Discrete event modeling and analysis for systems biology models

Hayssam Soueidan

N° d'ordre : 3916

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Hayssam SOUEIDAN**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Discrete event modeling and analysis for Systems Biology models

---

Soutenue le : 4 Décembre 2009

Après avis des rapporteurs :

Hidde DE JONG	Directeur de recherche INRIA
Olivier ROUX	Professeur

Devant la commission d'examen composée de :

Bedreddine AINSEBA	Professeur	Examinateur
André ARNOLD	Professeur	Membre Invité
Gilles BERNOT	Professeur	Examinateur
Hidde DE JONG	Directeur de recherche INRIA	Rapporteur
Macha NIKOLSKI ...	Chargée de recherche	Co-directrice
Olivier ROUX	Professeur	Rapporteur
David SHERMAN ...	Directeur de recherche INRIA	Directeur de thèse
Grégoire SUTRE	Chargé de recherche	Co-directeur

– 2009 –

Abstract

A general goal of systems biology is to acquire a detailed understanding of the dynamics of living systems by relating functional properties of whole systems with the interactions of their constituents. Often this goal is tackled through computer simulation. A number of different formalisms are currently used to construct numerical representations of biological systems, and a certain wealth of models is proposed using *ad hoc* methods. There arises an interesting question of to what extent these models can be reused and composed, together or in a larger framework.

In this thesis, we propose BioRica as a means to circumvent the difficulty of incorporating disparate approaches in the same modeling study. BioRica is an extension of the AltaRica specification language to describe hierarchical non-deterministic General Semi-Markov processes. We first extend the syntax and automata semantics of AltaRica in order to account for stochastic labeling. We then provide a semantics to BioRica programs in terms of *stochastic transition systems*, that are transition systems with stochastic labeling. We then develop numerical methods to symbolically compute the probability of a given finite path in a stochastic transition systems.

We then define algorithms and rules to compile a BioRica system into a stand alone C++ simulator that simulates the underlying stochastic process. We also present language extensions that enables the modeler to include into a BioRica hierarchical systems nodes that use numerical libraries (e.g. Mathematica, Matlab, GSL). Such nodes can be used to perform numerical integration or flux balance analysis during discrete event simulation.

We then consider the problem of using models with uncertain parameter values. Quantitative models in Systems Biology depend on a large number of free parameters, whose values completely determine behavior of models. Some range of parameter values produce similar system dynamics, making it possible to define general trends for trajectories of the system (e.g. oscillating behavior) for some parameter values. In this work, we defined an automata-based formalism to describe the qualitative behavior of systems' dynamics. Qualitative behaviors are represented by finite transition systems whose states contain predicate valuation and whose transitions are labeled by probabilistic delays. We provide algorithms to automatically build such automata representation by using random sampling over the parameter space and algorithms to compare and cluster the resulting qualitative transition system.

Finally, we validate our approach by studying a rejuvenation effect in yeasts cells population by using a hierarchical population model defined in BioRica. Models of ageing for yeast cells aim to provide insight into the general biological processes of ageing. For this study, we used the BioRica framework to generate a hierarchical simulation tool that allows dynamic creation of entities during simulation. The predictions of our hierarchical mathematical model has been validated experimentally by the micro-biology laboratory of Gothenburg.

Keywords: Systems biology, Discrete event systems, AltaRica, Cell ageing, General semi-Markovian processes, Qualitative abstraction

Contents

List of Figures

General introduction

Over the last decade, availability of large-scale technologies and high-throughput experiments have resulted in an unprecedented accumulation of biological data. In its turn, this accumulation of data has increased the gap between human capacity to generate data and that to analyze it. To complement usual "low-throughput" experiments, it is now possible to sequence complete genomes [HKKH91], to obtain gene expression data at the scale of an organism [LW00] and to characterize the possible gene/protein and protein/protein interactions [LRR$^+$02, UGC$^+$00]. This progress in the measurements of rates of molecular mechanisms as well as the characterization of possible interactions between cellular processes implied a paradigm shift in life sciences. Indeed, the growing number of possible combinations of molecular interactions at the system scale results in a range of possible explanations for any observed behavior that far exceeds the human capacity to evaluate them one by one [FMW$^+$03]. In this context, the combination of mathematical biology and computational tools to analyze such systems (for example to *a priori* rule out some seemingly reasonable hypothesis) *in silico* is getting more and more accepted by the cell biologists and neurobiologists community.

Not only our knowledge of possible interactions is increasing, it is also now widely accepted that biological processes can not be seen statically. Models, in their broadest sense, have always been used by biologists, e.g. when drawing a metabolic pathway or when considering a graph of protein/protein interactions. However, we cannot deduce from such static representations alone the function of the process under consideration. For example, although the nutritional stress response of the model organism *E.Coli* and several metabolic pathways of the yeast *Saccharomyces Cerevisiae* have been characterized since decades, we yet to have a clear understanding of how these interactions result in biological functions [DHP04, BRDJ$^+$05]. Biologists emphasize now the need for mathematical and *dynamic* models to describe time dependent molecular processes. Indeed, living cells are inherently *dynamic*: cells grow, divide, *communicate*, move and respond to the evolution of their environment.

Let us consider two examples of biological dynamic phenomena, the electrical activity in cells and the membrane transport mechanism.

Electrical activity in cells is one of the earliest dynamic behavior of cells that has been studied. Hodgkin and Huxley studied action potentials along a giant squid axon in 1939 and modeled it in 1952 [HH52, HH39]. The electrical activity of the squid giant axon has been studied through the response of the axon to a small positive current. As the current propagates along the axonal membrane as a pulse, it results in a single spike of electrical activity (an action potential). Hodgkin and Huxley model is *empirical*, since they used experimental results that were suitable to be represented without consideration of any underlying physiological and cellular mechanisms. In fact, only thirty years after this seminal work, this observed behavior was explained by a "realistic cartoon" of ionic currents in cells [Hil01].

Another dynamic biological process of interest is the membrane transport mechanism. The osmotic balance in cells is maintained by pumping Ca^{2+} and Na^+ ions from the cell and K^+ ions back into the cell. The maintenance of this balance is for example essential to give red cells their characteristic and functional shape.

The two previous biological processes are in fact coupled, and any system level analysis of membrane transport should account for the electrical activity in cells. In fact, Chay and Keizer have shown with a mathematical model in 1983 [CK83] that the increase of blood glucose level induces insulin secretion, which in its turn induces regular bursts of electrical activity in pancreatic cells. The action potentials occurring at the membrane of pancreatic cells are caused by rapid influx of Ca^{2+} and slower efflux of K^{+} that result in oscillations of Ca^{2+} concentration within the cytoplasm.

These two biological processes exhibit characteristics common to most biological processes: they are complex, can be seen at different levels of abstraction, and moreover they exhibit a non linear response and their interactions are essential for higher-level behavior. In order to study such complex networks of processes, the contribution of mathematical models and computing tools has been invaluable to understand their dynamics.

The combination of mathematical biology, experimental biology, biochemistry and computer science in order to unveil the behavior of *biological systems* lead to the "new" discipline of *Systems Biology*. Systems biology is a field that aims at elucidating the relationships between biological components and how these interactions result in complex system-wise behavior. The dogma at the base of Systems Biology is that due both to the presence of non linear responses and to multiple interactions between processes, the behavior of the whole system cannot be reduced to the behavior of its components. Furthermore, as the size and complexity of biological networks increase and will soon cross the threshold of manually understandable models [SRT06], one of the most important challenges for Systems Biology is to develop computational tools that ease the manipulation and analysis of huge systems [SHK+06].

Strikingly, the same complexity also appeared during the last decade in the analysis of engineered systems, and that in the both cases of software or hardware systems. Especially in the case where the system is deemed as being critical, the vast amount of possible interactions between disparate components imposed the need for *formal methods* to help the validation and analysis phase. One striking similarity between Systems Biology and engineering is that both fields are based on the compositional approach. Most biologists now agree that we will never be able to comprehend cellular networks in their whole, due to their "complexity" [SHK+06] and their growing size [SRT06]. As engineers have been faced with a similar limitation, their solution was to decompose large system into subsystems performing distinct, well defined and well understood functions. The reason behind this approach is that a complex system can be rationalized by considering it compositionally.

Composition in biological systems is however inherently different from the composition of engineered systems. First, we do not know how to decompose a cellular network functionally and thus what could be the definition of a biological "module". Second, biological processes can operate at various time scales and be modeled at different levels of abstractions. Thus, a single modeling formalism to represent any biological module seems out of reach within our current conceptions of Systems Biology. However, from the previous observations, any modeling formalism adapted for Systems Biology should be hierarchical, compositional and should allow for different modeling approaches.

Currently the most widely used modeling formalism is certainly that of systems of *ordinary differential equations* that represent the time dependent evolution of molecular concentrations. Under some conditions of homogeneity of the medium and of presence of chemical species at high concentration, systems of ODE have been shown to produce extremely precise predictions, that have been validated experimentally (e.g see the evolution of a cell cycle model

[Tys91b, TKA+00, TCC+04]). Still, not only ODE models are flat, they are also completely deterministic and thus can neither faithfully represent any *stochastic* behavior in cells, nor can they model adequately *discrete timed* behavior. For example, the model of [TCC+04] is a continuous differential equation model, built by accounting for the interactions between molecular species that control the budding yeast cell cycle. Although the mechanisms of regulation of the cell cycle are detailed, the authors completely abstract the cytokinesis process by representing it by a single discrete assignment that divides the mass of the cell by half, and that is triggered at a certain amount of time after a threshold is crossed. This example illustrates that although ODEs are powerful enough to concisely and adequately model non linear interactions between molecular species, they are not adequate to model more abstract processes for which little is known about the underlying mechanism or little is desired to be represented in the model. When considering other type of modeling formalisms, although a wide range of formalisms have been used (Petri Nets, constraint automata, differential equations, stochastic pi-calculus), it is a humbling observation that the most widely used tools in this technically sophisticated field remain for the modeling step PowerPoint and Excel, and for model analysis step Matlab [KLH+07]. We propose to circumvent these limitations by considering *multi-level and multi-formalism models*.

Multi-level and multi-formalism models may be used when different levels of detail and abstraction are possible within a hierarchical system [UDZ05]. That is, when a system accounts for multiple molecular processes, for example happening at different time or concentration scale, the level of details may be adapted in order to account for available knowledge and kinetics information. Existing simulation tools [TKHT04, BBHT06] can account for models where molecular processes involving small concentrations are simulated with a variant of Stochastic Simulation Algorithm. In both cases, exact knowledge of the biochemical reactions is required. In the case where biochemical information is missing (e.g. kinetic constants) some of the dynamics of a metabolic network can still be evaluated using pathway or genome scale approaches such as dynamic flux balance analysis [MED02] or structural analysis [SGSB06].

Accounting for unknown gene functions and/or interactions, is hard [vDMMO00]. We may then consider that the qualitative behavior is essential in order to understand the system as a whole, while the accurate values of the kinetic parameters are of secondary importance. Indeed, the whole research community starts to accept this idea [SHK+06], which means that the time is ripe for moving from simulation only towards more symbolic model analysis methods. Behavioral analysis of models has been extensively studied in the context of formal verification of hardware and software systems. Here, models are usually non-deterministic automata with variables over finite or infinite domains. Formal verification does not face theoretical limitations (i.e. it is decidable) for finite-state models. However, this approach is in practice limited by the so-called state explosion problem: the size of the model's state space, that must be explored by the verification tool, is (at least) exponential in the number of variables. The introduction of symbolic techniques [BCM+92] was a major breakthrough in this field, and nowadays models with a few hundreds of variables can be verified. Approximation techniques have been developed in order to extend the reach of formal verification to larger models, that can be finite-state (but with too many states) or infinite-state, such as automata with (unbounded) integer variables. A wide class of techniques of approximation are based on the notion of *model abstraction*. Model abstraction is based on the following observation: given a property under check, there are surely many details in the model that are not relevant to the property, and these details could therefore be abstracted away. Behavioral properties are preserved from an abstraction to its original model [DGG97], and abstractions can be

computed automatically from the original model [GS97]. Model abstraction is an approximation technique that have recently received a lot of attention since it permits automatic verification of software directly from the source code.

Another advantage of a multi-level hierarchical approach is clear when noticing how biological processes take place at different time-scales. For example, transforming metabolites is fast, while synthesizing an enzyme is slow. If modeled by ODEs, this multi-scale problem leads to the *stiffness problem*. It is the fast time-scaled reactions that are stable, but it is the slow reactions that determine the trajectory of the system. If done naively, a simulation will have to use extremely short time steps (which implies long simulation times) due to the fast time scales. Hierarchy can be seen as a way to overcome this issue. Indeed, simulation of hierarchical systems can be done efficiently by having dedicated solvers for processes at different granularities. Whenever a multi-component system is modeled, a semantics for module composition has to be provided. In particular, a mathematical programming language known as the stochastic pi-calculus allows the components of a biological system to be modeled independently [Pri95]. In particular, large models can be constructed by composition of simple components [BCP06]. The calculus also facilitates mathematical analysis of systems using a range of established techniques. Various stochastic simulators have been developed for the calculus, in order to perform virtual experiments on biological system models.

Objectives and outline of this thesis

Due to the availability of more and more data and possible mechanisms to explain observed emergent behavior, Systems Biology approaches require compositional modeling tools able to cope with multiple formalisms and admitting efficient analysis tools. The challenges under these requirements are both conceptual and computational. The conceptual difficulty is to provide a modeling language that is compositional and expressive enough to reuse existing models. The computational difficulty is to analyze models described in this language, either analytically / symbolically or by numerical simulation. The goal of this thesis is to provide an automata based framework to describe and analyze stochastic processes in the field of systems biology.

The BioRica framework we present and analyze in this thesis can combine different formalisms within a single framework and can efficiently simulate the resulting model. BioRica is a high-level modeling framework integrating discrete and continuous multi-scale dynamics within the same semantics domain, while offering an easy to use and computationally efficient numerical simulator. It is based on a generic formalism that captures a range of discrete and continuous formalism and admits a precise operational semantics.

We first recall the background material and the domain of this work in the chapters Chap. 1 and Chap. 2. The objectives of these two chapters is to define the methodology and the tools already available for the mathematical modeling of biological processes.

In the chapter Chap. 3, we define the BioRica language and its automata compositional semantics. BioRica is extension of the AltaRica specification able to specify non deterministic stochastic automata with timed transitions distributed arbitrarily. More precisely, we will take as a starting point for BioRica the work done on the AltaRica specification language, and specifically that on the AltaRica Dataflow subset([Rau02]), and we will extend its syntax and automata semantics in order to account for probabilistic descriptions. We will then define *composition* operations in order to describe hierarchical systems with BioRica.

In the following chapter Chap. 4, we provide the semantics of BioRica programs in terms of stochastic transition systems and we establish gradually a symbolic computation scheme to compute the formula denoting the probability of a finite path with deadlines in such a stochastic transition system. This analytical scheme is based on the computation of linear combinations of random variables denoting timer values of discrete events. We also show that by using this approach, computations with numerical values are only performed at the very end if needed.

We then consider the problem of simulating a BioRica hierarchical system, and we provide in chapter Chap. 5 the rules and algorithms to compile a BioRica system into a C++ program that simulates the underlying stochastic process. Our approach is based on on the automatic generation of a set of C++ classes mapping the BioRica system hierarchy. We then adapt the variable-time advance simulation schema to non deterministic hierarchical discrete event systems. The approach developed in this chapter were published in [SSN07].

In chapter 6, we provide abstraction based algorithms to build automata based representation of transient behavior of continuous systems and show that high level emergent behavior such as oscillations are preserved by our abstractions. In fact, given a range of possible qualitative values represented as abstraction functions, we show how to compute from the time series generated by numerical integrators a stochastic transition system. This approach has been presented and published in [SSN09].

Finally, we define and analyze in the chapter Chap. 7 a hierarchical and hybrid model of a yeast cells population that explains the rejuvenation effect in symmetrically dividing yeast cultures by a retention factor. To this end, we describe how to incorporate continuous systems into a discrete BioRica system by using composition. From a differential continuous model, we describe a "hybrid system" accounting for cell division, and then build a population model by parallel composition. We then interpret the predictions of this model with literature results concerning damage segregation in asymmetrically and symmetrically dividing yeast cells. The results presented in this chapter were published in [CSS+08].

The Blind Men and the Elephant

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind.

The First approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the Elephant
Is very like a wall!"

The Second, feeling of the tusk,
Cried, "Ho! what have we here
So very round and smooth and sharp?
To me tis mighty clear
This wonder of an Elephant
Is very like a spear!"

The Third approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "the Elephant
Is very like a snake!"

The Fourth reached out an eager hand,
And felt about the knee.
"What most this wondrous beast is like
Is mighty plain," quoth he;
" Tis clear enough the Elephant
Is very like a tree!"

The Fifth, who chanced to touch the ear,
Said: "Een the blindest man
Can tell what this resembles most;
Deny the fact who can
This marvel of an Elephant
Is very like a fan!"

The Sixth no sooner had begun
About the beast to grope,
Than, seizing on the swinging tail
That fell within his scope,
"I see," quoth he, "the Elephant
Is very like a rope!"

