ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

HABILITATION À DIRIGER LES RECHERCHES

Taking a Walk on the Expressiveness Road

Cinzia Di Giusto

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention de l'habilitation à diriger les recherche en INFORMATIQUE

Taking a Walk on the Expressiveness Road

Taking a Walk on the Expressiveness Road

Cinzia Di Giusto

⋈

Jury :

Rapporteurs
Luis Caires, Pr., Técnico Lisboa, Portugal
Roland Meyer, Pr., Universität Braunschweig, Germany
Alan Schmitt, Dr. HDR, Inria Rennes, France

Examinateurs
Simon Gay, Pr., University of Glasgow, United Kingdom
Paola Giannini, Pr., Università del Piemonte Orientale, Italy
Maciej Koutny, Pr., Newcastle University, United Kingdom

Université Côte d'Azur

Cinzia Di Giusto

Taking a Walk on the Expressiveness Road

xiii+50 p.

Ce document a été préparé avec LATEX2e et la classe `these-ISSS` version v. 2.10.

Impression : main.tex – 24/3/2024 – 15:35

Révision pour la classe : these-ISSS.cls,v 2.10 2020/06/24 14:16:37 mpelleau

À Anna, l'encadrement plus important.

Taking a Walk on the Expressiveness Road

Taking a Walk on the Expressiveness Road

Abstract

This dissertation explores the intricacies of concurrent systems and their formal analysis, aiming at understanding their behaviors and at verifying their properties through formal methods. Concurrent systems, central to modern computing, involve multiple entities working collectively to shape the system's overall behavior. This research delves into the distinctive challenges posed by these systems.
In concurrent systems, the focus shifts from program termination and correctness, typical of sequential systems, to responsiveness at all times. Behaviors become complex, involving interactions, data exchanges, and resource sharing among independent processes or entities. Verification priorities encompass synchronization, communication consistency, and resource management.
The dissertation comprises three interconnected research directions:

1. Process Algebras and Session Types
2. Communicating Automata
3. Applications: demonstrating the practical application of formal methods in diverse domains, bioinformatics, ecology and neural network.

In summary, this dissertation uncovers the complexities of concurrent systems, emphasizing their formal analysis and aiming to identify the boundary between descriptive power and decidability
The dissertation will be organized in chapters discussing my work and classifying them into those three categories: process calculi and session types, automata with memory, applications of concurrent models to various domains. Each chapter will be preceded by a general introduction, specifying and commenting the context and giving pointers to the publications concerning the discussed works.

Keywords: Concurrency, Verification, Session types.

Table of contents

Prologue

If you don't start, you'll never finish.

— Frank Sonnenberg, The Path to a Meaningful Life.

I have **finally** decided to write this manuscript. I have long thought of what should be the format that I should give it. This brief introduction comes as a sort of disclaimer. In this manuscript the reader will not find formal definitions or results. I am mainly discussing, giving my opinion and putting in perspective my past works. Full technical results and formal details are left in the published papers.

1.1 Research

I came across "concurrency" by chance. It was not part of my master studies, nor the subject of my master thesis. I wanted to pursue my studies with a PhD and a colleague in a project where I had been hired as a programmer, gave me a paper to read. I do not remember neither the authors nor the content of the paper just that it talked about composition of processes running in parallel. It was the beginning. Then I met my advisor Maurizio Gabbrielli and he shared with me one of the problems that he was investigating at the time: an expressiveness problem. So I learned about process algebra and several other concurrent formalisms (Petri nets, rewriting systems, etc.). Since then, my guiding light in research is not one single small topic but mainly what are the markers of expressiveness in formalisms that model concurrent behavior. The riveting part is that some of the markers are recurrent and once identified the creative part is to spot the proper technique to make it evident.

Besides, research/inventive work follows the ebbs and flows of history and behind each result there are always two stories. One that is the human one, the colleagues that we met and where the flow of discussion brought us and the story that is told in our published papers, that most of the time is an *a posteriori* reconstruction to justify choices. Clearly the first one would be much more witty but I would stick to the simplest one that is to make sense of (or rather summing up) my research activities of the last 10 years.

Teaching duties, even though they represent half (at least nominally, but practically much more) of my working days are briefly discussed below.

1.2 Teaching activities

I rapidly mention here my engagement around education and teaching activities and point to my webpage [48] for all details concerning my CV. My teaching activity is divided between the bachelor's and master's computer science programs at the Université Côte d'Azur. All lectures are taught in person (except during the COVID period, when I switched to distance and/or hybrid formats).

At the bachelor level, I'm in charge of the "Bases de l'informatique" course (L1 level, 18h CM + 100h TD) for a class of about 200 students on average (creation of material, lectures and about half of the TDs). I also help with the practice part for the Compilation course (L3 level, 36h TD) for around 30 students. The aim of the "Bases de l'informatique" course is to give the first concepts and tools needed in a career in Computer Science. The notions of function, recurrence principle, propositional and first-order logic, and an introduction to automata are discussed. Each lecture is followed by 2 tutorial sessions. The first is designed to discuss the arguments seen in class, with the proposal of exercises. The second session is designed to give students a greater sense of responsibility in mastering the concepts, and to help them self-check their understanding of the concepts (through exercises and the help of the tutor).

At the Master's level, I'm in charge of the courses: Parallelism, Communication and Concurrency at M1 level and Types at M2 level. The classes are made up of around 10 students, with each course comprising 24h in person lectures. The Parallelism course covers evaluation and parallel programming techniques. The course is complemented by a series of tutorials and a project (guided by a colleague) where the notions of conflict and deadlock are emphasized. Communication and Concurrency introduces the first concepts of distributed protocol modeling (through process algebras, Petri nets, etc.). Finally, Types introduces typing systems and their main properties.

I often propose projects for M1 and M2 students and supervise internships both at M1 level and end-of-study internships.

Finally from a more administrative point of view I am the coordinator of the M1 which includes selecting the students, preparing the schedule and handling the pedagogical committees.

1.3 Acknowledgements

First of all, I would like to thank the reviewers for taking the time to review this manuscript.

I would also like to thank a number of close colleagues/friends for their constant support and technical and everyday discussions: Ilaria Castellani, Hanna Klaudel, Etienne Lozes, and Jorge Perez. Thank you for your patience and friendship.

A special thanks to the whole group of PhD students who used to chat outside my office. Thank you for sharing your research ideas, your jokes, and your time with me.

Finally, I am grateful to my life partner, Enrico. It is a blessing to be able to share a technical discussion and, above all, to be understood. Thank you for the easy and the difficult times.

Chapter 1
Introduction

If you have to support yourself, you had bloody well better find some way that is going to be interesting
— Katherine Hepburn, .

I will start this dissertation from the end, stating the general topic of my work and then building up from there and going into details.

> *My research focuses on concurrent systems in which individual entities collectively shape the overall system behavior. I aim to explore the conditions and methods for verifying properties of such systems.*

Everything in the previous sentence needs to be explained, but two ingredients stick out: concurrent systems and checking properties. So what is a concurrent system, or said otherwise what is concurrency? And once we are handling a concurrent systems what are the key aspects that needs to be checked carefully and/or ensured?

Among the plethora of definitions of concurrent system, we will take the one given in [57] where Hoare and Lamport say:

> In a concurrent system, two or more activities (e.g., processes or programs) progress in some manner in parallel with each other.

The essential point is that in a concurrent system there are entities/activities (sometime we will call them participants) that work together towards a common objective. We will refer to this common objective as the protocol (e.g., the sequence of actions) of the system. "Working together" means that participants collaborate through synchronization, exchange of data or resource sharing. Cooperation becomes the central point of study of the discipline and it allows to draw a clear distinction between the sequential and the concurrent setting. Indeed in sequential programs, we are mainly concerned about whether programs terminate and/or if they are correct, i.e., whether what has been computed corresponds to the expected result. Now, in the concurrent setting, we often do not care about termination as systems are meant to be responsive at all times. Moreover, the behavior of a concurrent system cannot be expressed as an input-output function. In such systems we primarily care whether the synchronization/sharing of resources between entities is possible or whether exchanged data are consistent. Hence, some (those that I have focused on) of the key properties of concurrent systems include:

Safety: it ensures that certain undesirable events or conditions never occur during the system's execution. For example, a safety property might guarantee that no two processes access a shared resource simultaneously.

Liveness: it ensures that something desirable eventually happens in the system. This could include properties like deadlock freedom (the system does not reach a state where all processes are blocked, unable to make progress due to resource conflicts) or termination (all processes eventually finish execution).

Other important properties, that we will not directly touch here, are *isolation* that ensure that concurrent transactions in databases do not interfere with each other, preserving data consistency; *fault tolerance* checking that the system can recover from failures, continue to operate, and maintain data consistency; *scalability* which is related to resource utilization, response time, and the ability to handle increased workloads; or *temporal/causal properties* that deal with the timing and specific ordering of events.

There are a number of key factors, that while reasoning about concurrent systems and implementing them, results in systems where it is challenging to ensure the properties above. First of all concurrency introduces non-determinism, where the order of execution of concurrent components (e.g., threads, processes) is not strictly controlled. Identifying all possible interleavings and ensuring "correctness" for each is a complex, expensive task. Moreover, the incorrect use of synchronization can lead to deadlocks, race conditions, data inconsistencies and performance bottlenecks. For these reasons, direct techniques such as debugging or testing may not uncover all concurrency-related issues. Some bugs may only manifest under specific interleavings, making them difficult to reproduce and diagnose. In some sense, for concurrent systems, rigorous mathematical techniques such as formal methods are the only way to go and the family of formal methods is incredibly rich.

Most of the formalisms for talking about concurrency are based on a formal semantics establishing a precise understanding of the behavior of programs. As underlined above, we are mainly interested in the way a system synchronizes and exchanges data. In other words, we focus on how and when a system communicates internally and with the surrounding environment. All other features become unnecessary details and are usually abstracted away. The dominant models that have been proposed for dealing with concurrency are in historical order Petri nets (introduced in 1962 by Petri) [86, 80], process calculi (Milner and Hoare independently proposed in 1980 their CCS (Calculus of Communicating Systems) [78], and CSP (Communicating Sequential Processes)[56]), and automata with memory (let just consider for the time being communicating automata as presented in 1983 in [18]). All these models were followed by a huge success and up to now the theory have been greatly enriched with variants and extension of those preliminary works.

Petri nets provide a graphical representation of how entities interact and compete for resources in a system. They consists of two main components: places and transitions. Places represent states or conditions in the system, they can hold a certain number of tokens which symbolize the presence of resources, or events in a specific state. Transitions represent actions or events that can occur in the system. When a transition fires, it consumes tokens from its input places and produces tokens in its output places. Apart from the basic form, different types of Petri nets exist, each with its own features and variations to model specific types of systems. Notably we can mention stochastic Petri nets that include probabilistic aspects. High-level Petri nets allow to represent additional information, modeling more complex data. Reset nets and inhibitory nets introduce special arcs that enrich the expressiveness of the formalisms. For a survey see [32].

Process calculi bring concepts from algebra and they impose rules for describing the interactions and communications among processes in a system [5, 73]. A leading motivation in the development of process calculi has been properly capturing the dynamic character of concurrent behavior. Apart from the calculi mentioned above, over the time the process calculus "par excellence" has become the π-calculus [91]. Much of the success of the π-calculus can be fairly attributed to the way it departs from CCS so as to describe mobile systems in which communication topologies can change dynamically. Similarly to Petri nets, a lot of extensions have been proposed in the literature (stochastic, timed, with localities, higher order, to mention a few) [6].

Finally, automata with memory and in particular communicating automata build upon traditional automata theory, which focuses on the study of abstract machines that process input symbols and transition between states. In the context of concurrent systems, the concept of automata is extended to capture the interactions and communications among multiple automata or processes. Communication happens via channels through which processes exchange messages. Processes can synchronize their actions based on the availability of those messages. See [67] for a survey.

A precise mathematical formulation is the initial step to analyze a concrete system. Then, on top of such formulation, one needs to devise techniques to prove that a system satisfies or not a given property. Techniques include static analysis, encompassing abstract interpretation [28, 76, 77] or typing systems [58, 59], language comparison and model checking. Typing systems and behavioral types [4], in particular, introduce challenges related to synchronization, communication, and coordination among concurrently executing processes. In sessions typed systems, type soundness typically ensures communication safety (processes do not present communication errors) and session fidelity (processes respect their session protocol), as well as some progress property (processes do not get stuck). Combined, these properties can be seen as enforcing a protocol conformance for processes. Furthermore, language equivalence, typical of the sequential setting, although also used in concurrency, leaves the spot to bisimulation [89, 90], that is a finer equivalence capturing the fact that a system simulates the other and vice-versa so that two systems cannot be distinguished by an external observer. Finally model checking involves systematically exploring all possible states of a model (usually a finite-state representation of the system) to verify properties automatically.

Now, this brings us back to the underlying topic of my research which is exploring the conditions and methods for verifying properties of concurrent system. The problem, is that, even if we focus only on the exchanged messages, the obtained formalisms are, in their general formulation, already Turing powerful (except for Petri nets). This entails that all interesting properties are undecidable and it motivates the need of finding a compromise between what can be described and what can be studied/decided. This is what we mean by expressiveness studies.

