the current path, and not to extend this path with letter y whenever $x\,I\,y$ and $y \ll x$. This algorithm is given below:

```
EdgeLeanDfs(s0,ε);

proc EdgeLeanDfs(q,prev);
     local variable q';
     hash(q);
     forall q -a-> q' such that prev= ε or ¬(aIprev) or prev ≪ a do
            begin
                  if q' not hashed then EdgeLeanDfs(q', a);
end EdgeLeanDfs;
```

Theorem 1 `EdgeLeanDfs(`s_0`, `ϵ`)` *explores all states of* $\mathcal{A}$ *that are reachable from* s_0.

The proof of this theorem, which can be found in [icalp07, am&ai09], is actually far from trivial. It uses a well-founded order (based on the Parikh image of a word) and the number of occurrences of letters in the suffixes of paths leading to a given state.

We now give an example that illustrates the performance of our algorithm. Consider two processes p and p' with a counter from 1 to n on each process. These counters can be incremented through actions a and a', respectively, and decremented through actions b and b', respectively. Clearly, the independence relation between these actions is $I = \{(a,a'),(a',a),(a,b'),(b',a),(b,a'),(a',b),(b,b'),(b',b)\}$. Let $a \ll b \ll a' \ll b'$. The left part of Figure 3.2 shows in solid edges the paths explored by regular depth-dirst search (`Dfs`), and in dotted edges the transitions which lead to a state already explored. On the right, we indicate in the same way the path followed by `EdgeLeanDfs`, though we have deleted the transitions not considered by `EdgeLeanDfs`.

It is easy to see that `Dfs` considers almost twice as many transitions as `EdgeLeanDfs` (i.e., $4n(n-1)$ versus $(2n+2)(n-1)$). Moreover, the stack size in `DFS` is n^2-1 (path labeled by $(a^n a' b^n a')^{(n-1)/2}$), while for `EdgeLeanDfs` it is only $2n$. Generalizing this example from 2 to n processes, we obtain an example in which `EdgeLeanDfs` has an exponentially smaller maximum stack

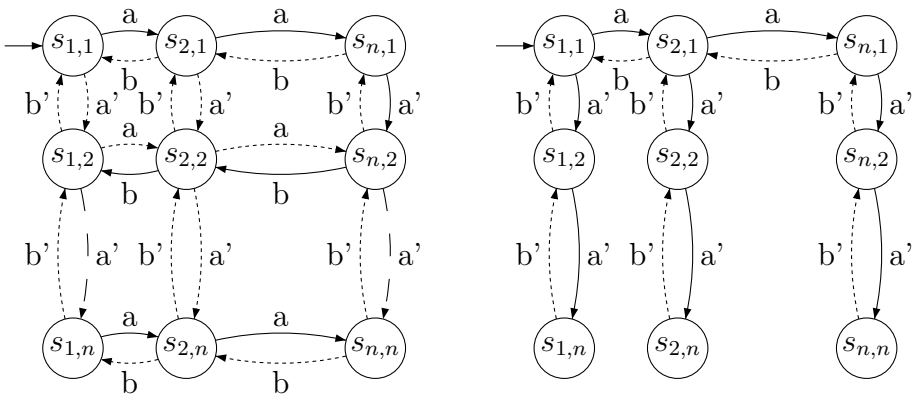


Figure 3.2: Paths explored by `Dfs` (left) and by `EdgeLeanDfs` (right).

size than `Dfs`. We indeed witnessed stack size reduction in experiments with real life systems (see Table 3.1 on the facing page, see [am&ai09] for details and more tests), where we report the number of states and edges explored, as well as the maximal stack size, the maximal memory taken and the time taken by the algorithms.

Although `LNF_Dfs` may fail to explore the entire state space (each of our examples contains cycles), there is only one case (RW6) where we observed a difference between the number of states generated by `EdgeLeanDfs` and `LNF_Dfs`. In most of the experiments, both of our algorithms explore considerably fewer transitions than regular depth-first search. On the other hand, the difference between `EdgeLeanDfs` and `LNF_Dfs` regarding the number of transitions is not very significant. With respect to the stack size (and thus memory consumption), our algorithms are up to 1000 times better than regular DFS (see, e.g., DMS examples).

3.6 Related Work

We now start the related work section with very closely related work, and end with different approach to tame the complexity for the model checking of distributed regular systems.

Set of Representatives for a finite set. In [WYKG08, KWG09], the authors apply Partial Order Reduction to Bounded Model Checking of distributed systems. Bounded model checking implies that only a finite set of

Table 3.1: Experimental Results.

DMSnoCC

	states	edges	stack	memory [MB]	time [s]
Spin with regular DFS	229M	1009M	26M	1060 (1126.5)	12623
Spin with `EdgeLeanDfs`	229M	296M	264.5K	40.9 (41.1)	12620
Spin with `LNF_Dfs`	229M	265M	32.2K	31.4 (33.8)	11638

DMSwithCC

	states	edges	stack	memory [MB]	time [s]
Spin with regular DFS	132M	541M	18.9M	760 (860)	8469
Spin with `EdgeLeanDfs`	132M	174M	384K	19.8 (30.9)	8523
Spin with `LNF_Dfs`	132M	151M	29.2K	5.2 (5.6)	8158

RW4

	states	edges	stack	memory [MB]	time [s]
Spin with regular DFS	263K	1.1M	2253	26.1 (27.4)	5.34
Spin with `EdgeLeanDfs`	263K	558K	1247	26.0 (26.5)	4.22
Spin with `LNF_Dfs`	263K	462K	625	26.0 (26.5)	3.87

RW6

	states	edges	stack	memory [MB]	time [s]
Spin with regular DFS	11.5M	65.6M	827K	42.9 (43.5)	560
Spin with `EdgeLeanDfs`	11.5M	59.6M	784K	42.0 (46.1)	556
Spin with `LNF_Dfs`	9.9M	41.3M	148K	12.3 (15.3)	432

paths is tested. The idea for Partial Order Reduction is simply to reduce the number of paths considered, by removing redundant ones. The procedure used in [KWG09] generates exactly the set of paths in LNF, while the procedure used in [WYKG08] generates the set of irreducible paths. The authors however miss this fact. They need tedious proof in order to prove that at least an equivalent path is explored by their algorithms. Also, they seem to miss the difference between Partial Order Reduction of states (see below) and Partial Order Reduction of transitions - which they and we do. It is somehow our impression that these differences are indeed not known by most of the researchers even in our field.

Partial Order Reduction for finite state Systems. The most related work with LNF_Dfs and with EdgeLeanDfs is the SleepSet method [God94, GHP95, GW91]. It also reduces the set of transitions considered by the search. We proved in [icalp07, am&ai09] that the transitions reduced by SleepSet and By LNF_Dfs are actually exactly the same, although the algorithms - and their overhead - differ. SleepSet was not designed tailored to a particular search strategy (DFS,BFS). Hence at least for DFS, applying it straight may fail to explore every reachable states. In [GHP95], it was proposed to reopen previously visited states (that is, consider path with loops) in order to obtain a correct algorithm. We believe that EdgeLeanDfs is a better tradeoff as it has extremely small overhead, preventing to explore a state twice at the cost of more transitions explored (but no transition can be explored twice). The result is that the computation time taken by SleepSet [God94] is sometimes worse than a straight DFS, which is never the case with EdgeLeanDfs. Also, for systems with loops, it cannot reduce the stack size as EdgeLeanDfs does.

The second kind - and the original one- of Partial Order Reduction is the one reducing the number of states visited. Not exploring a state is dangerous as it can change the result of model checking. The reduction is thus made carefully, according to the property being checked. Several heuristic have been developed, mainly Ample Sets [Pel94], Stubborn Sets [Val92], and Persistent sets [God94]. Overall, there is more time saved by not exploring some states than by not exploring some transitions, although the overhead can be negative when not many states can be pruned. On the other hand, reducing the number of states does not help the memory consumption (stack size) as well as reducing the number of transitions.

Unfoldings. Historically, *unfoldings* [McM95], [ERV02] were the first technique to handle distributed sytems as such, considering the associated (infinite) event structure. The important feature of event structures is that there is no duplication of states for two threads in parallel. Executing "ab" with $(a, b) \in I$ is simply represented as action a in parallel with action b. There is no need to say that nothing, a, b, ab and ba are possible. On the other hand, two (non-equivalent) runs leading to the same global states will not be merged as with an automata based approach. Hence, POR comes for free, and instead the analysis of the global states of the system - ensuring that only a finite number of paths are explored (cutting the branches at some point) while exploring every global states (complete prefix)- is more involved. *Adequate orders* exist ensuring that global states will be explored only once using BFS [ERV02]. However, when using DFS, some states may not be explored [EKS08]. These results on BFS and DFS are very similar to ours concerning LNF_Dfs. One important point is that both the adequate order and LNF_Dfs use LNF, but in a different way. To compare both methods, it is understood that unfolding works better when there are lots of concurrency and few non-equivalent runs leading to the same global states, while in the opposite case, it is more efficient to run an automata based technique considering interleavings plus POR. For instance, unfolding works very well with monitoring [FBHJ05], which consider a state space which is the product with an observation - which makes few runs reach the same global product state. Also, unfolding allows to represent the possible explanations in an efficient way [FBHJ05]. Very recently, [RSSK15] proposed a new POR method, using unfolding to keep a memory of configuration visited. The idea is to consider only one maximal path in the search, and only at the backtrack phase to visit transitions in local-conflict. Using the local conflict relation seems promising, as it allows to explore at most one execution per Mazurkiewicz trace, while ensuring to explore every configuration reached by maximal executions.

Tree Like Structure Concerning the new technique which represents the partial order as a graph and use tree automata techniques to handle it [CG14] (for the numerous cases where the graph has bounded treewidth), the general impression is that it is very useful to scale decidability to more complex distributed systems (using stacks, data, and possibly time). Concerning taming the complexity when decidability is obvious, as is the case in this chapter, there is not yet tangible evidence that this techniques is efficient.

3.7 Perspectives

Büchi automata. Results presented in this chapter mostly refer to finding all the reachable states using DFS. DFS is particularly interesting for model-checking Büchi automata, as one can use the Tarjan's algorithm (or the Double DFS alternative) in order to find loops in linear time. One can wonder if EdgeLeanDfs would find loops. The answer is negative if we consider only irreducible paths (a path spanning over both DFSs). However, if one reinitializes the path for the second DFS (that is, it is only asked that path in the first DFS are irreducible, and path in the second DFS are irreducible -but not the concatenation of them), then it is not too complex to prove that loops will be found, if they exist, as all the reachable states will be found by the second search from the backtracked state of the first DFS.

POR of states vs POR of transitions. It could seem interesting to compare the performance of Partial Order Reductions of states vs transitions. There is however no reason to oppose these two techniques, as they can be used together [God94]. It would be interesting to show how EdgeLeanDfs could be used together with POR of states (e.g. Persistent set [God94]).

Unfoldings vs POR of transitions. As mentioned, many results are similar concerning POR of transitions and unfoldings. However, there is no equivalent of the EdgeLeanDfs in terms of unfoldings. It might be feasible to define complete prefixes according to irreducible linearizations, that is duplicate some configurations when they represent the same prefix but different irreducible linearizations. A in-depth comparison between LNF_Bfs (which never consider two linearizations of the same trace) and BFS on unfoldings (which also never consider two linearizations of the same trace) could also be interesting to undestand if these two approaches are really different or closer than most people believe.

Chapter 4

Asynchronous Automata and the Zielonka Construction

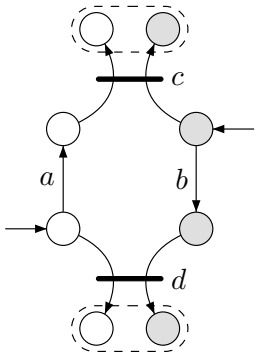
4.1 Introduction

Zielonka's theorem [Zie87], established almost 30 years ago, gives the theoretical background for distributing a global control over several processes. It states that any regular language closed under commutation is the language of an *asynchronous automaton* (a tuple of automata, one per process, exchanging information when performing common actions). We will show in this chapter that using representatives, the complexity of this construction can be improved.

4.2 Asynchronous Automata

Asynchronous Automata generalize Network of Automata by allowing processes involved in a common action to exchange information on their state. Thus, the local state on process p after a common action a with q may depend on the state process q was in before a. This generalization is necessary in order to implement in a distributed way every regular language closed by commutation.

We keep the same notation as in the overview chapter. Let $\mathcal{P}$ be a fixed set of processes, and Σ and alphabet and $\mathrm{dom} : \Sigma \to 2^{\mathcal{P}}$ a function assigning each letter to a set of processes executing that letter. For any $p \in \mathcal{P}$, we denote $\Sigma_p = \{a \in \Sigma \mid p \in \mathrm{dom}(a)\}$. Actions a and b are independent, that is



(a)

Figure 4.1: An asynchronous automata. States of process p (resp. q) are unshaded (resp. shaded). Dashed lines mark global final states.

$(a, b) \in I$, iff $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$.

An *asynchronous automaton* is a tuple $((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$, where for all $p \in \mathcal{P}$, S_p is the set of *local states of process* p, and for all $a \in \Sigma$, $\Delta_a \subseteq \prod_{p\in\mathrm{dom}(a)} S_p \times \prod_{p\in\mathrm{dom}(a)} S_p$ defines the (partial) *transition relation*. Note that while we define the transition relation on letters for ease of presentation, it is equivalent to a corresponding definition on processes. Any $s = (s_p)_{p\in\mathcal{P}} \in \prod_{p\in\mathcal{P}} S_p$ is called a *global state* and $In, Fin \subseteq (S_p)_{p\in\mathcal{P}}$ denote the set of global initial and final states, respectively.

The *semantics* of an asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ is given by the (sequential) automaton $S(AA) = (C, \to, In, Fin)$ over Σ, where $C = \prod_{p\in\mathcal{P}} S_p$ is the set of global states, and the global transition relation is given by $\to$: $C \to C$ with $(s_p)_{p\in\mathcal{P}} \xrightarrow{a} (s'_p)_{p\in\mathcal{P}}$ iff $(s'_p)_{p\in\mathrm{dom}(a)} \in \Delta_a((s_p)_{p\in\mathrm{dom}(a)})$ and $s'_p = s_p$ for all $p \notin \mathrm{dom}(a)$. The language $\mathcal{L}(AA)$ accepted by AA is by definition $\mathcal{L}(S(AA))$, the language accepted by $S(AA)$.

Notice that $S(AA)$ is *diamond* (cf. Chapter 3 on page 19) for any given asynchronous automaton AA, that is for all $s, s', t \in C$ and all $(a, b) \in I$, if $s \xrightarrow{a} s' \xrightarrow{b} t$, then there exists t' with $s \xrightarrow{b} t' \xrightarrow{a} t$. Thus $\mathcal{L}(AA)$ is closed by commutation: for all $v \in \mathcal{L}(AA)$ and $w \sim v$, we also have $w \in \mathcal{L}(AA)$.

4.3 Survey of the different Zielonka's constructions

In the past 30 years, several attempts have been made to construct from regular (commutation-closed) specifications asynchronous automata with *some* additional properties. We start by giving some of the properties:

Definition 3 (determinism) *We call an asynchronous automaton* $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ deterministic*, if* $|In| = 1$ *and* $|\Delta_a(s)| \leq 1$ *for all* $a \in \Sigma$ *and* $s \in \prod_{p\in dom(a)} S_p$.

Non-determinism allows a process to guess what another process is doing concurrently. Note that every asynchronous automaton can be transformed into a deterministic asynchronous automaton, albeit with an unavoidable doubly exponential blow-up in the number of states [KMS94].