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

So oft in theologic wars,
The disputants, I ween,
Rail on in utter ignorance
Of what each other mean,
And prate about an Elephant
Not one of them has seen!

John Godfrey Saxe (1816–1887)

Part I

Background

Chapter 1

From truth to lies: A journey through mathematical modeling for biology

The aim of this chapter is to show what is meant by a model, the nature and objectives of the modeling process and finally to discuss how models and especially mathematical models can be used within a scientific study in the case of biology. In particular, we will try to show how different, competing models represented in different formalisms may be useful. In essence, our objective is to show that new biological knowledge is "created" by the process of abduction, which is itself a logical fallacy. In order to reduce the impact of this fallacy, multiple hypothesis must be confronted on the basis of observations. Consequently, to bridge between observations and explanation or prediction, models are a fundamental (if not the only) way to test a hypothesis.

1.1 What is modeling? What is a system?

Systems and models are primitive concepts (like those of a set or a structure) that are used with so many meanings and in so many domains that it might be impossible to give an exact, consensual definition. Although their understanding could thus be left to intuition alone, we nonetheless provide some (in our sense) representative definitions found in literature.

- A system is a regularly interacting or interdependent *group of items forming an unified whole* (Webster).
- A system is a *combination of components that act together* to perform a function not possible with any of the individual parts.(IEEE Standard dictionary of Electrical and Electronic terms).
- A model is a *simplified description of a system* or process, or even of another model. The most common uses of models are to assist calculations, predictions and design choices.
- model (n): *a miniature representation* of something; *a pattern* of something to be made; *an example* for imitation or emulation; *a description* or analogy used to help visualize something (e.g., an atom) that cannot be directly observed; *a system* of postulates, data

and inferences *presented as a mathematical description* of an entity or state of affairs [Dym04].

As it is usually pointed out for example in [Hae05], these definitions of model and systems imply that virtually everything can be understood as a system and that models are present in every facet of human activity.

Nevertheless, what can be observed from these different definitions is that a system is made of interacting components, and that a model is an abstraction of a system. But more importantly, these definitions suggest that modeling is a cognitive activity in which we think and make models to describe how "real life" systems, processes or objects behave, by accounting for their components; and that we use models to communicate a view of the world.

1.1.1 Scientific method and modeling: the principle of abduction

Scientific method may be rudimentarily outlined as opposing two a priori distinct worlds, a "real world" and a "conceptual world". In the real world, we *observe* various phenomena and behaviors, whether produced artifacts or natural. On the contrary, in the conceptual world we try to understand and explain what is going on in that real, external world.

Dym and Ivey [Dym04], describe the scientific method as a body of techniques existing in the conceptual world in order to reason about the real world. They identify three activities that take place in the conceptual world when faced with a phenomenon: (1) we gather observations, (2) we analyze these observations by means of models, (3) we use the results of the analysis to describe precisely the phenomenon, to predict what would happen in a yet to be conducted experience or to or explain the phenomenon. Let us discuss the mode of inference used in this scientific method with holding to section Sec. 1.2 the details of these three possible uses of mathematical models. The base of the scientific method is the use of empirical data to form new body of knowledge. The mathematician Charles Sanders Peirce in "Deduction, Induction, and Hypothesis" (1878) (see [Nii99]) proposed different modes of inference proper to the scientific method and that explain how empirical evidence can be used to form theories. Peirce compared *deduction*, *induction* and *abduction*. Consider the following syllogisms:

Rule All philosophers wear a toga,

Case These humans are philosophers,

Result Therefore these humans wear a toga.

The result is obtained via deduction. On the contrary, induction can be used to explain how hypothesis and consequences are related.

Case A random sample of philosophers is selected.

Result All these philosophers wear a toga.

Rule All philosophers wear a toga.

However, experimental methods in a science like biology (and thus its use of models) rely on the use of abduction.

Rule All the philosophers wear a toga.

Result These humans wear a toga.

Case These humans are philosophers.

Indeed, the goal of abduction is to explain a "surprising" result. The surprising result is: seeing a group of humans wearing a toga. We must then seek an hypothesis that would make this surprising result not surprising. It is true that if these humans are philosophers then they would wear a toga, hence there is evidence to suspect that these humans are philosophers. By the observation of a surprising result, we seek a hypothesis that would explain this surprising result (and thus make it less surprising). Abduction thus seeks the most plausible hypothesis that would explain a result, given accepted knowledge.

Note that abduction is a logical fallacy. Let a and b be two logical propositions. Deduction allows us to derive b as a consequence of a. Induction allows us to derive $a \Rightarrow b$ by a set of samples of a and b. Abduction allow us to *infer* a from the observation of b. Abduction is the *post hoc ergo propter hoc* fallacy (affirming the consequent) since we do not account for all the possible explanations of b. In order to reduce the impact of this logical fallacy, we would have to consider all the possible explanation for an observed phenomenon, which is clearly impossible.

1.1.2 From abduction to soundness: Chamberlain's multiple hypothesis testing

Abduction is the only possible means to form a new body of knowledge by observation, but it is a mode of inference that is inherently wrong, since all the possible explanations of the observation are not accounted for. However, by encompassing a wide range of possible explanations, one can hope to compare multiple explanations on the basis of empirical evidence. In order to cover and on the same time to limit the universe of explanations, we are forced to simplify our view of reality and to keep only what we think is relevant for the observed phenomenon. In other words, we build models, and the possible explanations of an observed phenomenon are confronted by the use of multiple, competing models.

Thomas Chamberlain (1843-1928) advocated in a series of papers [Cha31](as cited by [And08]) that scientific investigation should focus on a set of alternative plausible scientific hypotheses that must be confronted. Each plausible hypothesis forms an explanation that should be evaluated with respect to its empirical support. In this sense, the core objective of the modeling process is to provide systematic and mathematical tools to aid in this evaluation.

As an example of a carefully shaped set of plausible hypotheses, Anderson [And08] highlights the process and methodology Caley and Hone [CH02] followed while they were studying bovine tuberculosis transmission in feral ferrets. Before performing any data collection, Caley and Hone derived from literature review 20 hypotheses relating the instant of transmission with the age of the animal. As an example we provide here 5 of their hypotheses ([And08]):

- Transmission occurs until the $1.5 \sim 2.0$ months of age, via suckling, from mother to offspring,
- Transmission occurs from the age of 10 months, via mating or fighting, during the breeding season,
- Transmission occurs from the age of $2 \sim 3$ months, via routine social activities (e.g. sharing dens),

- Transmission occurs from the age of $1.5 \sim 2$ months, via hunting,
- Transmission occurs from birth, via environmental contamination.

Each of the hypotheses targets at least two variables: the age of transmission and the medium of transmission; furthermore, these hypotheses are not mutually exclusive. Thus, evaluating and comparing each hypothesis with respect to empirical support should answer questions about the "how" and "when" of contamination (i.e. "What is the size of the effect?"), which should be contrasted with the "what" of contamination (i.e. "Is there an effect?").

From Anderson's point of view, which follows closely Chamberlain's and Francis Bacon's approach, the usual paradigm of investigation based on null hypothesis testing results most of time (Anderson is in the field of ecology and population biology) in the rejection of a faulty and hardly plausible null hypothesis. Anderson concludes the tuberculosis example by challenging the reader to devise a plausible null hypothesis and ironizes by suggesting that H_0 should be one of : "No transmission occurs", "Transmission is random", "Bears do no go in the woods".

Given a phenomenon, multiple models must be assessed. Although Anderson's work aims exclusively at providing tools and strategies for confronting multiple hypotheses in the field of statistical modeling, our approach of systems modeling starts with the same principle: each hypothesis should be supported by empirical data via the use of a model that is specific to the hypothesis under scrutiny. Since we need multiple models, one of the objectives of this thesis is to provide a modeling language that is able to describe models of different nature and to provide a mathematical interpretation of these models.

1.2 Utility of mathematical models

General definitions of models all agree on the fact that models represent reality with some required degree of approximation. Given that models are approximations and thus inherently "wrong", why is this approximation required? What is exactly the purpose of the modeling activity?

The general objective of the modeling process is to manage complexity. The overall complexity of technical systems is increasing. In several technical fields - telecommunication, information systems, logistics - the number of components involved in an object under design makes it impossible to manage and predict the global effect of local decisions [Zim08]. Similarly, as our understanding of living systems increases, we witness an increase of complexity as we wish to understand how the possible interactions between thousands of molecules form the basis of living organisms and their functional properties. In both of the technical and biological fields, models offer a way to deal with this complexity. We will now consider in more detail the attributes of this complexity in biology, then characterize the possible purposes of models.

1.2.1 Models as means to deal with biological complexity

The nature of biological systems we wish to model in systems biology or in physiology are characterized by their complexity [CC08]. Yates [Yat78] identified five features of biological processes that give birth to their complexity: the number of components, their interactions, non linearity, asymmetry and nonholonomic constraints.

The number of components in the system is immediately related to its complexity. But Flood et al. (1993) refined this point of view and have shown that it is from possible interactions between components that complexity emerges.

Nonlinear systems occur when one element of the system varies in a non linear manner with respect to another. Non linearity is ubiquitous in biological processes [CC08]. However, from a modeling perspective, it can be reasonable to consider systems as being linear under some conditions.

Asymmetry in organisms is fairly usual and is a key property of cellular processes such a cellular differentiation. Without the presence of asymmetry in one of the processes of growth, the successive divisions of a single cell would yield a population of identical cells and no specialized organs would emerge.

Holonomic constraints are global constraints that are expressible as a function of the state of the system and time. A system is said to be holonomic if all of its constraints are holonomic. Nonholonomy of biological processes appears whenever part of a system exhibits local regulation and control. At the unicellular level, consider for example glucose metabolism and cell cycle regulation of yeast cells. Even if these two processes are related (since they are parts of the same system) and that they interact indirectly, there is no global constraint that can express both processes, or equivalently, one of the processes as a function (in the mathematical sense) of the other. Nonholonomy implies that some parts of a system exhibit a high degree of freedom, and it is this absence of constraints that implies complexity. This implies that the state of the system is not a function of the state of some of the parts of the system.

In other words, complexity arises in biology through the nature of interactions of the components of a system where non linear effects and control appear at multiple organizational levels. As a consequence of this "real-life" complexity, it is often not possible to measure *in vivo* with sufficient precision the values of the quantities of interest, since they may be at the molecular level. Often, only indirect measures about the quantities of interest may be obtained, implying at least the need for some model to be able to infer the value of the quantity of interest.

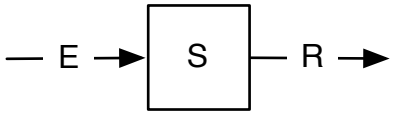
1.2.2 Use of mathematical models

> *Model[er]: A device for turning assumptions into conclusions Schimel (2002) (as cited by [Hae05])*

The degree of detail that is incorporated in a model is determined by its purpose. Karplus in 1983 [Kar83] provided a conceptual framework to define the possible use of a mathematical model. In Karplus's framework, a model is a triple $\langle S, E, R\rangle$ where S is a black box system with an input E and an output response R (see figure Fig. 1.1)

Considering all the possible pairs of elements of $\langle S, E, R\rangle$ as problem statements, the use of the model is to characterize the third missing component.

- If E and R are given, the modeling process aims at determining the system S. By inferring the system given its input and output, we solve a synthesis problem that arises whenever we want to use a model to understand a process.
- If E and S are given, the modeling process aims at characterizing the output response R. By using the knowledge of the component (modeled as the system S) and available



Type of Problem	Given	To Find	Uses of models
Synthesis	E, R	S	Understand
Analysis	E, S	R	Predict
Instrumentation	S, R	E	Control

Figure 1.1: Systems and the uses of models. Top: General system presented as a triple $\langle S, E, R\rangle$ relating the input E to the output R by the mean of a model S [Kar83].

experimental data (the input E) we solve an analysis problem that arises whenever we want to use a model to predict the response of the system.

- If S and R are given, the modeling process aims at finding an input E such that the response of the system S is R. After describing some degree of freedoms in S, we solve a control problem that arises whenever we want to use a model to instrument the system.

Mathematical models can also be used to coordinate research, summarize data, detect logical inconsistencies and perform scenario based testing. By giving a single and non ambiguous interpretation of a system in a mathematical language, large organizations (teams, companies) can coordinate experimental research and communicate upon a common ground. Summarizing experimental data in terms of a mathematical model can be used when a large set of experimental data needs to be compared, classified or searched. For example, a linear regression on the model of the equation $y = ax + b$ can summarize a set of data points in terms of two parameters a and b. Logical inconsistencies between hypothesis, experimental conclusions and assumptions can be be detected for example by using a logical language to specify each item. For example, Kam et al. in [KKM+08, KHK+04] modeled all experimental results related to the cell fate acquisition of the worm *C. Elegans* using the formalism of *live sequence charts*. Finally, a model can be used to perform scenario based testing where multiple models are compared in order to elect a model to concretize.

1.2.3 Misuse of mathematical models

Levins[Lev66] identified three properties of models related to the purpose of using a model: realism, precision and generality.

Realism is a property of the model structure characterizing the degree to which the model simulates the real world. Physical models of airplanes used to interpret results of wind tunnel experiments often include the maximal level of detail possible.

Precision is a property of the model response characterizing the degree to which the model's prediction is the same as the observed real world response.

Generality is a property of the model characterizing the degree to which the model is applicable to a large number of systems.

Levins argued that these three properties cannot be maximized by the same model, and that at most two of these properties may be maximized in a single model. For instance, maximizing realism cannot be performed without trading off for generality.

However, Orzack and Sober [OS93] argued that Levins argument should be clarified, since he does not define exactly the terms realism, precision and generality in his original work.

Orzack and Sober thus refined Levins thesis and have shown that when a model A is a special case of another model A', there is no trade-off between the three properties. They claimed that for any model with n independent variables, we can add an additional variable and obtain a model that is more realistic, precise and general that the first one.

Models are tools used to solve a specific problem, and nothing in the model itself forbids an inappropriate application. Holling [Hol78] and Karplus [Kar77a, Kar77b, Kar83] especially discussed the problem of the use of quantitative models and illustrated how highly precise models can be used in domains providing poor data or small understanding of the domain's dynamics and mechanisms.

Karplus went further and identified domains in which the poor quality of the available data and shallow understanding of the mechanisms led to inappropriate models. He positioned social science, economics and political science at the lower end of applicability of models. As an example, Karplus cited Jay Forrester's World Dynamics model which, more or less, accounted for everything [For71]. Essentially, inappropriate use of models is one of the consequences of the possibility to simulate (or solve) quantitative models with an arbitrary number of digits, and that regardless of the accuracy of the premises used to build the model. Thus, although the results are extremely precise with respect to the model (in the mathematical sense), this precision can be semantically skewed towards the precision of the model relative to the system, via an effect that Haefner [Hae05] qualified as "numbers which often acquire a reality of their own". Such semantic drift may be harmless, at best.

1.3 How do we model?

When using mathematics to model, two separate approaches to modeling are possible depending on the intended use of the model. These two approaches are data driven models and system oriented models.

When models are solely based on experimental data, we can apply a data driven modeling process that aims at building black-box models. In essence, data-driven models describe an input output relationship between experimental data collected on the system. Data driven models are mathematical descriptions of data, and have only an implicit correspondence with the underlying biological process [CCB01].

The second modeling approach is system oriented modeling, with the goal to represent the underlying biological process explicitly, up to a level of approximation and resolution dependent on the intended use of the model and on the availability of a priori knowledge and assumptions. Once the degree of approximation has been chosen, a mathematical formulation can be built.

An important decision at the first step of the modeling process is to decide the level of abstraction of the model. Deciding on the right level of abstraction is equivalent to deciding on what should be part of the model and what should be part of the environment. This step requires careful thinking, since by identifying initially what should be accounted for in the model, we also immediately delimit its application and purpose.

The result of this first step is a *conceptual model*. The conceptual model is usually not formalized and covers all the assumptions and approximations that are made before building the mathematical model. In [CCB01], the authors identify three classes of possible approximations: aggregation, abstraction and idealization. Approximation by *aggregation* considers a complex structure as an unified entity by lumping together its constituents. For exam-

ple, the proteins in a mitochondria may be lumped together in five classes defined by the proteins' function, or conversely they can be carefully modeled by accounting for the exact concentration of each possible protein present in the mitochondria matrix. Approximation by *abstraction* matches a complex structure to its functional role. A mitochondria may be abstracted as an ATP producing unit. Approximation by *idealization* aims at simplifying dynamics that are not considered as pertinent. For example, the response of a cell population to oxidative stress implies the mitochondria of each cell. The response of the mitochondria may be modeled either as being instantaneous and affecting all the mitochondria or modeled by accounting for the dynamics of the diffusion.

Once enough approximations are described in the conceptual model, the conceptual model is translated into mathematical equations describing the relationships between the *variables* of the model. A general hypothesis we have to apply to the system we wish to study by using mathematical modeling is to consider that it is always in some state. A state of a model fully describes the quantities we wish to measure with the model. This implies that the first step when translating the conceptual model in mathematical equations is to define the set of measurable variables that are chosen and associated with a given system.