Expressiveness is the common thread linking my research work. In this manuscript we start by commenting on variants of process algebra, trying to understand when properties like the ones mentioned above are decidable or not. Session types, for instance, give a clean characterization of processes for which some properties are valid by construction. This means that a well typed process is a program that conforms to its protocol and hence ensure that the protocol is executed as expected. Then profiting by the connection between session types and communicating automata we talk about special classes of communicating automata that exploit different notions of synchronization to (again) achieve decidability. Finally, we exhibit a mix of applications of expressiveness studies to formalism used in biology and ecology.

Organization of the dissertation. The manuscript is organized in chapters discussing my work and classifying them into 3 categories: process calculi and session types, automata with memory and finally applications of concurrent models to various domains. Each chapter is preceded by a general introduction, specifying and commenting the context. In each chapter, I will give pointers to the publications concerning the discussed works. The final chapter circles back discussing what are the directions and ideas to come.

Chapter 2

Process Calculi and Session Types

I would now like to turn to my other and greatest excitement, trying to understand how discrete systems communicate with each other.

— Robin Milner,

Speech on receiving an Honorary Degree from the University of Bologna.

The core of my research has always been trying to understand what are the cruces of concurrency: i.e., where to situate the limit and which are those special features that render the language expressive enough to model a precise situation but not too expressive that every interesting property becomes undecidable. Process algebra are one of those formalism that are prone to this kind of analysis.

Indeed, one of the principles of process algebra is to have a concise language with few operators that can *compositionally* build more complex behavior. Much like the λ-calculus, the first effort is to go to the essence of the language, e.g., what are the critical features that allow to count and test for zero. In the concurrent setting, the paradigmatic examples are CCS, CSP, and the π-calculus. All these languages feature message exchange, recursive behavior, and a way of creating new resources. All of them can be shown to be Turing powerful. Still, as mentioned in the Introduction, there is something fundamentally different in concurrent calculi from sequential ones. Encodings into Turing equivalent models can say something about the complexity of the model, but do not qualify entirely the language under consideration. This is the content of a research thread (among the most notable papers we can recall [84, 49]) where the process algebras under consideration are Turing equivalent but if we fix some desired properties of the encodings (e.g., compositionality) we can differentiate languages and show that some specific patterns can be modeled in one of the language but not in the other.

This can be considered sufficient motivation to consider novel operators that add the possibility of modeling more complex exchanges. Here we consider two possible directions of extending CCS: i) space and mobility considering how to treat movement and composition of processes and ii) time and, in particular, we focus on logic time.

Space and mobility. In process algebras, the notion of space typically refers to the way in which processes or process components are organized or structured within a system. Some process algebras incorporate spatial hierarchies [23], where processes are organized into nested structures or spaces. This hierarchy can represent system modularity and decomposition. Mobility, instead, refers to the dynamic movement or relocation of processes or channels within a system. In this chapter, we focus, in particular, on process mobility, which involves processes that can migrate from one location to another during their execution. Process mobility models, for instance, the idea that, in a distributed computing system, a process might move from one server to another to balance the load. For the papers that we consider here, the two notions are somehow intertwined.

We begin by considering space and process composition. In [103] we studied the expressiveness of a language designed for describing process composition. This work builds upon the foundation laid by [10], which introduces a formal component model known as BIP (Behavior, Interaction, Priority). In [103], we introduce a process calculus that models processes as compositions of a "glue" component (in BIP terminology) and subcomponents. Notably, this calculus represents a conservative extension of BIP, incorporating more dynamic forms of glues. Our investigation focuses on assessing the Turing completeness of various variants of this calculus, differing primarily in their language for defining glues. Our findings reveal that constraining the glue language to BIP's glues alone is sufficient to achieve Turing completeness. Conversely, the removal of priorities from the control language results in a loss of Turing completeness. Additionally, we demonstrate that reintroducing a simple form of dynamic component creation in the control language, albeit without priorities, is adequate to restore Turing completeness. These results offer complementary insights to those obtained in the context of BIP, emphasizing the pivotal role of priorities in determining expressiveness.

Next, a second set of research works places a particular emphasis on mobility. It concerns formal frameworks and methodologies to model runtime adaptation while upholding system correctness and reliability.

In our earlier work [19], we introduced the concept of adaptable processes within the realm of process algebra. Adaptable processes represent units of execution with the capacity for dynamic reconfiguration during runtime. They can change their behavior, structure, or communication patterns to adapt to changing conditions. We established an expressiveness hierarchy of languages, depending on their ability to create new processes and on the allowed patterns of evolution. Within this hierarchy, we investigated the decidability of two verification problems: bounded adaptation, which restricts the number of consecutive erroneous states during computation to a specified value k, and eventual adaptation, which ensures that if the system enters an error state, it will eventually transition to an error-free state.

Subsequently, in our work [101], we tame the expressiveness issues described above by introducing a binary session type discipline. We demonstrated that well-typed processes exhibit safety and consistency properties. Safety guarantees the absence of communication errors during runtime, while consistency ensures that active session behavior is never disrupted by adaptation actions.

Finally, in [102], we introduced an event-based approach to adaptation. Here, adaptation requests, originating from the system itself or its context, are treated as events capable of triggering runtime adaptation routines. These routines leverage type-directed checks to enable the reconfiguration of processes with active sessions. As in previous work, we developed a type system discipline for both binary and multiparty protocols.

Time. Turning our attention to the dimension of time, we explore an extension of session types that includes time. While this concept is not entirely novel, recent proposals, such as [11], have sought to enhance multiparty session types by guaranteeing that processes within the calculus adhere to specific time-windows stipulated by their protocols.

However, our approach diverges from conventional physical time as we consider logical time. Our contribution involves the development of a process calculus tailored for multiparty sessions. It draws inspiration from synchronous reactive programming, encompassing features such as broadcast signals, logical instants, and event-based preemption (as outlined in [21]).

Within this calculus, participants in a protocol gain the ability to broadcast messages, temporarily suspend their activities while awaiting specific messages, and respond to events. The principal innovation lies in the introduction of a session type system designed for this calculus. This session type system serves a dual purpose: it enforces session correctness by ensuring protocol conformance and additionally guarantees input timeliness, a time-related property critical for preventing livelock scenarios.

Organization. Three topics are summarized in the following, each exemplifying one of the extension discussed above. Revisiting Glue Expressiveness in Component-Based Systems [103], Adaptable processes [19, 101, 102] and Multiparty Reactive Sessions [21].

2.1 Revisiting Glue Expressiveness in Component-Based Systems

The first extension that we describe here, concern the space direction. We consider component-based software where complex systems can be built by composing, or *gluing* together components that may have been developed independently.

In their paper on glue expressiveness [10], Bliudze and Sifakis, have proposed to look at the expressive power of *glues* or composition operators in an effort to assess the relative merits of different component frameworks with respect to their composition capabilities. In essence, the criterion they use to compare two sets

G_1 and G_2 of composition operators with respect to a set of components C and an equivalence relation, is whether it is possible to find for every operator in G_1 an equivalent one in G_2. The equivalence criterion is based on the fact that each composition of components in C via G_1 can be obtained via a composition in G_2.

This work, however, leaves open a number of questions, in particular regarding the form glues can take, and their intrinsic expressiveness. Indeed, the notion of glue in [10] is essentially a static one. One may legitimately argue in favor of more dynamic forms of composition, e.g., to allow the creation of new components or the replacement of existing ones to accommodate different forms of software update.

We model component assemblages as terms in a process calculus, called CAB (for Components And Behaviors). The language is parametric with respect to a family of primitive components expressed as labelled transition systems. It features locations representing components. Each location can either contain a set of sub-components or a behavior from the class of primitive components together with a glue operator. The glue operators include action prefix, parallel composition and recursion. Actions govern how each component synchronizes with the others and how the behavior of a subcomponent is prioritized. Action prefixes allow the definition of dynamic glues. Parallel composition is not a synchronization operator, but it can be interpreted as an and operator among glues.

We study the expressiveness of CAB. It is easy to show that BIP glues can be recovered as single state processes of our glue language in CAB.

Theorem 2.1.1. *BIP systems defined over a set P of components can be encoded in CAB(P): any BIP system is strongly bisimilar to its encoding.*

We also show that even if the set of primitive components is empty and we only consider the set of static BIP glues, the resulting variant of CAB is Turing-complete. The result is obtained by resorting to an encoding of Minsky Machines and strongly relies on priorities. Numbers inside registers are encoded as the parallel composition of as many occurrences as the number to be encoded of the same component. An increment adds an occurrence of this component while a decrement removes one, if available. To be able to execute the test for zero, it is necessary to check that the register is indeed empty, this is the role of priorities. Such an encoding is reminiscent of the role of tokens in Petri nets. Indeed the expressiveness of the fragment without priorities falls back to that of Petri nets.

Finally, we show that Turing-completeness can be reobtained if we add a simple form of component creation. Thanks to the interplay between the creation of new components and recursion we can gain Turing equivalence. The result, is obtained again by resorting to an encoding of Minsky machines that exploits creations and nesting of components.

2.2 Adaptable processes

We mentioned in the Introduction that one of the motivation in the development of process calculi is to capture the dynamic nature of concurrent behavior. Here we describe behavioral patterns which concern changes at the process level and describe process evolution along time, in other words processes are able to suspend, replace or relocate a running activity. This is a central concept for the adaptation capabilities by which processes modify their behavior in response to exceptional circumstances in their environment.

In [19], we introduce $\mathcal{E}$, a process calculus of adaptable processes: a variant of CCS without restriction and relabeling. Adaptable processes have a location and are sensible to actions of dynamic update at runtime. Locations are useful to designate and structure processes into hierarchies, they are transparent and allow processes to evolve on their own or interact freely with their environment. A contextual update prefix allows to implement forms of process evolvability.

We consider several variants of $\mathcal{E}$, obtained via two orthogonal characterizations.

1. The first one is structural, and defines static and dynamic topologies of adaptable processes. In a static topology, the number of adaptable processes does not vary along the evolution of the system: they cannot be destroyed nor new ones can appear. In contrast, in the more general dynamic topology this restriction is lifted. We will use the subscripts s and d to denote the variants of $\mathcal{E}$ with static and dynamic topologies, respectively.

   The structural characterization is useful when one wants to preserve the main architecture of the operating system; conversely, an operating system could be designed to disallow runtime updates that alter its basic organization in dangerous ways. A static topology is also consistent with settings in which adaptable processes represent located resources, whose creation is disallowed or comes with a cost (as in cloud computing scenarios).

2. The second characterization is behavioral, and concerns update patterns, i.e., the shape of context in an update prefix. We consider three kinds of update patterns, which determine three families of $\mathcal{E}$ calculi, denoted by the superscripts 1, 2, and 3, respectively. The first update pattern admits all kinds of contexts, and so it represents the most expressive form of update. In particular, contexts can appear behind prefixes. The second update pattern forbids such guarded contexts. In the third update pattern we further require contexts to have exactly one hole, thus preserving the current behavior (and possibly adding new behaviors): this is the most restrictive form of update.

   The behavioral characterization is inspired by the forms of reconfiguration and/or update available in component models in which evolvability is specified in terms of patterns, such as SOFA 2 [55]. Update patterns are also related to functionalities present in programming languages such as Erlang [1].

	$\mathcal{E}_d$ – Dynamic Topology	$\mathcal{E}_s$ – Static Topology
$\mathcal{E}^1$	BA undec / EA undec	BA undec / EA undec
$\mathcal{E}^2$	BA dec / EA undec	BA dec / EA undec
$\mathcal{E}^3$	BA dec / EA undec	BA dec / EA dec

Figure 2.1 – Summary of (un)decidability results for dialects of $\mathcal{E}$.

Expressiveness. We study two *verification problems* associated to $\mathcal{E}$ processes and their (un)decidability. They are defined in terms of standard observability predicates (barbs), which indicate the presence of a designated error signal. We thus distinguish between correct states, states in which no error barbs are observable and error states: states exhibiting error barbs. The first verification problem, *bounded adaptation* (abbreviated BA) ensures that, given a finite k, at most k consecutive error states can arise in computations of the system, including those reachable as a result of dynamic updates. The second one, *eventual adaptation* (abbreviated EA), is similar but weaker: it ensures that if the system enters into an error state then it will eventually reach a correct state. Both properties are studied with respect to a set of possible updates that can be applied at runtime.

The main technical results are summarized in Figure 2.1. The calculus $\mathcal{E}^1$ is shown to be Turing complete thus both BA and EA are shown to be undecidable. The expressiveness comes from update actions in $\mathcal{E}$ that allow to "jump" from finite Petri nets to a Turing complete model. We, then, show that in $\mathcal{E}^2$ BA is *decidable*, while EA remains *undecidable*. Interestingly, EA is already undecidable in $\mathcal{E}^3_d$, while it is *decidable* in $\mathcal{E}^3_s$.