Definition 4 (deadend-freeness) *A global state* s *is called a* deadend*, if there does not exist a word* $w \in \Sigma^*$ *and global state* $s' \in Fin$ *with* $s \xrightarrow{w} s'$. *An asynchronous automaton is* deadend-free *iff no global state reachable from an initial state is a deadend: for all* $v \in \Sigma^*, s_0 \in In$*, and all* s *with* $s_0 \xrightarrow{v} s$*, the state* s *is not a deadend.*

Deadend-freeness prevents a process from performing actions that will not be observable in terms of the language.

Definition 5 (local acceptance) *An asynchronous automaton* $((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, In, Fin)$ *is said to be* locally accepting *(or said to* have local final states*), if* $Fin = \prod_{p\in\mathcal{P}} Fin_p$ *for some* $Fin_p \subseteq S_p$ *for all* $p \in \mathcal{P}$.

We summarize now the different results.

Theorem 2 *Let* L *be a regular language closed by commutation. Then, there exists three asynchronous automata* AA_1, AA_2, AA_3 *over* (Σ, dom) *with* $\mathcal{L}(AA_i) = L$ *for all* $i \in \{1, 2, 3\}$ *such that:*

1. AA_1 *is deterministic [Zie87, CMZ93, DR95, MS94, icalp06, icalp10],*
2. AA_2 *is deadend-free [Zie89],*
3. AA_3 *has local initial and final states [Bau09].*

We provide here the worst case space complexities (the number of local states) to obtain a deterministic or non-deterministic asynchronous automaton (Det AA, Non Det AA), given a deterministic or non-deterministic diamond automaton A over a set of processes $\mathcal{P}$:

complexity	Det AA	Non Det AA
Det A	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)} \cdot 2^{2^{\lvert\mathcal{P}\rvert^4}}$ [icalp10]	–
Non Det A	$2^{O(\lvert A\rvert\cdot\lvert\mathcal{P}\rvert^2+\lvert\mathcal{P}\rvert^4)}$ [icalp06]	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)}$ [Bau09]

The complexities stated to obtain a deterministic asynchronous automaton from [icalp06, icalp10] are optimal, while optimality is not proven for obtaining a non-deterministic asynchronous automaton (using [Bau09] for instance). Note that [Bau09] uses a construction not based on Zielonka's. Determinizing an asynchronous automaton is possible, but the blow-up is doubly exponential [KMS94]: constructing a deterministic asynchronous automaton directly is preferable.

4.4 Representatives to reduce the complexity

The original complexity of [Zie87], analyzed by [MS94], is stated as building a (monolithic) asynchronous automaton such that every process has at most $O(2^{\lvert A\rvert^{\lvert\Sigma\rvert}})$. Later, [MS94] improved the complexity to $O(2^{\lvert A\rvert^{\lvert\mathcal{P}\rvert^2}})$, giving an on the fly algorithm requiring only an exponential number of bits in the number of processes for a process to remember its configuration.

In parallel, [CMZ93] presented a (monolithic) algorithm with a complexity of $O(\lvert A\rvert^{2^{\lvert\Sigma\rvert}})$ number of states, granted that the original specification is a *deterministic* diamond automaton.

In order to obtain the complexity $2^{O(\lvert A\rvert\cdot\lvert\mathcal{P}\rvert^2+\lvert\mathcal{P}\rvert^4)}$ (even from a non-deterministic diamond specification), we resort to representatives once more [icalp06]. One can also obtain an on the fly algorithm requiring only a *polynomial* number of bits for a process to remember its configuration. The idea is the following.

First, events in the *primary information* of a labeled partial order $T = (E, \leq, \lambda)$ over (Σ, I) are important, as we show below. Formally, we denote $\text{last}_p(T)$ the last (for $\leq$) event $e \in E$ on process p ($e \in \Sigma_p$) (if it exists), that is such that there is no $f \in E$ on p with $e \leq f$. We denote $S_1(T) = \{last_p(T) \mid p \in \mathcal{P}\}$. Notice that we can have $\text{last}_p(T) = \text{last}_q(T)$ for $p \neq q$. Clearly, $S_1(T)$ has at most $\lvert\mathcal{P}\rvert$ elements.

We define the view of a process p as $\text{view}_p(T) = (E', \leq', \lambda')$ with $E' = \{e \in E \mid e \leq \text{last}_p(T)\}$, and $\leq' = \leq|_{E'}, \lambda' = \lambda|_{E'}$ being the restriction to E'. When a trace is presented as an equivalence class $[u]$ of a word $u \in \Sigma^*$, then $\text{view}_p([u])$ is also the shortest trace $[v]$ such that there exists v' with $w \sim vv'$, and each action $a \in \Sigma_p$ occurs as many times in v as in w. We have that $(E', \leq', \lambda')$ is the labeled partial order associated with $[v]$, both definitions are equivalent.

When p and q synchronize with action a, p needs the information on what had happened in q and which did not happend on p before a. That is, $\text{view}_q(T) \setminus \text{view}_p(T)$. What happens in $\text{view}_q(T) \setminus \text{view}_p(T)$ is entirely defined by $\text{view}_p(T) \cap \text{view}_q(T)$. This is entirely determined by $\text{view}_q(T)$ and by $\max(\text{view}_p(T) \cap \text{view}_q(T))$, the maximal events of $\text{view}_p(T) \cap \text{view}_q(T)$. We can use the following lemma from [MS94]:

Lemma 2 *[MS94] Let a partial order $T = (E, \leq, \lambda)$ and $p, q \in \mathcal{P}$. Then:*

$$\max(\mathit{view}_p(T), \mathit{view}_q(T)) \subseteq S_1(\mathit{view}_p(T)) \cap S_1(\mathit{view}_q(T)).$$

Assume that process q knows $\text{view}_q(T)$ but not $\text{view}_p(T)$ (which is what happens in a distributed system). Process q cannot know $\text{view}_q(T) \setminus \text{view}_p(T)$ with certainty. However, q can compute a set $\mathcal{F}$ such that $\text{view}_q(T) \setminus \text{view}_p(T)$ is one of the element in $\mathcal{F}$ no matter what p is doing in parallel. Formally, for a trace $T = (E, \leq, \lambda)$, a trace T' is a *factor* of T if there exists $F \subseteq E$ with $g \in F$ whenever there exists $e, f \in F$ and $e \leq g \leq f$, and $T' = (F, \leq|_F, \lambda|_F)$. Now, for each $S \subseteq S_1(\text{view}_q(T))$ (at most exponentially many in the number of processes), we compute the factor $F_S = \{e \in \text{view}_q(T) \mid \nexists f \in S, e \leq f\}$. Then $\mathcal{F} = \{F_S \mid S \subseteq S_1(\text{view}_q(T)\}$. We know that $\text{view}_q(T) \setminus \mathit{view}_p(T) \in \mathcal{F}$ thanks to lemma 2. The original constructions [Zie87, CMZ93, MS94] keeps all these factors explicitely, explaining the doubly exponential complexity for the number of states.

Instead, we keep only a small number of representatives which allows to reconstruct every possible combination [icalp06]. Our representatives are called *zones*. When a $\text{view}_q(T) \setminus \mathit{view}_p(T)$ is needed, we compose the zones included in $\text{view}_q(T) \setminus \mathit{view}_p(T)$. Zones are defined as equivalence classes of the following relation:

Let $T = (E, \leq, \lambda)$ be a trace. For an event $e \in E$ we define the set of events $S(e) = \{f \in S_1(T) \mid e \leq f\}$. We say that two events e, f are equivalent (denoted as $e \equiv f$) if and only if $S(e) = S(f)$. The equivalence classes of $\equiv$ are called zones.

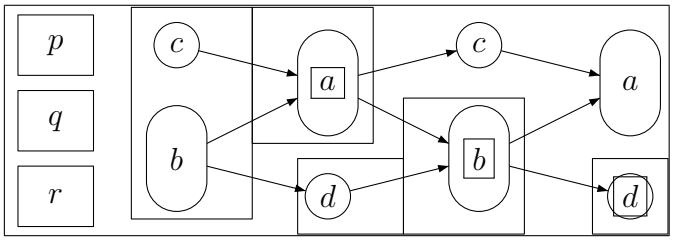


Figure 4.2: The three zones of $\text{view}_r(T)$. The distinguished nodes are those in $S_1(\text{view}_r(T))$.

Let Z be a zone and define $S(Z) = S(e)$ for some event $e \in Z$. Let also Z, Z' be two zones of some trace T. We write $Z < Z'$ if $Z \neq Z'$ and if $e < e'$ for some events $e \in Z$, $e' \in Z'$. By $\text{dom}(Z)$ we denote the set of processes occurring in the zone Z, i.e., $\text{dom}(Z) = \cup_{e \in Z} \text{dom}(e)$. The following lemma is easy to show:

Lemma 3 *1. A zone of T is a factor of T.*

2. The set of zones partitions the set of events of T.

3. The relation $<$ on zones is acyclic. It induces the least partial order such that $S(Z) \supsetneq S(Z')$ and $dom(Z) \cap dom(Z') \neq \emptyset$ implies $Z < Z'$.

Figure 4.2 depicts the trace $T = [cbadcbad]$ with $\mathcal{P} = \{p, q, r\}$, $\text{dom}(a) = \{p, q\}$, $\text{dom}(b) = \{q, r\}$, $\text{dom}(c) = \{p\}$, $\text{dom}(d) = \{r\}$. We have $S_1(\text{view}_r(T))$ consists of the first a, the second b and the second d. There are three zones in $\text{view}_r(T)$: Z_1 is the first a, b and c, Z_2 is the first d and the second b, and Z_3 is the second d (see also Figure 4.2). We have $Z_1 < Z_2 < Z_3$. Also, $S(Z_2)$ consists of the second b and the second d.

Zones enjoy some crucial properties, stated in the following Proposition.

Proposition 2 *[icalp06] Let T be a trace, $p, q \in \mathcal{P}$ be processes, and Z a zone of $view_p(T)$. Then either $Z \subseteq view_p(T) \cap view_q(T)$, or $Z \cap (view_p(T) \cap view_q(T)) = \emptyset$.*

Further, the number of zones is polynomial.

Proposition 3 *[icalp06] Let T be a trace. There are at most $|\mathcal{P}|^2 + |\mathcal{P}|$ zones in T.*

We obtain:

Theorem 1 *[icalp06] Let $\mathcal{A}$ be a (non-deterministic) I-diamond automaton over the independence alphabet (Σ, I). We can construct an equivalent deterministic asynchronous automaton $\mathcal{B}$ with less than $2^{|\mathcal{A}|^2\times(|\mathcal{P}|^2+|\mathcal{P}|)+2|\mathcal{P}|^4}$ states. Each process has a memory of size $O(|\mathcal{A}|^2 \times |\mathcal{P}|^2 + |\mathcal{P}|^4)$, and computes its next state in time $O(|\mathcal{A}|^2 \times |\mathcal{P}|^2 + |\mathcal{P}|^4)$.*

Notice that in order to obtain the complexity $|A|^{O(|\mathcal{P}|^2)}\cdot 2^{2|\mathcal{P}|^4}$ from a *deterministic* diamond specification, we apply the trick of [CMZ93] to the previous construction, although adapting it to obtain an on-the-fly algorithm with a polynomial time complexity and logarithmic (in the number of states of A, quadratic in the number of processes) for the number of bits to remember to encode a configuration of a process is non-trivial [icalp10]. We detail the algorithm and give some experimental results in the next section.

4.5 Synthesis Algorithm from Deterministic Specification

Let $\mathcal{A} = (V, \Sigma, \to, v^0, F)$ be a *deterministic* I-diamond automaton. We explain in the following how to build a deterministic asynchronous automaton $\mathcal{B}$ with the same language. The local p-state of $\mathcal{B}$ reached on a prefix $\text{view}_p(T)$ is the tuple $(TS, \text{dom}, \langle \text{dom}(Z_i), S(Z_i), s_{Z_i}\rangle_{i=1,\dots m})$, where:

- $\{Z_1 \dots, Z_m\}$ is the set of zones of $\text{view}_p(T)$.
- The timestamping $TS : \mathcal{P} \to \{0, \cdots, |\mathcal{P}|\}^{\mathcal{P}}$ associates every process q with the timestamp of the last event on q in $\text{view}_p(T)$ [DR95, Zie87].
- The domain $\text{dom} : \mathcal{P} \to 2^{\mathcal{P}}$ associates every process q with the domain of the last event on q in $\text{view}_p(T)$.
- For a zone Z, $\text{dom}(Z)$ denotes the set of processes occurring in Z.
- For a zone Z, $S(Z) \subseteq \mathcal{P}$. That is, $q \in S(Z)$ means that there is an event of Z before the last event $\text{last}_q(\text{view}_p(T))$ on q in $\text{view}_p(T)$.
- s_Z is the state reached from the initial state by executing events in zones $Z' \leq Z$ (state s_Z is unique as $\mathcal{A}$ is deterministic).

We now define the local transition function δ_p of the asynchronous automaton $\mathcal{B}$. Recall that the order on zones $Z_i < Z_j$ can be computed from the knowledge of $S(Z_i)$ and of $\mathrm{dom}(Z_i)$, for all zones Z_i (see Lemma 3). We do not recall how to update TS and dom (see [Zie87, DR95]). Notice that we do not need the function $S : \mathcal{P} \to 2^{\mathcal{P}}$ as S_{Z_i} includes this information.

We first compute a prestate, that is a set of pre zones for $\mathrm{view}_p(T)a$: some (final) zones of $\mathrm{view}_p(T)a$ can be decomposed into several prezones. We denote $dom(a) = \{p, q_1, \cdots q_m\}$. We first add the new information given by process $q = q_i$, inductively on i, to the information that p gets. Assume that we have already computed the state for the set of processes $P = \{p, q_1, \cdots q_{i-1}\}$.

Assume that the process q is in local state $(TS^q, \mathrm{dom}^q, \langle \mathrm{dom}(Z_i^q), S(Z_i^q), s_{Z_i^q}\rangle_{i=1,\ldots,n})$ with history $\mathrm{view}_q(T)$. Moreover, the current prestate (associated with events $\mathrm{view}_P(T)$ seen by $P = \{p, q_1, \cdots q_{i-1}\}$) is $(TS, \mathrm{dom}, \langle \mathrm{dom}(Z_i), S(Z_i), s_{Z_i}\rangle_{i=1,\ldots,m}\rangle$. We now explain how to update the prestate, taking into account the state of q. The updated prestate for $P \cup \{q\}$ after reading $q = q_i$ is denoted $(TS', \mathrm{dom}', \langle \mathrm{dom}(Z'_i), S(Z'_i), s_{Z'_i}\rangle_{i=1,\ldots,k})$, with $\mathrm{view}_{P\cup q}(T)$ as history, where:

1. Let $\mathrm{Common} = \{r \in \mathcal{P} \mid \exists p \in \mathcal{P}, (TS^q, \mathrm{dom}^q)(r) \neq (TS, \mathrm{dom})(p)\}$ be the set of process r such that the primary event $\mathrm{last}_r(\mathrm{view}_q(T))$ is a primary event of $\mathrm{view}_p(T)$.
2. Let $J = \{i \in \{1, \ldots, n\} \mid S(Z_i^q) \cap \mathrm{Common} = \emptyset\}$, and $k = m + |J|$,
3. Set $(TS', dom')(r) = (TS^q, dom^q)(r)$ if there exists $i \in J$ with $r \in S(Z_i^q)$, and $(TS', dom')(r) = (TS, dom)(r)$ otherwise,
4. Let $\langle \mathrm{dom}(Z'_j), S(Z'_j), s_{Z'_j}\rangle_{j=1,\ldots,m} = \langle \mathrm{dom}(Z_j), S(Z_j), s_{Z_j}\rangle_{j=1,\ldots,m}$,
5. $\langle \mathrm{dom}(Z'_{m+j}), S(Z'_{m+j}), s_{Z'_{m+j}}\rangle_{j=1,\ldots,|J|} = \langle \mathrm{dom}(Z_i^q), S(Z_i^q), s_{Z_i^q}\rangle_{i\in J}$,
6. The partial order $<'$ on the new zones is given by the transitive closure of the relation $< \cup <^q \cup \{(Z'_i, Z'_j) \mid i \leq m < j, \mathrm{dom}(Z'_i) \cap \mathrm{dom}(Z'_j) \neq \emptyset\}$,
7. For all $Z'_i <' Z'_j$, $S'(Z'_i) \leftarrow S'(Z'_i) \cup S'(Z'_j)$,

Once the information from all processes have been added, we add a last prezone Z_{m+1} corresponding to action a. Assume that the current p-prestate is $(TS, \mathrm{dom}, \langle \mathrm{dom}(Z_i), S(Z_i), s_{Z_i}\rangle_{i=1,\ldots,m})$.