Once state variables are defined, a general way to describe the behavior of a system is to describe the structure of its state as well as how the system changes its state over time.

We will now introduce a simple classification based on the mathematical structure used to represent the variables, the time and the dynamics.

1.3.1 Mathematical model formulation

A simple basis for the classification of models can rely on the structure of the underlying mathematical theory used in the model. We consider that models can be:

- *Process oriented* when the mechanistic processes are explicited in the model. Otherwise, we say that the model is descriptive and *phenomenological*.
- *Dynamic*, when the state of the model evolves over time. Otherwise, we say that the model is *static*.
- *Continuous time*, when the evolution over time is supported by a continuous variable. Otherwise, we say that the model is *discrete time*.
- *Continuous*, when the possible configurations of the model are real valued. Otherwise, we say that the model is *discrete*. For models over a discrete state space, we further distinguish between finite and unbounded state space.
- *Non deterministic*, when the model is under-specified and allows for a set of possible executions instead of a single execution. Otherwise, we say that the model is *deterministic*.
- Stochastic, when an execution of a model depends on random variables. Otherwise we say that the model is *non random*.

This classification of models serves as a methodological guideline during the modeling process.

We will review in the next chapter 2 instances of the previous classes and give an overview of the analysis that can be performed on them.

In this thesis, we consider models that have a discrete state space, that evolve under continuous time, by means of instantaneous transitions with random fluctuations. This choice is backed up by the following reasons. The type of the state values and of passage of time is not a property of the system under consideration, but a property of the model we want to use to gain a better understanding of the system. This implies that this choice is independent of the question whether real systems are inherently continuous or not. When the state space is discrete and when time evolves continuously, the dynamic behavior of a model is described by timed activities. The start of a timed activity is triggered by a precondition on the state of the model, and their finishing events happen after some delay. Once the finishing event happens, the model evolves to its next state, where subsequent activities may start. By simply modeling the causality of activities within a model, one can use this model to answer qualitative questions such as whether a given state of the model can ever be reached. However, when quantitative properties are of interest, the model has to include additional information. As we saw, the environment and the inner part of a model are subject to uncertainty. The process of modeling itself relies on abstracting some part of the system. This implies that whatever the level of abstraction chosen for the model, there will always be some level of detail that is not part of the model. This results in events that are unpredictable at this level, and that are therefore modeled as being random. Note that this is independent of the question of whether there is true randomness or not. For example, conflicts between activities or events are a source of unpredictable behavior. Such conflicts can occur whenever two activities that are competing may alter the system's state. Whenever the outcome of a conflict is not known at the level of abstraction chosen for the model, we can still describe how the outcome should be distributed by using probabilities. Another source of uncertainty in modeling arises when the exact time of an event is unknown. In this situation, we can consider the delay of an event as a random variable following a given probability distribution function.

1.3.2 Hierarchical systems: from molecular to systems biology

> Systems biology is a new field of biology that aims to develop a system-level understanding of biological systems. [Kit01].

The objective of systems biology is to relate functional properties of whole systems with the interactions of their constituents (Alberghina, 2005). The system's behavior is often considered as an emerging behavior, that is a qualitative property that emerges from non linear interactions between components. This emergent behavior is often important for the survival of living organisms. The classical example is oscillation in networks of components that would not oscillate on their own [GGH$^+$01]. This implies that living systems should be studied at multiple levels, between the molecular level and the systems level, and that we should explain the behavior of the whole system as emerging from the behavior of its constituents. According to [BBHW07], this leads to an apparent paradox inherent to systems biology: living systems are composed of nothing but molecules, but the functional properties of living systems cannot be understood by molecular biology alone. In essence, the paradox can be resolved by stressing the word "understood". Bruggeman et al., 2002 advocate that there is no systematic choice between a purely reductionist approach (microscopic "nothing but" statements) and an antireductionist approach (macroscopic "all or nothing") approach, but rather a pragmatic choice of a middle position, with the aim to *explain* the system's behavior in terms of its functional organization.

In order to explain the emerging behavior of complex living systems, hierarchical decomposition is a modeling practice that decreases the complexity of the task. The simpler hierarchical structure of a system is based on the "part-of" relationship. For example, our perception of the human organism can be described in such terms: from genes, through cellular subsystems, cells, organelles and organs to the complete physiological organism [KS98]. In fact, this can be said about systems in general, as hierarchy is deeply ingrained in the way we perceive nature and thus in the way we model it as a system:

> Whether nature is truly organized hierarchically is moot. Men's perception of nature is hierarchical.[Web79]

By decomposing a system into its components, a top-down or bottom-up approach can be applied to the modeling process, and this increases the "manageability" of a complex system [KE03]. However, as implied by the definition of the systems biology approach, the behavior of the whole system can not be expressed as a linear function of the behavior of its constituents, and thus a model of a system must account for non linear interactions.

Not only the complexity of the modeling task is decreased by decomposing a system, this decomposition is essential in order to exhibit regularity and patterns. When a system is decomposed in its parts (for a given relationship), it offers the possibility of detecting similarities between two different systems. As an example, the stress response circuitry of a prokaryotic organism *Escherichia Coli* is present (albeit slightly modified) even in eukaryotic cells [Kit01]. This implies that the model of the heat shock stress response in *E.Coli* may be reused to build a more complex machinery adapted to eukaryotic cells. By decomposing the *E.Coli* cell as a system of interacting networks, some networks may be uncovered to be part of other systems.

Hierarchical modeling aims at decomposing a system into components that have enough local regulation and control to be analyzed in isolation and such that they produce emergent effects when analyzed within a system. As such, hierarchy is not a characteristic of a system, but a characteristic of the modeling process used to model the system. When a system is analyzed, the objective is to explain the behavior at any level in terms of the level below, and how this behavior explains the level above.[Web79]

1.3.3 Randomness in biology

The role of random fluctuations in biology can be seen in three conditions: limitations of the mass action modeling assumption, accounting for uncertainties that concern the experimental data and finally as a necessary condition for explaining some biological patterns, like gene expression or neuron firing.

In biochemical kinetics, the *law of mass action* states that the rate of a single reaction is proportional to the product of the concentrations of the reactants. This law is a key assumption when deriving a set of differential equations from a network of biochemical reactions (see 2.7.2 for an application and a discussion of this assumption). However, the stochastic approach to biochemical kinetics is by far the methodology that is the most justified by statistical physics considerations [Wil06]. Indeed, the conventional approach of modeling biochemical reaction networks by using continuous, deterministic rate equations can not be applied to systems with multi-protein complexes or to systems where the dynamics depend on a species that is present in very low concentrations [Kit01, Gil77]. For example, behavior induced by signaling pathways is known to be sensitive to operations and reactions that happen for a small number of molecules [GHLG03, Hal89]. For neutrophil signaling pathways, the study of Hallet

[Hal89] showed that only the behavior of 200 K^+ and Na^+ ionic channels is responsible for the intracellular concentration of Ca^+.

Probabilistic models can be used *in the modeling methodology* and we will argue that this is particularly useful in the case where a biochemical model covers a range of possible behaviors and where little is known about the parameters used in the derivation of ODEs from biochemical networks. The translation of a biochemical reaction network into a deterministic continuous equation requires at least the rate laws for all parameters. Once rate laws have been determined, the parametrized system obtained by the application of the mass action law can be studied, analyzed and solved. However, mass action kinetics fails to correctly capture the saturation of enzymes, which is a common phenomenon. In order to account for enzymatic saturation, more realistic kinetics can be applied (e.g. Michaelis Menten) which requires additional parameters (e.g. the dissociation constant) and additional knowledge (functional form of the enzyme kinetics). But this approach is severely hampered by the the lack of available enzymatic data, either concerning the value of the kinetic parameter or the functional form of the enzymatic rate equations. In the best case where both are available, parameter values may have been obtained under different experimental conditions (temperature, physiological conditions) or for different strains or tissue type, which leads to thermodynamic errors when used in the same model. To overcome this limitation, a probabilistic kinetics that accounts for thermodynamic constraints has been proposed in [LK06b, LK06a]; [SGSB06] proposes a statistical exploration of the comprehensive parameter only requiring stoichiometric information. In both approaches, authors consider the parameters as being random variables and they both provide techniques to characterize the *plausible* distribution of the parameters, thus leading to models that are not random, once initial conditions and parameter values have been selected by a randomized mechanism.

Finally, the behavior of a system can be explained by a high variability in its constituents and this variability can be easily modeled by random variables. In the field of neurobiology, several studies outlined that the pattern of firing of individual nerve cells, and most importantly the range of possible patterns in a population of nerve cell, is explained by a probabilistic gating model of voltage-dependent ion channels[WRK00, SZ96]. In the case of black box models, data models using stochastic functions have been shown to be essential to capture *important* unmeasurable disturbances in the field of neurophysiology [MN72]. Stochastic fluctuations are also abundant in gene regulation mechanisms, and McAdams [MA99] has shown that some regulatory mechanisms do rely on variability induced by low intracellular concentrations.

1.4 Model (in)validation for biology

Model validation is a problem specific process where we compare a set of competing model by measuring their validity with respect to their intended purpose [CCB01] and deduce their domain of validity. In [CCB01], the author states that within a necessary domain of validity, the output of the model should "correspond sufficiently" to the experimental data, and he further states that "an acceptably small difference" is usual. This approach can be facilitated for parametric models by parameter estimation tools such as [ZK06]. Basically, given a set of data and a parametric model, tools can reduce the least square difference between a numerical solution of an ODE and the initial data.

For models built using experimental data, Anderson [And08] emphasizes the need to define

the impact of stochasticity in the initial data and thus the need to formalize any statistical assumptions before comparing the output of the model with the experimental data. If the purpose of the model is to have a perfect correspondence with the experimental data, it is always possible to perform a fit by using high order Fourier series or polynomial terms. Goodness of fit of n data points is always perfect when we use a polynomial of order n. This implies that the objective of the modeling process is not to fit the *data*, but the *information in the data*. Thus, defining for a specific modeling task what is the "signal" and what is the "noise" should be equivalent to the definition of the model. The model is supposed to parsimoniously represent the information contained in the data.

In the case of Chamberlain's approach, the goal is to build a one-to-one correspondence between the set of hypothesis and the set of competing models. We loosely used the notion of "a set of competing models" up to now and we now clarify what it means for two model to be competing and link this notion with parametric models. In order to have two competing models, the notion of difference between models needs to be defined. As we saw in section Sec. 1.3.1, different mathematical formalisms may underly the model and we can safely assume that two models described in different formalisms are different. However, for models in the same formalism the notion of different models can not be based on structural differences, but should be refined to account for parameters. Consider for example the logistic growth equation

$$dN/dt = rN(1 - N/K)$$

As N decreases to 0, the limit of this model is the exponential growth equation

$$dN/dt = rN.$$

Although the exponential and logistic growth models are usually considered as being "different" since they exhibit different equations and structure, the exponential model is the limit case of the logistic growth model. In other words if the parameter N is free, the logistic and exponential model may be equivalent. Thus, the non equivalence between two (parametric) models can only be evaluated once the parameters are set; the structure and equations of the models can not be used as a basis. Even in the case of more complex models, such as the one obtained when deriving ODEs from a biochemical reaction network, the limit case when one of the parameter is 0 is a structurally different model. Thus, when building the set of competing models, each hypothesis should be mapped to a parametric model and its parameter values. In other words, model identification and parameter estimation are exactly the same process.

In the case where validation relies on experimental data, we are facing the task of selecting among a set of competing models (or equivalently competing hypotheses), the "best model". Anderson further recommends that one restate the problem as follows:

> The issue becomes the *evidence* of each model, given the data.[...] So now we may ask if hypothesis C is 10 times as *likely* as hypothesis A ? *(from [And08])*

The theory presented in detail in [And08] defines exactly what is a *good* fitted model, given the data. Anderson provides statistical tools that can be used to rank the models by using the Kullback-Leibler information principle and its restatement by Akaike in terms *entropy maximization principle*. The beauty of Akaike's information criterion is that it formally links the Kullback-Leibler principle, Boltzman entropy and Fischer maximum likelihood [And08]. By using this criterion as a basis for model ranking and estimation of the quality of models,

Anderson provides a framework where quality of models is made relative to competing models and in which parsimony and goodness-of-fit are accounted for. Contrary to the approach of Levins and of Orzack, in Anderson's framework blindly adding free variables and parameters does not provide a better model.

In the case where validation can not be performed against experimental data, validation must be made on the basis of qualitative results. The same biological system is described in multiple formalisms. Indeed, as we saw in section Sec. 1.3.1 , there is a wealth of possible formulations, and except when a formulation requires assumptions that are obviously false for the modeled system (such as the assumption of high number of reactants for the law of mass action), the same system can be described by multiple models, each having their own set of assumptions and specific abstractions.

Richard Levins [Lev66] suggested determining whether the predictions of the model result from its specificities or from the details in the simplifying assumptions, by the use of a technique he called *robustness analysis*. Robustness analysis is defined as searching for common predictions among multiple models of the same system [WR08]. In the case where the same result holds for each of these models, Levins qualifies the result as a *robust theorem*. Weisberg and Forber [WR08] applied this robustness analysis to the Lotka-Volterra model of prey predation (that we study in more details in section Sec. 2.7.2). The author used four different models: one is the original formulation in terms of ODEs, the three others being stochastic models based either on population dynamics, individual based dynamics, or density dependent individual based dynamics. They explored each model in order to validate the following property

> Ceteris paribus, if the abundance of predators is controlled mostly by the growth of the prey and the abundance of the prey controlled mostly by the death of predators, then a general biocide will increase the abundance of the prey and decrease the abundance of predators.

The authors have shown that this property holds for all their possible model formulations, while other simpler properties don't (stating e.g. that the average number of predators and prey corresponds to a stable equilibrium point). The authors refer to [CD89] for an illustration of a "real life" application of this principle: by using the DDT pesticide as a treatment against *Icerya purchasi*, a scale insect, orchardists in fact have aggravated the pest invasion since the insecticide has also killed the pest's main predator.

1.5 Concluding remarks

In this chapter we have defined the terms and concepts that underly the work done during this thesis. We gave an overview of how the modeling process may fit in the scientific method, why models are required for biological sciences, what are the inherent limitations of models, and provided some guidelines for the modeling process and for model validation.

In summary, mathematical models are abstractions that *depict reality*. When the experimental data is present, the use of mathematical models can be seen as that of maximizing the information that can be gained by measurements, by a formalization of the impact of noise and random fluctuations. When measures of quantities of interests are not available, mathematical models can be used to *infer* plausible values from the experimental data. In the extreme case where experimental data is not available, mathematical models may still be used to exhibit general principles about the behavior of complex systems.

In any case, models are means to increase our understanding of complex systems, by the relying on approximation. In this sense, models are never "true", they always reflect only the parts of reality that we conceptually think are relevant to explain the observed behavior. To have a true model, such as a perfect model for the whole cell, would require an extraordinary quantity of experimental data to estimate its parameters, and an extraordinary amount of time to understand and interpret its output behavior.

In order to build hierarchical systems, models should not be reused directly without interpretation or further approximations. Model reuse is a major research effort in modeling and simulation [UDLK05]. In the field of systems biology, the complexity of the modeling process makes reuse of existing models extremely appealing. The challenges currently faced by the systems biology community include the interpretation of such composed models, the technical interoperability of the corresponding simulation systems, and modeling and simulation that accounts for multi-formalisms and hierarchical systems . This problem has been identified as a grand challenge in the systems biology community[SHK$^+$06]. As illustrated by the BioModels database [LNBB$^+$06], the output of the modeling process is often reduced to its mathematical artifact, namely the model itself, without all the assumptions, experimental data, conceptual considerations and validation methods that have been used to elect a candidate model as the "true" model. Even if technical and mathematical tools provide a framework where model composition is possible (and this thesis indeed provides one of them), a complex model built by reusing existing models that have been obtained with different assumptions seems hardly plausible.

Chapter 2

Modeling formalisms

The aim of this chapter is to present the preliminary notions and modeling formalisms used thorough this thesis. After introducing the discrete model of finite state automata and of transition systems, we recall the definitions of *constraints automata* and their extension that serve as a basis for the semantics of the AltaRica formalism. Afterwards, we recall some definition of probability theory and present two widely used stochastic processes class: *discrete time discrete state Markov chain* and *continuous time discrete state Markov chains*. We then finish this preliminary section by presenting the notion of *discrete event systems* and by introducing model formalisms widely used in mathematical biology.

2.1 Discrete and finite models

2.1.1 Finite Automata, transition systems

The first model of behavior we present is the model of an *automaton*, also called a *finite state machine*. An automaton is a directed graph with labeled edges. Each node of the graph is meant to represent a possible state of the system, while edges represent *transitions* that are discrete steps that can change the state of the system. A transition has a source and a target state. The source state represents the state before "executing" (or "firing") the transition, and the target state is the state reached after executing the transition. A self explaining example is given in Fig. 2.1.

```
In[17]:= Q = {"Light off", "Light on"};
         Σ = {"switch"};
         rel = {
             {"Light off" → "Light on", "switch"},
             {"Light on" → "Light off", "switch"}
           };
         GraphPlot[rel, DirectedEdges → True, VertexLabeling → True]
```

Figure 2.1: The structure of a simple two state automaton with labeled edges, displayed as a graph over Q.