The undecidability results are obtained via encodings of Minsky machines. The encodings that we provide in $\mathcal{E}^2$ and $\mathcal{E}^3_d$ do not reproduce faithfully the corresponding machine, but we can detect that only finitely many steps are wrongly simulated. The decidability of EA in $\mathcal{E}^3_s$ is proved by resorting to Petri nets: EA is reduced to a problem on Petri nets, that we call infinite visits which, in turn, can be reduced to place boundedness. For the decidability of BA we appeal to the theory of well-structured transition systems [41] and its associated results. In our case, such a theory must be coupled with Kruskal's theorem [66] (which allows to deal with terms whose syntactical tree structure has an unbounded depth), and with the calculation of the predecessors of target terms in the context of trees with unbounded depth (which is necessary in order to deal with arbitrary aggregations and dynamic updates that may generate new adaptable processes).

Typing systems. To rule out the decidability issues, in [101] we extend $\mathcal{E}$ with a session type discipline. In our language communicating behavior coexists with update actions. This raises the need for disciplining both forms of interaction, in such a way that protocol abstractions given by session types are respected and evolvability requirements are enforced. We draw inspiration from the session types in [42], which ensure that sessions within Ambient hierarchies are never disrupted by Ambient mobility steps. By generalizing to the context of (session) processes which execute in arbitrary, possibly nested locations, we obtain a property which we call consistency: update actions over located processes which are engaged in active session behavior cannot be enabled. The semantics enables adaptation actions within arbitrary process hierarchies and, following [42], endows each located process with a suitable runtime annotation, which describes its active session behavior. Runtime annotations for locations are key in avoiding undesirable update actions.

Our typing system extends existing session type systems with the notion of interface, which allows for simple and intuitive static checking rules for evolvability constructs. In particular, interfaces are essential to rule out careless updates guaranteeing that services should always be available in multiple copies, this is the Service Channel Principle (SCP) identified in [22].

In [102], we relax the type discipline to enable adaptation steps also on locations that contain already established sessions. To enable this behavior we propose update processes in which the adaptation routine dynamically checks the current state of the protocols running in it and determines a new process behavior based on that state. The semantics for our update processes exploits session monitors: processes that maintain the (current) protocol state at a given channel, and constructs for dynamic type inspection, as put forward in [65].

We propose an event-based approach. Our process syntax includes adaptation signals that are similar to communication prefixes and that can be used to issue an adaptation request for a given location. Each location is endowed with a queue that stores its pending adaptation signals/requests. The use of adaptation requests and the arrival predicate allows us to maintain a separation between structured communications (disciplined by session types) and the events that ultimately trigger update processes.

As before the session type system ensures safety: well-typed programs do not exhibit communication errors (e.g., mismatched messages) and consistency: well-typed programs do not allow adaptation actions that disrupt already established session protocols. We considered a typed process model for both binary and multiparty sessions. The multiparty session type discipline is built upon the work in [26] which uses global types and monitored processes as main ingredients. A global type abstracts a sequence of communications between two or more participants. By projecting a global type onto each participant, one obtains a series of local types. Such local types are used as monitors for the processes that implement each partner: a monitor enables the visible actions of the process. The association of a monitor and a process is called a monitored process. We extend the framework of [26] with typeful update processes and event-based constructs. As a result, we obtain an alternative framework in which events handle adaptation requests:

— Internal adaptation requests are initiated by a participant that announces all other participants that the current global protocol should be abandoned, and that a new global protocol should be adopted instead.

— External adaptation requests are meant to dynamically update or upgrade the local implementation of a single participant.

To express internal adaptation requests, processes may explicitly output a message whose communication object denotes a choreography. This message can be accessed by all protocol participants (using the shared queue) so to determine their updated behavior. Networks in our model are distributed collections of locations, each containing a monitored process, a local collection of processes, and a queue. This queue is useful to model external adaptation requests which, as described above, are targeted to a particular participant. As a result, we obtain a setting in which internal adaptation is no longer anticipated at the level of types and concerns all participants of the choreography, while external adaptation requests focus on a single participant.

Summing up:

1. We have introduced $\mathcal{E}$, a core calculus of adaptable processes. $\mathcal{E}$ allows to express a wide range of patterns of process evolution and runtime adaptation. By means of structural and behavioral characterizations, we identify different meaningful variants of the language.
2. We have shown the expressiveness of $\mathcal{E}$ by studying the (un)decidability of bounded and eventual adaptation
3. We introduced a binary and multiparty session type discipline that marries structured communications and runtime adaptation .

2.3 Multiparty Reactive Sessions

In many realistic scenarios, message-passing does not operate alone but coexists with other programming paradigms and abstractions. In particular, the interplay of message-passing concurrency with time- and event-based requirements is very common. Many protocols are subject to time-related constraints (as in, e.g., "the request must be answered within n seconds"). Also, protocols may depend on events that trigger run-time adaptations (as in, e.g., "react to a timeout by executing an alternative protocol").

In this work [21], we study the integration of synchronous reactive programming (SRP) [15, 74] and session-based concurrency giving birth to the language Multiparty Reactive Sessions (MRS). We devise a uniform programming model for message-passing systems in which some components are reactive and/or timed. Synchronous reactive programming is a well-established model rooted on a few features: broadcast signals, logical instants, and event-based preemption. This makes it an ideal vehicle for specifying and analyzing timed reactive systems.

Indeed existing frameworks based on session types lack the ability of broadcasting messages that are not fetched by all participants ("orphan messages" become the norm rather than the exception) and a preemption mechanism, allowing participants' behaviors to be simultaneously reset in reaction to some event.

To address this gap, we propose a new typed framework for multiparty protocols, which supports reactive, time-dependent behaviors. Our framework builds on a language which combines constructs from π-calculi with typical features of SRP. Such features include logical instants, which are periods in which all components compute until they cannot evolve anymore (instants are what make SRP "synchronous"). Constructs to control the evolution of processes among instants are: a pause construct, which explicitly suspends execution for the current instant; a watch construct implementing preemption, a sort of exception handler which triggers an alternative behavior in reaction to a given event and event emission, which is used to control the watch construct. Finally, instead of point-to-point communication, the language features broadcast communication.

In MRS, a process resides within a configuration with its memory and emitted events. Semantics is based on two reduction relations on configurations: the first formalizes small-step execution within an instant, until the configuration suspends. Suspension occurs when all participants have sent/received in the current instant, are waiting for a missing message, or have reached a "pause". The second reduction relation formalizes how suspended configurations evolve across different instants. A design choice of MRS (typical of SRP languages) is to require that, during each instant, every participant can broadcast at most once and receive at most once from the same sender. This requirement is enforced by the semantics itself; it facilitates both message handling during instants and the typing of processes.

We prove that the calculus MRS satisfies (bounded) reactivity, a standard soundness property of SRP, which states that small-step execution converges to a suspension or termination point at every instant [74, 92]. This property, also called instant termination, is a prerequisite for establishing protocol correctness, as it ensures that reactive computations evolve through a succession of instants and thus proceed as expected.

Theorem 2.3.1. *Let C be a reachable configuration of a MRS process. Then C reduces to a suspended configuration in a fixed number of steps.*

The calculus is enriched by a typing systems that ensures properties typical of session type systems: communication safety (processes do not feature communication errors), session fidelity (processes respect their session protocol), as well as some progress property (processes do not get stuck). In addition, MRS processes satisfy input timeliness, a new time-related property stating that every unguarded input is matched by an output during the current instant or the next one.

Our type system relies on the usual ingredients of multiparty session types: global types describe the whole multiparty protocol; local types describe the contribution of each participant to the protocol and a projection function relates global and local types. However, because of the interplay between sessions and SRP, these ingredients have rather different definitions/consequences in our framework.

In particular we require a pre-processing phase over global types called saturation, that enrich protocols with so-called implicit pauses, i.e., those that are not specified by processes but induced by the synchronous reactive semantics between two broadcasts by the same participant, or between two inputs by the same participant from the same source. Unique to our setting, saturation is essential to conduct our static analysis on MRS processes and, ultimately, to reconcile the conceptual gap between SRP and session-based concurrency.

Moreover to prove subject reduction, we need to show that suspension preserves typing. A key property is the absence of circular dependencies in well typed configurations. Indeed, because of bounded reactivity, MRS is deadlock-free: all potential deadlocks due to missing messages are turned into livelocks. We prove that livelocks are not possible for well-typed configurations, i.e., all configurations where processes are typeable and each local type is the projection of a given global type do not contain circular dependencies.

2.4 Conclusion

In this chapter we have discussed how to extend basic process algebra with new constructs and how to evaluate the overall expressiveness of the obtained variant. Part of our analysis is based on evaluating whether a fragment is Turing powerful or not. When a fragment is not Turing powerful we resort to Petri net encodings, or well-structured transition systems. Another complementary way of restricting the expressiveness of variants lies in the use of typing disciplines that allow building programs that are correct by construction.

In the literature (or better in recent conference venues) we see less and less contributions of these kind, maybe suggesting that most of the family of extensions have already been treated. All these extensions have also underlined a weakness of process algebra. The syntax and semantics of such variants are convoluted, they often use too many symbols and loose the initial aim of being compact and intuitive.

Instead other formalisms such as automata with memory have become more *à la mode*. They benefit from having graphical representations that make them more intuitive and they also benefit from a long well established theory. This will be the content of the next chapter.

Still I do not think that process algebra theory is dead, I believe that we will see many of the techniques born for process algebra applied to other domains, it is already the case for the theory behind bisimilarity. On top of this, the connection with automata theory is tight, see for instance how session types can be read as communicating automata and how projection can be defined in that context [72]. As we venture further into this dissertation, we will explore the intricate web of connections between process algebra and communicating automata, revealing the nuanced ways in which they converge and complement each other in the pursuit of unraveling the complexities of distributed systems and communication protocols.

Chapter 3

Automata with Memory

I have a grand memory for forgetting.

— Robert Louis Stevenson, Kidnapped.

In this chapter, we move away from process algebra and delve into communicating automata. While this transition might appear as a shift in formalism, it is important to recognize the profound interconnection between these two theoretical frameworks. In essence, both process algebra and communicating automata share a fundamental objective: they both serve as vehicles for modeling and understanding the exchange of information within systems comprised of independent processes or autonomous parties. In the literature, one can find formal connections as [40, 69] or the more recent [72]. These works shed light on the strong relationship between session types over process algebra and communicating automata. One of the key insights is the concept that the projection of global types, a pivotal aspect in multiparty session types for process algebra, can be interpreted as a form of communicating automata. In some sense, this proves that those formalisms may serve as complementary lenses through which we can gain a richer understanding of how systems interact and communicate.

Let us begin with informally recalling the definition of a system of communicating automata. It consists of a set of finite-state machines (one for every process/participant) where transitions are labelled over send or receive actions. Depending on the presence and the configuration of buffers, different semantics can be defined, giving rise to different communicating models. The simplest semantics one can define is the synchronous one, where there are no buffers and send and receive action can only happen simultaneously. This model is the most tractable, as the semantics can be represented by a finite state machine. This way all interesting properties can be translated into classic problems over finite state automata, that we know to be decidable. Hence synchronous communicating automata are a simple model but not very flexible nor expressive. Indeed most of the real-life systems are asynchronous where send and receive actions are decoupled. The most common communicating model amounts to the presence of a memory (usually FIFO buffers) among each pair of participants and per direction of communication. It is referred to as peer-to-peer communication (other models will be discussed in Section 3.2). It models a bit closely reality but it is much more complex as it can also encode Turing machines

[18], thus rendering model-checking undecidable. Now, similarly to the previous chapter, the question is to tune the model so that decidability can be retrieved loosing as little as possible in term of expressiveness. One of the directions to go to retrieve decidability is to "force" the synchrony of the system. This can be achieved, for instance by bounding the size of the buffers, giving rise to existentially and universally bounded models as in [47] or by organizing executions into phases where only a fixed number of send actions is allowed as within the notion of synchronizability in [14]. This is also the direction that we take in this chapter.

The first group of works that we discuss focus exactly on this last notion of synchronizability. We provided a generic framework, based on monadic second order logic and special tree-width, that unifies existing definitions, explains their good properties, and allows one to easily derive other, more general, definitions and decidability results [13].

Next, we show that this same technique can be applied to extend our reasoning across a spectrum of communicating models. These models span from synchronous rendezvous communications to fully asynchronous and out-of-order communications [94]. We presented a unified hierarchy of communication models, distinguished by their concurrent behaviors and the degree of synchronization inherent in their operations. We established that all the communication models we consider can be axiomatized within monadic second-order logic and may therefore benefit from several bounded verification techniques based on bounded special treewidth.

Finally, we close this chapter by discussing the closest work to process algebra and multiparty session types [98]. The systems under consideration are the closest to synchronous communication as all traces are causally equivalent, up to permutations of independent actions, to ones where each reception is immediately preceded by the corresponding send. We show that it is decidable to model-check them and check whether a system satisfy such a property.

Organization. The works discussed on this chapter are mostly taken from the results of the two PhD thesis of L. Laversa and L. Germerie Guizouarn (that I have co-supervised with E. Lozes). They correspond to the following publications: deciding synchronizability of systems [99, 100, 13] and hierarchy of message-passing communication models [94, 95], finally for Multiparty protocols and synchronous communications [98].