The new p-prestate is $(TS', \mathrm{dom}', \langle \mathrm{dom}(Z'_i), S(Z'_i), s_{Z'_i}\rangle_{i=1,\ldots,m+1})$, with:

1. Let $\mathrm{dom}(Z'_{m+1}) = \mathrm{dom}(a)$, and $S(Z'_{m+1}) = \mathrm{dom}(a)$.
2. Let $\langle \mathrm{dom}(Z'_j), S(Z'_j), s_{Z'_j}\rangle_{j=1,\ldots,m} = \langle \mathrm{dom}(Z_j), S(Z_j), s_{Z_j}\rangle_{j=1,\ldots,m}$.
3. For all $i \in \{1,\ldots,m\}$, let $S(Z'_i) \leftarrow S(Z'_i) \cup \mathrm{dom}(a)$,
4. Hence $<' = < \cup \{(Z'_i, Z'_{m+1} \mid i \in \{1,\ldots,m\}\})$
5. Compute a new Timestamp TT for a. For all $p \notin \mathrm{dom}(a)$, let $(TS', dom')(p) = (TS, dom)(p)$. For all $p \in \mathrm{dom}(a)$, let $dom'(p) = \mathrm{dom}(a)$ and $TS'(p) = TT$.

Now, it remains to do two things. First, clean the prestate in order to obtain the (final) set of zones, mainly merging zones together. Finally, we will compute the state information corresponding to these zones.

Assume that the current p-prestate is $(TS, \mathrm{dom}, \langle \mathrm{dom}(Z_i), S(Z_i), s_{Z_i}\rangle_{i=1,\ldots,m})$. We first merge prezones together. Intuitively, prezones Z_i, Z_j with the same value $S(Z_i) = S(Z_j)$ will be merged, such that all (final) zones will have different S-values. More formally, let $X = \{S(pz) \mid pz \text{ is a prezone}\}$. For each $x \in X$, we create a zone Z_x. We set $S(Z_x) = x$, and $\mathrm{dom}(Z_x) = \bigcup_{\text{prezone } pz, S(pz)=x} \mathrm{dom}(pz)$. We also set $Z_x < Z_y$ for all zone x, y such that there exists two prezones $pz < pz'$ with $S(pz) = x$ and $S(pz') = y$.

Now, we describe how to create the state information s_Z for each zone Z. The state information s_Z is computed using function *trick* defined below. The inputs we pass to function *trick* are state information on prezones that we already have.

The function $trick(r, s, t, Q)$ is defined for three states r, s, t and a domain $Q \subseteq \mathcal{P}$: first, compute any word w such that $\mathrm{dom}(w) \subseteq Q$, and such that $r \xrightarrow{w} s$ (using a depth first search restricted to actions a with $\mathrm{dom}(a) \subseteq Q$). Then compute the state t' such that $t \xrightarrow{w} t'$. Then set $trick(r, s, t, Q) = t'$. We have the following result:

Lemma 4 *[icalp10] Let r be a state of a deterministic diamond automaton. Let $r \xrightarrow{w} s \xrightarrow{w'} t'$ be a path such that for all letters $a \in w', b \in w$, we have $(a, b) \in I$. Let t such that $r \xrightarrow{w'} t \xrightarrow{w} t'$ (it exists because the automaton is diamond). Then $trick(r, s, t, \{q \in \mathcal{P} \mid \exists a \in w, q \in dom(a)\}) = t'$.*

Here is how we compute s_{Z_x}, for all x. First, take $\{Z_{y_1}, \cdots, Z_{y_n}\}$ the set of zones smaller or equal to Z_x. We assume that the order is defined such

that $Z_{y_i} \not> Z_{y_j}$ for all $i < j$ (i.e. it is a linearization). We then enumerate (in a linearized order) for all i the prezones $pz_1^i \cdots pz_k^i$ contained in Z_i, that is with $S(pz_j^i) = y_i$. Using function *compute* defined below, we finally set $s(Z_x) = compute(pz_1^1, \cdots, pz_k^1 \cdots pz_1^n, \cdots pz_{k'}^n)$.

We now define the function $compute(pz_1, \cdots, pz_n)$ for a list of prezone $pz_1, \cdots, pz_n$, given as a linearization (that is $pz_i \not< pz_j$ for $i > j$). For all $i \leq k \leq n$, we will compute inductively the state $s_{i,k}$ corresponding to the state reached after reading the set of prezones $B_{i,k} = \{pz_1, \ldots, pz_i\} \cup \{pz_j \mid pz_j \leq pz_k, j > i\}$.

$Compute(pz_1, \cdots, pz_n)$:

- set $s_{1,1} = s_{pz_1}$
- for all $k = 2 \cdots n$
  - set $s_{0,k} = s_{pz_k}$
  - for all $i = 1 \cdots k$
    1. If $pz_i \leq pz_k$, then set $s_{i,k} = s_{i-1,k}$
    2. else, set $s_{i,k} = trick(s_{i-1,i-1},\ s_{i,i},\ s_{i-1,k},\ \mathrm{dom}(pz_i))$
- return $s_{n,n}$

To compute s_{pz} for pz the last prezone corresponding to the last letter a, we modify $Compute(pz_1, \cdots, pz_n)$ to take into account the last letter a.

We now report some experiments. See [fsttcs13] for more details. For 7 systems, we report the number of states and processes. We also report the number of global state produced by the *heuristic* of [SEM03], by the original Zielonka's construction [Zie87], as well as the number of global as well as local states of our construction [icalp10]. Notice that the original and the heuristic build only a global state space, the reason why we cannot report a number of local states. Notice also the in an on-the-fly algorithm (used e.g. for runtime verification or runtime computation of causal knowledge [fossacs15]), not all the local states need to be constructed by [icalp10]. Overall, heuristic and our new constructions are (almost) always better than the original Zielonka's construction. In term of global state space, the heuristic manages to find very compact representation for some systems, while for other systems, our new construction is better for all metrics.

	$\|A\|$	$\|\mathcal{P}\|$	*heuristic*	*original*	*global*	*local*
mutex-a	13	3	13	1493	271	126
mutex-b	14	4	14	34	22	16
simple	3	3	5	12	12	9
phil	5	4	5	70	71	60
prop2	6	2	188	188	36	21
prop3	11	3	639	639	240	92
L4	8	2	N/A	10	10	5

4.6 Related Work

A very thorough survey on synthesis is [Dar07].

Other Constructions Most of the constructions [Zie87, CMZ93, MS94, DR95, icalp06, icalp10] are close in spirit to the the Zielonka construction, improving various part of the constructions. It is worth noting that the Timestamping used is roughly unchanged from [Zie87]. When the complexity is stated in terms of letters rather than processes, [Zie06] provides actually a better timestamping, requiring only 1 bit for $|\Sigma|^2$ events instead of $\log(p)$ bits for $|\Sigma|^2$ (labeling letters) or $|\mathcal{P}|^2$ (labeling processes) events as in [Zie87].

In [BM05, Bau09], a different construction is given, guessing a correct path for each process. This construction thus gives rise to *non-deterministic* Asynchronous Automaton, albeit with a good complexity (polynomial in the number of states of the *non-deterministic* Automaton given as specification). Obtaining a deterministic Asynchronous Automaton from this non deterministic Asynchronous Automaton can be done, but at a doubly exponential price [KMS94], making the construction interesting only when determinism is irrelevant (for implementation, determinism seems important). In [Pig93], a construction is given as a structural induction on a rational language. Comparisons are thus harder to draw with other constructions.

Last, some constructions are given with better complexity for restricted systems. A simpler construction is given in [DR95] for the case of chordal graphs, extending the case where the dependency relation D is acyclic, but with exponential complexity in the number of states of the automaton given as specification. For acyclic communication pattern, where there is a unique flow of information from a process to another process, [KM13] provides a quadratic algorithm. In this latter case, synthesis with non-controllable actions is also decidable [icalp13] (see the next section).

Realistic Synthesis None of these constructions (nor [icalp10]) gives a general *realistic* distributed implementation: either the constructions are plagued by deadends [Zie87, CMZ93, MS94, DR95, icalp06, icalp10, Bau09, BM05], non-deterministic guesses [Zie89, BM07, Bau09], or the acceptance condition or choice of actions are not distributed [Zie87, CMZ93, MS94, DR95, icalp06, icalp10]. Further, while the initial state is trivially distributed in Zielonka's construction (since it is unique, due to determinism), this is not the case in [Zie89, BM07]. There are systems for which it is not possible to obtain an implementation satisfying all these conditions. In the case of prefix-closed and *forward diamond* [DR95] specifications, [Muk02, SEM03] gives a deterministic deadlock-free implementation with local final states. In the general case, we provide in [fsttcs13] semantical and syntactical characterizations of languages of asynchronous automata with deadends, and/or local accepting states, and/or local decisions.

4.7 Perspectives

Language equivalence is not always sufficient for implementation. A bisimulation equivalence between the specification and the transition system of the asynchronous automaton would be more precise. In the case of a deterministic specification, [fsttcs13] actually provides a bisimilar Asynchronous Automaton as soon as the specification is forward diamond [DR95] (which is necessary and sufficient to obtaining a deadlock-free implementation, needed for the bisimulation). It is much harder to obtain a bisimilar Asynchronous Automaton from a non-deterministic specification. The main problem is to obtain a necessary and sufficient condition for the existence of a bisimilar (deadlock-free) implementation, as it is possible that n edges labeled by a letter a from a state correspond to the same actions in a different interleaving as $m \neq n$ edges labeled by a from a different state. Starting from a non-deterministic I-diamond (interleaved) automaton does not seem easy, and a different non-deterministic specification seems needed in that case, may be under the form of an unfolding.

An important extension of this work is when the I-diamond specification has some non-controllable actions. One can ask whether there exist asynchronous distributed controllers reaching a goal (for instance reaching some global state). A distributed controller on a process has only a local view of what happens, using information received from other processes during past

common actions, and can enable and disable controllable actions. Non controllable actions can never be disabled, and may or may not happen. In this context, several work have been done, providing decidability of distributed control for various subcases, ranging from serial parallel alphabets [GLZ04], often-communicating processes through satisfiability of MSO [MTY05], to acyclic communication pattern [icalp13]. There is hope in the future to obtain decidability in the general case of distributed control of an I-diamond regular specification, as well as extending MSO satisfiability [MTY05] to grid free system. Complexity is usually non elementary when the number of processes is not fixed, with matching lower bound proved in [icalp13]. Complexity can be improved for subclasses, for instance EXPTIME-complete when non-controllable actions never commute [FO14].

Finally, synthesis with message passing instead of synchronous communication will be considered in the next chapter.

Chapter 5

Message Passing Systems

5.1 Introduction

Exchanging information by handshaking, that is by performing an action synchronizing all the processes performing it (as for Network of Automaton, Asynchronous Automaton, Petri Nets, etc.), can seem restrictive. For instance, the symmetry in the communication ensures that process p knows when process q receives a message from p tp q. We explain in this chapter how results concerning model checking and implementation can be extended to message passing systems, where the communication itself is asynchronous, made by sending and receiving messages. We also explain how set of representatives can be used to specify non-regular message passing specifications, under the form of MSC-graphs.

5.2 Channel Bound

5.2.1 Message Sequence Charts

Let $\mathcal{P}$ be a finite set of processes. Let C be a finite set of *message contents* or *control messages*. The set of actions is $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p = \Sigma_! \cup \Sigma_?$, with:

- $\Sigma_! = \{p!q(c) \mid p, q \in \mathcal{P}, c \in C\}$ is the set of send actions. The action $p!q(c)$ represent a send from p to q of a message of content c.
- $\Sigma_? = \{p?q(c) \mid p, q \in \mathcal{P}, c \in C\}$ is the set of send actions. The action $p?q(c)$ represent a receive in p from q of a message of content c.

- The set of actions on process p is $\Sigma_p = \{p?q(c), p!q(c) \mid q \in \mathcal{P}, c \in C\}$.

We assume a unidirectional channel from p to q for each pair of processes $(p, q) \in \mathcal{P}^2$. For simplicity, we ignore internal actions local to a process. We assume that the channels are *FIFO*, that is the n-th action $p?q(c)$ receives the n-th sending $q!p(d)$ of message from p to q, and further $c = d$. A word $w \in \Sigma^*$ is *well-formed* if for all prefix v of w ending with some $p?q(c)$, we have at least as many sends $q!p$ as receives $p?q$ in v, that is $|v|_{q!p} \geq |v|_{p?q}$. In particular, there cannot be more messages $p?q$ received than sent in any prefix. Word w is *complete* if further, $|w|_{q!p} = |w|_{p?q}$ for all p, q, c.

As with traces, we can associate a labeled partial order with any word over Σ^*. For simplicity, we will explicitly specify the message matching function m, but it could be recovered from $\leq$ as well thanks to FIFO. We will call such structures *Message Sequence Charts* (*MSCs* for short). Let $w = a_1 \cdots a_n \in \Sigma^*$ be a well-formed word. We define the MSC $(E, \leq, m, \lambda)$ with:

- $E = \{e_1, \ldots, e_n\}$ is the set of events, with the same number n of events as w,
- $\lambda(e_i) = a_i$,
- $m : E \to E$ is a partial one to one function, with $m(e_i) = e_j$ if $\lambda(e_i) = p!q(c)$ and $\lambda(e_j) = q?p(c)$ for some p, q, c, and $|a_1 \cdots a_i|_{p!q} = |a_1 \cdots a_j|_{q?p}$,
- $e_i \leq e_j$ iff $m(e_i) = e_j$; or $e_i, e_j \in \Sigma_p$ for some process p and $i \leq j$.

Notice that as w is well-formed, $m(e_i) = e_j$ implies that $i \leq j$. We say that 2 words over Σ are equivalent if they have isomorphic MSCs. In general, it is not possible to generate MSCs using a trace alphabet, because one has to count the numbers of messages of every type, which cannot be done with a finite alphabet. There exist two positive results though, in the case of universal and existential channel bounds [LM04].

5.2.2 Universal Channel Bound

Consider a set $W \subseteq \Sigma^*$ of words. Assume that there exists a b such that for all $w \in W$, for all prefix v of w, $|v|_{p!q} - |v|_{q?p} \leq b$ for all p, q. In such a case, we say that the channels of W are *universally b-bounded*. One can then

use the Kuske's trace alphabet [Kus02], defined below, in order to obtain the same set of partial orders.

Consider the trace alphabet (Σ', I), with $\Sigma' = \Sigma \times \{0, \ldots, b\}$. We define $D = \Sigma^2 \setminus I$ as follows: $((a,i),(b,j)) \in D$ iff there exists $p \in \mathcal{P}$ with $a, b \in \Sigma_p$, or $i = j$ and $\{a, b\} = \{p!q(c), q?p(c)\}$ for some p, q, c. For $a, b \in \Sigma$, we denote $a \equiv b$ whenever $a = p!q(c)$ and $b = p!q(d)$, or $a = p?q(c)$ and $b = p?q(d)$ for some p, q, c, d.