Automata are probably the most used formalism to model the behavior of discrete sys-

tems. See, for example, the automata representation of programming languages semantics [Plo81, Win93], and model checking [QS82, CE81]. By using automata as the mathematical representation of the possible behaviors of a system, model checking algorithms and techniques based on relations over automata such as language inclusion [HMU06] or bisimulation [Mil82] can be used to *validate* the compliance of a system with respect to a set of behaviors described using an appropriate logic (e.g. LTL, CTL [CE81, Pnu77]).

While the structure of an automaton is always described as a graph structure, automata are usually *decorated graphs*, giving rise to many different subclasses. To begin with, we recall the definition of *labeled transition systems* that are automata whose edges are labelled with *events*.

Definition 2.1 (**Labeled transition system**)**.** A *finite labeled transition system* (LTS) is a triple $\langle Q, \Sigma, \rightarrow \rangle$ where

- Q is a finite set of states $\{q, q', q_0, \ldots, q_n\}$ called the *state space*,
- Σ is a finite alphabet of events $\{e, e_0, \ldots, e_m\}$,
- $\rightarrow \subseteq (Q \times \Sigma \times Q)$ is a (finite) set of labeled transitions.

To simplify the notations, a transition $(q, e, q') \in \rightarrow$ is written $q \xrightarrow{e} q'$. For a given LTS $\mathcal{L} = \langle Q, \Sigma, \rightarrow \rangle$, we write $q \not\xrightarrow{e} q'$ if $(q, e, q') \notin \rightarrow$.

The LTS structure $\langle Q, \Sigma, \rightarrow \rangle$ from the example in figure Fig.2.1 is $Q = \{\text{“Light on”}, \text{“Light off”}\}, \Sigma = \{\text{switch}\}$ with the transition relation defined by

$$\rightarrow = \{(\text{“Light on”}, \text{“switch”}, \text{“Light off”}), (\text{“Light off”}, \text{“switch”}, \text{“Light on”})\}.$$

In order to introduce some terminology, let us consider the example of figure Fig. 2.2.

In the LTS defined in Fig. 2.2, it should be noted that not all events are enabled in every state. For example, there is no outgoing transition from the state y labeled with the event c. We say that the event c is *inactive* in y. Furthermore, note that in state x, there are two outgoing transitions labeled with the same event a, we say that this state is *non deterministic*, in the sense that whenever the event a happens, the model does not provide sufficient information to decide on a single successor state, and thus *all* the possible successors can potentially be reached by a single transition.

2.1.2 Composition of labeled transition systems

Labeled transition systems can be composed by using the synchronized product of Arnold-Nivat [Arn94, Arn92]. Let $\{S_i\}$ be a set of LTS with $S_i = \langle Q_i, \Sigma_i, \rightarrow_i \rangle$ and consider a *synchronization constraint* $I \subseteq \prod_i \Sigma_i$. The synchronized product of the S_i with respect to the synchronization constraint I is the LTS $S = \langle Q, \Sigma, \rightarrow \rangle$ where $Q = \prod_i Q_i$, $\Sigma = \prod_i \Sigma_i$, and where

$$(s_1, \ldots, s_n) \xrightarrow{(e_1, \ldots, e_n)} (s'_1, \ldots, s'_n) \Leftrightarrow (e_1, \ldots, e_n) \in I, s_i \xrightarrow{e_i}_i s'_i, \forall i, 1 \leq i \leq n.$$

In the case where $I = \prod_i \Sigma_i$, the synchronized product is equivalent to the free product.

In[25]:=
```
Q = {x, y, z};
Σ = {a, b, c};
rel = {
    {x → y, c}, {x → z, a}, {z → z, b}, {z → y, a}, {z → y, c}, {y → y, b},
    {y → x, a}, {x → y, a}
  };
GraphPlot[rel, DirectedEdges → True, VertexLabeling → True]
```

Out[28]=

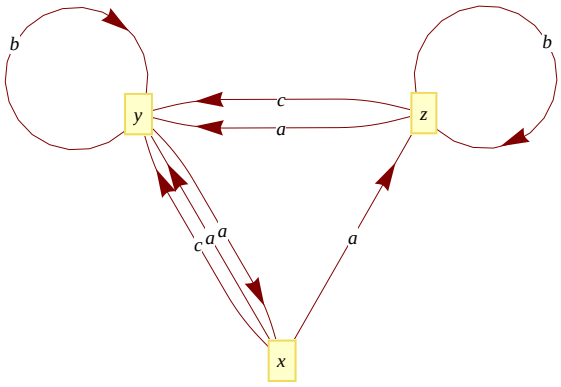


Figure 2.2: Example of a non deterministic LTS.

2.1.3 Semantics

An important notion we use in this manuscript is the notion of *semantics* associated with a class of models. We (loosely) define a semantics as a function mapping a model in a given *modeling formalism* to a model in another formalism (in this we follow [NN92]). For example, for the LTS we can provide a semantics in terms of *languages*. A language over an alphabet Σ is a set of *words* that are finite sequences $(e_i)_{i\leq n}$ where $e_i \in \Sigma$. Given an LTS $\mathcal{L} = \langle Q, \Sigma, \rightarrow \rangle$, the semantics of $\mathcal{L}$ in terms of languages is the set of words denoted $[\![\mathcal{L}]\!]$ such that for any word $(e_i)_{i\leq n} \in [\![\mathcal{L}]\!]$, there exists a sequence of states $(q_i)_{i\leq n+1}$ such that $q_i \xrightarrow{e_i} q_{i+1}$ for all $i \leq n$. Note that the semantics of a structured object such as an LTS is given in terms of less structured objects: sets generated by application of the binary operation of concatenation. In this sense, this implies that all the semantics defined in this thesis are not necessarily surjective. For instance, for the semantics of an LTS in terms of languages, it is known that there is no finite transition system or finite automaton whose semantics is language $\{a^n b^n\}$ for some integer n [HMU06]. In other words, there exist (in fact an infinite number of) languages that do not represent the semantics of any LTS.

2.1.4 Variables, Logic

The core limitation of using LTS to model a dynamic system is that the state space and the transition relation are explicit. In order to ease the description of complex systems with LTS, we can use a higher level (i.e. more structured) formalism the semantics of which will be given in terms of LTS. We will introduce the notion of *constraint automata* [BR94], that are automata with an implicit transition relation and state space, that manipulate *variables* by means of *assignments* whenever the *current valuation* of the automaton satisfies a *constraint*. In order to formally define this object we first need a logical language to define the notions

of variable, valuation and constraint. We only give here an overview of the logic, for a more thorough treatment, see [LC03, HR04].

A constraint automaton manipulates *variables* that are defined over a finite *domain* limiting the possible valuations. More formally, we have domains and variables as provided in the following definition.

Definition 2.2 (**Domains and variables**)**.** The domains, variables, domain of variables and valuations are defined as follows.

- $Z = \{a, b, c, \ldots\}$ is a finite set called the *domain* the elements of which are called *constants*. We consider for constraint automata that Z is a finite subset of $\mathbb{Z}$.
- $X = \{x, y, z, \ldots\}$ is a finite set of variables.
- An application $\mathrm{dom} : X \to \mathcal{P}(Z)$, such that for any x in X, $\mathrm{dom}(x) \neq \emptyset$. The application $\mathrm{dom}(x)$ is called the *domain of the variable* x, i.e. this is the set of possible values for the variable x.
- The set extension of *dom*, i.e. for any $Y \subseteq X, \mathrm{dom}(Y) = \prod_{y \in Y} \mathrm{dom}(y)$.
- A valuation v is an application $X \to Z$ that assigns a variable x to its value $v(x) \in \mathrm{dom}(x)$.
- For any variable $x \in X$, for any valuation v, for any value a in $\mathrm{dom}(x)$, we define $v[a/x]$ to be an updated valuation of v where $v[a/x](x) = a$ and where $v[a/x](y) = v(y)$ if $x \neq y$.

The expressions and formulas used in constraint automata are terms and formulas of a quantifier free first order language. The elements of Z are the constants of our language and these elements are considered as being nullary functions. Let F be a set of functions containing at least Z, *the terms* over F are defined inductively with the following Backus Normal Form grammar:

$$t ::= x \mid c \mid f(t, \ldots, t)$$

where x ranges over the set of variables X, c over nullary function symbols in F, and f over the elements of F with arity $n > 0$.

Let P be a set of predicate symbols, the formulas of a constraint automaton are defined by the following Backus Normal Form grammar:

$$\phi ::= \mathbb{1} \mid \mathbb{0} \mid p(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi$$

where $\mathbb{1}$ and $\mathbb{0}$ are symbols denoting the usual boolean constants true and false, $p \in P$ is a predicate symbol of arity $n \geq 1$ and t_i are terms over the functions F. Let $Y \subseteq X$ be a set of variables, we will denote by $\mathcal{T}^Y$ and $\mathcal{F}^Y$ the set of terms and formulas built with variables in Y. In the case where $Y = X$, we omit the superscript and simply write $\mathcal{T}$ and $\mathcal{F}$.

We will denote by $\mathrm{vars}(\phi)$ or $\mathrm{vars}(t)$ the set of variables occurring in the formula ϕ or in the term t.[1] Any formula $\phi(x, y, z, \ldots)$ is intended to be interpreted with respect to the finite set of variables X. Thus, the *semantics* $[\![\phi]\!]$ of a formula ϕ is a subset of $\mathrm{dom}(X)$. We define the

[1] For first order language with quantifiers, this is usually the set of free variables occurring in a formula.

semantics of the boolean constants $\mathbb{1}$ and $\mathbb{0}$ by $[\![\mathbb{1}]\!] = \text{dom}(X)$ and $[\![\mathbb{0}]\!] = \emptyset$. Finally, given a formula ϕ, a variable x and a term t, we define $\phi[t/x]$ to be the formula obtained by replacing each occurrence of the variable x with t. In the following, we fix the functions and predicates to be $F = \{+, -, *, /\}$ and $P = \{=, <, >, \leq, \geq\}$.

Example 2.1. Suppose n, f and g are function symbols respectively nullary, unary and binary. Then $g(f(n), n)$ and $f(g(n, f(n)))$ are terms, but not $g(n)$ (it violates the arity). Suppose $0, 1, \ldots$ are nullary, s is unary and $+, -$ are binary. Then $-(2, +(s(x), y))$ is a term. Usually, binary symbols are written in the infix manner rather than in the prefix manner; thus, the term is written $2 - (s(x) + y)$. For a set of variables $X = \{x, y, z\}$ with domains $Z = dom(x) = dom(y) = dom(z) = \{0, \ldots, 10\}$, the formula $\phi(x, y, z) = x + y + 3 \leq 4$ admits the semantics $[\![\phi]\!] = \langle 0, 0, Z\rangle \cup \langle 0, 1, Z\rangle \cup \langle 1, 0, Z\rangle \cup \langle 1, 1, Z\rangle$ where $\langle a, b, Z\rangle = \{\langle a, b, c\rangle \mid c \in Z\}$.

2.2 Constraint automata

A constraint automaton ([BR94] as cited by [Pag04] and [FP93]) is defined as follows.

Definition 2.3 (**Constraint automaton, not interfaced version**)**.** A *constraint automaton* is a tuple $\mathcal{A} = \langle S, \Sigma, A, M\rangle$ where

- S is a set of variables called *state variables*,
- Σ is a finite alphabet,
- $A \in \mathcal{F}^S$ is a formula over variables called the *global assertion*,
- $M \subset \mathcal{F} \times \Sigma \times (S \to \mathcal{T})$ is a set of *macro transitions* $\langle g, e, a\rangle$ where:
  - $g \in \mathcal{F}^S$ is called the *guard*,
  - $e \in \Sigma$ is the *label* of the event representing the transition,
  - $a : S \to \mathcal{T}^S$ maps every state variable to a term denoting its value after the transition. This mapping is called the *assignment* of the transition. Note that this application is complete, therefore every state variable is related to a term, that can be reduced to the variable itself if it is not affected by the transition.

In the next section we will formally define the semantics of a variant of constraint automata called *interfaced constraint automata*. Since we reuse most of the construction of the semantics of constraint automata, here we only provide an intuition of their LTS semantics. Consider a constraint automaton $\mathcal{A} = \langle S, \Sigma, A, M\rangle$ where $S = \{x\}$, $F = \emptyset$, $\Sigma = \{e\}$, $A = \mathbb{1}$, $M = \{(x \leq 3, e, x \to x + 1), (x > 3, e, x \to x + 2)\}$, we will manually build the LTS $[\![A]\!] = \langle Q, \Sigma, \to\rangle$. The set Q is the domain of the variables, that is $Q = dom(x)$ that we set to $Q = \{1, 2, \ldots, 6\}$. An element of Q is called a *configuration* in order to distinguish easily between a state and state variables. Consider the configuration $\langle 3\rangle$ corresponding to the valuation $x : 3$. The set of macro-transitions that have a guard satisfied by this configuration is $(x \leq 3, e, x \to x + 1)$. Therefore, in the underlying LTS, we have the transition $\langle 3\rangle \xrightarrow{e} \langle 4\rangle$. The configuration $\langle 5\rangle$ of the LTS corresponds to the valuation $x : 5$, for this valuation, the set of macro-transitions that have a guard satisfied is $\{(x \geq 3, e, x \to x + 2)\}$, however the assignment $x \to x + 2$ is forbidden since it will cause an overflow for the variable x. The set of transitions of this LTS is depicted in figure Fig. 2.3.

```
In[79]:= Q = Range[6];
         Union[
           If[# ≤ 3 ⋀ # + 1 ≤ 6,
              {# → # + 1, "e"}] & /@ Q,
         If[# ≥ 3 ⋀ # + 2 ≤ 6,
              {# → # + 2, "e"}] & /@ Q
          ][[2 ;;]]]
         GraphPlot[%, VertexLabeling → True, DirectedEdges → True,
          SelfLoopStyle → None]

Out[80]= {{1 → 2, e}, {2 → 3, e}, {3 → 4, e}, {3 → 5, e}, {4 → 6, e}}
```

Out[81]=

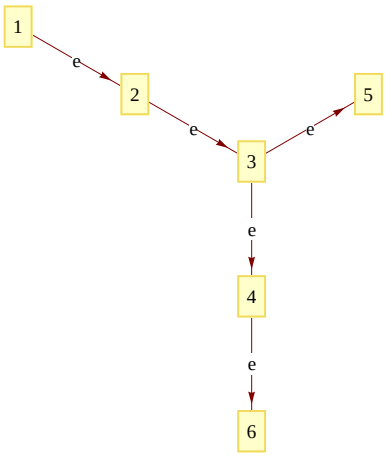


Figure 2.3: Semantics of a constraint automaton in terms of an LTS.

2.3 The AltaRica formalism

Although the constraint automata described in the previous section provide a more concise description of systems than LTS by means of variables and arithmetic operations, they are not *a priori* adapted to describe systems of interacting components. The *AltaRica formalism* [AGPR00, Poi00] has been designed to describe the possible interactions between components. AltaRica is a *hierarchical* specification language based on *interfaced constraint automata* and in which compositions are performed by *the synchronized product* of interfaced constraint automata.

An interfaced constraint automaton, also called a *component*, is defined as follows.

Definition 2.4 (Constraint automaton, interfaced version). An *interfaced constraint automaton* is a tuple $\mathcal{A} = \langle S, F, \Sigma, A, M \rangle$ where

- S is a set of variables called *state variables*,
- F is a set of variables called *flow variables*,
- Σ is a finite alphabet containing at least the letter ϵ; the letter ϵ denotes the label of the *invisible event*,
- $A \in \mathcal{F}^{S \cup F}$ is a formula over variables called the *global assertion*,

- $M \subset \mathcal{F} \times \Sigma \times (S \to \mathcal{T})$ is a set of *macro transitions* $\langle g, e, a \rangle$ where:
  - $g \in \mathcal{F}^{S \cup F}$ is called the guard,
  - $e \in \Sigma$ is the label of the event representing the transition,
  - $a : S \to \mathcal{T}^{S \cup F}$ maps every state variable to a term denoting its value after the transition. This mapping is called the *assignment* of the transition. Note that this application is complete, therefore every state variable is related to a term, that can be reduced to the variable itself if it is not affected by the transition.

  The macro transitions M always contain the macro transition $\langle \mathbb{1}, \epsilon, \mathrm{Id} \rangle$ where Id is the identity function.

The main differences between the interfaced and not interfaced version of a constraint automaton are the set of flow variables and the presence of a specific event ϵ denoting an invisible event. These two additions are the basis for hierarchical composition, and this is illustrated in the semantics of a constraint automaton is terms of LTS. This semantics is defined as follows.

Definition 2.5 (**Semantics of a constraint automaton in terms of LTS [Poi00]**)**.** The semantics of a constraint automata $\mathcal{A} = \langle S, F, \Sigma, A, M \rangle$ is the LTS $[\![\mathcal{A}]\!] = \langle Q, \Sigma, \to \rangle$ where $Q = \{(s, f) \in \mathrm{dom}(S \cup F) \mid (s, f) \in [\![A]\!]\}$ and where $(s, f) \xrightarrow{e} (s', f')$ if $(s, f, e, s') \in M$ and if $(s', f') \in [\![A]\!]$.