3.1 Deciding k-synchronizability of Systems

This first section focuses on the notion of synchronizability and boundedness. In particular we study systems where in order to tame expressiveness we bound the number of exchanged messages. This generates a group of families of systems that are not Turing equivalent where some interesting properties (e.g., reachability of configurations) turn out to be decidable. Hence a crucial step is to be able to

	Peer-to-Peer	Mailbox
Weakly synchronous	Undecidable [13]	EXPTIME [13]
Weakly k-synchronous	Decidable [14, 100, 13]	
Strongly k-synchronous	—	Decidable [13]
Existentially k-p2p-bounded	Decidable [47]	
Existentially k-mailbox-bounded	—	Decidable [13]

Figure 3.1 – Summary of the decidability of the membership problem in various classes. The – stands for a formalism that does not exist in literature

determine whether a systems belongs or not to one of those families (the so-called membership problem). Figure 3.1 sums up the membership problem for several families.

We mentioned before existentially k-bounded systems [47] where all executions can be re-ordered into an execution where the size of the buffer does not exceed k. This property is undecidable, even for a given k, but it becomes decidable when considering deadlock-free systems. In 2018, Bouajjani et al., called a system (weakly) *k-synchronizable* [14] if every execution can be divided into blocks of at most k messages. After each block, a message is either read, or will never be read. The semantics is represented by sets of message sequence charts (MSC for short) a graphical representation of the behavior that only depicts the relative order of actions. Processes are assigned vertical lines representing the order in which actions are executed. Matching send/receive actions are represented by connecting arrows. Given this representation of semantics, division into blocks does not imply that buffers are bounded to k messages. A key aspect between these works is that they consider different communication architectures. Existentially bounded systems have been studied for p2p, whereas k-synchronizability has been studied for mailbox communication, for which each process merges all its incoming messages in a unique queue. The decidability results for k-synchronizability have been extended to p2p communications [99], but it is unknown whether the decidability results for existentially bounded systems extend to mailbox communication. Moreover, variants of those definitions can be obtained depending on if we consider or not unmatched messages (i.e., messages that are sent but never read). Indeed the challenges that arise in [14] are due to mailbox communication and unmatched messages blocking a channel so that all messages sent afterwards will never be read. To clarify and overcome this issue, we proposed *strong k-synchronizability*, a new definition that is suitable for mailbox communication: an execution is called *strongly k-synchronizable* if it can be rescheduled into another k-bounded execution such that there are at most k messages in the channels before emptying them.

The decidability results mentioned above have been obtained by resorting to a general framework based on monadic second-order logic (MSO) and (special) tree-width —that is a graph measure that indicates how close a graph is to a tree [27]. Such a framework captures most of the existing definitions of systems that may work

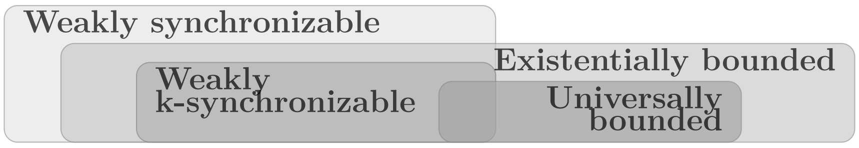


Figure 3.2 – Hierarchy of classes for p2p systems

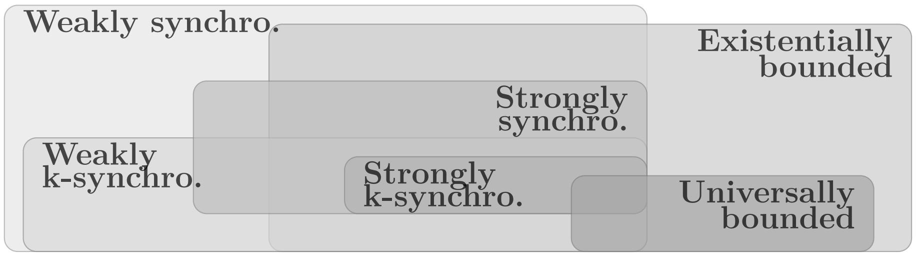


Figure 3.3 – Hierarchy of classes for mailbox systems

with bounded channels. It is quite valuable, as on top of membership one can show that reachability and model checking (if the logic formula is MSO definable) are decidable. Indeed it is enough to show that the communication model, combined with the bounding class, enforces a bounded treewidth of the class of behaviors represent via MSCs.

The decidability results are possible if the bound k is known, but computing k is, in general, a difficult problem. For instance, in the case of existentially-bounded classes, computing such bound is undecidable. We have been able to compute such a k for the class of mailbox weakly synchronous systems [100]. This is doable as the set of possible blocks (also called exchanges) represents a regular language, hence if the bound exists it will be the size of the largest word of the language.

Finally, we also compared the relations between each class of languages, depicted in Figures 3.2 and 3.3. In [71] one can find a complete discussion with examples, distinguishing each of the classes.

3.2 Hierarchy of Message-Passing Communication Models

Closely related to the previous set of works, in this section we explore expressiveness problems linked to synchronizability [94]. We focus on the same four classes as the previous section, this time varying the communicating model. We consider a range of communicating models spanning from asynchronous communication to realizable with synchronous communication (RSC). All the analyzed models can be axiomatized using monadic second-order logic. Moreover, it can be shown that most of those classes have finite treewidth, thus by employing the frameworks described

	Weakly sync	Weakly k-sync	∃k bounded	∀k bounded
`asy`	unbounded STW	✓	✓	✓
`p2p`	✗	✓	✓	✓
`co`	✗	✓	✓	✓
`mb`	✓	✓	✓	✓
`onen`	✓	✓	✓	✓
`nn`	✓	✓	✓	✓

Figure 3.4 – (Un)decidability results for the synchronizability problems.

above one can show decidability of membership and bounded model checking. Figure 3.5 summarizes the decidability results.

Interestingly, one can prove that for the class of weakly synchronous `p2p` MSCs, it is possible to build a system than generates MSCs with arbitrarily large grids (thus with unbounded treewidth). Such a construction can be employed to provide an encoding of a system into a weakly synchronous system with three processes, allowing to show that reachability of a configuration is undecidable [95].

Next, we briefly define the communicating models:

Fully Asynchronous Communication: (`asy`) messages can be received at any time once they have been sent, and send events are non-blocking. It can be modeled as a bag where all messages are stored and retrieved by processes when necessary.

Peer-to-Peer Communication: (`p2p`) any two messages sent from one process to another are always received in the same order as they are sent. This is usually implemented by processes pairwise connected with FIFO channels.

Causally Ordered Communication: (`co`) messages are delivered to a process according to the causality of their emissions. In other words, if there are two messages m_1 and m_2 with the same recipient, such that there exists a causal path from m_1 to m_2, then m_1 must be received before m_2. Causal ordering was introduced by Lamport in [68] with the name "happened before" order.

Mailbox Communication: (`mb`) any two messages sent to the same process, regardless of the sender, must be received in the same order as they are sent. In other words, if a process receives m_1 before m_2, then m_1 must have been sent before m_2. An implementation of the mailbox communication model consist in a single incoming FIFO channel for each process, in which all processes enqueue their messages.

FIFO `1−n` Communication: (`onen`) is the dual of `mb`, it coordinates a sender with all the receivers. Any two messages sent by a process must be received in the same order as they are sent. These two messages might be received by different processes and the two receive events might be concurrent. An implementation of the FIFO `1−n` communication model consist in a single

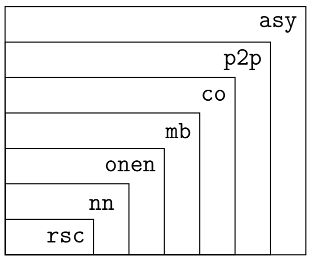


Figure 3.5 – The hierarchy of MSC classes.

outgoing FIFO channel for each process, which is shared by all the other processes. A send event would then push a message on the outgoing FIFO channel.

FIFO `n—n` Communication: (`nn`) messages are globally ordered and delivered according to their emission order. Any two messages must be received in the same order as they are sent. These two messages might be sent or received by any process and the two send or receive events might be concurrent. The FIFO `n—n` coordinates all the senders with all the receivers. An implementation of the FIFO `n—n` communication model consist in a single FIFO channel shared by all processes.

RSC Communication: (`rsc`) it imposes the existence of a scheduling such that any send event is immediately followed by its corresponding receive event.

From these definitions and their axiomatization in MSO, we can deduce a hierarchy of communication models, Figure 3.5. Surprisingly, the FIFO `1—n` class, that could be thought of as the "dual" of the mailbox class, is a subclass of mailbox class. This strongly contrasts with Chevrou et al. sequential hierarchy, where FIFO `1—n` and mailbox are incomparable [24].

3.3 Multiparty protocols and synchronous communications

In this contribution [98] , we study systems realizable with synchronous communication (RSC). RSC systems are a class of communicating automata designed to capture the essence of synchronous communication between concurrent processes. In such systems, all executions can be rescheduled in such a way that all receptions immediately follow their corresponding sends. In [98] (differently from the definition of RSC discusses in the previous section) we allow executions that contain unmatched messages. One of the main advantages of RSC systems is that most of the interesting properties turn out to be decidable regardless of the communication model considered.

The first good property is that, if we fix the number of processes, the membership problem, i.e., whether a system is RSC, is decidable in polynomial time. The proof proceeds in three steps: first, we show that RSC executions are characterized by acyclic conflict graphs, that visualizes the causal dependencies between send and receive actions. Second, we show that the fact that a system is not RSC is revealed by the existence of "bad" executions of a certain shape, called borderline violations [14]. These are executions for which every strict prefix is RSC. Finally, we show that the graphical characterization can be exploited to show the regularity of the language of borderline violations, from which we get the decidability of the RSC property.

The second class of good properties for RSC systems is that all regular safety properties, which includes reachability, absence of unspecified receptions, progress and boundedness are also decidable in polynomial time. Given a property P, the P safety problem consists in checking whether the intersection between the set of reachable states and those represented by P is empty. For RSC systems, the set of reachable states may not be regular, it is possible that buffers may contain configurations that are context free or context sensitive. Nonetheless, we show that for a computable regular property P, the P safety problem remains decidable.

3.4 Conclusion

This chapter explores the expressiveness of communicating automata, more specifically, exploiting the concept of synchronizability. The main result is a general framework. The decidability of membership and bounded model checking follows (i) from the definability of models into monadic second-order logic and (ii) from showing that the semantics, when interpreted as a classes of graphs, have finite tree-width. We have studied synchronizability taking into consideration different communicating models spanning from asynchronous to RSC.

In particular, RSC seems to be a good candidate to study the relations with multiparty session types, much similar to what has been done in [72]. Session types community mostly considers p2p buffers, the works discussed in this paper pave the way to generalize notions (syntax, semantics, types) to other communicating means.

On top of this, in the whole chapter we always admit the presence of unmatched messages, hinting at the fact that they may represent messages that are sent but not received, while multiparty session types always dismissed the presence of such messages, considering them as programming errors. Is this the only feasible interpretation? Maybe unmatched messages can be studied in the setting of an open systems, representing messages that are sent "outside" and so their control goes beyond the system under consideration.

Chapter 4

Applications

Variety's the very spice of life
That gives it all its flavor
— William Cowper, The Timepiece.

This chapter rounds up this manuscript by commenting some of the research directions that I have developed since my PhD and that have been left out in previous chapters. These works cannot be directly categorized as expressiveness studies but are worthy of mention. They are brought together by the fact that they all concern modeling and verifying concurrent systems from diverse disciplines like bioinformatics, ecology and neurosciences.

My first approach to bioinformatics came through the expressiveness analysis of rewriting models [39]. Indeed, biological processes are usually given in terms of pathways which are causal chains of responses to stimuli (a rewriting system in the computer science terminology). The κ-calculus has been introduced by Danos and Laneve in [31] to describe chemical and gene interactions. Since its introduction, and in its more general stochastic form [30, 16, 17], it has become an important reference for the description and analysis of such systems ([64, 83, 106] to mention a few examples). Building on [39], I become interested in rewriting systems for biological and bio-inspired systems and more specifically in reaction systems [20]. In [38], we extended reaction systems with discrete time constraints to model toxicity scenarios and introduced *ANDy* (for Activity Networks with Delays). It allows to model time-dependent biological systems, for instance regulatory pathways where toxic behaviors are represented by temporal logic formulae that define the reachability of bad (toxic) states or non-viable paths. Reaction rules called activities involve entities playing the role of activators, inhibitors or products of biochemical network operation. Activities may have a given duration, i.e., the time required to obtain results. Moreover entities are associated with levels (e.g., concentration, strength). Entity levels may change as a result of an activity or may decay gradually as time passes by. We chose discrete time constraints to ensure decidability (that we showed via an encoding into Petri nets) and leveraged the connection to Petri nets, making use of existing model checking tools for analysis.

Along the same line of research, in [97, 96] we considered applying rewriting systems to the modeling and analysis of ecosystems. The formalism was introduced in [45] to represent discrete qualitative models of ecosystems. Rewriting rules model conditions of appearance/disappearance of living (animals, plants etc.) and non-living entities (air, water, soil etc.). In [45] Gaucherel et al. cover the analysis of temporal features, such as basins of attraction (i.e., strongly connected states), tipping points (critical transitions along trajectories) and various kinds of collapses (functioning systems whose structures were nevertheless fixed), while, in our papers, we consider a way of comparing ecosystems. The method relies on a measure of similarity and on an optimization algorithm. The proposed method allows to detect patterns, i.e., ecological processes or sets of processes.