We define a function $f : \Sigma^* \to \Sigma'^*$, such that for each word $w = a_1 \ldots a_n \in W$, we have $f(w) = a'_1 \cdots a'_n$ with $a'_i = (a_i, k)$ with k being the number modulo $b+1$ of letters $a_j \equiv a_i$ in $a_1 \cdots a_i$. We define the set $W' \subseteq \Sigma'^*$ with $W' = \{w' \mid \exists w \in W, f(w) = w'\}$.

Then one can show [Kus02] that the set of MSCs associated with words in W is isomorphic to the set of labeled partial orders associated with words in W'. This encoding allows to lift many results from trace theory to *universally bounded* communicating systems, including the Zielonka's implementation theorem [Zie87] (see also Chapter 4).

5.2.3 Existential Channel Bound

Set of representatives can also be useful in the context of asynchronous message passing. Consider a set $W \subseteq \Sigma^*$ of words closed by commutation, that is such that every word w' equivalent with $w \in W$ (that is with the same MSC) satisfies $w' \in W$. We say that W is *existentially b-bounded* if there exists a set W' of representatives for W (that is $W' \subseteq W$ and for each $w \in W$, there exists $w' \in W'$ associated with the same MSC), such that W' is universally b-bounded.

Now, the same Kuske's trace alphabet can be used, but with a twist. It is no more the case that the set of MSCs associated with words in W is isomorphic to the set of labeled partial orders associated with words in W'. Indeed, taking the word $p!q(c)q?p(c)p!q(c)q?p(c)$, this word is existentially one bounded but not universally 1 bounded (it is universally 2 bounded). Applying the Kuske's alphabet with $b = 1$ gives a partial order on set of events $\{s_1, r_1, s_2, r_2\}$ with $\lambda(s_1) = \lambda(s_2) = p!q(c)$, $\lambda(r_1) = \lambda(r_2) = q?p(c)$, $s_1 < s_2$, $r_1 < r_2$, $s_1 < r_1$, $s_2 < r_2$ (which are relations of the MSC), plus the reversed relation $r_1 < s_2$, which is not present in the MSC. What we have is that W' captures exactly the set of b bounded words in W. Usually, this can be used to lift results from Mazurkiewicz trace theory. Actually, even Zielonka's implementation theorem [Zie87] can be lifted. However, the

implementation obtained is non-deterministic [i&c06], and this is unavoidable [i&c06].

In the following, we introduce several classes of systems, and give for each class the different results.

5.3 Communicating Finite State Machines

Communicating Finite State Machines (*CFSM* for short) are a class of models in this framework, close to the implementation of programs communicating with asynchronous messages.

Definition 6 *A* communicating finite-state machine *is a tuple* $\mathcal{A} = ((\mathcal{A}_p)_{p\in\mathcal{P}}, F)$ *where*

- *For every* $p \in \mathcal{P}$, $\mathcal{A}_p = (S_p, \to_p, s_p^0)$ *is a finite labeled transition system over the alphabet* $\Sigma_p \times C$ *for any* $p \in \mathcal{P}$ *(i.e.,* $\to_p \subseteq S_p \times (\Sigma_p \times C) \times S_p$*) with initial state* $s_p^0 \in S_p$.
- $F \subseteq \prod_{p\in\mathcal{P}} S_p$ *is a set of global final states.*

The CFSM $\mathcal{A}$ *is* deterministic *if*

- $s \xrightarrow{p!q,m_1}_p s_1$ *and* $s \xrightarrow{p!q,m_2}_p s_2$ *implies* $s_1 = s_2$ *and* $m_1 = m_2$
- $s \xrightarrow{p?q,m}_p s_1$ *and* $s \xrightarrow{p?q,m}_p s_2$ *implies* $s_1 = s_2$.

To define the semantics of a CFSM, we can define a global (infinite) state system $\mathcal{A}$ on set of states: $S = \prod_{p\in\mathcal{P}} \mathcal{A}_p \times (C^*)^{\mathcal{P}^2}$. Namely, each configuration c of $\mathcal{A}$ is of the form $c = ((s_p)_{p\in\mathcal{P}}, (\nu_{p,q})_{p\neq q\in\mathcal{P}}) \in S$, where for each $p, q \in \mathcal{P}$, $\nu_{p,q}$ is the sequence of message contents, from messages sent from p to q and not yet received, in their send order. We have $((s_p)_{p\in\mathcal{P}}, (\nu_{p,q})_{p\neq q\in\mathcal{P}}) \xrightarrow{a} ((s'_p)_{p\in\mathcal{P}}, (\nu'_{p,q})_{p\neq q\in\mathcal{P}})$ iff

- for $a \in \Sigma_p$, we have $s_p \xrightarrow{a} s'_p$ and $s'_r = s_r$ for all $r \neq p$,
- for $a = p!q(c)$ or $a = q?p(c)$, $\nu'_{q,r} = \nu_{q,r}$ for all $(r, s) \neq (p, q)$,
- for $a = p!q(c)$, $\nu'_{p,q} = \nu_{p,q}c$, that is c is appended at the end of $\nu_{p,q}$,

- for $a = q!p(c)$, $c\nu'_{p,q} = \nu_{p,q}$, that is c is removed from the beginning of $\nu_{p,q}$,

An execution of a CFSM ($\mathcal{A}_p$)is a word on alphabet Σ which is in the language $\mathcal{L}(\mathcal{A})$. It is easy to see that if $w \in \mathcal{L}(\mathcal{A})$, then for all w' equivalent with w (that is with the same MSC), we have $w' \in \mathcal{L}(\mathcal{A})$ as well, that is $\mathcal{L}(\mathcal{A})$ is closed by commutation.

Thanks to the closure by commutation, one can apply the technique described in the previous section. We call a CFSM $\mathcal{A}$ existentially b-bounded if its language $\mathcal{L}(\mathcal{A})$ is. We can state the decidability model checking for existentially bounded CFSM against an MSO formula [MM01].

Proposition 4 *Let $\mathcal{A}$ be a CFSM existentially b-bounded. Then for all MSO formula φ on MSCs, it is decidable whether $\mathcal{A}$ has an execution which does not satisfy φ.*

Existentially bounded CFSM having good properties, it is interesting to wonder whether it is a decidable class. For decidability, deadend-free CFSM are important, namely CFSMs such that from every reachable configuration, a final state can be reached. We have the following results:

Theorem 2 *[fi07] Let $\mathcal{A}$ a CFSM. It is undecidable to know whether there exists b such that $\mathcal{A}$ is existentially b-bounded, even if $\mathcal{A}$ is deadend-free.*

Given b, it is undecidable to know whether $\mathcal{A}$ is existentially b-bounded.

Given b, knowing whether $\mathcal{A}$ is existentially b-bounded is PSPACE-complete.

We can actually apply a semi-algorithm to know if there exists b such that $\mathcal{A}$ is existentially b-bounded in the case where $\mathcal{A}$ is deadend-free. If we interpret $\mathcal{A}$ as weak FIFO, that is messages with different contents can overtake each other, we can interpret the CFSM as a Petri Net and modify the algorithm for boundedness in Petri Net to answer this question. This interpretation does not remove runs from the CFSM, but can create (unbounded) ones. Thus if the algorithm answers that the CFSM is bounded with weak FIFO, then it is bounded with the usual FIFO interpretation as well. We state the theorem for the case where there is no content of message, for which both interpretations collapse.

Theorem 3 *[i&c10] Let $c \in C$. Let $\mathcal{A}$ be a deadend-free CFSM sending only message of content c. Then it is decidable whether there exists b such that $\mathcal{A}$ is existentially b-bounded.*

5.4 Regular Set of Representatives as Model

Another way to specify a communicating system is to use representative sets as model. That is, assume that we have an automaton $\mathcal{A}$ over Σ whose language $\mathcal{L}(\mathcal{A})$ accepts only well-formed and complete runs. However, $\mathcal{L}(\mathcal{A})$ is not necessarily closed by commutation, that is, there may be two words w, w' with the same MSC such that $w \in \mathcal{L}(\mathcal{A})$ and $w' \notin \mathcal{L}(\mathcal{A})$. We can give an MSC interpretation to this automaton: $\mathcal{L}_{MSC}(\mathcal{A})$ is the set of MSCs associated to runs of $\mathcal{A}$. We denote by $[\mathcal{L}(\mathcal{A})] = \{w \mid w \text{ is associated with an MSC in } \mathcal{L}_{MSC}(\mathcal{A})\}$. That is, we consider every word equivalent with a word of $\mathcal{L}(\mathcal{A})$. It is easy to see that $\mathcal{L}(\mathcal{A})$ is a regular set of representatives for $[\mathcal{L}(\mathcal{A})]$. In general, $[\mathcal{L}(\mathcal{A})]$ is not regular, although $\mathcal{L}(\mathcal{A})$ is.

This specification corresponds to *safe Compositional MSC-graphs* [GMP03], which have nice properties. First, it is easy to show that there exists a b such that all the words in $\mathcal{L}(\mathcal{A})$ are b-bounded [jcss06]. That is, $[\mathcal{L}(\mathcal{A})]$ is existentially b-bounded.

Using the set of representatives idea, as the set of words satisfying an MSO formula and the language of a CFSM are both closed by commutation, by checking only those words in $\mathcal{L}(\mathcal{A})$, we obtain:

Theorem 3 *[jcss06] Let $\mathcal{A}$ be a safe Compositional MSC-graph, φ be an MSO formula over MSCs, and let $\mathcal{B}$ be a CFSM. Then*

- *One can check whether there exists a word in $[\mathcal{L}(\mathcal{A})]$ which satisfies φ.*
- *One can check in PSPACE whether there exists a word in the intersection of $[\mathcal{L}(\mathcal{A})]$ and $\mathcal{L}(\mathcal{B})$.*

5.5 Related work

There are many more results, on the topic of communicating finite state machines, but the ones stated in this chapter are quite representative of what can be done. We now compare with other connected results.

Direct proofs for asynchronous communication. Historically, proofs for asynchronous message passing systems have been performed without using the trace alphabet, either for model checking [AY99] or for implementation [HMNST05]. The proofs tend to be more tedious as messages in transit

need to be kept explicitly. Techniques like bounding phases (cycles between processes of unbounded queues) [BE12] can however handle well complexity.

Implementation Results. Concerning implementation, many results have been proved. The first results concern implementation of CFSM equivalent to a global specification, either with FIFO [AEY03] or with weak FIFO [Mor02]. The latter enjoys decidability in slightly more cases thanks to the theory of Petri Nets. In the case where processes of a CFSM are allowed to pass some information together with messages, equivalent implementation can be obtained in more cases, using the Zielonka's construction. This has been done directly [HMNST05] or using trace encoding [Kus02] for bounded channels, later extended to existentially bounded channels [i&c06]. Non-Zielonka construction has also been used [BM07], resulting to non-deterministic implementation. Last, implementation can also be made from a somehow local EMSO logic using direct message and process order [BL06], using Hanf [Han65] and Gaifman [Gai82] theorems.

Serializability. Existential channel bounds are still under studies in the Software Engineering community, particularly concerning web services. Names and setting may slightly differ though (e.g. conversation protocols where message receives are not specified). This line of work started with the serializability notion of Bultan [BB14], which is roughly equivalent to existentially-1-bounded. A system is serializable [BB14] if assuming that each message are received as soon as they are sent (ie. synchronous messages), then the behavior of the system is not reduced. In [BB14, SY15], this is extended to the existence of a bound such that no behavior is lost once this bound is fixed on message channels, similar to existential-boundedness.

Programming Languages. Recently, a new global specification formalism has been defined in the realm of programming language and type theory, namely (multiparty) *session types* [DY12, DY13]. In [DY12], well-formed global session types defines a subset of local-choice MSC-graphs [HJ00, jcss06]. They are transformed to CFSM by considering the projection to each processes [AEY03]. In [DY13], it is shown how to a build global session type automatically from a serializable CFSM.

Quasi Static Scheduling. *Quasi Static Scheduling* (*QSS*) [CKLPW04] asks to design a scheduler which does not disallow any local choice, such that a distributed system under schedule is regular. This control problem is actually tightly related to existential channel bound (which also generate a sort

of scheduler which does not disallow any local choices). In [concur08, i&c10], we showed that QSS is undecidable in general (even for system without deadends). We show that QSS is equivalent with existential channel bound, and thus decidable, in the case where the system is deadend-free and data-branching, that is there is no state with two outgoing receive actions (the choice of process p cannot be made by another process q sending two different messages to p). Previously, only heuristic were known [CKLPW04].

Extensions of Communicating Systems. Communicating systems have been extended in several ways. Different kind of product forms have been considered for specifying communicating systems: parallel composition of different modules synchronized on common actions [fossacs08], or low level fibered interleaving [tcs09]. It is also possible to automatically implement *parametric CFSM* where the number of processes can evolve [Bol14]. Last, recently, the techniques expressing the partial order as a bounded treewidth graphs have been used to prove the decidability of model check of existentially bounded CFSMs with stacks and data [CG14].

5.6 Perspectives

We now have solid foundations for the theory of CFSMs. Properties are well understood, as well as numerous techniques to exploit them. For instance, it should not be too difficult to lift results on the control of asynchronous automata (see previous chapter) to the control of bounded CFSMs.

There are still several things which remain to be done. The first one is to export these foundations to other field, such as Programming Languages and Software Engineering, and in particular Web Service choreography (conversation protocols, etc). For instance, allowing some liberty in the implementation would allow implementing more specification automatically, as was done for CFSM. A particularly interesting class is the class of Local choice specification [jcss06], which has not yet been considered in Web Service Choreography.

As was recently shown [CG14], new techniques can still bring new result concerning extensions of CFSMs. For instance, analyzing quantified values in CFSMs is important. We will discuss about *timed* system in the next chapter. Probabilistic CFSMs are also an important topic which has seen very few results so far. One reason is that it is not trivial to define an intuitive probability law on distributed systems [Abb14].

Chapter 6

Timed and Distributed Systems.

6.1 Introduction

Systems involving both time and concurrency are notoriously difficult to analyze. The main problem is that reasoning about partial orders is difficult in the presence of global time, which imposes a total order on executed events. In this chapter as in the previous one, the concurrent systems we consider use mainly message to communicate. In this setting, existing decidability results apply when clocks on different processes cannot be compared or when the set of timed executions is regular [AD94]. We now show how using representatives, we can get decidability results for message passing systems with time, requiring neither restriction. First, we consider a non-regular timed system given under the form of a set of representatives, namely time-constrained MSC-graphs (TC-MSC graphsfor short) introduced in [AMN07]. We study if the set of timed executions generated by a TC-MSC graph is empty. This emptiness problem is known to be undecidable in general [GMN09].

We describe two different approaches. The first one consists in finding a regular set R of *representative* timed executions, i.e., such that every consistent timed-constrained scenario of the system has a timed execution in R. This extends the results of the previous chapter, showing that set of representatives can be useful also in a timed setting. This allows us to decide the emptiness problem under the assumption that the TC-MSC graph is prohibited from (1) *forcing* any basic scenario to take an arbitrarily long time to complete and (2) *enforcing* unboundedly many events to occur within one unit of time [ipl12].