For a component in isolation, the other components in the system are abstracted by flow variables (representing variables visible by many components) and by the epsilon invisible event (representing external transitions updating the flow variable). This is implied by the LTS semantics since after each transition of a component, the flow variables are allowed to take any value within their domain, as long as the global assertion is respected. For a component in isolation, each transition triggers non deterministically an update of the valuation of flow variables. In the case where the transition of a component is guarded by a flow variable, the semantics in terms of LTS allows the component to make a empty transition $\mathbb{1} \xrightarrow{\epsilon} Id$ in order to always allow an update of the value of this flow variable. By this construction, the possible behavior of a component with flow variables uses non determinism to encompass all the possible behaviors of other components, even if they are not yet specified.

Flow variables are not assigned but constrained via the global assertion of a component. According to the definition of macro-transitions, it is not possible to assign a value to a flow variable as a result of a transition. In fact, the only possible way to "assign" a value to a flow variable is to constrain it by the use of the global assertion. For example, in a component with a state variable x and a flow variable f, the flow variable can be constrained to always have $f = x + 1$.

Hierarchical models in AltaRica are defined in terms of *AltaRica nodes* that are structures that contain other AltaRica nodes or components. An AltaRica node specifies synchronization constraints that are used when giving its semantics in terms of LTS. Basically, the semantics of an AltaRica node is defined as the synchronized product of the LTS that are the semantics of the components of the AltaRica node. The synchronization constraints are used to model events that should always occur simultaneously in the hierarchical model, while a global assertion is used to represent how flow variables should be coordinated in the hierarchy. We will come back to these notions in the next chapter, when we define the syntax and semantics of a *BioRica node*.

The AltaRica formalism also accounts for priorities between events and for scoping of event and flow variable visibility in the hierarchy. Priorities are a partial order $<$ over events specifying that if events e and e' are enabled in the same state and if $e < e'$, then only the transitions labeled with e' are effective. Events and flow variables can be scoped. This permits one to restrict an event synchronizability to its parent node or to restrict the visibility of a flow variable to itself.

2.4 Probabilities and measure

This section presents some notions of probabilities which are necessary for this thesis. The following brief introduction is based on a synthesis of [Haa02], [FF98],[Zim08],[Gut05] and [F$^+$57].

Let Ω be a set, called the *sample space*, that represents the possible outcome of a random probabilistic experiment. A subset $A \subseteq \Omega$ is called a (probabilistic) event. An event A occurs whenever the outcome of a probabilistic experiment is in A. In order to model the set of possible events, a σ-field (also called a σ-algebra) $\mathcal{F}$ is associated with Ω. The σ-field $\mathcal{F}$ is a collection of events such that

- $\emptyset, \omega \in \mathcal{F}$,
- for a subset $A \subseteq \Omega$, $A \in \mathcal{F}$, we have $A^c \in \mathcal{F}$, where $A^c = \Omega - A = \{\omega \in \Omega \mid \omega \notin A\}$,
- $\left(\bigcup_{i=1}^{+\infty} A_i\right) \in \mathcal{F}$ whenever $A_i \in \mathcal{F}$ for all $i \geq 1$.

The $\sigma-$field generated by a collection $\mathcal{A} \subseteq \mathcal{F}$ is the smallest σ-field containing $\mathcal{A}$, it is the intersection of all σ-fields containing $\mathcal{A}$.

We can then define a *probability measure* P over $\mathcal{F}$ as a non negative real-valued function such that

- $P(\emptyset) = 0$,
- $P(\Omega) = 1$,
- $P\left(\bigcup A_i\right) = \sum P(A_i)$,

whenever (A_i) is a (finite or countably infinite) collection of disjoint sets in $\mathcal{F}$. Then we immediately have for any $A, B \in \mathcal{F}$,

- $0 \leq P(A) \leq 1$,
- $P(A^c) = 1 - P(A)$,
- $A \subseteq B$ implies $P(A) \leq P(B)$.

In particular, if there exists a countable set $A \in \mathcal{F}$ such that $P(A) = 1$ then P is said to be a *discrete probability measure* and $\langle \Sigma, \mathcal{F}, P \rangle$ is called a *discrete probability space*.

The *support set* [Rud74] of a probability measure is the smallest subset of the sample space whose measure is 1. Formally, the support set of f is the set

$$supp(f) = \Omega - \bigcup \{A \in \mathcal{F} \mid A \text{ is open and } P(A) = 0\}.$$

Definition 2.6 (**Measurable and probability space**)**.** The pair $\langle \Omega, \mathcal{F} \rangle$ is called a *measurable space*, the triple $\langle \Omega, \mathcal{F}, P \rangle$ is called *a probability space.*

Definition 2.7 (**Measurable functions**)**.** Let $\langle \Omega, \mathcal{F} \rangle$ and $\langle \Omega', \mathcal{F}' \rangle$ be two measurable spaces, a function $f : \Omega \to \Omega'$ is said to be measurable if the set $f^{-1}(A) = \{a \mid f(a) \in A\}$ is in $\mathcal{F}$, for any set $A \in \mathcal{F}$.

In order to push forward a measure P, we have the following result.

Proposition 2.1. *If f is a measurable function as before and if $\langle \Omega, \mathcal{F}, P \rangle$ is a probability space, then $P \circ f^{-1}$ is a probability measure on $\langle \Omega', \mathcal{F}' \rangle$ and the triple $\left\langle \Omega', \mathcal{F}', P \circ f^{-1} \right\rangle$ is a probability space.*

For an event B such that $P(B) > 0$, the probability of an event A conditional on B, denoted $P(A \mid B)$ is defined as

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

The mapping P_B from $\mathcal{F}$to $[0, 1]$ with $P_B(A) = P(A \mid B)$ defines a probability measure P_B on $\mathcal{F}$.

Definition 2.8 (**Countable partition**)**.** A collection of events $(B_i)_{i \in I}$ is called a (countable) partition of Ω if

1. I is countable,
2. $i \neq j$ implies that $B_i \cup B_j = \emptyset$,
3. $P(B_i) \neq 0$ for all $i \in I$,
4. $\Omega = \bigcup_{i \in I} B_i$.

Theorem 2.2 (**Law of total probability**)**.** *Let (B_i) be a partition of Ω and $A \in \mathcal{F}$ be an event, we have*

$$P(A) = \sum_{i \in I} P(A \mid B_i) P(B_i).$$

We say that the events A, B are *independent* if $P(A \cap B) = P(A)P(B)$. We say that the events $\{A_0, \ldots, A_n\}$ are *mutually independent* if

$$P(A_{i_1} \cap \ldots A_{i_k}) = P(A_{i_1}) \times \ldots \times P(A_{i_k})$$

for $2 \leq k \leq n$ and for $0 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq n$.

2.4.1 General measures and Borel spaces

For a σ-field $\mathcal{F}$, a non negative real valued function μ over $\mathcal{F}$ is a *general measure* if μ is a probability measure without the constraint that $\mu(\omega) = 1$. If $\mu(A) > 0$ for some $A \in \mathcal{F}$, μ is said to be non trivial.

Let $\Re$ be the σ-field generated by the set of open intervals of the real line (a, b), the elements of $\Re$ are called the *Borel sets.* Then let μ be the unique measure on the measurable space $\langle \mathbb{R}, \Re \rangle$ such that $\mu((a, b)) = b - a$ for each interval (a, b). This measure μ is called the *Lebesgue measure.*

Let $\mathbb{R}^n$ be the n-dimensional real space, and let $\Re^n$ be the σ-field generated by the class of *rectangle sets* of the form $(a_1, b_1) \times \ldots \times (a_n, b_n)$, then the n-dimensional Lebesgue measure μ_n is the unique measure such that

$$\mu_n(A) = \prod_{i=1}^{n}(b_i - a_i).$$

In this thesis, we consider only probability spaces that are either countable or isomorphic to some Borel space defined on a real hyperspace.

2.4.2 Random variables and Probability distribution functions

Definition 2.9 (**Random variable**)**.** A *random variable* (r.v.) X on a probability space $\langle \Omega, \mathcal{F}, P \rangle$ is a real-valued function defined on Ω and *measurable with respect to* $\mathcal{F}$, that is $\{\omega \in \Omega \mid X(\omega) \leq x\} \in \mathcal{F}$ for $x \in \mathbb{R}$.

The previous definition is usually generalized for n dimensional r.v. that are measurable functions from a probability space to $\langle \mathbb{R}^n, \Re^n \rangle$.

The σ-field $\sigma\langle X \rangle$ generated by a r.v. X is the smallest σ-field containing all sets of the form $\{\omega \mid X(\omega) \leq x\}$. We always have $\sigma\langle X \rangle \subseteq \mathcal{F}$.

Note that in the definition of a random variable, the probability measure of the initial probability space is not used at all. In fact, we can dispose completely of this initial probability space and only consider the probability measure induced by a random variable, as stated by the following proposition.

Proposition 2.3. *Let X from $\langle \Omega, \mathcal{F}, P \rangle$ to $\langle \mathbb{R}^n, \Re^n \rangle$ be a r.v., then the function P_X from $\Re^n$ to $[0, 1]$ defined by*

$$P_X(B) = P(X^{-1}(B))$$

for any borel set B is a probability measure on $\langle \mathbb{R}^n, \Re^n \rangle$.

The *distribution* of a r.v. X is the probability measure μ_X defined on the measurable space $\langle \mathbb{R}, \Re \rangle$ such that $\mu_X(A) = P(\{\omega \in \Omega \mid X(\omega) \in A\})$ for $A \in \Re$. We note by $P\{X \in A\}$ the value $P(\{\omega \in \Omega \mid X(\omega) \in A\})$. The right continuous function F_X defined by

$$F_X(x) = P\{X \leq x\} = \mu_X((-\infty, x])$$

for $x \in \mathbb{R}$ is called the *cumulative distribution function* (CDF) of X. For a proper F_X we have

$$\lim_{x \to +\infty} F_X(x) = 1.$$

Given μ_X, we have an immediate and unique CDF F_X. Conversely, given a function F, μ is the unique measure on $\langle \mathbb{R}, \Re \rangle$ that satisfies $\mu((a, b)) = F(b) - F(a)$ for each interval $(a, b]$. This implies that given F, we have an immediate and unique measure μ and thus a r.v. X.

The *probability density function* (PDF) f_X of a r.v. X is given by the derivative of F_X if it exists.

$$\forall x \in \mathbb{R} : f_X(x) = \frac{d}{dx} F_X(x) \text{ and } F_X(x) = \int_{-\infty}^{x} f_X(y)dy.$$

Continuous random variables are used in this thesis to describe random delays of timed activities. Delays being obviously greater than or equal to 0, we restrict ourselves to distributions that have a support that is a subset of $[0, +\infty)$. We define the set of non negative

probability distribution functions $\mathbb{F}^+$ as

$$\mathbb{F}^+ = \{F_X \mid \forall x \in \mathbb{R}, x < 0 \Rightarrow F_X(x) = 0\}.$$

Discrete, continuous and mixed delays are described uniformly by using generalized distributions and density functions. A discrete probability mass at a point x leads to a step in the distribution function. The step function $s(x)$ (also called *Heaviside* function) is defined for any $x \in \mathbb{R}$ as

$$s(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ otherwise.} \end{cases}$$

The Dirac impulse δ_0 is defined by a rectangular function with constant area of one, for which the length of the basis is taken as zero. That is, $\delta_0(x)$ denotes a (generalized) function with an area of size one at 0, and represents a generalized derivative of the step function[Zim08]. Both the step function and the Dirac impulse can be translated by $b \in \mathbb{R}$ and scaled by $a \in \mathbb{R}^+$ to be combined with other parts of PDF or of a CDF as follows

$$\frac{d}{dx}(a \times s(x-b)) = a \times \delta_0(x-b) \text{ and } \int_{-\infty}^{x} a \times \delta_0(y-b)dy = a \times s * x - b.$$

By using a weighted sum of step functions, distribution functions of discrete random variables can be captured.

2.4.3 Joint distribution functions

We now consider probability statements concerning two or more random variables. Let X and Y be two (real-valued) r.v., then the joint cumulative distribution function of X and Y is defined by

$$F(a,b) = \mathbf{P}\{X \leq a, Y \leq b\}, -\infty < a, b, < +\infty.$$

The marginal distribution F_X of X is defined by $F_X(a) = \lim_{b\to\infty} F(a,b)$.

In the case where X and Y are continuous, we consider their *joint probability density functions*. We say that X and Y are jointly continuous if there exists a function $f(x,y)$ defined on $\mathbb{R}^2$ such that for every set $C \subseteq \mathbb{R}^2$, we have

$$\mathbf{P}\{(X,Y) \in C\} = \int\int_{(x,y)\in C} f(x,y)dxdy,$$

in which cases the function f is called the joint probability density function of X and Y. In order to compute such double integrals, we can apply the theorem of Fubini.

Theorem 2.4 (Fubini's Theorem [Gut05]). *Let $\langle \Omega_1, \mathcal{F}_1, P_1 \rangle$ and $\langle \Omega_1, \mathcal{F}_1, P_1 \rangle$ be probability spaces and consider the product space $\langle \Omega_1 \times \Omega_2, \mathcal{F}_1 \times \mathcal{F}_2, P \rangle$ where $P = P_1 \times P_2$ is the product measure as defined above. Suppose that $\mathbf{X} = \langle X_1, X_2 \rangle$ is a two dimensional r.v., and that g is $\mathcal{F}_1 \times \mathcal{F}_2$ measurable and (1) non-negative or (2)-integrable. Then*

$$\int_\Omega g(\mathbf{X})dP = \int_{\Omega_1\times\Omega_2} g(X_1,X_2)d(P_1 \times P_2) = \int_{\Omega_1} \left(\int_{\Omega_2} g(X)dP_2 \right) dP_1.$$

Therefore, in the case where the previous subset C can be decomposed so that $C = \{(x, y) \mid x \in A, y \in B\}$, we have

$$\mathbf{P}\{X \in A, Y \in B\} = \int_B \int_A f(x, y)dxdy.$$

In the case where the r.v. are independent, we can further decompose the previous integrals. Indeed, in the case where X and Y are independent, we have $\mathbf{P}\{X \leq a, Y \leq b\} = \mathbf{P}\{X \leq a\}\mathbf{P}\{Y \leq b\}$, which implies (for the jointly continuous case) that we have

$$\begin{aligned} F(a, b) &= F_X(a)F_Y(b) \\ f(a, b) &= f_X(a)f_Y(b). \end{aligned}$$

The previous definitions and properties hold for n r.v. and will be mainly used in this thesis to describe the probability of subsets of $\mathbb{R}^n$ in chapter 4. For an example of how we use these results, consider that X and Y are independent r.v. that have uniform distributions over the range $[0, 60]$. Then the probability $\mathbf{P}\{X + 10 < Y\}$ is obtained as follows,

$$\begin{aligned} & \int\int_{x+10<y} f(x, y)dxdy \\ = & \int\int_{x+10<y} f_X(x)f_Y(y)dxdy \\ = & \int_{-\infty<x<+\infty} \int_{-\infty<y<10-x} f_X(x)f_Y(y)dydx \\ = & \int_0^{60} \int_0^{10-x} f_X(x)f_Y(y)dxdy \\ = & \int_0^{60} f_X(x)F_Y(10 - x)dxdy. \end{aligned}$$

2.5 Stochastic processes

When observing a random experiment over time there is usually a (random) *sequence* of state changes. This is captured by the notion of a *stochastic process*. A stochastic process is a collection of r.v. $\{X(t) \mid t \in T\}$ that are indexed by the time t. The index set T may either be the set of natural numbers $\mathbb{N}$ or the set of real numbers $\mathbb{R}$. In the case where the state space of the process is discrete, we say that the stochastic process is *discrete state* process. Otherwise, we call it a continuous-state process. We now review two important classes of stochastic processes, namely,

1. Discrete Time Markov chains (DTMCs), that are discrete-state and discrete-time stochastic processes satisfying the Markov memoryless property for state transitions;

2. Continuous Time Markov chains (CTMCs), that are discrete-state continous-time stochastic process with exponentially distributed inter-event time.

We will further detail in chapter Chap. 4 the class of *Generalized Semi-Markov processes*.

2.5.1 Discrete Time Markov Chains

The simplest model of the stochastic processes that we consider are discrete-time Markov chains (DTMCs). A DTMC is a discrete time and discrete space process characterized by the Markov (memoryless) property. Let Q be a finite or countably infinite set called the *state space*.

Definition 2.10 (**Markov Chain**)**.** A sequence $(X_n)_{n\in\mathbb{N}}$ of r.v. is called a Markov chain if

$$\mathbf{P}(X_n = q_n \mid X_{n-1} = q_{n-1}, X_{n-2} = q_{n-2}, \ldots, X_0 = q_0) = \mathbf{P}(X_n = q_n \mid X_{n-1} = q_{n-1}).$$

An intuitive interpretation of a DTMC is that the probability of the next state only depends on the current state and not on any past history. We further consider *time-homogeneous Markov chains* where for any $i, j \in S$, we have $\mathbf{P}(X_n = i \mid X_{n-1} = j) = \mathbf{P}(X_{n+1} = i \mid X_n = j)$ for all $n \geq 0$. Time homogeneity implies that probabilities are not dependent on the time to reach a state and this implies that we can represent a Markov chain by its transition probability P_{ij} that is usually represented as a $Q \times Q$ matrix. In the case where Q is finite, we can define a DTMC as follows.