The last topic that I will present here concerns the modeling of biological neural networks [36, 35, 34]. We formalized spiking neural networks as networks of timed automata [2]. Timed automata are extensions of finite state automata which exhibit both discrete and continuous behaviors over time. They are well suited to model neural networks as they allow to model the influence of past inputs, or the presence of a refractory period, a lapse of time immediately following the spike emission in which the neuron cannot emit. The model is validated against some relevant properties expressed as temporal logical formulae. However, practical usability is limited due to the rapid explosion of the number of states, and one should find how to represent the behavior of groups of neurons instead of single entities. Beyond model-checking, we devised an algorithm reminiscent of supervised learning to infer synaptical weight values, enabling the display of expected behaviors.

Organization. Five topics are summarized in the following. The Kappa-Lattice: Decidability Boundaries for Qualitative Analysis in Biological Languages [39], Hunting Distributed Malware with the κ-Calculus [29], Activity Networks with Delays: An Application to Toxicity Analysis [38], Analysis of Discrete Models for Ecosystem Ecology [97, 96] and Spiking Neural Networks Modelled as Timed Automata: With Parameter Learning [36, 35, 34].

4.1 The κ-lattice

In this work, we explore the decidability boundaries of the κ-calculus. As mentioned before κ is a formalism for modeling molecular biology where molecules are terms with internal state and with sites, bonds are represented by names that label sites, and reactions are represented by rewriting rules. From a more mathematical point of view, κ is language modeling graph rewriting where molecules are nodes with fields (the state) that can have as many connections as the number of sites, and bonds are the arcs (Figure 4.1). Reactions have the shape $R \to P$, where R and P are set of entities (pre-solutions using the biology terminology) called reactants and products, respectively. They are of three kinds: Creations may produce new bonds between two unbound sites, or synthesize new molecules. Destructions behave the other way around. Exchanges are reminiscent of the π calculus because they define

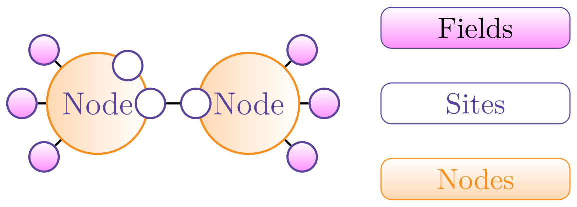
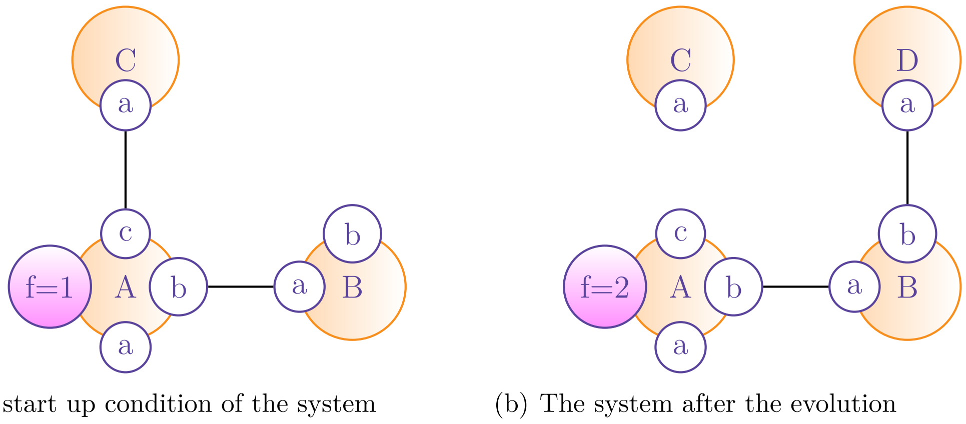


Figure 4.1 – Elements in κ



(a) The start up condition of the system (b) The system after the evolution

Figure 4.2 – A system and the application of a rule

a migration of a bond from one reactant to the other. We distinguish two types of exchanges: the one occurring between connected molecules, called (connected) bond flipping, and the one occurring between disconnected molecules, called free (bond) flipping. Take for instance the system represented in Figure 4.2(a)

$$A[f = 1](a^{\varepsilon} + b^{1} + c^{2}), B(a^{1} + b^{\varepsilon}), C(a^{2})$$

together with the application of rule (Figure 4.2(b)

$$A[f = 1](c^{2}), B(b^{\varepsilon}), C(a^{2}) \rightarrow A[f = 2](c^{\varepsilon}), B(b^{3}), C(a^{\varepsilon}), D(a^{3})$$

.

We consider a number of κ dialects, shown in Figure 4.3, that are inspired by biological phenomena (see [39] for some examples on how each fragment can represent a family of real biological reactions). Languages are ordered by inclusion and we move from κ along two different axes:

1. vertically we restrict the shape of destructions rules:
   - the superscript $-n$: we restrict destructions forbidding cancellations of molecules, as a consequence the number of molecules never decreases,
   - the superscript $-d$: we disallow destruction rules, no molecules nor bonds can be removed,

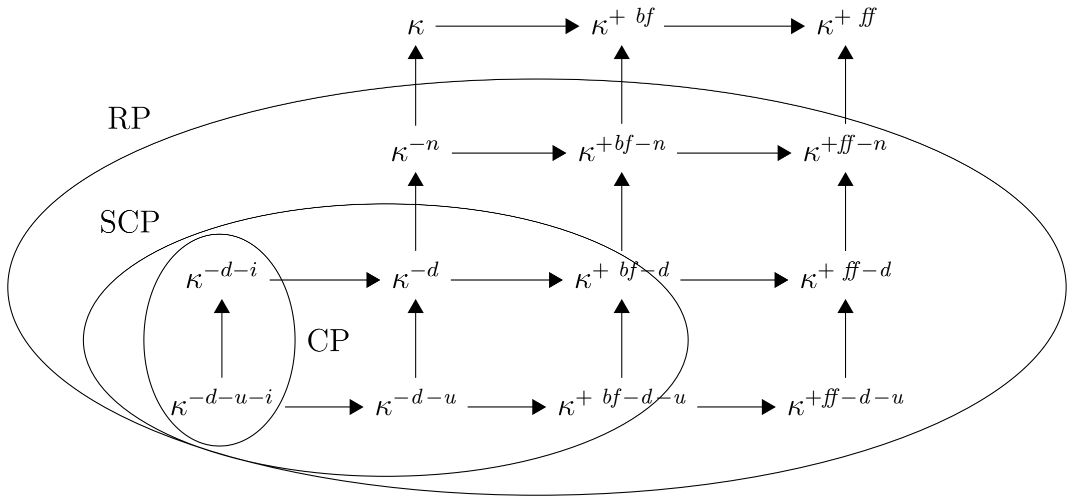


Figure 4.3 – The κ lattice, languages are ordered by the sublanguage relation

— the superscript $-d-u$: we disallow destructions and consider species with no fields,

— the superscript $-d-u-i$: we disallow destructions, fields and all creations are constrained so that one cannot verify whether reactants are linked, in other words no bond occurs on the left-hand side except for the one considered for flipping ,

2. horizontally we add capabilities coming from the exchange rule:
   — the superscript + *bf*: only allows bond-flipping,
   — the superscript + *ff*: allows the full free flipping, where exchanges can occur also between two unbounded molecules.

For all of the 14 dialects of κ we investigate three problems:

the reachability problem (RP): whether there exists a derivation from an initial to a target solution.

the simple coverability problem (SCP): whether there exists a solution reachable from an initial one containing exactly the same complexes as a given one, possibly with a greater multiplicity.

the coverability problem (CP): whether there exists a reachable solution containing a given (sub)solution. Differently from SCP, the shape of complexes in the target solution is not fixed a priori. We only require that it contains a given solution.

where a *complex* is a maximal set of connected molecules; and a *solution* is a multiset of complexes. We say that a solution is *reachable* from another when it is obtained from the latter one by using a finite sequence of reactions.

Theorem 4.1.1. *The following properties are decidable:*

1. *CP is decidable in* κ^{-d-i}.
2. *SCP is decidable for* $\kappa^{+\ bf-d}$.
3. *RP is decidable in* κ^{+ff-n}.

The following properties are undecidable:

1. *CP is undecidable in* κ^{-d-u}.
2. *SCP is undecidable in* κ^{-n} *and* $\kappa^{+ff-d-u}$.
3. *RP is undecidable in* κ.

The results on the (un)decidability of RP, SCP, and CP in the κ lattice are illustrated in Figure 4.3. Notice that for decidability results it is enough to prove the theorem for the largest calculus in the class and conversely for undecidability for the smallest.

Undecidability is obtained by modeling Turing complete formalisms in the calculi. The most surprising result is the undecidability of CP in κ^{-d-u}. We prove that this quite poor fragment of κ – in which molecules have no state and bonds can be neither destroyed nor flipped – is powerful enough to encode Two Counter Machines [79]. Observe that this result relies on the possibility to test at least the presence of bonds (allowing to test whether a complex, representing the termination of a computation can be produced). While the dialects that include κ^{-d-u} are Turing complete, many of them retain decidable SCP and/or RP properties. These facts, apparently contrasting with Turing universality of the calculi, are consequences of the following monotonic properties: reactions cannot decrease (i) the total number of molecules in the solution nor (ii) the size of the complexes in the solution. In the calculi satisfying the form of monotonicity (i) we show that it is possible to compute an upper-bound to the number of molecules in the solutions of interest for the analysis of RP. In this way, we reduced our analysis to a finite state system. For the calculi satisfying the form of monotonicity (ii) we show that it is possible to compute an upper-bound to the size of the complexes in the solutions of interest for the analysis of SCP. In this case, even if it is not possible to reduce to a finite state system (because there is no upper-bound to the number of instances of the complexes in the solutions of interest), we could reduce to Petri-nets in which target reachability and coverability are decidable.

4.2 Distributed Malware with the κ-Calculus

In this work, we present a proof of concept, using the κ-calculus, for detecting and analyzing distributed malware together with their propagation mechanisms.

The term *malware* is used to refer generically to a malicious code, i.e., any software program designed to move from computer to computer and network to network in order to intentionally modify computer systems without the consent of the owner [51]. Since the threat of malware attacks is an unavoidable problem in computer security, it is crucial to develop both sophisticated models for expressing and understanding distributed malware (e.g., botnets) and efficient techniques of defense from them. A promising approach regards the monitoring of malware propagation in a network both for studying properties of the considered malware and/or for finding ways to immunize the network. Indeed, the monitoring of malware propagation turns out to be particularly interesting for analyzing the topology of botnets and for understanding, detecting and stopping bot propagation [25].

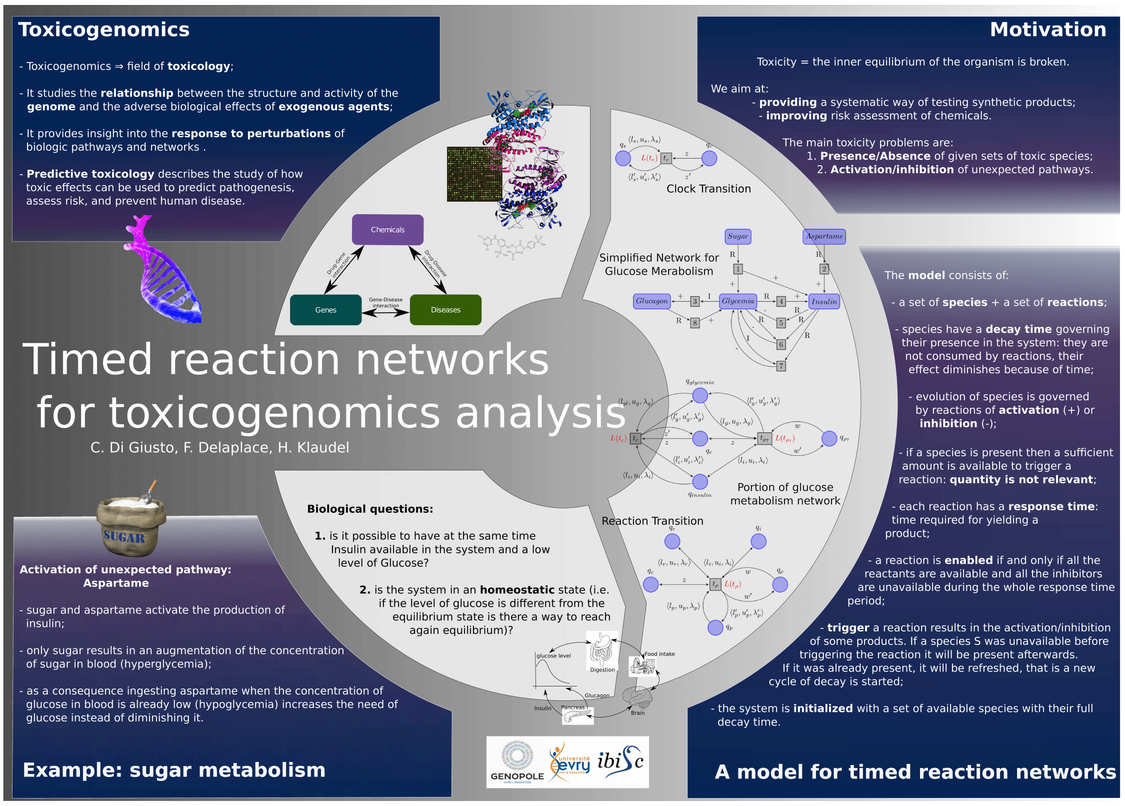
We propose to represent malware as formal structures in the κ-calculus, the framework captures the dynamic behavior and interactions of malware agents across different nodes in the network. As biological virus are generally associated to malware, we found it natural to look for a suitable calculus among the ones designed for computational biology. We look for a formalism that not only could express concurrent interactions but that could also describe spatial aspects such as containment or relative position (i.e., topology).