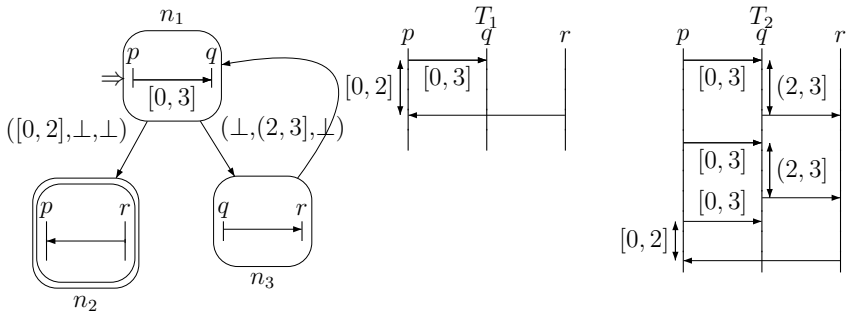


Figure 6.1: A TC-MSC graph G_1 and two TC-MSCs it generates

The second approach is more ad hoc but also more efficient, as we do not keep several equivalent interleaved representations. Instead, time constraints are encoded symbolically as a system of inequalities. We obtain decidability when condition (1) above is met. That is, we can drop condition (2) above. Also, we can effectively check condition (1): for a given K, it is decidable whether *a path* of the TC-MSC graph *forces* some node to take more than K time units to complete [ictac12].

6.2 Time-Constrained MSC graphs

Let $\mathcal{I}(\mathbb{N})$ denote the set of open and closed intervals whose end points are in $\mathbb{N}$, plus the intervals of the form $[c, \infty)$, (c, ∞), where $c \in \mathbb{N}$. We shall use intervals in $\mathcal{I}(\mathbb{N})$ to constrain the lower and upper bounds on the difference of occurrence times of events in a scenario. We remark that in what follows, intervals involving non-negative rational numbers can be easily simulated by scaling them to integers. We adopt the basic definitions from [AMN07].

Definition 7 *A* time-constrained message sequence chart *(*TC-MSC*) over a set $\mathcal{P}$ of processes is a tuple $T = (E, (<_p)_{p\in\mathcal{P}}, \lambda, \mu, \delta)$ where E is a finite non-empty set of* events*; $\lambda : E \to \Sigma$ labels each event with an action type in Σ such that:*

(i) Each $<_p \subseteq E_p \times E_p$ is a total order, where $E_p = \lambda^{-1}(\{p\} \times \{!, ?\} \times \mathcal{P})$. Members of E_p are termed p-events.

(ii) The message relation μ is a bijection from $E_{send} = \lambda^{-1}(\mathcal{P} \times \{!\} \times \mathcal{P})$ (send events) to $E_{recv} = \lambda^{-1}(\mathcal{P} \times \{?\} \times \mathcal{P})$ (receive events). For any e, f with $\mu(e) = f$, for some p, q, we have $\lambda(e) = p!q$ and $\lambda(f) = q?p$. For each e, e' with $\lambda(e) = \lambda(e') = p!q$ for some $p, q \in \mathcal{P}$, we have $e <_p e'$ iff $\mu(e) <_q \mu(e')$. (FIFO)

(iii) Writing $<$ for the transitive closure of $(\bigcup_{p \in \mathcal{P}} <_p) \cup \mu$, the time constraint labeling function δ associates an interval in $\mathcal{I}(\mathbb{N})$ to each pair of events $(e, f) \in E \times E$ with $e < f$.

With a slight abuse of notation, we write a TC-MSC as $T = (E, <, \lambda, \mu, \delta)$, with $<$ as above. A *linearization* of T is a sequence $\sigma = a_1 \dots a_\ell$ over Σ^*, where $\ell = |E|$ and such that E can be enumerated as $e_1 \dots e_\ell$ with $a_i = \lambda(e_i)$, and $e_i < e_j$ implies $i < j$ for any i, j in $\{1, \dots, \ell\}$. Note that due to the FIFO condition (see Condition (ii) in the definition above), the enumeration $e_1 \dots e_\ell$ is uniquely determined by $a_1 \dots a_\ell$. A TC-MSC T defines a collection of linearizations augmented with occurrence times such that the relative delay between each pair of causally ordered events falls within the interval dictated by δ. To avoid confusion, we shall term occurrence times as *dates*: A *timed execution* w of T is a sequence $(a_1, d_1) \dots (a_\ell, d_\ell)$, where $a_1 \dots a_\ell$ is a linearization of T, each date d_i is a non-negative real for $i = 1, \dots, \ell$, and $d_1 \leq \dots \leq d_\ell$. Let $e_1 \dots e_\ell$ be the enumeration corresponding to the linearization $a_1 \dots a_\ell$. Then $e_i < e_j$ implies $d_j - d_i$ is in the interval $\delta(e_i, e_j)$.

To describe infinite collections of TC-MSCs, we use TC-MSC graphs:

Definition 8 *Let $\mathcal{T}$ be a finite non-empty set of TC-MSCs. A* TC-MSC graph *over $\mathcal{T}$ is a tuple $G = (N, \longrightarrow, n_{ini}, N_{fin}, \Lambda, \Delta)$ where N is a finite set of nodes, $\longrightarrow \subseteq N \times N$ a transition relation, $n_{ini} \in N$ the initial node, $N_{fin} \subseteq N$ the subset of final nodes, and $\Lambda : N \to \mathcal{T}$ labels each node with a TC-MSC from $\mathcal{T}$. Further, the mapping Δ associates each transition (n, n') in $\longrightarrow$ with a $\mathcal{P}$-indexed family of intervals in $\mathcal{I}(\mathbb{N})$, such that if either n or n' has no p-event, then the p-component of $\Delta(n, n')$ is $[0, \infty)$.*

For each p, we write $\Delta_p(n, n')$ for the p-th component of $\Delta(n, n')$. The interval $\Delta_p(n, n')$ specifies the range of relative delay on p when moving from n to n'. We write $\perp$ for the interval $[0, \infty)$. Figure 6.1 displays a TC-MSC graph G_1 whose nodes are n_1, n_2, n_3, with n_1 being the initial node and n_2 the final node. In n_1, the relative delay between the send event of p and the receive event of q is constrained to lie within $[0, 3]$. The $([0, 2], \perp, \perp)$ on

transition (n_1, n_2) indicates $\Delta_p(n_1, n_2) = [0, 2]$, $\Delta_q(n_1, n_2) = \bot$, $\Delta_r(n_1, n_2) = \bot$. It asserts that the relative delay between the last event of p of n_1 and the first event of p of n_2 should be in $[0, 2]$. To reduce clutter in the figures, we omit time constraints of the form $\bot$ inside a TC-MSC labeling a node, and $(\bot)^{|\mathcal{P}|}$ on transitions.

We fix a TC-MSC graph $G = (N, \longrightarrow, n_{ini}, N_{fin}, \Lambda, \Delta)$. We write $n \longrightarrow n'$ for $(n, n') \in \longrightarrow$ and speak interchangeably of a node n and its associated TC-MSC $\Lambda(n)$. A TC-MSC graph defines a collection of TC-MSCs arising from concatenating TC-MSCs in paths of G. First, for a TC-MSC $T = (E, <, \lambda, \mu, \delta)$, we call the $<_p$-minimal event in E_p the *first* p-event, and the $<_p$-maximal event in E_p the *last* p-event. Simply put, for a transition (n, n'), the concatenation of n with n' is the TC-MSC resulting from placing n' after n, and for each process p, take $\Delta_p(n, n')$ to be the time constraint between the last p-event of n and the first p-event of n'. Formally, letting $\Lambda(n) = (E, <, \lambda, \mu, \delta)$ and $\Lambda(n') = (E', <', \lambda', \mu', \delta')$, the *concatenation* of $\Lambda(n)$ and $\Lambda(n')$, denoted $\Lambda(n) \circ \Lambda(n')$, is the TC-MSC $(E'', <'', \lambda'', \mu'', \delta'')$ detailed as follows. Firstly, E'' is the disjoint union of E and E'; λ'' agrees with λ on events in E, and with λ' on events in E'. Secondly, for each p, $<''_p$ is $<_p \cup <'_p \cup E_p \times E'_p$; μ'' is the union of μ and μ'. Lastly, for $e, f \in E''$ with $e <'' f$, $\delta''(e, f)$ is given as follows: (i) if $e, f \in E$, then $\delta''(e, f) = \delta(e, f)$; (ii) if $e, f \in E'$, then $\delta''(e, f) = \delta'(e, f)$; (iii) suppose $e \in E$, $f \in E'$. If for some p, e is the last p-event of n and f the first p-event of n', $\delta''(e, f) = \Delta_p(n, n')$, otherwise, $\delta(e, f) = \bot$. Note that the restriction $\Delta_p(n, n') = \bot$ whenever $E^n_p = \emptyset$ or $E^{n'}_p = \emptyset$ in Definition 8 is equivalent to the restrictions in [AMN07, GMN09]. It implies that $\circ$ is associative.

A *path* of G is a sequence of nodes $\rho = n_0 \dots n_\ell$ of G such that $n_0 = n_{ini}$ and $n_i \longrightarrow n_{i+1}$ for $i = 0, \dots, \ell - 1$. Since $\circ$ is associative, we can unambiguously define the TC-MSC induced by ρ, denoted T^ρ, to be $\Lambda(n_0) \circ \dots \circ \Lambda(n_\ell)$. The path ρ is *final* if $n_\ell \in N_{fin}$. The *TC-MSC language* of G is the set of TC-MSCs induced by final paths of G. For a TC-MSC T, let $\mathcal{L}(T)$ denote its set of timed executions. For TC-MSC graph G, the *timed execution language* of G, denoted $\mathcal{L}(G)$, is the union of $\mathcal{L}(T^\rho)$ over final paths ρ of G. That is, in $\mathcal{L}(G)$, there may be events from one node appearing before all the events of the previous nodes are executed, because G is defines as a regular set of representatives representing all the equivalent executions $\mathcal{L}(G)$. We say that a TC-MSC T (resp. a path ρ) is *consistent* iff $\mathcal{L}(T) \neq \emptyset$ (resp. $\mathcal{L}(T^\rho) \neq \emptyset$). In what follows, timed executions of the TC-MSC T^ρ are

sometimes refered to as timed executions of ρ and $\mathcal{L}(\rho)$ refers to $\mathcal{L}(T^\rho)$.

We tackle the *emptiness problem for TC-MSC graphs*, which is: given a TC-MSC graph G, determine whether $\mathcal{L}(G)$ is empty. The emptiness of $\mathcal{L}(G)$ implies that for any TC-MSC T^ρ induced by a final path ρ of G, no assignment of dates to events in T^ρ can satisfy all the time constraints in T^ρ. Thus, such a G with $\mathcal{L}(G) = \emptyset$ should be considered ill-specified, and should be checked for. However, it is known from [GMN09] that the emptiness problem for TC-MSC graphs is undecidable. In [AMN07], decidability is obtained for locally-synchronized TC-MSC graphs. This syntactical restriction limits concurrency, and implies that the timed execution language is regular, which is a severe restriction. Indeed, even simple examples, such as G_1 from Figure 6.1 or the producer-consumer protocol, do not have regular timed execution languages.

6.3 Regular Set of Representative Timed Executions

We demonstrate how *regular sets of representatives* can be used to obtain decidability of the emptiness problem for TC-MSC graphs. Here, regular stands for *timed regular*, i.e., languages accepted by finite timed automata [AD94]. Notice that timed regularity implies regularity of the untimed projection of the timed language.

Let R be a set of *representative timed executions* for G. That is, for all consistent final path ρ of G, there exists a timed execution w in R such that w is a timed execution of $\mathcal{L}(\rho)$. We have $\mathcal{L}(G) = \emptyset$ iff $R = \emptyset$. Now, many timed executions of a TC-MSC graph G are equivalent, in the sense that they are timed executions of the TC-MSC induced by the same final path of G. To check for emptiness of $\mathcal{L}(G)$, it suffices to consider emptiness of a set R of representatives for G, instead of $\mathcal{L}(G)$ itself. If R turns out to be regular and effective, then the emptiness problem for TC-MSC graphs can be decided. For example, consider G_2 in Figure 6.2. The language $\mathcal{L}(G_2)$ is not regular. However, the set $\{\sigma_0, \sigma_0\sigma_1, \sigma_0\sigma_1\sigma_2, \ldots\}$, where $\sigma_i = (p!q, 4i)(q?p, 4i+1)(s!r, 4i+2)(r?s, 4i+3)$ for all $i \in \mathbb{N}$, is a regular set of representatives for G_2.

We fix TC-MSC graph $G = (N, \longrightarrow, n_{ini}, N_{fin}, \Lambda, \Delta)$, a path $\rho = n_0 \ldots n_\ell$ of G, a timed execution $w = (a_1, d_1) \ldots (a_h, d_h)$ of ρ, and $e_1 \ldots e_h$ the enu-

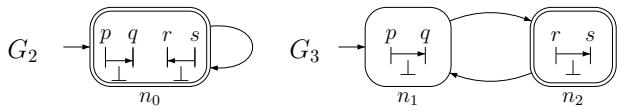


Figure 6.2: Two TC-MSC graphs G_2, G_3. Specification G_2 is scenario-connected and G_3 is not.

meration of E associated with $a_1 \dots a_h$. We start by giving a first set of representatives.

Definition 9 *Let K be an integer. We call w* K-drift-bounded *if for each $0 \leq u \leq \ell$, and $i, j \in \{1, \dots, h\}$, if e_i, e_j are in $\Lambda(n_u)$, then $|d_i - d_j| \leq K$.*

Thus w is K-drift-bounded if the difference between the first and last date associated with an event of any TC-MSC $\Lambda(n_u)$ is bounded by K. Interpreting the scenario in each node of a TC-MSC graph as one phase or transaction of a distributed protocol, it is realistic to believe that at least some (but not necessarily all) executions of an implemented system are K-drift-bounded.

Now, for a TC-MSC graph G and an integer K, we say that G is *K-drift-bounded* if for every consistent path ρ of G, there *exists* a K-drift-bounded timed execution in $\mathcal{L}(\rho)$. We emphasize that *all* timed executions of $\mathcal{L}(\rho)$ are not required to be K-drift-bounded. Observe that, G being K-drift bounded implies that the set $L_K(G)$ of K-drift-bounded executions of G is a set of representatives of G. Unfortunately this set may not be regular because of arbitrarily many events executed in one time unit. We thus introduce the following.

For $K' \in \mathbb{N}$, w has *at most K' events per unit of time* if for any $i, j \in \{1, \dots, h\}$, $d_j - d_i \leq 1$ implies $j - i < K'$. A language L is *strongly non-Zeno* [BBBB09] if there exists $K' \in \mathbb{N}$ such that every execution of L has at most K' events per unit of time. It turns out that by imposing the following syntactical condition, a TC-MSC graph has a strongly non-Zeno set of representatives (this is one consequence of Proposition 5 below). We say that a transition (n, n') of G is *positively constrained* if for every p, $\Delta_p(n, n')$ is *not* $[0, 0]$ (but can be $[0, 1)$, $[3, 3]$, $[2, \infty) \dots$). G is *positively constrained* if every transition of G is positively constrained. This restriction does not imply that $\mathcal{L}(G)$ is itself strongly non-Zeno: consider the positively constrained TC-MSC graph G_2 of Figure 6.2 (where transitions without labels are implicitly labeled by $\Delta_p = \perp$ for all p). $\mathcal{L}(G_2)$ is *not* strongly non-Zeno since

unboundedly many events can occur at date 0 (and hence within a unit of time). However, there exist timed executions where time elapses between positively constrained transitions.

We now present our regular set of representatives, namely the set $L_{K,K'}(G)$ of (K, K')-well-behaved timed executions, as well as the restriction needed on G for representativity.

Definition 10 *A TC-MSC graph is* K-well-formed *if it is K-drift-bounded and positively constrained. It is* scenario-connected *if for every transition (n, n'), there exists a process p s.t. both n and n' have at least one p-event.*

For instance, in Figure 6.2, G_2 is scenario-connected while G_3 is not.