Definition 2.11 (**Discrete Time Markov Chain**)**.** A discrete-time Markov chain is a tuple $\langle Q, q_0, M\rangle$ where

1. Q is finite set of *states*,
2. $q_0 \in Q$ is the initial state,
3. $M : Q \times Q \to [0, 1]$ is the transition probability matrix.

We further require M to be a stochastic matrix, that is for any state q, we have $\sum_{q'\in Q} M(q, q') = 1$. An element $M(q, q')$ of the transition probability matrix gives the probability of making a transition from the state q to the state q'.

We now show how to give a semantics to a DTMC in terms of paths. A path of a DTMC is a non empty sequence $\omega = q_0q_1q_2, \ldots$ where $q_i \in Q$ and $M(q_i, q_{i+1}) > 0$ for all $i \geq 0$. We say that a finite path ω of length n is a prefix of an infinite path ω' if the first n states of ω' are exactly the sequence ω. We build a probabilistic measure on the set of finite paths by using the cylinder set construction ([KSK66], as cited by [Par02b], see also [BKH99]). Let q be a state of a DTMC $\mathcal{M} = \langle Q, q_0, M\rangle$. The probability measure $Prob_q$ on the set of paths $Path_q$ starting from q is defined inductively by $\mathbf{P}(q) = 1, \mathbf{P}(qq_1q_2 \ldots q_n) = M(q, q_1) \cdot M(q_1, q_2) \cdot \cdots \cdot M(q_{n-1}, q_n)$. The cylinder set $Cyl(\omega)$ is then the set of all paths with prefix ω. Then, let Σ_q be the smallest σ-field on $Path_q$ which contains $Cyl(\omega)$ where ω are all the the finite paths starting in q. The probability measure $Prob_q$ is then the unique measure $Prob_q(Cyl(\omega)) = \mathbf{P}(\omega)$.

2.5.2 Exponential distribution

Before moving to the continuous time Markov chains, we first review in this subsection the properties of an important family of distributions called *the exponential distribution*. Consider a r.v. X, we say that X has an exponential distribution of parameter $\lambda \in [0, \infty)$ (called the rate) if its PDF is given by

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

or equivalently

$$F_X(x) = \int_{-\infty,x} f_X(y)dy = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0. \end{cases}$$

The main characteristic of the exponential distribution is its memoryless property, that is

$$P(X > s + t \mid X > t) = P(X > s), \qquad \text{for all } s, t \geq 0. \tag{2.1}$$

Equation Eq. 2.1 implies immediately that we have

$$\mathbf{P}(X > s + t) = \mathbf{P}(X > s)\mathbf{P}(X > t).$$

Furthermore, the minimum of several independent exponentially distributed r.v. is again a random variable with a rate equal to the sum of rates[Ros96].

2.5.3 Continuous time Markov chains

Let Q be a set of states. A continuous-time Markov chain is a stochastic process $\{X(t)\}$ such that

$$\mathbf{P}(X(t_n) = q_n \mid X(t_{n-1}) = q_{n-1}, X(t_{n-2}) = q_{n-2}, \ldots, X(0_{=}q_0) = \mathbf{P}(X(t_n) = q_n \mid X(t_{n-1}) = q_{n-1})$$

for any increasing sequence $t_0 \leq t_1 \leq \cdots \leq t_n$. Similarly to discrete space discrete-time Markov chains, if the current state x_n is known, the value taken after the next transition depends only on x_n and not on any past history. Analogously to the discrete time case, we consider time homogeneous CTMCs and define one step stationary transition probabilities from the state i to the state j in less than time t as $P(i, j, t) = \mathbf{P}(X(s + t) = j \mid X(s) = i) = \mathbf{P}(X(t) = j \mid X(0) = i)$. Unfortunately, continuous transition probabilities can not be represented by a finite real-valued matrix, and CTMCs are thus not usually defined by their stationary transition probabilities. We will instead show that a higher level structure inspired by discrete event systems admits a semantics in terms of continuous time Markov chains.

Let $S = \langle Q, \Sigma, \rightarrow, \Lambda \rangle$ be a decorated LTS where $Q \subset \mathbb{N}$, Σ is a finite alphabet and Λ is a mapping from $Q \times Q$ to $\mathbb{R}$ that maps to each pair of states (i, j) a rate $\Lambda(i, j)$ called *the rate of transition from state i to state j* . We further suppose that $\rightarrow$ is deterministic, that is for all state $q \in Q$, for all label $e \in \Sigma$, the set of states $\left\{ q' \in Q \mid q \xrightarrow{e} q' \right\}$ has zero or one element. If needed, Σ and $\Rightarrow$ can be built from an unlabeled transition relation $(q, q') = Q \times Q$ by setting $\rightarrow = \{(q, qq', q')\}$ and Σ as the second coordinate projection.

Suppose that movement from the state i to the state j is determined by "competing" clocks that go off at random, exponentially-distributed times. Such clock are usually called *timers*. For each state i, there is a timer associated with every event enabled in i, and the process moves from state i to the state associated with the first expiring timer. let C_i be the set of events active in state i, equivalently this is the set of active timer in state i. As a consequence of the assumptions that the timers follow an exponential distribution, the probability that two timers expire at the same time is 0. Let T_i be the time at which the first timer goes off in state i and let N_i be the *index* of the first timer that goes off. That is

$$T_i \sim min_j T_{i,j},$$
$$N_i \sim argmin_j T_{i,j}.$$

By the property of the exponential distribution that states that the minimum of independent exponential r.v. is itself an exponential r.v., the distribution of T_i is immediate

$$\mathbf{P}(T_i \leq t) = 1 - e^{-\lambda_i t} t \geq 0,$$

where λ_i is the sum of the rates $\Lambda(i,j)$ for j in the possible successor of i, that is

$$\lambda_i = \sum_{j \in Q | \exists e \in \Sigma, i \xrightarrow{e} j} \Lambda(i,j). \tag{2.2}$$

Furthermore, we have

$$\mathbf{P}(N_i = j) = \frac{Q(i,j)}{\lambda_i} \text{ for } i \neq j.$$

Finally, T_i and N_i are independent random variables, and thus we have

$$\mathbf{P}(T_i \leq t, N_i = j) = \mathbf{P}(T_i \leq t)\mathbf{P}(N_i = j) = (1 - e^{-\lambda_i t}) * \frac{Q(i,j)}{\lambda_i}.$$

From this characterization, the continuous time process defined by $\{X(t)\}$ satisfies the continuous time Markov property.

This construction of the CTMC exhibits that the Markov "memoryless property" is in fact two fold: First, all past *state* information is irrelevant (no state memory), and second, *how long* the process has been in the current state is irrelevant (no state age memory needed). In fact, it is mostly the second characterization that restricts the CTMC to have exponentially distributed inter-event time, and we will discuss in chapter 4 the class of *Semi Markov processes* that relax the second property and the class of *generalized semi-Markov processes* that relax both properties.

2.6 Discrete Event Systems

In this section, we present the framework of *Discrete Event Systems* by following the presentations made in [Zim08] and [CL08].

A discrete event system (DES) is a system which stays in a state during some time after which an atomic event might happen that changes the state immediately. Just like for the constraint automata, the possible states of the system are described by state variables. A state of the system is then characterized by the association of a certain value with each of the state variables, which is called a valuation.

DES are studied because they model both the static and dynamic information about a system, thus making them useful for evaluating the behavior of dynamic system. In DES, the behavior of a system is represented by the visited states and the *events* transitioning from one state to the next. Events may change the values of some state variables and thus represent a set of possible transitions. Such events are meant to represent activities, that might become enabled once their conditions are satisfied, start, take some time to complete and are finally fired resulting in a immediate transition from one state to another.

The time between two subsequent transitions depends on the time that the corresponding events have been enabled. Whenever an event becomes newly enabled, a *delay* is sampled from a *delay distribution* and represents the time before this event fires. When the time is used up, the event fires which results in an immediate transition from one state to another. In this new state, the remaining events delays yields the next event to fire. The time that is spent in one individual state is called the *sojourn time*.

2.7 Mathematical modeling of biological systems

Mathematical and computational techniques are essential for systems biology [Kit01] as they provide the foundations for describing formally models of systems with interacting components.

2.7.1 Biochemical reaction networks

Among all the possible representations of a biological mechanism, a biochemist will tend to view systems as networks of biochemical reactions, and it appears that most biological processes "exist" at this level of detail [Wil06].

A network of biochemical reactions is a set of general chemical reactions of the form

$$\sum_{i \leq r} m_i R_i \rightarrow \sum_{j \leq p} n_j P_j$$

where r is the number of reactants, p the number of products, R_i represents a reactant molecule, P_i a product molecule, m_i is the number of molecules of R_i consumed in a single reaction step, n_i the number of molecules P_i produced in a single reaction step. The coefficients m_i and n_j are called *stochiometries*. The stochiometries are usually (but not necessarily) assumed to be natural numbers. Furthermore, there is no assumption that the R_i and P_i are disjoint, and it is possible for a molecule to be both consumed and produced in a single reaction step.

2.7.2 Kinetics

Biochemical kinetics is a field aimed at studying the dynamics of a biochemical reaction networks. Essentially, biochemical kinetics is concerned with the time-evolution of a chemical mixture whose substances react according to a biochemical reaction network. In other words, biochemical kinetics aims at determining the *reaction rates* of each biochemical reaction possible in the system.

Law of mass action Given a set of biochemical reactions, the *law of mass action* states that the rate of a single reaction is proportional to the product of the concentrations of the reactants. For example, consider the following set of biochemical reactions, known as the Lotka-Volterra system (LV) ([Vol31] as cited by [Hae05]):

$$\begin{aligned} A &\rightarrow 2A \\ A + B &\rightarrow 2B \\ B &\rightarrow \emptyset \end{aligned}$$

This system of biochemical reactions represents a predator-prey dynamics, where A is the prey and B is the predator. Here, we assume an open system, where prey reproduces "out of nothing" (unlimited food is available) while predators reproduce while eating prey and can also die.

This model admits an intuitive interpretation where the amount of prey and predators are non negative integer amounts, that change only by a discrete amount when a reaction event occurs. This picture is correct, and will be used in the next subsection 2.7.3. However, we

will introduce here the dynamics of this system with a classical macroscopic setting. In this setting, the amount of each reactant and product is measured as real-valued *concentrations*, measured in (say) moles per liter, M, which can vary continuously as the reactions occur. Conventionally, the concentration of a species X is denoted $[X]$.

According to the law of mass action, the instantaneous rate of a reaction is directly proportional to the concentration of each reactant raised to the power of its stoichiometry. So for the Lotka-Volterra system, the second reaction will proceed at the rate $k_2[A][B]$ where $k_2 \in R^+$ is the constant of proportionality of this reaction. Since this reaction consumes one reactant A, $[A]$ will also decrease at the instantaneous rate $k_2[A][B]$; and $[Y]$ will increase at the same rate. When all three equations are considered, we can obtain for the Lotka-Volterra system a dynamic model in terms of ordinary differential equations:

$$\begin{aligned} \frac{d[A]}{dt} &= k_1[A] - k_2[A][B], \\ \frac{d[B]}{dt} &= k_2[A][B] - k_3[B]. \end{aligned}$$

In order to solve this system numerically, the three rate constants, k_1, k_2, k_3 and the initial conditions $A(0), B(0)$ must be specified. Once this has been done, the complete dynamics of the system of ODEs is specified and can be revealed by numerically approximating the solution using iterative methods (e.g. Euler, Runge-Kutta, see [PTVF02] for a more thorough treatment on numerical approximation of solutions of ODEs).

Properties of the ODE system The solutions depicted in figure Fig. 2.4 as ϵ decreases to 0 exhibit two properties of the Lotka-Volterra system when modeled using continuous deterministic dynamics in terms of ODE. First, the system exhibits periodic solutions for any value of $\epsilon > 0$. Second, the system is at a stable equilibrium when $\epsilon = 0$. It can be shown that indeed $A = k_3/k_2, B = k_1/k_2$ is an equilibrium solution, that is for which $A(t), B(t)$ are constants. Furthermore, by studying the Jacobian of the system, we can characterize the nature of this equilibrium point and see that it is stable and not attractive. This implies that for any value of $\epsilon > 0$, the system will never reach this equilibrium point. More detailed explanation on the qualitative analysis of equilibrium points can be found in [Mur03] and in the specialized monograph [Fra04].

2.7.3 Mass action stochastic kinetics

The interpretation of a biochemical reaction network in terms of ODEs is based on a macroscopic assumption where discrete quantities (number of molecules) are approximated by continuous quantities (concentrations). This implies for the example of the Lotka-Volterra system that a population never really goes extinct, since small residuals (due both to the system and to its numerical approximation) will imply that $A(t) \neq 0$ almost surely for any $t \geq 0$. This is illustrated in the following analysis where extreme values are sought. In the simulation presented in Fig 2.5, the value of the species A ranges from 10^{-11} to 10^{11}, but is never 0.

The extreme cases for the macroscopic approach to kinetics fail to capture the discrete and stochastic nature of chemical kinetics when the number of some of interacting molecules are very low. As many intra-cellular processes involve extremely low concentrations, such discrete stochastic effects are often relevant for systems biology models. For example, this effect has

been shown to have a qualitative impact on a model of the circadian clock where stochastic effects enhance the resistance of a chemical oscillator [VKBL02].

We will follow the presentation given in [Wil06] of the approach initiated by Gillespie in [Gil77]. We suppose that the biochemical reactions under study happen in a container of fixed volume where reactants are well stirred. In such a container, the position of the molecules in the system can be modeled as uniformly distributed random variables; and this distribution does not depend on time. Furthermore, since we assume thermal equilibrium, the probability that two molecules are within a reaction distance is also independent of time, and thus only dependent on the quantity of each reactant. Just like in the study of stochastic processes, we now seek to compute the probability that a reaction will occur within t time units given the current state of the system. More formally, let $\vec{x}(t)$ denote the state of the system at time t, i.e. $\vec{x}(t)$ is a vector of integers representing the quantities of each reactant in the biochemical reaction network. We now define the probability $h_i(\vec{x})$ that, conditional on $\vec{x}(t)$, a reaction R_i occurs within dt time unit. This probability is called the *stochastic rate law*.

Suppose that we are considering a zeroth-order reaction of the form $R_i : \emptyset \rightarrow P$. Then, the probability that such a reaction occurs is given by c_i where c_i is a constant called the *stochastic rate constant*. In the case of a first order reaction we have $R_i : X_j \rightarrow ?$ where ? can be anything. The probability $h_i(\vec{x})$ that such a reaction happens in the state $\vec{x}$ is $c_i\vec{x}_j$, where c_i is interpreted as the probability that 1 molecule will undergo this reaction. In the case of a second order reaction $R_i : X_j + X_k \rightarrow ?$, suppose that the hazard that a particular pair of molecules will react is denoted by c_i, then since there are $\vec{x}_j\vec{x}_k$ possible pairs of these particular molecules, we have $h_i(\vec{x}) = c_i\vec{x}_j\vec{x}_k$. For the second kind of second order reactions $R_i : 2X_j\emptyset ?$, we have $h_i(\vec{x}) = c_i * 1/2 * \vec{x}_j(\vec{x}_j - 1)$ since there are $1/2 * \vec{x}_j(\vec{x}_j - 1)$ possible pairs of the same molecule X_j.