We characterize malware in κ as programs (i.e., molecules or better in this context components, together with the rules describing their evolution). They exhibits the three defining behaviors of malware: The malware has to *imitate* the behavior of the host (e.g., evolving using the rules of the host). It has to possibly perform an *injury*, meaning that the payload is observable in the environment, and it is independent from the hosting program. Finally the malware replicates and *infects* the environment. We give examples on how to code different kinds of viruses: spywares, worms and botnets among the others.

We demonstrate how the κ-calculus could be used to perform simulations and analyses of malware behavior. By studying the topology of the network in which the malware spreads, it is possible to identify the best nodes to immunize, or to decide whether a given infection will evolve into an epidemics. This enables the identification of potential attack vectors, vulnerabilities, and patterns of infection. κ offers a framework where the granularity of the model can be chosen depending on the specific problem to represent: i.e., one can choose to model programs inside a single machine, or model a complex set of machines abstracting from the programs they are running. Finally the analysis of malware formalized in κ can benefit from the existence of tools for the simulation of κ-systems [16]. Moreover, these tools – developed in the biological setting – allow the specification of stochastic measures that can model the probability of a certain node to be infected, the rate of propagation of a given infection, etc..

4.3 Toxicity Analysis

In the contributions that follows, we keep working on rewriting systems. Such systems are slightly different than κ. On one side we do not need to consider links (bonds in the κ terminology) and on the other side we add time constraints and the ability of talking about entities that are not present.


Toxicogenomics

- Toxicogenomics ⇒ field of **toxicology**;

- It studies the **relationship** between the structure and activity of the **genome** and the adverse biological effects of **exogenous agents**;

- It provides insight into the **response to perturbations** of biologic pathways and networks .

- **Predictive toxicology** describes the study of how toxic effects can be used to predict pathogenesis, assess risk, and prevent human disease.

Motivation

Toxicity = the inner equilibrium of the organism is broken.

We aim at:
- **providing** a systematic way of testing synthetic products;
- **improving** risk assessment of chemicals.

The main toxicity problems are:
1. **Presence/Absence** of given sets of toxic species;
2. **Activation/inhibition** of unexpected pathways.

Timed reaction networks for toxicogenomics analysis

C. Di Giusto, F. Delaplace, H. Klaudel

The **model** consists of:

- a set of **species** + a set of **reactions**;

- species have a **decay time** governing their presence in the system: they are not consumed by reactions, their effect diminishes because of time;

- evolution of species is governed by reactions of **activation** (+) or **inhibition** (-);

- if a species is present then a sufficient amount is available to trigger a reaction: **quantity is not relevant**;

- each reaction has a **response time**: time required for yielding a product;

- a reaction is **enabled** if and only if all the reactants are available and all the inhibitors are unavailable during the whole response time period;

- **trigger** a reaction results in the activation/inhibition of some products. If a species S was unavailable before triggering the reaction it will be present afterwards. If it was already present, it will be refreshed, that is a new cycle of decay is started;

- the system is **initialized** with a set of available species with their full decay time.

A model for timed reaction networks

Biological questions:

1. is it possible to have at the same time Insulin available in the system and a low level of Glucose?

2. is the system in an **homeostatic** state (i.e. if the level of glucose is different from the equilibrium state is there a way to reach again equilibrium)?

Activation of unexpected pathway: Aspartame

- sugar and aspartame activate the production of insulin;

- only sugar results in an augmentation of the concentration of sugar in blood (hyperglycemia);

- as a consequence ingesting aspartame when the concentration of glucose in blood is already low (hypoglycemia) increases the need of glucose instead of diminishing it.

Example: sugar metabolism

Figure 4.4 – The poster presented at Evry'14, Thematic Research School Poster session

In [38] we introduce a formal framework called Activity Networks with Delays (ANDy) for modeling and analyzing time-dependent biological systems, specifically focusing on toxicity scenarios.

The paper addresses the challenge of modeling biological systems where activities have associated durations and the timing of events is crucial. Our proposal originates from reaction systems [20]. The formalism encompasses entities and activities. Each entity is characterized by one attribute, called *level*, representing activation/inhibition status or expression degree (e.g., low, medium, high). *Activities* are rules describing the evolution of systems, involving three sets of entities: *activators*, which must be present, *inhibitors*, which must be absent, and *results* whose expression levels have to be modified (increased or decreased). The introduction of (discrete) time concerns both activities and entities. Activities have a duration and entities are subject to a *decay* process that models the action of a non-specified environment: i.e., levels decrease with time progression. Differently from reactions systems, reactants are not consumed by activities but their levels naturally decrease as an effect of decay. Another difference is in the semantics of activities: while in reaction systems a maximal parallelism model is considered, here we adopted two types of activities: *mandatory* and *potential* ones. The former set of activities, once enabled, must take place altogether in a time unit (this is similar to what happens in reaction system). While the latter, non-deterministically and separately, may be performed or not.

The idea of adding activity duration and decay fits perfectly toxicity problems where exposure time and threshold dosage are crucial parameters. Indeed toxicology [107] studies the adverse effects of the exposure to chemicals at various levels of living entities: organism, tissue, cell and intracellular molecular systems. A toxicity process is a sequence of physiological events that causes the abnormal behavior of a living organism with respect to its healthy state. Such processes can be modeled either by identifying some pathological states or by characterizing sequences of states (traces), e.g., non viable behaviors. All these properties can be given in terms of temporal logic formulae.

We show that ANDy has finite state space and we provide an encoding into Petri Nets and Timed Automata that have associated tools to perform the toxicity analysis. We showcase the use of ANDy, modeling the blood glucose regulation in human body and interpreting the interaction of aspartame, instead of normal sugar, as a potential toxic behavior (see Figure 4.4). A prototype implementation of ANDy networks using the Snakes toolkit [87], Snoopy [53] and its related analysis tool Charlie [54] is available at [93].

4.4 Discrete Models for Ecosystem Ecology

Another application of rewriting systems are ecosystems. They are complex processes of highly different nature: e.g., bio-ecological, physico-chemical and socio-economical. The dynamics of such systems is difficult to grasp as it is the result of an intricate interplay between a large number of processes: the functioning of living species (e.g., fauna and flora) and inert components (e.g., dynamics of soil and climate), as well as human activities. In ecology, discrete qualitative modeling is still pioneering and under exploited. Approaches such as those by Gaucherel et al. [104, 43, 44], where the authors study the driving rules needed to change agricultural mosaics and model contrasted landscapes, are promising. As a starting point of our developments we took a general discrete qualitative formalism proposed by Gaucherel and Pommereau in [46]. Ecosystems are modeled as a set of (living and nonliving) entities together with a set of rewriting rules expressing the conditions of their appearance/disappearance (i.e., the ecosystem component responses).

We propose two techniques to complement the analysis of ecosystems modeled as in [46] without fully developing their semantics. On one side we want to remove redundant information that could have been (unintentionally) introduced during the modeling process and that could greatly lengthen computations. On the other side we want to compare models by focusing on their syntax.

For the first part we identify four strategies to simplify models divided into two groups. First those that find and remove redundant information immutable entities and correlations and second, those that are intended to find and hide some local behavior in a controlled way: cascading, and spontaneous production.

The former two (immutable entities and correlations) are conservative in the sense they do not modify the state space of the model but remove redundancy in entities and rules. *Immutable entities* are those that cannot change their state along the evolution of a system. They are never produced nor consumed by a rule, i.e., they are never on the right hand side. Thus if they are included in the initial state, they will always be present and conversely if they do not belong to it they will never appear in the ecosystem. As a consequence, we can simplify the model by removing occurrences of those entities or entire rules depending on the initial state and whether the rule requires the presence or the absence of such an entity. For *correlation*, we look for sets of entities that appear in rules always together and with consistent polarity. To find such sets we build a lattice whose infima represents the searched correlations.

The latter strategies (cascading and spontaneous production) slightly modify the state space but always in a controlled way. With the *cascading strategy* we remove rules which may be reduced as an effect of transitivity. This way we can remove stuttering and the introduction of intermediate states that are not used. For *spontaneous production*, we tackle rules that model production of entities without any precondition. Such rules entail that those entities are always present. In this case we raise an alert and leave the choice to the user. If the user is interested in the asymptotic behavior (the entity should not disappear from the system) then it is better to add the entities to the initial state and simplify the model by eliminating all the effects of those entities on rules. Otherwise, if the user is interested in observing possibly "oscillating" behaviors then leave the system as it is. This means that the entities will appear non-deterministically when the rule is used. They can disappear as an effect of other rules to reappear back when the rule is used again.

For the second part, we notice that many processes are common to most ecosystems: e.g., for species interactions: predation, competition, symbiosis, etc. and it is crucial to be able to identify them to better describe the ecosystem under study. We propose a method to detect whether a given process is present in an ecosystem. This can help to understand if the introduction of a new entity could cause the appearance of a known process, and to understand the real nature of new processes. Moreover, it is more efficient to search for patterns by referring only to the syntactical system specification instead of considering the whole semantics (that is exponential in the number of entities) based on the assumption that similar processes look similar at the rule level.

In order to compare two models of ecosystems, we introduce a pair of mappings, the first identifying entities and the latter rules, and a similarity measure expressed as a scoring function. This scoring majors the number of matched entities and rules, and penalizes those that do not perfectly match. Similarity is then defined as an optimization problem through the scoring function. Indeed, the scoring function with optimal value uniquely determines the mappings of entities and rules. The definition of the scoring function is used to search for interaction patterns in the rule-based models of ecosystems. As the complexity of this kind of search is exponential, it is not always possible in realistic cases to find optimal solutions in a reasonable time. Nevertheless, optimization tools generally allow obtaining a sub-optimal solution quite efficiently.

We implemented a prototype that allows encoding the matching of two models into a pseudo-Boolean optimization problem and invoked a satsolver to solve it. We applied this prototype to systematically match a collection of predefined interaction patterns against a set of models of realistic ecosystems.

4.5 Spiking neural networks as timed automata

The last topic discussed here concerns the modeling of biological neural networks. No rewriting system is considered here, instead we model networks via timed automata. We take the so-called *third generation* models of neuron behavior, also known as spiking neural networks (SNNs). Neurons may fire output spikes according to threshold-based rules which take into account input spike magnitudes and occurrence times [85]. We focus on the Leaky Integrate and Fire Model (LI&F) model originally proposed in [70], especially we consider the discretized formulation given in [33], which relies on the notion of logical time. Time is considered as a sequence of logical discrete instants, and an instant is a point in time where external input events can be observed, computations can be done, and outputs can be emitted. We encode LI&F networks into Timed Automata networks where each neuron is an automaton. We show how to define the behavior of a single neuron and how to build a network of neurons. The behavior of a neuron consists in accumulating the weighted sum of inputs, provided by a number of ingoing weighted synapses, for a given amount of time. Then, if the potential accumulated during the last and previous accumulation periods overcomes a given threshold, the neuron fires an output over the outgoing synapse. Synapses are channels shared between the timed automata, while spike emissions are represented by broadcast synchronizations occurring over such channels. We coded a prototype of our neuron using Uppaal [3] and used the built-in model checker to evaluate the model.

We analyze some intrinsic properties of the proposed model: the minimum threshold value allowing a neuron to emit, the minimum delay between neuron emissions and the lack of inter-spike memory, preventing the behavior of a neuron from being influenced by what happened before the last spike. Moreover we evaluate our model with respect to a classification based on a work by Izhikevich [60] that classifies spiking neuron models according to some behavior that they should exhibit in order to be considered biologically relevant. We encode in temporal logics all the behaviors (or capabilities) a LI&F model should be able to reproduce and we exploit model checking to prove these behaviors are reproducible in our model. Izhikevich also identifies a set of behaviors which are not expected to be reproducible by any

LI&F model. We prove these limits to hold for our model as well. We show that our model can reproduce:

tonic spiking: the behavior of a neuron producing a periodic output sequence as a response to a persistent excitatory constant input sequence;

the integrator behavior: the behavior of a neuron producing an output spike whenever it receives at least a specific number of spikes from its input sources in the same accumulation period;

excitability: the behavior of a neuron emitting sequences having a decreasing inter-firing period, i.e., an increasing output frequency, when stimulated by an increasing number of excitatory inputs.