Proposition 5 *[ipl12] Let $K \in \mathbb{N}$. If G is K-well-formed and scenario-connected, then $L_{K,K'}(G)$ is a regular set of representative timed executions of $\mathcal{L}(G)$, with $K' = (4(|\mathcal{P}| + 1) + 2) \cdot |\mathcal{P}| \cdot M$, where M is the max number of events in a node of G.*

With a regular set of representatives, we can obtain the decidability of emptiness. We can actually lift the scenario-connected restriction still using this method by building an equivalent TC-MSC graph which is scenario-connected from any non scenario-connected TC-MSC graph[ipl12].

In the following, we give another technique based on a symbolic representation which allows to lift the non-Zeno restriction as well as enable decidability of whether G is K-well-formed.

6.4 Symbolic Encoding

To avoid clutter, we assume that constraints are only of the form $[a, b]$ and $[a, \infty)$. We also assume that the TC-MSC graph H is full, that is every node has at least one event on every process. We show in [ictac12] how to generate an equivalent drift bounded full TC-MSC graph from any drift bounded TC-MSC graph by adding dummy events. We first describe intuitively the key ingredients of the proof.

- First, we observe that checking consistency of a path ρ of H, i.e., $\mathcal{L}(T^\rho) \neq \emptyset$, is equivalent to checking for the existence of a solution to a system of inequalities over (real-valued) variables x_e depicting the dates of events e of T^ρ.

- Next, checking whether a dated MSC can be extended by a node by assigning appropriate dates to events of this node can be done with information only on the relative difference of dates of the last event of the dated MSC on each process. This motivates us to associate a symbolic profile $PF(\rho)$ to each path ρ. A symbolic profile is a system of inequalities whose solutions correspond to the dates of final events of dated MSCs generated by T^ρ, and vice versa. In particular, $PF(\rho)$ has a solution iff ρ is consistent.

- We remark that constants appearing in symbolic profiles can be chosen as integers. Restricting constants to be within $[-K, K]$ does not exclude any consistent *K-drift-bounded* path of H. We can then represent with a finite automaton the set of consistent *K-drift-bounded* paths of H.

Systems of inequalities and Fourier-Motzkin elimination. We first fix basic terminologies for systems of difference inequalities. Let X be a finite nonempty set of real-valued variables. A *(difference) inequality* is an inequality of the form $x - y \leq a$, where x, y are two different variables in X.

Definition 11 *A* system of (difference) inequalities φ *over* X *is* $\wedge_{(x,y)\in R}\, x - y \leq a_{xy}$ *where* $R \subseteq X \times X$ *is an irreflexive relation. We say that* φ *has* integral coefficients *whenever* a_{xy} *is a* (possibly negative) integer *for all* $(x, y) \in R$.

We assume that the system is *simplified*, that is, for each $x, y \in X$, there is at most one inequality of the form $x - y \leq a$. If $x - y \leq a$ appears in φ, we say that φ contains an *edge* (x, y), and the weight of this edge is a. We say that two systems φ, ψ of inequalities are *equivalent* when φ has a solution (in the real domain) iff ψ has a solution (in the real domain).

A key idea is to propagate constraints concerning variables in a subset $Y \subsetneq X$ on variables in $X \setminus Y$, and then safely remove variables in Y while keeping an equivalent system, using the *Fourier-Motzkin* elimination method.

For $F \subseteq X$, let $Res\varphi$ denote the (unique) system of inequalities over variables F obtained by performing Fourier-Motzkin elimination of variables in $X \setminus F$ following a fixed order. We have that φ and $Res\varphi$ are equivalent. If φ has *integral coefficients*, then so does $Res\varphi$.

Symbolic Profiles. Let $T^\rho = (E, (<_p), \mu, \lambda, \delta)$ be the TC-MSC associated with some path $\rho = n_0 \dots n_\ell$ of H. We denote by x_e a $\mathbb{R}_{\geq 0}$-valued variable, standing for the date of event $e \in E$, and let $X_E = \{x_e \mid e \in E\}$. We associate path ρ with a system of linear inequalities $\Phi(\rho)$ *with integral coefficients* as follows:

Definition 12 *The system $\Phi(\rho)$ associated with ρ is the smallest system of inequalities over the set of variables X_E such that, for any $e, f \in E$ with $e \lessdot f$,*

- *if $\delta(e, f) = [L, U]$, then $\Phi(\rho)$ contains both $x_f - x_e \leq U$ and $x_e - x_f \leq -L$;*
- *if $\delta(e, f) = [L, \infty)$, then $\Phi(\rho)$ contains $x_e - x_f \leq -L$.*

We easily have that ρ is consistent iff $\Phi(\rho)$ has a solution. Let e_p be the last event of T^ρ on p, for each process p. Let E_{last} be the set $\{e_p \mid p \in \mathcal{P}\}$. Using Fourier-Motzkin elimination of variables $X' = \{x_e \mid e \notin E_{last}\}$, we obtain a system $\Phi(\rho)|_{X_{last}}$ over variables $X_{last} = \{x_e \mid e \in E_{last}\}$, with integral coefficients, equivalent with $\Phi(\rho)$. Once simplified, this system has at most $|\mathcal{P}|^2$ inequalities with integral coefficients. We encode this system as a *symbolic profile*.

Definition 13 *A* symbolic profile *σ is a function from $\mathcal{P} \times \mathcal{P}$ to $\mathbb{Z} \cup \{\infty\}$. We denote by $\mathcal{PF}$ the (infinite) set of all profiles.*

Notice that symbolic profiles are *syntactically* similar to Difference Bounded Matrices (DBMs) [BY03] over $|\mathcal{P}|$ clocks. However, unlike a DBM, a symbolic profile may not correspond to a timed linearization, and the update function (defined below) is very different when compared to DBMs.

Let φ be a system of inequalities with integral coefficients over $X_{last} = \{x_p \mid p \in \mathcal{P}\}$. We define the symbolic profile $prof\,\varphi$ induced by φ as $prof\,\varphi[p, q] = a_{pq}$ if $x_p - x_q \leq a_{pq}$ belongs to φ, and $prof\,\varphi[p, q] = \infty$ otherwise. Intuitively, $prof\,\varphi[p, q] = \infty$ means that there is no inequality of the form $x_p - x_q \leq a_{pq}$ in φ. We abusively use $prof\,\varphi$ as a system of inequalities in the following, and denote x_p for x_{e_p}. For a path ρ, we denote $prof\,\rho = prof\,(\Phi(\rho))|_{X_{last}}$. We say that a symbolic profile $\sigma \in \mathcal{PF}$ is *satisfiable* if it has a solution. It is easy to check whether $prof\,\rho$ is satisfiable, either by using Fourier-Motzkin elimination till reaching a trivial equation, or by using Shostak characterization.

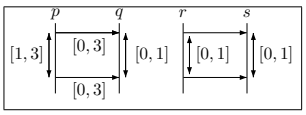


Figure 6.3: The TC-MSC induced by path $n_1 \cdot n_1$ of G_1 and its profile

Proposition 6 *$prof\,\rho$ is satisfiable iff ρ is consistent.*

As an example, consider the TC-MSC $T^{n_1 \cdot n_1}$ in Figure 6.3. Let e^i_j denote the i^{th} event on process j and E be the set of events of $T^{n_1 \cdot n_1}$. We obtain $\Phi(n_1 \cdot n_1)$ to be the set of inequalities over $X = \{x_e \mid e \in E\}$, where for instance the inequations $x_{e^2_p} - x_{e^1_p} \leq 3$ and $x_{e^1_p} - x_{e^2_p} \leq -1$ capture the timing constraint $[1, 3]$ between e^1_p and e^2_p. Now eliminating variables $x_{e^1_p}, x_{e^1_q}, x_{e^1_r}, x_{e^1_s}$ results in a set of equations on $X_{last} = \{x_{e^2_p}, x_{e^2_q}, x_{e^2_r}, x_{e^2_s}\} = \{x_p, x_q, x_r, x_s\}$ as shown. E.g., $prof\, n_1 \cdot n_1)[p, q] = \min(3, -1+3+1) = 3$ and $prof\, n_1 \cdot n_1)[s, r] = \infty$. This system of inequalities has many solutions.

Bounded profiles. Notice that the set of symbolic profiles as defined above is not finite in general (the coefficients range over $\mathbb{Z}$), and so, it cannot be recorded by a finite state automaton. Instead, we use bounded profiles.

Let $L \in \mathbb{N}$. The set $\mathcal{PF}_L$ is finite. We denote by $\Phi_L(\rho)$ the system of inequalities obtained from $\Phi(\rho)$ by the following modification: for each $i = 0, \ldots, \ell$, for any two different events e, f in the same node n of ρ, if $\Phi(\rho)$ contains $x_e - x_f \leq a_{e,f}$, then replace it by $x_e - x_f \leq min(a_{e,f}, L)$. If $\Phi(\rho)$ does not have an edge (e, f), then add the inequality $x_e - x_f \leq L$.

Proposition 7 *Let ρ be a path of a full TC-MSC graph H. Then $PF_L(\rho) \in \mathcal{PF}_L$, and $PF_L(\rho)$ is satisfiable iff ρ is consistent and L-drift-bounded.*

It is not possible to obtain $PF_L(\rho)$ directly from $PF(\rho)$. Instead, it can be computed inductively along the path ρ [ictac12].

This enables us to check both the emptiness of a K-drift-bounded TC-MSC graph G, and the K-drift-boundedness of any TC-MSC graph G, by working with a full TC-MSC graph constructed from G.

Theorem 4 *[ictac12] Let $K \in \mathbb{N}$ and G be a K-drift-bounded TC-MSC graph. Then checking whether $\mathcal{L}(G)$ is empty is decidable in PSPACE.*

The drift-boundedness hypothesis of Theorem 4 can be effectively checked using bounded profiles, giving rise to an effective decidability procedure.

Theorem 5 *[ictac12] Let $K \in \mathbb{N}$ and G be a TC-MSC graph. Then checking whether G is K-drift-bounded is decidable in PSPACE.*

6.5 Related Work

Other Constructions Compared with [AGMN10], which builds an automaton accepting every timed linearizations of a regular TC-MSC graph, [ictac12] provides a construction with a much smaller automaton in the worst case (exponential in $|\mathcal{P}|^2$ instead of exponential in $|G|$ for [AGMN10]). Further, being symbolic, the worst case is not always reached, contrary to constructions based on zones of timed automata [AGMN10, ipl12, AMN07, ZXLZZ02]. In [DL07], decidability is obtained for non-global time. That is, time is never compared between processes.

Time and Timed Petri Nets. In order to go beyond regular timed specification, well-structured transition systems is another possibility [FS01], whose most well-known member is Petri Nets. Two main extensions of Petri Nets with time have been made. In Time Petri Nets, transitions are labeled with a time constraint intervals: a transition is firable if it has been enabled for an amount of time inside the interval. Also, when a transition has been enabled for the maximal amount of time according to its associated interval, it must fire. This is called *urgency*. Most problems (reachability, termination, control-state reachability, boundedness) are undecidable for Time Petri Nets. To obtain decidability, one either restricts to bounded TPNs [BD91], or gives up urgency [RS09]. In this latter case, the untimed language of a TPN without urgency (also known as its weak-time semantics) is the language of the associated Petri Net without timing constraints.

On the other hand, Timed Petri Nets (also called Timed-arc Petri Nets) associate an age to each token [AN01]. The number of ages is in general unbounded. Each arc can be constrained by a timing interval: only tokens with age in the interval can be consumed by this transition. Timed Petri Nets cannot encode urgency [AN01]. Although the number of token ages is unbounded, the theory of well-structured transition systems [FS01] can be applied because of monotonicity (a token is always allowed to stay forever at a place as there is no urgency). Thus, control-state reachability and boundedness are decidable for Timed Petri Nets [AN01]. However, (marking) reachability is undecidable. These decidability results hold beyond regularity, for global time, as in [ipl12, ictac12]. It can be proved that

decidability is preserved in the presence of urgency on the bounded part of the net [icatpn16].

Unfolding and Partial Order Techniques. In [ZXLZZ02], a set of finite paths is analyzed using Partial Order Techniques, allowing to reduce the complexity by not considering all the paths but only a set of representative timed executions. In [BHR06] and [CCJ06], complete finite prefix of unfoldings are defined for network of timed automata, but it stays in the realm of regular systems.

6.6 Perspectives

The results of [ictac12] is interesting but holds in a particular setting of TC-MSC graphs. In order to generalize the result to other class of systems, one possibility could be to use the encoding of a timed distributed systems in a bounded treewidth graph [CG14], handling time symbolically as in [ictac12], using Fourier-Motzkin elimination.

In the last decade, robustness properties started to be considered, i.e, whether the system can withstand infinitesimal timing errors. This has been extensively studied for timed automata [Pur00, BMS13], etc. It would be interesting to extend the study started for TPNs (e.g. [AHJR12]) to distributed timed systems.

Chapter 7

Perspectives

This thesis presents several applications of the technique of regular set of representatives. This technique can be used to specify in a finite way infinite states distributed systems; to avoid some combinatorial blow up for model checking of distributed systems; and even to improve the complexity of implementing and controlling distributed systems. It can be applied to different classes of distributed models: network of automata, asynchronous automata, etc., and it can be extended to message passing systems and to distributed timed systems.

Using representatives to improve the complexity [Pel93] and to handle infinite states systems [GMP03, MM01] has been done before, even though not always totally explicitly [GMP03, MM01]. However, we believe that our work, spanning over a decade, showed that regular set of representatives should be considered seriously as a main technique to *tame the complexity* of any form of distributed systems. Still, there are other main techniques which should be considered as well to tame the complexity of distributed systems. Further, in many contexts, other techniques work simply better.

- First, other Partial Order Reduction techniques exist, in particular to reduce the number of global states of the system [Pel94, Val92, GW91], which is more effective at improving the speed of the algorithm.

- More importantly, unfoldings [McM95, ERV02] have shown their benefit to handle extremely distributed systems, for instance in diagnostic [FBHJ05].

- Recently, a new technique representing the partial orders as bounded

treewidth graphs [CG14] gives promising results to extend decidability to more complex distributed systems, with data, stacks, etc.

- An efficient method to find bugs is to assume some bound on the system, and look for bugs within this bounded framework, e.g. bounded phase analysis [BE12]. Such bounds can be controlled by efficient type systems [GMO06]. Type systems can actually model realistic message passing protocols [DY12, DY13].

However, there are still many contexts for which its simplicity and efficiency is clear (bounded model checking [WYKG08, KWG09], etc.). Last, it can be used in combination with other techniques to produce better results, for instance with POR of states [God94].

Concerning our perspectives on distributed systems, several levels should be considered. First, several topics, for which a lot of time has been invested, need to be completed (POR of transitions, control of distributed systems, etc.). The details are in the perspective section of each chapter. The techniques developed in this thesis should also be better advertised, in particular to other communities (Software engineering, etc.) in order to find more applications.

I devoted more than a decade working on algorithms for distributed systems, and I am slowly shifting away: as a large part of the model checking community, I recently switched to more quantitative matters: quantitative models and quantitative properties, in particular stochastic ones. I follow two tracks at the same time. First, discrete techniques (automata techniques, etc.) can be used in order to solve some quantitative problems. Concerning continuous time, it is well known [AD94] that discrete abstractions (regions, etc.) of the system allows to answer many questions, including quantitative questions (see also Chapter 6 on page 53). Concerning stochastic systems, discrete techniques can also be used to analyze many qualitative properties (e.g. almost all paths conforming to a property (that is the set of paths not conforming to the property is of measure zero)). We had some result on that front, solving qualitative questions for 2-player games with imperfect information [lics09], optimizing the number of queries in one-and-a-half-player games with imperfect information [fsttcs11], or solving the value-one problem for MDP with 1 continuous clock [qest14].