We now consider the system as a stochastic process. Since the reaction hazards are dependent on the current state of the system and that they are time homogeneous, the time evolution of the system's state is seen as a continuous-time discrete-state Markov process, namely a CTMC. In fact, we have from [Gil77] that the time before one of the reactions happens is an exponentially distributed r.v. of rate $h_0(\vec{x}) = \sum h_i(\vec{x})$, while the probability that this next reaction is R_i is given by $h_i(\vec{x})/h_0(\vec{x})$. This gives us the direct and exact stochastic simulation algorithm presented by the original author, of which an implementation is given in figure Fig. 2.6.

```
In[108]:= LV := {
            A'[t] == k1 * A[t] - k2 *A[t]*B[t],
            B'[t] == k2 * A[t]*B[t] - k3*B[t],
            A[0] == initA, B[0] == initB
           }

In[115]:= conditions = {k1 → 1, k2 → 0.1, k3 → 0.1, initA → 0.1 / 0.1 + ϵ,
            initB → 1 / 0.1 + ϵ}

Out[115]= {k1 → 1, k2 → 0.1, k3 → 0.1, initA → 1. + ϵ, initB → 10. + ϵ}

In[119]:= Table[
           sol = NDSolve[LV /. conditions /. {ϵ → 1}, {A, B}, {t, 0, 100}];
           Plot[{A[t], B[t]} /. sol, {t, 0, 100}],
           {ϵ, 1, 0, -0.2}
          ]
```

Out[119]=

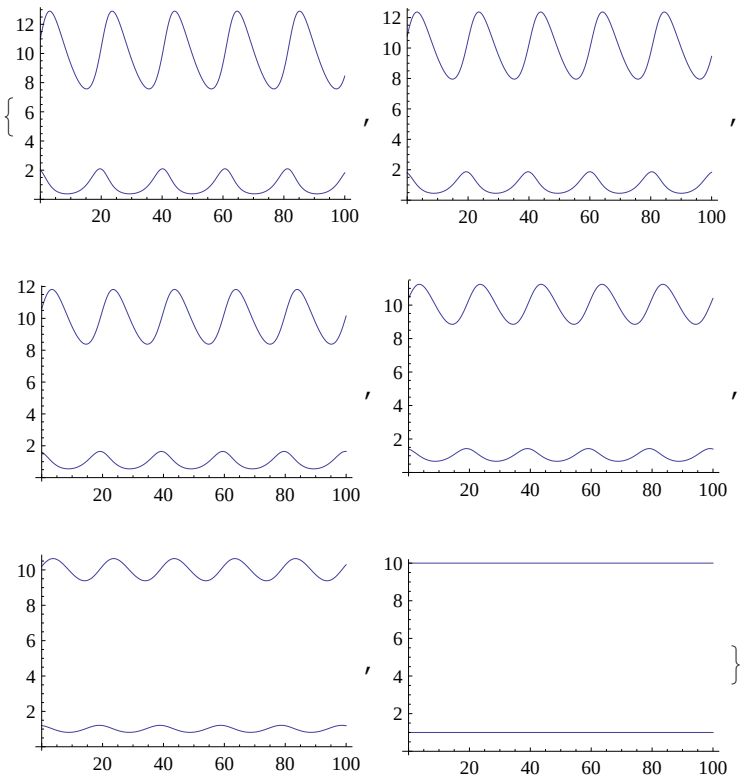
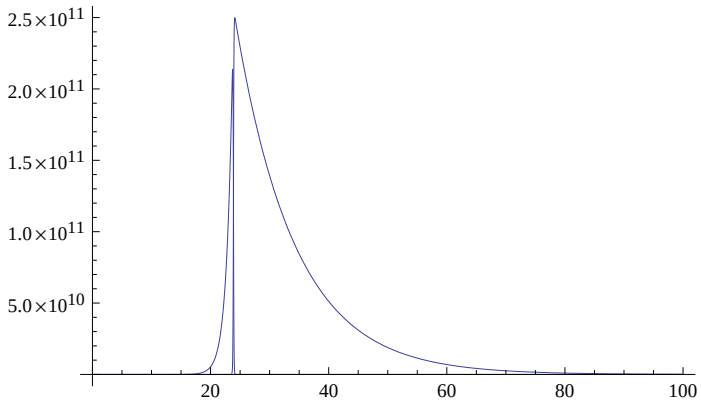


Figure 2.4: Lotka-Volterra dynamics for the rates $k_1 = 1, k_2 = 0.1, k_3 = 0.1$ and initial conditions $A(0) = k_3/k_2 + \epsilon, B(0) = k_1/k_2 + \epsilon$ as $\epsilon \in \mathbb{R}$ ranges over $[0, 1]$ by step of 0.2.

In[309]:=
```
conditions = {k1 → 1, k2 → 0.0000000001, k3 → 0.1, initA → 1 + ϵ,
   initB → 10 + ϵ}
sol = NDSolve[LV /. conditions /. {ϵ → 10}, {A, B}, {t, 0, 100}];
Plot[{A[t], B[t]} /. sol, {t, 0, 100}, PlotRange → All]
```

Out[309]=
$$\left\{k1 \to 1,\ k2 \to 1.\times 10^{-10},\ k3 \to 0.1,\ initA \to 1+\epsilon,\ initB \to 10+\epsilon\right\}$$

Out[311]=



In[328]:=
```
FindMinimum[A[t] /. sol, {t, 50}]
FindMaximum[A[t] /. sol, {t, 20}]
```

Out[328]=
$$\left\{3.82574\times 10^{-11},\ \{t \to 56.353\}\right\}$$

Out[329]=
$$\left\{2.13992\times 10^{11},\ \{t \to 23.7395\}\right\}$$

Figure 2.5: Simulation of the LV system with extreme parameter value for k_2.

In[194]:=
```
UpdateState[state_, reactionIndex_] := Switch[reactionIndex,
  1, {state[[1]] + 1, state[[2]]},
  2, {state[[1]] - 1, state[[2]] + 1},
  3, {state[[1]], state[[2]] - 1}
 ]
```

In[238]:=
```
StepSystem[{t_, state_}] := Module[{hazards, nextT, reactionIndex},
  If[state == {0, 0}, Return[{t, state}]];
  hazards = {
    rates[[1]] * state[[1]],
    rates[[2]] * state[[1]] * state[[2]],
    rates[[3]] * state[[2]]};
  nextT = RandomReal[ExponentialDistribution[Total[hazards]]];
  reactionIndex = RandomChoice[hazards -> Range[3]];
  {t + nextT, UpdateState[state, reactionIndex]}
 ]
```

In[248]:=
```
t0 = 0; state0 = {200, 100};
rates = {1, 0.005, 0.6};
```

In[235]:=
```
res = FixedPointList[StepSystem, {t0, state0}, 50 000];
timeSeries = Partition[Flatten[res], 3];
ListLinePlot[{timeSeries[[All, {1, 2}]], timeSeries[[All, {1, 3}]]}]
```

Out[237]=

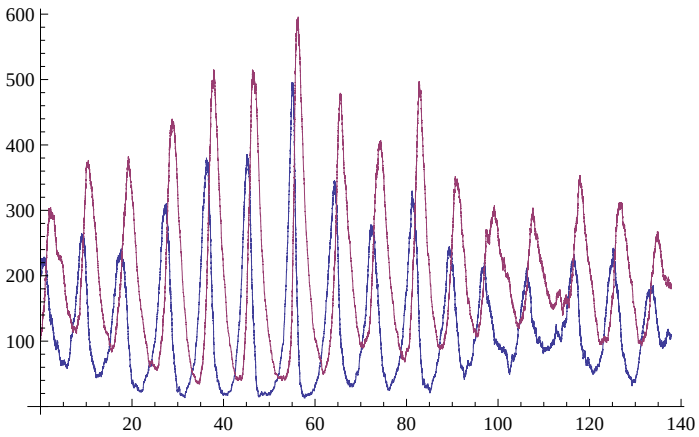


Figure 2.6: A single realization of a stochastic Lotka-Volterra system for the stochastic rate constants $(1, 0.005, 0.6)$ and initial conditions $(200, 100)$. This realization is obtained with the function Step, implementing the Gillespie direct method.

Part II

Modeling

Chapter 3

The BioRica language

In this chapter, we define the syntax and transition semantics of the BioRica language. The BioRica language is a hierarchical and declarative language to describe discrete dynamic systems. A discrete dynamic system is a system in which a component may only change due to a discrete step called a transition.

In BioRica the transitions are declarative. That is, the transitions express the logic of a computation and not necessarily its control flow. To this end, the possible dynamics of the system are described via macro transitions. Macro transitions are guarded transitions indicating both the constraints that must be met for a transition to occur and how they affect the variables of the system.

The mathematical foundation of this approach is the notion of guarded automata. Guarded automata are like non deterministic finite automata except that the state and transitions are kept implicit, i.e. are described by constraints and assignments over variables. This generalizes both finite state machines and Petri Nets with bounded places. One of the key advantages of such a declarative language is that it can shorten the description of complex dynamics.

This chapter is organized as follows. Constraints being fundamental in our language, we will first define how they can be described with first order formulas denoting the conditions under which an event can occur. We will then introduce BioRica nodes, that are the base unit of our systems. We then give the semantics of BioRica nodes in terms of stochastic mode automata, a generalization of transition systems with private and public variables, and with a stochastic labeling. Finally, we introduce composition operations that can be used to build a system by reusing components. The composition operation we will consider are the parallel product, flow connections and event synchronization. We will then show that our semantics is *compositional*, in the sense that a BioRica system made of composed nodes can be flattened into a single BioRica node.

Comparison with the AltaRica formalism

The notable differences between BioRica and AltaRica are the following. In BioRica, variables and transitions can only be defined at the leaves of the hierarchy; while in AltaRica, variables and transitions can appear at any level in the hierarchy. Assertions in BioRica are restricted to equality between an output variable and a term of state or input variables. As such, they are used to constraint to value of output flows. In AltaRica, assertions can be arbitrary quantifier free formulas and can be used to model complex dependencies between variables. Furthermore, we used a new approach for the flow connections and synchronization mechanism. Finally,

the main difference between BioRica and AltaRica is the stochastic labeling associated to some events.

3.1 BioRica node

3.1.1 Abstract syntax

BioRica nodes are structures denoting the abstract representation of a BioRica program containing a single node. This structure follows closely the concrete syntax of the BioRica language, and each component of this structure corresponds to a possible clause in the concrete declaration of a node.

Definition 3.1 (**BioRica Node**)**.** A BioRica node is an 9-uple $\mathcal{B} = \langle S,\ F^{in},\ F^{out},\ \text{dom},\ \Sigma,\ A,\ M,\ \Gamma_C,\ W\rangle$ where:

- S, F^{in}, F^{out} are disjoint sets of variables respectively called *state variables, input flow variables,* and *output flow variables.*
- $\text{dom} : S \cup F^{in} \cup F^{out} \to \mathcal{P}(Z)$ is an application that maps each variable of the node to its domain.
- Σ is a finite alphabet containing at least the letter ϵ. The letter ϵ denotes the label of the *invisible event.*
- $A : F^{out} \to \mathcal{T}^{S \cup F^{in}}$ is a total application denoting the node assertion. The node assertion maps every output flow variable o to a term $A(o)$ s.t. $\text{vars}(A(o)) \subset S \cup F^{in}$.
- $M \subset \mathcal{F} \times \Sigma \times (S \to T^{S \cup F^{in}})$ is a set of *macro transitions* $\langle g, e, a\rangle$ where:
  - $g \in \mathcal{F}^{S \cup F^{in}}$ is called the guard,
  - $e \in \Sigma$ is the label of the event representing the transition,
  - $a : S \to \mathcal{T}^{S \cup F^{in}}$ maps every state variable to a term denoting its value after the transition. This mapping is called the *assignment* of the transition. Note that this application is complete, therefore every state variable is related to a term, that can be reduced to the variable itself if it is not affected by the transition.

  The macro transitions M always contain the macro transition $\langle \mathbb{1}, \epsilon, \text{Id}\rangle$ where Id is the identity function.
- $\Gamma_C : \Sigma \to \left(\mathcal{T}^{S \cup F^{in}}\right)^n \times (\mathbb{R}^n \to \mathbb{F}^+)$, where $n \in \mathbb{N}$ and where $\mathbb{F}^+$ is the set of probability distributions which support is included in $[0, \infty)$. The application Γ_C maps every event e to a vector of terms and a constructor of delay distributions.
- $W : \Sigma \to \mathbb{N} \cup \{\sharp\}$ maps to any event e a strictly positive value or an indefinite value called the *weight.*

The difference in nature between state and flow variables is that the values of the state variables are purely local and the environment (i.e. other components of the system) can never access directly to their values. On the contrary, the flow variables are precisely used to exchange information between the component and its environment.

The configuration of a component may only change due to discrete steps called transitions. These transitions are caused by events that are of two types. The first type are called local events. Local events are events that the component know about, and may alter the value of state variables. These are the events that are declared in the alphabet Σ and whose macro transitions are defined in the macro transition set M. The second type of events is called *invisible events*, and are all represented by the same symbol, noted ϵ. For these invisible events, the configuration is modified because of an external event, e.g. a modification in the configuration of the environment. Such events may only alter the value of flow variables and are kept implicit in the model.

Two different stochastic labeling are possible in a BioRica node. The first stochastic labeling Γ_C, maps each event to a probabilistic distribution $f \in \Psi$ denoting the delay of the event. The elements of Ψ are distributions over $[0, \infty)$. These distributions can be either degenerate, discrete or continuous distributions. We thus consider (with an abuse of notation) that Ψ contains *functions* from $[0, \inf)$ to $[0, 1]$, although some elements of Ψ are not functions (e.g. the Dirac delta distribution). As such, an event may be modeled as instantaneous by associating it with a delay following a Dirac distribution δ_0 centered around 0 or (equivalently) by a discrete degenerate distribution.

The second stochastic labeling W is a partial mapping from events to $\mathbb{N}$, that associates event labels with an integer denoting the weight of the event. The weight of an event is used to assign a probability to an event in the case where multiple events are concurring.

3.1.2 Transition semantics

In order to define the semantics of a BioRica node, we first detail the semantics of the macro-transitions as an unfolding in terms of labeled transition systems. As we stated earlier, each macro-transition describes the set of configurations that enables a transition as well as the assignments that are performed after a transition has been fired. Following the work done on AltaRica [AGPR00], macro transitions in BioRica are unfolded by accounting both for their *preconditions* (stated in the guard) and their *postconditions* (domain or assertion violation).

Example 3.1. As a starting simple example, we consider the following non stochastic bounded counter described in BioRica.

```
1  node counter
     state
3      val :[0,10];
     flow
5      working :[0,10]: i;
       result  :[0,2]: o;
7    event
       incr ,decr;
9    trans
       (working≥1) ⊢incr →val:=val+1;
11     (working≥1) ⊢decr →val:=val−1;
     assert
13     result =working+val;
   edon
```

This program defines a single BioRica node *counter*, with an integer state variable *val* ranging from the 0 to 10. The node *counter* also uses two flow variables, *working* that is an input variable, and *result* that is an output variable. The two events *incr* and *decr* label two guarded macro transitions that results resp. in an incrementation and in a decrementation of the variable *val*. Finally, an assertion indicates how the value of the output variable *result* is determined given the value of the variables *working* and *val*.

The transition system semantics of this model is depicted in the fig. 3.1. This transition

system illustrates four characteristics of the transition system semantics of BioRica: 1) *pre-conditions*, 2)*post-conditions*, 3)*state space reduction by assertion*, and 4)*non deterministic fluctuations*.

First, as explicitly stated in the model, the two events labeled *incr* and *decr* can only happen when $working \geq 1$. Thus, no transitions labeled with *incr* or *decr* appear in the transition system when $working = 0$. This is called the *pre-condition* of the macro transition.

Second, the *decr* macro transition (line 11) does not explicitly forbid a decrementation to occur when the counter *val* is 0. Such a decrementation would yield a configuration where $val = -1$. However, since we want to ensure that each state of the transition system corresponds to a valuation of the variables that respects the domains of each variable, the *decr* event is not enabled in any state where $val = 0$, or more generally, is not enabled in any state where the assignments would lead to a configuration violating the domain of the variable *val*. In other words, a macro-transition is only possible if it reaches a state respecting the domain invariant of the system. This is called the *post-condition* of a transition.

Third, notice that although the domain of the variables indicates that the state space of the model should be the discrete set $\{0, \ldots, 10\} \times \{0, 1\} \times \{0, 2\}$, the assertion $result = working + val$ reduces this state space to the six valuations satisfying $result = working + val$. Those are the six states depicted in fig. 3.1.

Fourth, we stated earlier that state and flow variables are inherently different: the former can only be modified via assignments while the latter can only be modified by the environment during an invisible transition. This is illustrated in all the transitions of the transition system that are labeled with "()". However, the transition system also includes the transition $val = 0, working = 1 \xrightarrow{incr} val = 1, working = 0$, which is not invisible and where the value of *working* changes. In fact, we consider that input variables can be modified at any time by the environment, and thus that they can be modified *during* a transition. In other words, the possible states after an assignment is the set of states obtained by applying the assignments to the state variables, and where all the input variables can fluctuate in their domain, as long as the resulting valuation does not violate the assertion of the node.

We can now formalize the previous remarks to define the transition system semantics of a BioRica node. We first show how the formula denoting the pre and post-condition of a transition can be built. Let $\mathcal{B} = \langle S, F^{in}, F^{out}, \text{dom}, \Sigma, A, M, \Gamma_C, W \rangle$ be a BioRica node, the pre- and post-conditions are defined as follows.

Definition 3.2 (**Pre and post-conditions of a macro transition**)**.** Let D be the first order formula representing the domain of the variables defined as $[\![D]\!] = dom(S \cup F^{in} \cup F^{out})$. The pre and post-conditions $\text{Pre}(t), \text{Post}(t)$ of a macro-transition $t = \langle g, e, a \rangle$ are the first order formulas defined by

$$\text{Pre}(t) = \; g \wedge D \wedge \bigwedge_{o \in F^{out}} (o = A(o)) \,,$$

and by

$$\text{Post}(t) = D[a(s_1)/s_1, \ldots, a(s_n)/s_n, s_i \in S]$$

where $D[a(s_1), \ldots, a(s_n)/s_1, \ldots, s_n, s_i \in S]$ represents the substitution of every $s_i \in S$ by the term $a(s_i)$.