For the negative results we show that our model cannot reproduce:

phasic spiking: the behavior of a neuron producing a single output spike when receiving a persistent and excitatory input sequence and then remaining quiescent for the rest of it;

bursting: the ability of producing a finite sequence of high frequency spikes;

spiking frequency adaptation: the behavior of a neuron producing a decreasing frequency output sequence as a response to a persistent excitatory input sequence;

spike latency: the behavior of a neuron firing delayed spikes, with respect to the instant when its potential reached or overcame the threshold

threshold variability: the behavior of a neuron allowing its threshold to vary according to the strength of its inputs;

bistability: the behavior of a neuron alternating between two operation modes: periodic emission and quiescence;

inhibition-induced spiking: the behavior of a neuron producing a spike output sequence as a response to a persistent inhibitory input sequence

rebound spiking: the behavior of a neuron producing an output spike after it received an inhibitory input.

For each non-reproducible behavior, we provide an extension of the model allowing to reproduce it.

Then, we exploit our automata-based modeling and model checking to propose a new methodology for parameter inference in SNNs: i.e., how to determine a parameter assignment for a network with a fixed topology and a given input such that a desired output behavior is displayed. Our analysis is inspired by the SpikeProp algorithm [12], which deals with fully connected feedforward networks of spiking neurons with layers and aims at attaining a set of target firing times of the output neurons for a given set of input patterns. An error function is obtained by computing the least mean squares of desired spike times and actual firing times; such an error is then back-propagated in the network. The main differences between the SpikeProp approach and our, are that our networks are discrete and not multi-layered, and we allow to have both positive and negative inputs spikes. The learning algorithm is then applied to find suitable parameters in mutual inhibition networks, a well studied class of networks in which the constituent neurons inhibit each other activity [75], and to a class of networks known as archetypes from [33]. These small circuits (e.g., simple series, parallel composition, negative loop, inhibition of a behavior, etc.) can be seen as the constituting bricks of bigger networks.

4.6 Conclusions

All the works discussed in this chapter concern mainly discrete modeling of complex systems. Such systems are intrinsically difficult because they describe interactions between a large number of entities and sometimes even consider interactions with parts that are under or non-specified (open-systems). These systems are often modeled via continuous functions (hence justifying the omnipresence of stochastic extensions of the models discussed above). Such models capture the "real" world more accurately and are considered to be better suited for analysis than discrete systems.

In my experience, discrete modeling works well when it is intended to be a prototype to guide further investigation. But they fall short when one is interested in finding mainstream applications of the model. This is evidenced by the proliferation of stochastic extensions of the classical models (Petri nets, κ, etc.).

Chapter 5

Future and beyond

The best is yet to come

— Carolyn Leigh, Cy Coleman.

In this dissertation, we have discussed expressiveness properties of three large families of formalisms: process algebras, communicating automata, and a variety of discrete models to be applied to bio-inspired systems. At the end of each chapter we have already hinted at possible future developments, here we quickly mention more general future directions.

I envisage pursuing three main aspects (perhaps possible topics for future PhD theses):

— Causal dependencies and reversible computations,

— Temporal aspects,

— Complexity.

Regarding causality and reversible computations, there is a strong connection between the expressiveness results discussed in Chapter 3 and causality notions. Indeed most of those results embed a notion of causality that is used to represent the order in which messages fill buffers. Such notions are also the foundation of reversible computations. As a matter of fact, one needs to know the order in which actions have been performed to restore a correct computation. In a concurrent world this entails that one needs to know (and store) information about which are the parallel/concurrent actions and where are the causal dependencies.

The second topic concerns temporal aspects which are, in general, a difficult subject. Temporal/timed extensions already exist under the format of timed automata [2] that employ real time clocks, timed process algebra and timed Petri nets. I would like to explore the meaning of timing the life of messages (hence encoding the idea of expiring messages) instead of actions. The main issue is to determine whether this can be encoded or not with existing formalisms or whether time is an orthogonal matter and expressiveness results can be ported in the new setting.

Finally, on complexity, it seems to me that the concurrent calculus community generally produces results on (un)decidability, but less often focuses on complexity classes. I believe that this will change and that we need to work more in this direction. To conclude this dissertation I will briefly summarize a recent work along these lines [9].

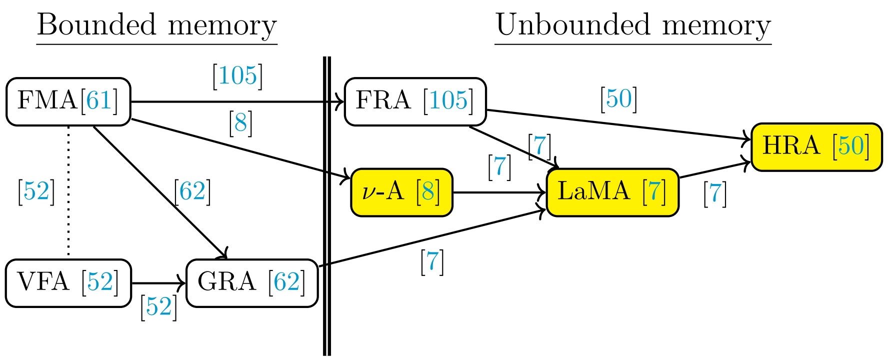


Figure 5.1 – A classification of memory automata from [7]. Arrows represent strict language inclusion, while dotted lines denote language incomparability.

5.1 Complexity in Unbounded Memory Automata

We study unbounded memory automata for words over an infinite alphabet, as introduced in [61, 82]. Such automata model essentially dynamic generative behaviors, i.e., programs that generate an unbounded number of distinct resources each with its own unique identifier (e.g., thread creation in Java, XML). Figure 5.1 depicts a hierarchy of unbounded memory automata, for a precise discussion on the relations among these formalisms see [7]. We focus, in particular, on three formalisms, ν-automata (ν-A) [37, 8], Layered Memory Automata (LaMA) [7] and HRA for History-Register Automata [50]. All these models are extensions of finite state automata with memory-storing letters. The memory for HRA is composed of registers (that can store only one letter) and histories (that can store an unbounded number of letters), whereas the memory for the other two models consists of a finite set of variables. Among the distinctive features of HRA, they can reset registers and histories, and select, remove and transfer individual letters. In ν-A and LaMA, variables must satisfy an additional constraint, referred to as injectivity, meaning that they cannot store shared letters. Moreover, variables can be emptied (reset) but single letters cannot be removed. We know that LaMA are more expressive than ν-A as the former are closed under intersection while it is not the case for the latter ones.

We tackle the complexity of two problems: membership (whether a word belongs to the recognized language of the considered automaton) and non-emptiness (whether the language recognized by an automaton is empty or not) for ν-A, LaMA and HRA (see Figure 5.2). We knew from [50] that the non-emptyness problem is Ackerman-complete for HRA. We show that concerning the membership problem, all three kinds of automata fall in the NP-complete class. We first prove that testing membership for HRA, ν-A and LaMA is an equivalent problem. This is obtained by providing two simulations of respectively LaMA and HRA into ν-A. Then we

Model	Membership	Non-emptyness
ν-A	NP-comp	PSPACE-comp
LaMA	NP-comp	Ackermann-comp
HRA	NP-comp	Ackermann-comp
FMA	NP-comp	NP-comp
VFA	NP-comp	NL-comp
FRA	NP-comp	?
GRA	?	?

Figure 5.2 – Summary of complexity classes

address complexity and show that the problem is NP-complete with a reduction of 3-SAT to LaMA. Non-emptiness is more delicate. We prove that the non-emptiness problem for ν-A is PSPACE-complete by reducing TQBF (True Fully Quantified Boolean Formula) to ν-A, while the proof in [50] can be adapted to show that the non-emptiness is Ackermann-complete also for LaMA. For finite-memory automata (FMA), it is known that membership and non-emptiness are NP-complete [88]. Knowing whether a language is included in another is undecidable for FMA when considering their non-deterministic version, but it is PSPACE-complete for deterministic ones [81]. In [52] it is shown that the non-emptiness problem for Variable Finite Automata (VFA) is NL-complete, while membership is NP-complete. For FRA and Guessing-Register Automata (GRA, [63]) we only know that both problems are decidable but we do not know the accurate complexity class.

This paper provides a comprehensive understanding of the computational complexity of unbounded memory automata, offering insights into their expressive power and limitations. I believe complexity studies may contribute to give a broader context within the field of formal language theory, paving the way for further exploration and finer comparisons among concurrent languages. For instance, all the complexity results of Chapter 3 are upper bounds, and little is known about the lower bounds.

Bibliography

[1] Open Source Erlang. http://www.erlang.org/, 2011.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[3] Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Ansgar Fehnker, Thomas Hune, Bertrand Jeannet, Kim G. Larsen, M. Oliver Möller, Paul Pettersson, Carsten Weise, and Wang Yi. *UPPAAL - Now, Next, and Future*, pages 99–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[4] Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Found. Trends Program. Lang.*, 3(2-3):95–230, 2016.

[5] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005. Process Algebra.

[6] Jan Bergstra, Alban Ponse, and Scott Smolka, editors. *Handbook of Process Algebra*, volume 335. Elsevier Science, 01 2001.

[7] Clément Bertrand, Hanna Klaudel, and Frédéric Peschanski. Layered memory automata: Recognizers for quasi-regular languages with unbounded memory. In Luca Bernardinello and Laure Petrucci, editors, *Application and Theory of Petri Nets and Concurrency - 43rd International Conference, PETRI NETS 2022, Bergen, Norway, June 19-24, 2022, Proceedings*, volume 13288 of *Lecture Notes in Computer Science*, pages 43–63. Springer, 2022.

[8] Clément Bertrand, Frédéric Peschanski, Hanna Klaudel, and Matthieu Latapy. Pattern matching in link streams: Timed-automata with finite memory. *Sci. Ann. Comput. Sci.*, 28(2):161–198, 2018.

[9] Clément Bertrand, **Cinzia Di Giusto**, Hanna Klaudel, and Damien Regnault. Complexity of membership and non-emptiness problems in unbounded memory automata. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 33:1–33:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[10] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR*, volume 5201 of *LNCS*, pages 508–522. Springer, 2008.

[11] Laura Bocchi, Maurizio Murgia, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Asynchronous timed session types. In Luís Caires, editor, *Programming Languages and Systems*, pages 583–610, Cham, 2019. Springer International Publishing.

[12] Sander M. Bohte, Han A. La Poutré, Joost N. Kok, Han A. La, Poutre Joost, and N. Kok. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48:17–37, 2002.

[13] Benedikt Bollig, **Cinzia Di Giusto**, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. A unifying framework for deciding synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[14] Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 372–391, Cham, 2018. Springer International Publishing.

[15] Frédéric Boussinot and Robert de Simone. The SL synchronous language. *Software Engineering*, 22(4):256–266, 1996.

[16] Pierre Boutillier, Jérôme Feret, Jean Krivine, and Walter Fontana. *The Kappa Language and Tools (V4)*. R Foundation for Statistical Computing, kappalanguage.org, 2020.

[17] Pierre Boutillier, Mutaamba Maasha, Xing Li, Héctor F Medina-Abarca, Jean Krivine, Jérôme Feret, Ioana Cristescu, Angus G Forbes, and Walter Fontana. The Kappa platform for rule-based modeling. *Bioinformatics*, 34(13):i583–i592, 06 2018.

[18] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, apr 1983.

[19] Mario Bravetti, **Cinzia Di Giusto**, Jorge A. Pérez, and Gianluigi Zavattaro. Adaptable processes. *Log. Methods Comput. Sci.*, 8(4), 2012.

[20] Robert Brijder, Andrzej Ehrenfeucht, Michael G. Main, and Grzegorz Rozenberg. A tour of reaction systems. *Int. J. Found. Comput. Sci.*, 22(7):1499–1517, 2011.

[21] Mauricio Cano, Ilaria Castellani, Cinzia Di Giusto, and Jorge A. Pérez. Multiparty Reactive Sessions. Research Report 9270, INRIA, April 2019.

[22] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March*

24 - April 1, 2007, Proceedings, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[23] Ilaria Castellani. Process algebras with localities. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 945–1045. North-Holland / Elsevier, 2001.

[24] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Aspects Comput.*, 28(5):847–879, 2016.

[25] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *SRUTI 2005*, pages 39–44, 2005.

[26] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Self-adaptive multiparty sessions. *Serv. Oriented Comput. Appl.*, 9(3-4):249–268, 2015.

[27] Bruno Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS*, volume 8 of *LIPIcs*, pages 13–29, 2010.

[28] Patrick Cousot and Radhia Cousot. Semantic analysis of communicating sequential processes (shortened version). In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1980.

[29] Mila Dalla Preda and **Cinzia Di Giusto**. Hunting distributed malware with the κ-calculus. In Olaf Owe, Martin Steffen, and Jan Arne Telle, editors, *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings*, volume 6914 of *Lecture Notes in Computer Science*, pages 102–113. Springer, 2011.

[30] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 17–41. Springer, 2007.

[31] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[32] René David and Hassane Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175–202, 1994.

[33] Elisabetta De Maria, Alexandre Muzy, Daniel Gaffé, Annie Ressouche, and Franck Grammont. Verification of Temporal Properties of Neuronal Archetypes Using Synchronous Models. In *Fifth International Workshop on Hybrid Systems Biology*, Grenoble, France, 2016.