In a more original fashion for someone from formal methods, I am also devoted to develop approximation algorithms for the quantitative analysis of stochastic systems. It is a novel topic we discussed with the late Philippe

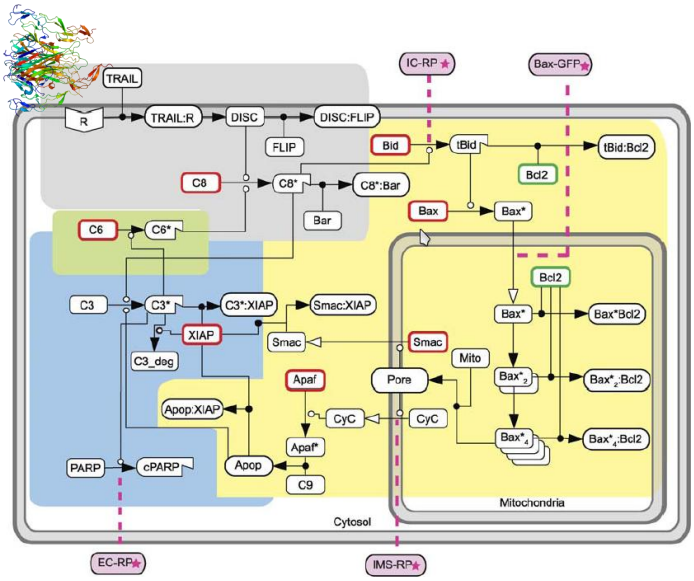


Figure 7.1: The TRAIL-induced apoptosis pathway.

Darondeau. The idea is to mix techniques from the formal methods with techniques from the AI and the Information Theory communities, which scales to large systems. One of the applications is computational system biology, developed inside ANR project STOCH-MC that I am leading. For instance, take the apoptosis pathway (one way for a cell to die), depicted in Fig.7.1. On reception of a biochemical message (e.g. Trail drug) binding to the cell, a set of biochemical reactions eventually create holes in the Mitochondria, and the cell dies. It involves around fifty biochemical species, and around an hundred chemical reactions. It is very hard to understand how this system works as a whole. Low level models of this biological system exist, generating distribution of time of death and survivability comparable to what is observed biologically. This model can be used only through time-consuming simulations with micro steps, making time evolve by few micro seconds only each step. It is thus very time consuming to analyze a model of this form using many simulations for extended period of time, e.g. modeling medical trials. The first challenge is to obtain a stochastic model behaving like the low level model, usually under the form a Dynamic Bayesian Network, that is a compact version of a Markov Chain [cmsb15, sub-1]. Analytical algorithms need to be designed to understand the property of the model.

The second challenge is to handle such a large system, usually corresponding to Markov chains with more than 10^{20} states. That's where approximations can be very useful: [cmsb11] builds on AI community approximate inference [MW01] in order to obtain algorithms running in polynomial time in the number of variables, that is logarithmic in the number of states of the Markov chain. The algorithm is parameterized by a value having direct impact on the accuracy and the runtime. Further, [tcbb12] performs an error analysis to bound errors made at each step of the approximation. If errors are too large, an approximation with better accuracy (but also slower) should be performed. We plan to use Information Theory techniques to get more efficient approximation schemes.

Here, properties are not expressed under the usual PCTL* format, familiar to many people in the model checking community. Indeed, the problem is not to know a measure on the set of paths satisfying a property (hence the time it takes for a path to satisfy a property can take any value), but rather to know whether there exists a time point for which the proportion of paths satisfying a property (which we can match with biological experiments at a given time point) is large or small enough. This latter kind of properties will be called distribution based properties. Surprisingly, these two kinds of properties are incomparable [BRS02]. Worse, while PCTL* have efficient model checking procedure - polynomial time for Markov Chain - this is not the case for distribution based properties. It has been shown [AAOW15] that checking whether the proportion of paths of a Markov chain which falls in a state never goes above some threshold at any time point is actually equivalent with Skolem, whose decidability is a long-standing open problem. This calls for a second type of approximations, in order get around undecidability [lics12, jacm15]. A challenge we will tackle in the forthcoming years is to obtain decidable fragments and approximation scheme for Markov Decision Processes for distribution based properties, which is highly non-trivial.

Index

(External) Bibliography

[AAOW15] S. Akshay, Timos Antonopoulos, Joel Ouaknine, James Worrell. Reachability problems for Markov Chains. *Information Processing Letters 115(2)*, pp. 155-158, 2015.

[Abb14] Samy Abbes. Processus probabilistes asynchrones. Habilitation, University Paris 7 Diderot, 2014.

[ABGM09] Serge Abiteboul, Pierre Bourhis, Alban Galland, Bogdan Marinoiu. The AXML Artifact Model. In *TIME'09*, pp. 11-17, IEEE, 2009.

[AD94] Rajeev Alur, David Dill. A Theory of Timed Automata. *Theoretical Computer Science 126(2)*, pp. 183-235, 1994.

[AEY03] Rajeev Alur, Kousha Etessami, Mihalis Yannakakis. Inference of Message Sequence Charts. *IEEE Transaction on Software Engineering 29(7)*, pp. 623-633, 2003.

[AGMN10] S. Akshay, Paul Gastin, Madhavan Mukund, K. Narayan Kumar. Model checking time-constrained scenario-based specifications. In *FSTTCS'10*, pp. 290-302, LNCS 4855, 2010.

[AHJR12] S. Akshay, Loic Hélouët, Claude Jard, Pierre-Alain Reynier. Robustness of Time Petri Nets under Guard Enlargement. In *RP'12*, pp. 92-106, LNCS 7550, 2012.

[AMNN05] Bharat Adsul, Madhavan Mukund, K. Narayan Kumar, Vasumathi Narayanan. Causal closure for MSC languages. In *FSTTCS'05*, pp. 335-347, LNCS 3821, 2005.

[AMN07] S. Akshay, Madhavan Mukund, K. Narayan Kumar. Checking Coverage for Infinite Collections of Timed Scenarios. In *CONCUR'07*, pp. 181-196, LNCS 4703, 2007.

[AN01] Parosh Abdulla, Aletta Nylén. Timed Petri Nets and BQOs. In *ICATPN'01*, pp. 53-70, LNCS 2075, 2001.

[AY99] Rajeev Alur, Mihalis Yannakakis. Model Checking of Message Sequence Charts. In *CONCUR'99*, pp. 114-129, LNCS 1664, 1999.

[Bau09] Nicolas Baudru. Compositional synthesis of asynchronous automata. *Theoretical Computer Science*, 412(29), pp. 3701-3716, 2011.

[BB14] Samik Basu, Tewfik Bultan. Automatic verification of interactions in asynchronous systems with unbounded buffers. In *ASE'14*, pp. 743-754, ACM-IEEE, 2014.

[BBBB09] Christel Baier, Thomas Brihaye, Nathalie Bertrand, Patricia Bouyer. When are timed automata determinizable? In *ICALP'09 (2)*, pp. 43-54, LNCS 5556, 2009.

[BD91] Bernard Berthomieu, Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. in Software Engineering 17(3)*, pp. 259-273, 1991.

[BE12] Ahmed Bouajjani, Michael Emmi. Bounded Phase Analysis of Message-Passing Programs. In *TACAS'12*, pp. 451-465, LNCS 7214, 2012.

[BHR06] Patricia Bouyer, Serge Haddad, Pierre-Alain Reynier. Timed Unfoldings for Networks of Timed Automata. In *ATVA'06*, pp. 292-306, LNCS 4218, 2006.

[BKM10] Benedikt Bollig, Dietrich Kuske, Ingmar Meinecke. Propositional Dynamic Logic for Message-Passing Systems. *Logical Methods in Computer Science 6(3)*, 2010.

[BL06] Benedikt Bollig, Martin Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science 358(2-3)*, pp. 150-172, 2006.

[BM05] Nicolas Baudru, Rémi Morin. Unfolding Synthesis of Asynchronous Automata. In *CSR'06*, pp. 46-57, LNCS 3967, 2006.

[BM07] Nicolas Baudru, Rémi Morin. Synthesis of Safe Message-Passing Systems. In *FSTTCS'07*, pp. 277-289, LNCS 4855, 2007.

[BMS13] Patricia Bouyer, Nicolas Markey, Ocan Sankur. Robustness in Timed Automata. In *RP'13*, pp. 1-18, LNCS 8169, 2013.

[Bol14] Benedikt Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*, pp. 18:1-18:10, ACM, 2014.

[BRS02] Danièle Beauquier, Alexander Rabinovich, Anatol Slissenko. A Logic of Probability with Decidable Model Checking. In *CSL'02*, pp. 306-321, LNCS 2471, 2002.

[BY03] Johann Bengtsson, Wang Yi. On Clock Difference Constraints and Termination in Reachability Analysis of Timed Automata. In *ICFEM'03*, pp. 491-503, 2003.

[BZ83] Daniel Brand, Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM 30(2)*, pp. 323-342, 1983.

[CCJ06] Frank Cassez, Thomas Chatain, Claude Jard. Symbolic Unfoldings For Networks of Timed Automata. In *ATVA'06*, pp. 307-321, LNCS 4218, 2006.

[CG14] Aiswarya Cyriac, Paul Gastin. Reasoning About Distributed Systems: WYSIWYG. In *FSTTCS'14*, pp. 11-30, LIPIcs, 2014.

[CKLPW04] Jordi Cortadella, Alex Kondratyev, Luciano Lavagno, Claudio Passerone, and Yosinori Watanabe. Quasi-static scheduling of independent tasks for reactive systems. In *Lectures on Concurrency and Petri Nets*, pp. 345-401, LNCS 3098, 2004.

[CMZ93] Robert Cori, Yves Métivier and Wiesla Zielonka. Asynchronous Mappings and Asynchronous Cellular Automata. *Information and Computation*, 106(2), pp. 159-202, 1993.

[DG04] Volker Diekert, Paul Gastin. Pure Future Local Temporal Logics Are Expressively Complete for Mazurkiewicz Traces. In *LATIN'04*, pp. 232-241, LNCS 3328, 2004.

[Dar07] Philippe Darondeau. Synthesis and Control of Asynchronous and Distributed Systems. In *ACSD'07*, pp. 13-22, IEEE, 2007.

[DL07] Catalin Dima and Ruggero Lanotte. Distributed Time-Asynchronous Automata. In *ICTAC'07*, pp. 185-200, LNCS 4711, 2007.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[DY12] Pierre-Malo Deniélou, Nobuko Yoshida. Multiparty Session Types Meet Communicating Automata. In *ESOP'12*, pp. 194-213, LNCS 7211, 2012.

[DY13] Pierre-Malo Deniélou, Nobuko Yoshida: Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP'13*,pp. 174-186, LNCS 7966, 2013.

[EKS08] Javier Esparza, Pradeep Kanade, Stefan Schwoon. A negative result on depth-first net unfoldings. *STTT 10(2)*, pp. 161-166, 2008.

[ERV02] Javier Esparza, Stefan Römer, Walter Vogler. An improvement of McMillan's unfolding algorithm. *FMSD 20*, pp. 285-310, 2002.

[FBHJ05] Eric Fabre, Albert Benveniste, Stefan Haar, Claude Jard. Distributed Monitoring of Concurrent and Asynchronous Systems. *J. Discrete Event Dynamic Systems 15(1)*, pp. 33-84, 2005.

[FO14] Bernd Finkbeiner, Ernst-Rüdiger Olderog. Petri Games: Synthesis of Distributed Systems with Causal Memory. In *GandALF'14*, EPTCS 161, pp. 217-230, 2014.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science 256 (1-2)*, pp. 53-92, 2001.

[Gai82] Haim Gaifman. On local and non-local properties. In *Logic Colloquium'81*, North Holland, 1982.

[GHP95] Patrice Godefroid, Gerard Holzmann, Didier Pirottin. State-Space Caching Revisited, *Formal Methods in System Design 7(3)*, pp. 227-242, 1995.

[GLZ04] Paul Gastin, Benjamin Lerman, Marc Zeitoun. Distributed Games with Causal Memory Are Decidable for Series-Parallel Systems. In *FSTTCS'04*, pp. 275-286, LNCS 3328, 2004.

[GMP03] Elsa Gunter, Anca Muscholl, Doron Peled. Compositional message sequence charts. *STTT 5(1)*, pp. 78-89, 2003.

[GMN09] Paul Gastin, Madhavan Mukund, K. Narayan Kumar. Reachability and boundedness in time-constrained MSC graphs. In *Perspectives in Concurrency – A Feststichrift for P. S. Thiagarajan*. Universities Press, 2009.

[GMO06] Dan Ghica, Andrzej Murawski, Luke Ong. Syntactic control of concurrency. *Theoretical Computer Science 350(2-3)*, pp. 234-251, 2006.

[God94] Patrice Godefroid. Partial Order Methods for the Verification of Concurrent Systems—An Approach to the State-Explosion Problem, PhD thesis, University of Liege, Computer Science Department, November 1994.

[GW91] Patrice Godefroid, Pierre Wolper. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties, in *CAV'91*, pp. 176-185, 1991.

[Han65] William Hanf. Model-theoretic methods in the study of elementary logic. In *The Theory of Models*, pp. 132-145, North Holland, 1965.

[HJ00] Loïc Hélouët, Claude Jard. Conditions for synthesis of communicating automata from HMSCs. In *FMICS'00*, GMD Report 91, 2000.

[HMNST05] Jesper Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind Sohoni, P. S. Thiagarajan. A theory of regular MSC languages. In *Information and Computation 202(1)*, pp. 1-38, 2005.

[KM13] Siddharth Krishna, Anca Muscholl. A quadratic construction for Zielonka automata with acyclic communication structure. *Theoretical Computer Science 503*, pp. 109-114, 2013.

[KMS94] Nils Klarlund, Madhavan Mukund and Milind Sohoni. Determinizing Asynchronous Automata. In *ICALP'94*, pp. 130-141, LNCS 820, 1994.

[Kus02] Dietrich Kuske. A Further Step towards a Theory of Regular MSC Languages. In *STACS'02*, pp. 489-500, LNCS 2285, 2002.

[Kus03] Dietrich Kuske. Regular sets of infinite message sequence charts. *Information and Computation 187(1)*, pp.80-109, 2003.

[KWG09] Vineet Kahlon, Chao Wang, Aarti Gupta. Monotonic Partial Order Reduction: An Optimal Symbolic Partial Order Reduction Technique. In *CAV'09*, pp. 398-413, LNCS 5643, 2009.

[LM04] Markus Lohrey, Anca Muscholl. Bounded MSC communication. *Information and Computation 189(2)*, pp. 160-181, 2004.

[McM95] Ken McMillan. A technique of state space search based on unfolding. *FMSD 6(1)*, pp. 45-65, 1995.

[Maz86] A. Mazurkiewicz. Trace semantics. In *Advances in Petri Nets 1986*, pp. 279-324, LNCS 255, 1986.

[MM01] P. Madhusudan, B. Meenakshi. Beyond Message Sequence Graphs. In *FSTTCS'01*, pp. 256-267, LNCS 2245, 2001.

[Muk02] Madhavan Mukund. From global specification to local implementations. In *Synthesis and Control of Discrete Event Systems*, Kluwer, pp. 19-34, 2002.

[MNS03] Madhavan Mukund, K. Narayan Kumar and Milind Sohoni. Synthesizing Distributed Finite-State Systems from MSCs. *Theoretical Computer Science* 290(1), 221-239, 2003.

[Mor02] Rémi Morin. Recognizable Sets of Message Sequence Charts. In *STACS'02*, pp. 523-534, LNCS 2285, 2002.