[34] Elisabetta De Maria and **Cinzia Di Giusto**. Inferring the synaptical weights of leaky integrate and fire asynchronous neural networks: Modelled as timed automata. In Alberto Cliquet Jr., Sheldon Wiebe, Paul E. Anderson, Giovanni Saggio, Reyer Zwiggelaar, Hugo Gamboa, Ana L. N. Fred, and Sergi Bermúdez i Badia, editors, *Biomedical Engineering Systems and Technologies - 11th International Joint Conference, BIOSTEC 2018, Funchal, Madeira, Portugal, January 19-21, 2018, Revised Selected Papers*, volume 1024 of *Communications in Computer and Information Science*, pages 149–166. Springer, 2018.

[35] Elisabetta De Maria and **Cinzia Di Giusto**. Parameter learning for spiking neural networks modelled as timed automata. In Paul Anderson, Hugo Gamboa, Ana L. N. Fred, and Sergi Bermúdez i Badia, editors, *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 3: BIOINFORMATICS, Funchal, Madeira, Portugal, January 19-21, 2018*, pages 17–28. SciTePress, 2018.

[36] Elisabetta De Maria, **Cinzia Di Giusto**, and Laetitia Laversa. Spiking neural networks modelled as timed automata: with parameter learning. *Nat. Comput.*, 19(1):135–155, 2020.

[37] Aurelien Deharbe and Frédéric Peschanski. The omniscient garbage collector: A resource analysis framework. In *14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014*, pages 102–111. IEEE Computer Society, 2014.

[38] Franck Delaplace, **Cinzia Di Giusto**, Jean-Louis Giavitto, Hanna Klaudel, and Antoine Spicher. Activity networks with delays an application to toxicity analysis. *Fundam. Informaticae*, 160(1-2):119–142, 2018.

[39] Giorgio Delzanno, **Cinzia Di Giusto**, Maurizio Gabbrielli, Cosimo Laneve, and Gianluigi Zavattaro. The *kappa*-lattice: Decidability boundaries for qualitative analysis in biological languages. In Pierpaolo Degano and Roberto Gorrieri, editors, *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings*, volume 5688 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2009.

[40] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[41] Alain Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[42] Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. BASS: boxed ambients with safe sessions. In Annalisa Bossi and

Michael J. Maher, editors, *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy*, pages 61–72. ACM, 2006.

[43] Cédric Gaucherel, Frédéric Boudon, Thomas Houet, Mathieu Castets, and Christophe Godin. Understanding patchy landscape dynamics: Towards a landscape language. *PLoS ONE*, 7(9):16, April 2012.

[44] Cédric Gaucherel, François Houllier, Daniel AUCLAIR, and Thomas Houet. Dynamic Landscape Modelling : The Quest for a Unifying Theory. *Living reviews in landscape Research*, 8(2):5–31, 2014.

[45] Cedric Gaucherel and Franck Pommereau. Using discrete systems to exhaustively characterize the dynamics of an integrated ecosystem. *Methods in Ecology and Evolution*, 10(9):1615–1627, 2019.

[46] Cédric Gaucherel and Franck Pommereau. Using discrete systems to exhaustively characterize the dynamics of an integrated ecosystem. *Methods in Ecology and Evolution*, pages 1–13, 06 2019.

[47] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.

[48] Cinzia Di Giusto. Personal webpage. https://niouze.i3s.unice.fr/digiusto/node/1.

[49] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010.

[50] Radu Grigore and Nikos Tzevelekos. History-register automata. *Log. Methods Comput. Sci.*, 12(1), 2016.

[51] R. A. Grimes. Malicious mobile code: Virus protection for windows. *O'Reilly & Associates, Inc.*, 2001.

[52] Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Variable automata over infinite alphabets. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, 2010.

[53] Monika Heiner, Mostafa Herajy, Fei Liu, Christian Rohr, and Martin Schwarick. Snoopy - A unifying petri net tool. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*, volume 7347 of *Lecture Notes in Computer Science*, pages 398–407. Springer, 2012.

[54] Monika Heiner, Martin Schwarick, and Jan-Thierry Wegener. Charlie - an extensible petri net analysis tool. In Raymond R. Devillers and Antti Valmari, editors, *Application and Theory of Petri Nets and Concurrency - 36th*

International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings, volume 9115 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2015.

[55] Petr Hnetynka and Frantisek Plasil. Dynamic reconfiguration and access to services in hierarchical component models. In *Proc. of CBSE'06*, volume 4063 of *LNCS*, pages 352–359. Springer, 2006.

[56] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[57] C. A. R. Hoare and Leslie Lamport. *Concurrent and Distributed Systems*, pages 295–358. Springer London, London, 1999.

[58] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[59] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008.

[60] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004.

[61] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[62] Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010.

[63] Ahmet Kara. *Logics on data words: Expressivity, satisfiability, model checking.* PhD thesis, Technical University of Dortmund, Germany, 2016.

[64] Yarden Katz and Walter Fontana. Probabilistic inference with polymerizing biochemical circuits. *Entropy*, 24(5):629, 2022.

[65] Dimitrios Kouzapas, Nobuko Yoshida, Raymond Hu, and Kohei Honda. On asynchronous eventful session semantics. *Math. Struct. in Comp. Science*, 2013. To appear.

[66] Joseph B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.

[67] Dietrich Kuske and Anca Muscholl. Communicating automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 1147–1188. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[68] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[69] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 221–232. ACM, 2015.

[70] L. Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635, 1907.

[71] Laetitia Laversa. *La synchronisabilité pour les systèmes distribués. (Synchronizability for distributed systems).* PhD thesis, University of Côte d'Azur, Nice, France, 2021.

[72] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 350–373. Springer, 2023.

[73] Bas Luttik. What is algebraic in process theory? *Bull. EATCS*, 88:66–83, 2006.

[74] Louis Mandel and Marc Pouzet. ReactiveML: a reactive extension to ML. In *PPDP'05, July 11-13 2005, Lisbon, Portugal*, pages 82–93. ACM, 2005.

[75] Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological cybernetics*, 56(5-6):345–353, 1987.

[76] Jan Midtgaard, Flemming Nielson, and Hanne Riis Nielson. Iterated process analysis over lattice-valued regular expressions. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 132–145. ACM, 2016.

[77] Jan Midtgaard, Flemming Nielson, and Hanne Riis Nielson. Process-local static analysis of synchronous processes. In Andreas Podelski, editor, *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, volume 11002 of *Lecture Notes in Computer Science*, pages 284–305. Springer, 2018.

[78] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[79] Marvin Minsky. *Computation: finite and infinite machines.* Prentice Hall, 1967.

[80] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[81] Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Polynomial-time equivalence testing for deterministic fresh-register automata. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International*

Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK, volume 117 of *LIPIcs*, pages 72:1–72:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[82] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, July 2004.

[83] Ngan Nguyen, Ondrej Strnad, Tobias Klein, Deng Luo, Ruwayda Alharbi, Peter Wonka, Martina Maritan, Peter Mindek, Ludovic Autin, David S. Goodsell, and Ivan Viola. Modeling in the time of COVID-19: Statistical and rule-based mesoscale models. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):722–732, feb 2021.

[84] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Math. Struct. Comput. Sci.*, 13(5):685–719, 2003.

[85] Hélène Paugam-Moisy and Sander Bohte. *Computing with Spiking Neuron Networks*, pages 335–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[86] Carl A. Petri. *Kommunikation mit Automaten.* Rheinisch-Westfälisches Institut für Instrumentelle Mathematik Bonn: [Schriften des Rheinisch-Westfälischen Instituts für Instrumentelle Mathematik. Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ., 1962.

[87] Franck Pommereau. Quickly prototyping petri nets tools with SNAKES. In Sándor Molnár, John R. Heath, Olivier Dalle, and Gabriel A. Wainer, editors, *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, March 3-7, 2008*, page 17. ICST/ACM, 2008.

[88] Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theoretical Computer Science*, 231(2):297 – 308, 2000.

[89] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, 2012.

[90] Davide Sangiorgi and Jan Rutten, editors. *Advanced Topics in Bisimulation and Coinduction.* Cambridge University Press, 2012.

[91] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001.

[92] Olivier Tardieu and Robert de Simone. Loops in Esterel. *ACM Trans. Embed. Comput. Syst.*, 4(4):708–750, November 2005.

[93] **Cinzia Di Giusto**. Additional material. `https://webusers.i3s.unice.fr/~cdigiusto/andy/`.

[94] **Cinzia Di Giusto**, Davide Ferré, Laetitia Laversa, and Étienne Lozes. A partial order view of message-passing communication models. *Proc. ACM Program. Lang.*, 7(POPL):1601–1627, 2023.

[95] **Cinzia Di Giusto**, Davide Ferré, Étienne Lozes, and Nicolas Nisse. Weakly synchronous systems with three machines are turing powerful. In Olivier Bournez, Enrico Formenti, and Igor Potapov, editors, *Reachability Problems - 17th International Conference, RP 2023, Nice, France, October 11-13, 2023, Proceedings*, volume 14235 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2023.

[96] **Cinzia Di Giusto**, Cédric Gaucherel, Hanna Klaudel, and Franck Pommereau. Analysis of discrete models for ecosystem ecology. In Ana Cecília Roque, Arkadiusz Tomczyk, Elisabetta De Maria, Felix Putze, Roman Moucek, Ana L. N. Fred, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies - 12th International Joint Conference, BIOSTEC 2019, Prague, Czech Republic, February 22-24, 2019, Revised Selected Papers*, volume 1211 of *Communications in Computer and Information Science*, pages 242–264. Springer, 2019.

[97] **Cinzia Di Giusto**, Cédric Gaucherel, Hanna Klaudel, and Franck Pommereau. Pattern matching in discrete models for ecosystem ecology. In Elisabetta De Maria, Ana L. N. Fred, and Hugo Gamboa, editors, *Proceedings of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2019) - Volume 3: BIOINFORMATICS, Prague, Czech Republic, February 22-24, 2019*, pages 101–111. SciTePress, 2019.

[98] **Cinzia Di Giusto**, Loic Germerie Guizouarn, and Étienne Lozes. Multiparty half-duplex systems and synchronous communications. *J. Log. Algebraic Methods Program.*, 131:100843, 2023.

[99] **Cinzia Di Giusto**, Laetitia Laversa, and Étienne Lozes. On the k-synchronizability of systems. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2020.

[100] **Cinzia Di Giusto**, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. In Sebastian Maneth, editor, *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Virtual Event, July 19-22, 2021, Proceedings*, volume 12803 of *Lecture Notes in Computer Science*, pages 102–114. Springer, 2021.

[101] **Cinzia Di Giusto** and Jorge A. Pérez. Disciplined structured communications with disciplined runtime adaptation. *Sci. Comput. Program.*, 97:235–265, 2015.

[102] **Cinzia Di Giusto** and Jorge A. Pérez. Event-based run-time adaptation in communication-centric systems. *Formal Aspects Comput.*, 28(4):531–566, 2016.

[103] **Cinzia Di Giusto** and Jean-Bernard Stefani. Revisiting glue expressiveness in component-based systems. In Wolfgang De Meuter and Gruia-Catalin Roman, editors, *Coordination Models and Languages - 13th International Conference, COORDINATION 2011, Reykjavik, Iceland, June 6-9, 2011. Proceedings*, volume 6721 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2011.

[104] Colin Thomas, Maximilien Cosme, Cédric Gaucherel, and Franck Pommereau. Model-checking ecological state-transition graphs. *PLoS Comput. Biol.*, 18(6), 2022.

[105] Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011.

[106] William Waites, Matteo Cavaliere, Vincent Danos, Ruchira Datta, Rosalind M Eggo, Timothy B Hallett, David Manheim, Jasmina Panovska-Griffiths, Timothy W Russell, and Veronika I Zarnitsyna. Compositional modelling of immune response and virus transmission dynamics. *Philosophical Transactions of the Royal Society A*, 380(2233):20210307, 2022.

[107] Michael D Waters and Jennifer M Fostel. Toxicogenomics and systems toxicology: aims and prospects. *Nature reviews. Genetics*, 5(12):936–48, December 2004.

Taking a Walk on the Expressiveness Road

Cinzia Di Giusto

Abstract

This dissertation explores the intricacies of concurrent systems and their formal analysis, aiming at understanding their behaviors and at verifying their properties through formal methods. Concurrent systems, central to modern computing, involve multiple entities working collectively to shape the system's overall behavior. This research delves into the distinctive challenges posed by these systems.
In concurrent systems, the focus shifts from program termination and correctness, typical of sequential systems, to responsiveness at all times. Behaviors become complex, involving interactions, data exchanges, and resource sharing among independent processes or entities. Verification priorities encompass synchronization, communication consistency, and resource management.
The dissertation comprises three interconnected research directions:

1. Process Algebras and Session Types
2. Communicating Automata
3. Applications: demonstrating the practical application of formal methods in diverse domains, bioinformatics, ecology and neural network.

In summary, this dissertation uncovers the complexities of concurrent systems, emphasizing their formal analysis and aiming to identify the boundary between descriptive power and decidability
The dissertation will be organized in chapters discussing my work and classifying them into those three categories: process calculi and session types, automata with memory, applications of concurrent models to various domains. Each chapter will be preceded by a general introduction, specifying and commenting the context and giving pointers to the publications concerning the discussed works.

Keywords: Concurrency, Verification, Session types.