[MS94] Madhavan Mukund and Milind Sohoni. Keeping Track of the Latest Gossip in a Distributed System. In *Distributed Computing* 10(3), pp. 137-148, 1997.

[MTY05] P. Madhusudan, P. S. Thiagarajan, Shaofa Yang. The MSO Theory of Connectedly Communicating Processes. In *FSTTCS'05*, pp. 201-212, LNCS 3821, 2005.

[MW01] Kevin Murphy, Yair Weiss. The factored frontier algorithm for approximate inference in DBNs. In *UAI'01*, pp. 378-385, 2001.

[Pig93] G. Pighizzini. Synthesis of Nondeterministic Asynchronous Automata. In *Algebra, Logic and Applications* 5, pp. 109-126, 1993.

[PR90] Amir Pnueli, Roni Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *FOCS'90*, pp. 746-757, IEEE, 1990.

[Pur00] Anuj Puri. Dynamical Properties of Timed Automata. *J. Discrete Event Dynamic Systems 10 (1-2)*, pp. 87-113, 2000.

[Och95] E. Ochmanski. Languages and Automata. In *The Book of Traces*. V. Diekert, G. Rozenberg (eds.), pp. 167-204, 1995.

[Pel93] Doron Peled. All from One, One for All: on Model Checking Using Representatives. In *CAV'93*, pp. 409-423, LNCS 697, 1993.

[Pel94] Doron Peled. Combining Partial Order Reductions with On-the-fly Model-Checking. In *CAV'94*, pp. 377-390, LNCS 818, 1994.

[RS09] Pierre-Alain Reynier, Arnaud Sangnier. Weak Time Petri Nets Strike Back! In *CONCUR'09*, pp. 557-571, LNCS 5710, 2009.

[RSSK15] César Rodríguez, Marcelo Sousa, Subodh Sharma, Daniel Kroening. Unfolding-based Partial Order Reduction. In *CONCUR'15*, pp. 456-469, LIPIcs, 2015.

[SEM03] Alin Stefanescu, Javier Esparza, Anca Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR'03*, pp. 27-41, LNCS 2761, 2003.

[SY15] Gwen Salaun, Lina Ye. Stability of Asynchronously Communicating Systems. INRIA Resaech Report, 2015.

[Tar71] R. E. Tarjan. Depth-First Search and Linear Graph Algorithms. in *FOCS'71*, pp. 114-121, 1971.

[TW02] P. S. Thiagarajan, Igor Walukiewicz. An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. *Information and Computation 179(2)*, pp. 230-249, 2002.

[Val92] Antti Valmari. A Stubborn Attack on State Explosion. *Formal Methods in System Design 1(4)*, pp. 297-322, 1992.

[WYKG08] Chao Wang, Zijiang Yang, Vineet Kahlon, Aarti Gupta. Peephole Partial Order Reduction. In *TACAS'08*, pp. 382-396, LNCS 4963, 2008.

[Zie87] Wieslaw Zielonka. Notes on Finite Asynchronous Automata. *RAIRO ITA 21(2)*, pp. 99-135, 1987.

[Zie89] Wieslaw Zielonka. Safe Executions of Recognizable Trace Languages by Asynchronous Automata. In *Logic at Botik 1989*, pp. 278-289, LNCS 363, 1989.

[Zie06] Wieslaw Zielonka. Time-stamps for Mazurkiewicz traces. *Theoretical Computer Science 356(1-2)*, pp. 255-262, 2006.

[ZXLZZ02] Jianhua Zhao, He Xu, Xuandong Li, Tao Zheng, Guoliang Zheng. Partial Order Path Technique for Checking Parallel Timed Automata. In *FTRTFT'02*, pp. 417-432, LNCS 2469, 2002.

Chapters of Collective Books

[LecCPN03] Blaise Genest, Anca Muscholl, and Doron Peled. Message Sequence Charts. Survey in *Lectures on Concurrency and Petri Nets*, pages 537–558, volume LNCS 3098, 2003.

International Journals

[i&c06] Blaise Genest, Dietrich Kuske, and Anca Muscholl. A Kleene theorem and model checking for a class of communicating automata. *Information and Computation (I&C)* 204(6):920–956, Elsevier, 2006.

[jcss06] Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level MSCs: Model-checking and Realizability. *Journal of Computer and System Science (JCSS)* 72(4):617–647, Elsevier, 2006.

[fi07] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae* 80(2):1–21, IOS Press, 2007.

[tocs08] Blaise Genest and Anca Muscholl. Pattern Matching and Membership for Hierarchical Message Sequence Charts. *Theory of Computing Systems (ToCS)* 42(4):536–567, Springer, 2008.

[am&ai09] Dragan Bosnacki, Edith Elkind, Blaise Genest, and Doron Peled. On commutativity based edge lean search, special issue of BISFAI 2007. *Annals of Mathematics & Artificial Intelligence (Annals Math & AI)* 56(2): 187-210, Springer, 2009.

[tcs09] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P.S. Thiagarajan, and Shaofa Yang. Causal Message Sequence Charts. *Theoretical Computer Science (TCS)* 410(41): 4094-4110, Elsevier, 2009.

[ijfcs10] Edith Elkind, Blaise Genest, Doron Peled, and Paola Spoletini. Quantifying the discord: Order discrepancies in Message Sequence Charts, special issue of ATVA 2007. *International Journal of Foundations of Computer Science (IJFCS)* 21(2): 211-233, WorldSciNet, 2010.

[i&c10] Philippe Darondeau, Blaise Genest, P.S. Thiagarajan, and Shaofa Yang. Quasi-Static Scheduling of Communicating Tasks, special issue of CONCUR 2008. *Information and Computation (I&C)* 208(10):1154-1168, Elsevier, 2010.

[ipl12] S. Akshay, Blaise Genest, Loic Helouet, Shaofa Yang. Regular Set of Representatives for Time-Constrained MSC Graphs. *Information Processing Letters (IPL)* 112(14-15):592-598, Elsevier, 2012.

[tcbb12] Sucheendra Palaniappan, S. Akshay, Bing Liu, Blaise Genest, P.S. Thiagarajan. A Hybrid Factored Frontier Algorithm for Dynamic Bayesian Networks with a Biopathways Application. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 9(5): 1352-1365, 2012.

[jdeds14a] Loïc Hélouet, Hérvé Marchand, Blaise Genest, Thomas Gazagnaire. Diagnosis from Scenarios. *Discrete Event Dynamic Systems: Theory and Applications (JDEDS) 24(4)*, 353-415, 2014.

[jdeds14b] Debmalya Biswas, Blaise Genest. Minimal Observability and Privacy Preserving Compensation for Transactional Services. *Discrete Event Dynamic Systems: Theory and Applications (JDEDS) 24(4)*, 611-646, 2014.

[jacm15] Manindra Agrawal, S. Akshay, Blaise Genest, P.S. Thiagarajan. Approximate Verification of the Symbolic Dynamics of Markov Chains. *Journal of the ACM 62(1)*, 34-65, 2015.

International conferences

[latin02] Blaise Genest, and Anca Mushcoll. Pattern Matching and Membership for Hierarchical Message Sequence Charts. In *Theoretical Informatics, 5th Latin American Symposium (LATIN 2002)*, volume 2286, pages 326–340, 2002.

[icalp02] Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level MSCs: Model-checking and Realizability. In *29th International Colloquium on Automata, Languages and Programming (ICALP'02, track B)*, volume LNCS 2380, pages 657–668, 2002.

[concur03] Blaise Genest, Loïc Hélouët, and Anca Muscholl. High-Level Message Sequence Charts and Projections. In *14th International Conference on Concurrency Theory (CONCUR'03)*, volume LNCS 2761 , pages 308–322, 2003.

[fossacs04] Blaise Genest, Marius Minea, Anca Muscholl, and Doron Peled. Products of Message Sequence Charts. In *7th International Conference on Foundations of software science and computation structures (FOSSACS'04)*, volume LNCS 2987, pages 195–210, 2004.

[dlt04] Blaise Genest, Dietrich Kuske, and Anca Muscholl. A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. In *8th International Conference on Developments in Language Theory (DLT'04)*, volume LNCS 3340, pages 300–48, 2004.

[tacas05a] Blaise Genest, Dietrich Kuske, Anca Muscholl, and Doron Peled. Snapshot Verification. In *11th Joint Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'05)*, volume LNCS 3440, pages 510–525, 2005.

[tacas05b] Blaise Genest. Compositional Message Sequence Charts (CMSCs) Are Better to Implement than MSCs. In *11th Joint Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'05)*, volume LNCS 3440, pages 429–444, 2005.

[concur05] Blaise Genest. On Implementation of Global Concurrent Systems with Local Asynchronous Controllers. In *16th International Conference on Concurrency Theory (CONCUR'05)*, volume LNCS 3653, pages 443–457, 2005.

[forte06] Edith Elkind, Blaise Genest, Doron Peled, and Hongyang Qu. Grey box checking. In *26th International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, volume LNCS 4229, pages 420–435, 2006.

[icalp06] Blaise Genest and Anca Muscholl. Constructing exponential-size deterministic Zielonka automata. In *33rd International Colloquium on Automata, Languages and Programming (ICALP'06, track B)*, volume LNCS 4052, pages 565–576, 2006.

[tacas07] Edith Elkind, Blaise Genest, and Doron Peled. Detecting races in ensembles of message sequence charts. In *13th Joint Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07)*, volume LNCS 4424, pages 420–434, 2007.

[icalp07] Dragan Bosnacki, Edith Elkind, Blaise Genest, and Doron Peled. On commutativity based edge lean search. In *34th International Colloquium on Automata, Languages and Programming (ICALP'07, track A)*, volume LNCS 4596, pages 158–170, 2007.

[concur07] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P.S. Thiagarajan, and Shaofa Yang. Causal Message Sequence Charts. In *18th International Conference on Concurrency Theory (CONCUR'07)*, volume LNCS 4703 , pages 166–180, 2007.

[atva07] Edith Elkind, Blaise Genest, Doron Peled, and Paola Spoletini. Quantifying the discord: Order discrepancies in Message Sequence Charts. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume LNCS 4762, pages 378–393, 2007.

[fossacs08] Philippe Darondeau, Blaise Genest, and Loïc Hélouët. Products of Message Sequence Charts. In *11th International Conference on Foundations of software science and computation structures (FOSSACS'08)*, volume LNCS 4962, pages 459–474, 2008.

[seke08] Debmalya Biswas and Blaise Genest. Minimal Observability for Transactional Hierarchical Services. In *20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, volume ISBN 1-891706-22-5, pages 531–536, 2008.

[concur08] Philippe Darondeau, Blaise Genest, P.S. Thiagarajan, and Shaofa Yang. Quasi-Static Scheduling of Communicating Tasks. In *19th International Conference on Concurrency Theory (CONCUR'08)*, volume LNCS 5201, pages 310–324, 2008.

[atva08] Blaise Genest, Anca Muscholl, Olivier Serre, and Marc Zeitoun. Tree Pattern Rewrite Systems. In *6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08)*, volume LNCS 5311, pages 332–346, 2008.

[hase08] Debmalya Biswas, Thomas Gazagnaire, and Blaise Genest. Small Logs for Transactional Services. In *11th IEEE High Assurance Systems Engineering Symposium (HASE'08)*, volume IEEE 978-0-7695-3482-4, pages 97-106, 2008.

[lics09] Nathalie Bertrand, Blaise Genest, and Hugo Gimbert. Qualitative Determinacy and Decidability of Stochastic Games with Signals. In *24th IEEE Symposium on Logic In Computer Science (LICS'09)*, volume IEEE, 2009.

[icalp10] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal Zielonka-Type Construction of Deterministic Asynchronous Automata. In *37th International Colloquium on Automata, Languages and Programming (ICALP'10, track B)*, volume LNCS 6199, pages 52–63, 2010.

[fsttcs10] Blaise Genest, Anca Muscholl, and Zhilin Wu. Verifying Recursive Active Documents with Positive Data Tree Rewriting. In *30th Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, LIPIcs , pages 469-480, 2010.

[cmsb11] Sucheendra Palaniappan, S. Akshay, Blaise Genest, P.S. Thiagarajan. A Hybrid Factored Frontier Algorithm for Dynamic Bayesian Networks. In *9th International Conference on Computational Methods in Systems Biology (CMSB'11)*, 35-44, ACM/IEEE 978-1-4503-0817-5, 2011.

[fsttcs11] Nathalie Bertrand, Blaise Genest. Minimal Disclosure in Partially Observable Markov Decision Processes. In *31st Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, 411-422, LIPIcs, 2011.

[ictac12] S. Akshay, Blaise Genest, Loic Helouet, Shaofa Yang. Symbolically Bounding the Drift in Time-Constrained MSC Graphs. In *9th International Colloquium on Theoretical Aspects of Computing 2012 (ICTAC'12)*, 1-15, LNCS 7521, 2012.

[lics12] S. Akshay, M. Agrawal, Blaise Genest, P.S. Thiagarajan. Approximate Verification of the Symbolic Dynamics of Markov Chains. In *27th Annual ACM/IEEE Symposium on Symposium on Logic In Computer Science (LICS 2012)*, pages 55–64. IEEE, 2012.

[icalp13] Blaise Genest, Hugo Gimbert, Anca Muscholl, Igor Walukiewicz. Asynchronous Games over Tree Architectures. In *40th International Colloquium on Automata, Languagesand Programming (ICALP'13, track B)*, volume LNCS 7966, pages 275–286, 2013.

[fsttcs13] S.Akshay, Ionut Dinca, Blaise Genest, Alin Stefanescu. Implementing Realistic Asynchronous Automata. . In *the 33st Foundations of Software Technology and Theoretical Computer Science (FSTTCS'13)*, 213-224, LIPIcs, 2013.

[qest14] Nathalie Bertrand, Thomas Brihaye, Blaise Genest. Deciding the value 1 problem in 1-clock Decision Stochastic Timed Automata. In *11th International Conference on Quantitative Evaluation of SysTems (QEST'14)*, 313-328, LNCS 8657, 2014.

[fossacs15] Blaise Genest, Doron Peled, Sven Schewe. Knowledge = observation + memory + computation. In *18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'15)*, 215-229, LNCS 9034, 2015.

[cmsb15] Sucheendra Palaniappan, François Bertaux, Matthieu Pichené, Eric Fabre, Gregory Batt, Blaise Genest. Approximating the dynamics of the Hybrid Stochastic-Deterministic Apoptosis pathway. **Poster** in *14th International Conference on Computational Methods in Systems Biology (CMSB'15)*, LNCS, 2015.

[stacs16] S. Akshay, Blaise Genest, Bruno Karelovic, Nikhil Vyas. On Regularity of unary Probabilistic Automata. In *33rd International Symposium on Theoretical Aspects of Computer Science (STACS'16)*, LIPIcs, 2015.

[icatpn16] S. Akshay, Blaise Genest, and Loïc Hélouët. Decidable classes of unbounded Petri nets with time and urgency. In *37th International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN'16)*, LNCS, 2016.

International Workshops

[wodes06] Loïc Hélouët, Thomas Gazagnaire, and Blaise Genest. Diagnosis from scenarios. In *proc. of the 8th Int. Workshop on Discrete Events Systems, (WODES'06)*, volume IEEE 1-4244-0053-8, pages 307–312, 2006.

[xsym09] Debmalya Biswas, Ashwin Jiwane, and Blaise Genest. Atomicity for XML Databases. In *6th International XML Database Symposium (XSym'09) at VLDB*, volume LNCS 5679, pages 160-167 (short paper), 2009.

Submitted

[sub-1] Sucheendra Palaniappan, François Bertaux, Matthieu Pichené, Eric Fabre, Gregory Batt, Blaise Genest. Discrete Stochastic Abstraction of Biological Pathway Dynamics: A case study of the Apoptosis Pathway.