The pre- and post-conditions formulas denote the conditions under which a transition is enabled. More formally, we have $s \xrightarrow{e} s' \Leftrightarrow s \in [\![Pre(t) \wedge Post(t)]\!]$ for a macro-transition

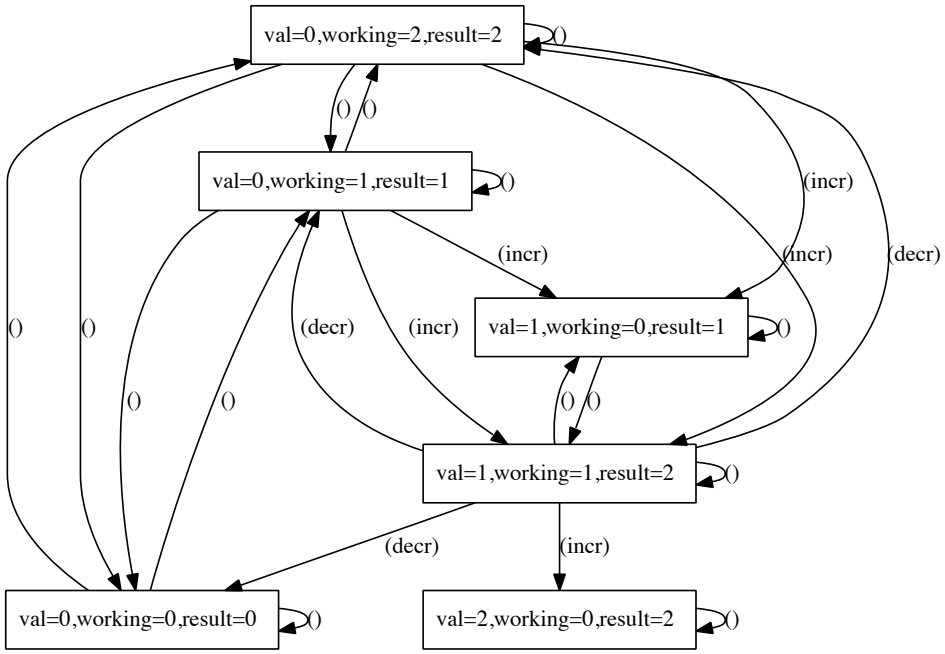


Figure 3.1: Transition system of a bounded counter.

$m = \langle g, e, a \rangle$. We now characterize the result of an assignment by a first order formula.

Definition 3.3 (**formulas denoting assignments**)**.** The formula $\mathrm{Aff}(t, X)$ denoting the assignments of a macro-transition $t = \langle g, e, a \rangle$ from the state $X \in \mathrm{dom}(V)$ is the formula built with

$$\mathrm{Aff}(\langle g, e, a \rangle, X) = \bigwedge_{v \in S} (v = (a\,(v)\,[X]))$$

where $a(v)[X]$ is the term $a(v)$ where variables are substituted by their values taken from X.

By using the pre, post-conditions and the logical characterization of an assignment, the transition relation is straightforward. Let Q be the set of state defined by

$$Q = \mathrm{dom}(S \cup F^{in} \cup F^{out}) \cap [\![\bigwedge_{o \in F^{out}} o = A(o)]\!].$$

We construct the labeled transition relation as follows.

Definition 3.4 (**Labeled transition relation**)**.** The labeled transition relation of a BioRica node $\mathcal{B} = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, A, M, \Gamma_C, W \rangle$is the relation $\rightarrow \subseteq \mathrm{Q} \times \Sigma \times \mathrm{Q}$ defined by

$$src \xrightarrow{e} tgt \Leftrightarrow \begin{cases} \exists t = \langle g, e, a \rangle \in M \\ tgt \in [\![\mathrm{Pre}(t)[src] \wedge \mathrm{Post}(t)[src] \wedge \mathrm{Aff}(t, src)[tgt] \wedge A[tgt]]\!] \end{cases}$$

The labeled transition system $\mathcal{S} = (Q, \rightarrow, \Sigma)$ is called the transition system of the node $\mathcal{B}$.

Example 3.2. We consider again the bounded counter (see example 3.1). We will explicitly build the transition relation for the unique macro transition m labeled by *incr*. For brevity of the following formulas, we resp. abbreviate the name of the variables $val, working$, and $result$ by the letters v, w, and r. The domain formula is defined by

$$D = (0 \leq v \leq 10) \wedge (0 \leq w \leq 10) \wedge (0 \leq r \leq 2).$$

The assertion A is defined by $A(r) = w + v$. Thus we have

$$\begin{aligned} Pre(m) &= 0 \leq r \leq 2 \wedge 0 \leq v \leq 2 \wedge 0 \leq w \leq 2 \wedge r = v + w \wedge w \geq 1 \\ Post(m) &= 0 \leq r \leq 2 \wedge 0 \leq v + 1 \leq 2 \wedge 0 \leq w \leq 2 \end{aligned}$$

The state space Q is defined by $Q = [\![0 \leq v \leq 10 \wedge 0 \leq w \leq 10 \wedge 0 \leq r \leq 2 \wedge r = w + v]\!]$ and thus we have $Q = [\![((0 \leq w < 2 \wedge 0 \leq v \leq 2 - w) \vee (w = 2 \wedge v = 0)) \wedge r = v + w]\!]$. Let $X = \langle \mathbf{v}, \mathbf{w}, \mathbf{r} \rangle$ be a configuration, the formula denoting the assignments of *incr* from X is $\mathrm{Aff}(m, X) = v = \mathbf{v} + 1$. Finally, for the configuration $X = \langle 0, 1, 1 \rangle$, and for any configuration $X' \in Q$ we have $X \xrightarrow{incr} X'$ iff $X' \in [\![v = 1]\!]$, that is $X' \in [\![(r = 1 \wedge v = 1 \wedge w = 0) \vee (r = 2 \wedge v = 1 \wedge w = 1)]\!]$.

The computation of the semantics of a BioRica node can be automatized as follows. We model a BioRica node as follows.

```
In[714]:= domainMapping = {w -> 0 ≤ w ≤ 10, v -> 0 ≤ v ≤ 10, r → 0 ≤ r ≤ 2}
          assertionTerms = {r → v + w}
          epsTrans = {True, "ϵ", {v → v}};
          Trans = {
            epsTrans,
            {w ≥ 1, "incr", {v → v + 1}},
            {w ≥ 1, "decr", {v → v - 1}}
           }
Out[714]= {w → 0 ≤ w ≤ 10, v → 0 ≤ v ≤ 10, r → 0 ≤ r ≤ 2}

Out[715]= {r → v + w}

Out[717]= {{True, ϵ, {v → v}}, {w ≥ 1, incr, {v → 1 + v}}, {w ≥ 1, decr, {v → -1 + v}}}
```

The domain, assertion, pre and post-conditions of the transition labeled with "incr" are the following conjunctions and substitutions.

```
In[718]:= domain = r ∧ v ∧ w /. domainMapping
          assertion = r == (r /. assertionTerms)
          Pret = domain ⋀ assertion ⋀ Trans[[2, 1]]
          Postt = (domain /. Trans[[2, 3]])
Out[718]= 0 ≤ r ≤ 2 && 0 ≤ v ≤ 10 && 0 ≤ w ≤ 10

Out[719]= r == v + w

Out[720]= 0 ≤ r ≤ 2 && 0 ≤ v ≤ 10 && 0 ≤ w ≤ 10 && r == v + w && w ≥ 1

Out[721]= 0 ≤ r ≤ 2 && 0 ≤ 1 + v ≤ 10 && 0 ≤ w ≤ 10
```

Then let `SolveFO` be a function mapping a first order quantifier free formula to the set of valuations satisfied by this valuation. We compute the set of transitions labeled with "incr" as follows.

```
In[712]:= srcs = SolveFO[Postt ⋀ Pret]

Out[712]= {{v → 0, w → 1, r → 1}, {v → 0, w → 2, r → 2}, {v → 1, w → 1, r → 2}}
```

```
In[713]:= Flatten[
           Map[
            Function[src,
             nextStateFormula = Map[#[[1]] → (#[[2]] /. src) &, trans[[3]]];
             nextVals = SolveFO[(domain ⋀ assertion /. nextStateFormula) ⋀
                 domain ⋀ assertion];
             {src → #, trans[[2]]} & /@ nextVals
            ], srcs]
           , 1]

Out[713]= {{{v → 0, w → 1, r → 1} → {v → 1, w → 0, r → 1}, incr},
           {{v → 0, w → 1, r → 1} → {v → 1, w → 1, r → 2}, incr},
           {{v → 0, w → 2, r → 2} → {v → 1, w → 0, r → 1}, incr},
           {{v → 0, w → 2, r → 2} → {v → 1, w → 1, r → 2}, incr},
           {{v → 1, w → 1, r → 2} → {v → 2, w → 0, r → 2}, incr}}
```

3.2 Node semantics in terms of stochastic mode automata

When a BioRica model only contains a single node, the transition system semantics defined in the previous section unambiguously defines its non stochastic behavior. In order to define the semantics of interacting nodes, we first define in this section an intermediate structure called the Stochastic Mode Automata (hereafter SMA). SMA are partial unfoldings of BioRica nodes, in which we preserve information about the type and domain of variables. By defining composition operators at the SMA level instead of the transition level (as it is done in the original Arnold-Nivat formalism), we can perform semantical checks before the applying the compositions. As we will see in section 3.3, we can use the types of variables and the assignments of macro transitions to verify the soundness of a *connection* or of a *synchronization* before applying it. We can thus restrict composition operations to the ones producing non empty transition systems. The structure of an SMA is defined as follows.

Definition 3.5 (**Non deterministic stochastic mode automata**)**.** A non deterministic stochastic mode automata (hereafter SMA) is a tuple $\mathcal{A} = (S, F^{in}, F^{out}, \text{dom}, \Sigma, \delta, \sigma, W, \Gamma)$ where

- S, F^{in}, F^{out} are mutually disjoint sets of variables called resp. *state variables, input variables, output variables*,
- $\text{dom} : S \cup F^{in} \cup F^{out} \to \mathcal{P}(Z)$ is an application that maps each variable of the node to its domain,
- Σ is a finite alphabet,

- δ is a mapping from $\mathrm{dom}(S) \times \mathrm{dom}(F^{in}) \times \Sigma$ to $\mathcal{P}(\mathrm{dom}(S) \times \mathrm{dom}(F^{in}))$ that maps a configuration and event to the set of successor configurations,

- σ is a partial mapping $\mathrm{dom}(S) \times \mathrm{dom}(F^{in})$ to $\mathrm{dom}(F^{out})$ that maps a configuration to the value of output variables,

- W is a mapping $\Sigma \to \mathbb{N} \cup \{\sharp\}$ called the weight of the even that maps every $e \in \Sigma$ to a real number or an undefined value. In the case where $W(e) = \sharp$, we say that e is *not weighted*,

- Γ is a mapping from $\Sigma \times \mathrm{dom}(V)$ to $\mathbb{F}^+$ that maps every event e and configuration v to a probability distribution $\Gamma(e, v)$ that has a support included in $\mathbb{R}_+$. For the special event ϵ we have, for any valuation $v \in \mathrm{dom}(V)$, $\Gamma(\epsilon, v) = \delta_0$. The distribution $\Gamma(e, v)$ is called *the delay of the event* e in the configuration v.

The semantics of a BioRica node is the SMA built using the following construction.

Definition 3.6 (BioRica node semantics). Let $\mathcal{B}$ be a BioRica node and let $\to$ be the labeled transition relation associated with $\mathcal{B}$ (see def. 3.4). The semantics $[\![\mathcal{B}]\!]$ of a BioRica node $\mathcal{B} = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, A, M, \Gamma_C, W\rangle$ is the SMA $[\![\mathcal{B}]\!] = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, \delta, \sigma, W, \Gamma\rangle$ defined as follows.

- Let $X = \langle \langle v_1, ..., v_n\rangle, \langle v_{in_1}, ..., v_{in_m}\rangle\rangle \in \mathrm{dom}(S) \times \mathrm{dom}(F^{in})$ be a configuration and let $\langle x_1, ..., x_n\rangle$ and $\langle y_1, ..., y_m\rangle$ be state and input variables of this configuration. The formula f characterizing X is

$$f_X = \left(\bigwedge_{i\in\{1,...,n\}} x_i = v_i \right) \wedge \left(\bigwedge_{i\in\{1,...,m\}} y_i = v_{in_i} \right),$$

  the global assertion of the node is

$$\mathrm{A} = D \wedge \bigwedge_{o\in F^{out}} (o = A(o)),$$

  and the mapping σ, from $\mathrm{dom}(S) \times \mathrm{dom}(F^{in})$ to $\mathrm{dom}(F^{out})$ is the partial application defined by

$$\sigma(v, v_{in}) = \pi_{F^{out}}([\![\mathrm{A} \wedge f_X]\!]) \text{ if } [\![\mathrm{A} \wedge f_X]\!] \neq \emptyset.$$

  where $\pi_X()$ is the projection of a valuation on a set of variables V onto the set of variables X.

- The mapping δ from $\mathrm{dom}(S) \times \mathrm{dom}(F^{in}) \times \Sigma$ to $\mathcal{P}(\mathrm{dom}(S) \times \mathrm{dom}(F^{in}))$ is the application defined by

$$\delta(v, v_{in}, e) = \left\{ \langle u, u_{in}\rangle \;\middle|\; \begin{array}{c} u \in \mathrm{dom}(S), u_{in} \in \mathrm{dom}(F^{in}), \\ \sigma(u, u_{in}), \sigma(v, v_{in}) \text{ are defined}, \\ \langle v, v_{in}, \sigma(v, v_{in})\rangle \xrightarrow{e} \langle u, u_{in}, \sigma(u, u_{in})\rangle \end{array} \right\}.$$

- The mapping Γ from $\Sigma \times \mathrm{dom}(V)$ to $\mathbb{F}^+$ is built with Γ_C and is defined for every $e \in \Sigma$ and for every $v \in \mathrm{dom}(V)$ by considering the distribution constructor $C_e = \Gamma_C(e)$ defined by

$$C_e = \left\langle \overrightarrow{t}, K \right\rangle \in \left(\left(\mathcal{T}^{S \cup F^{in}} \right)^n \times \left(\mathbb{R}^n \to \mathbb{R}^{[0,1]} \right) \right), C_e = \Gamma_C(e)$$

  and by accounting for the delay parameters, i.e. the vector of terms obtained after substitution of variables by their values in a given configuration. We thus have

$$\Gamma(e, v) = K(\overrightarrow{t}[v]) \text{ where } \left\langle \overrightarrow{t}, K \right\rangle = \Gamma_C(e).$$

Notice that in the previous definition, the application σ is a partial application, since not every valid valuation of state and input flow variables correspond to a valid valuation of output flow. Indeed, consider the following pathological example.

Example 3.3. The following program partially defines a BioRica node in which the domain of the output flow variables *outp* constraints the possible valuations of the state variable s and of the input flow variable *inp*. Indeed, the possible values for the variables $\langle s, inp \rangle$ are restricted to the set $\{\langle 0,0 \rangle ; \langle 0,1 \rangle ; \langle 1,0 \rangle\}$. Although other valuations are valid w.r.t. the domains of s and *inp*, they would violate the node assertion.

```
/*  ....  */
  state
    s :[0,100];
  flow
   inp :[0,100]: i;
   outp :[0,1]: o;
  assert
   outp=s*s*s*s*s*inp*inp*inp;
/*  ....  */
```

Given an SMA, rebuilding the transition relation (and thus the transition system semantics of a node) is immediate, as stated by the following property.

Property 3.1 (From SMA to transition systems). *Let $\mathcal{C}$ be a BioRica node, let $\mathcal{S} = (Q, \to, \Sigma)$ be the transition system of $\mathcal{C}$ and let $[\![\mathcal{C}]\!] = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, \delta, \sigma, W, \Gamma \rangle$ be the SMA denoting the semantics of $\mathcal{C}$. We have*

$$\langle v, v_{in}, v_{out} \rangle \xrightarrow{e} \langle v', v'_{in}, v'_{out} \rangle \Leftrightarrow \begin{cases} \delta(v, v_{in}, e) & = \langle v', v'_{in} \rangle \\ \sigma(v, v_{in}) & = v_{out} \\ \sigma(v', v'_{in}) & = v'_{out}. \end{cases}$$

Example 3.4. We describe here how the semantics of a BioRica node can be described in terms of a SMA. Let $\mathcal{B} = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, A, M, \Gamma_C, W \rangle$ be a BioRica node with $S = \{s\}$, $F^{in} = \{working\}$, $F^{out} = \emptyset$, $\Sigma = \{incr, decr\}$, $A = \mathbb{1}$, $P(e) = \sharp$ and $\Gamma(e) = \Delta_0$ for any $e \in \Sigma$. The macro transitions of $\mathcal{B}$ are defined by

$$M = \{(working, incr, s := s + 1); (working, decr, s := s - 1); (\mathbb{1}, \epsilon, \mathbb{1})\}$$

The semantics $[\![\mathcal{B}]\!]$ of this node is given by the SMA $\langle S,\ F^{in},\ F^{out},\ \mathrm{dom},\ \Sigma,\ \delta,\ \sigma,\ W,\ \Gamma\rangle$ defined with $\mathrm{dom}(s) = \{0,1,2\}$, $\mathrm{dom}(working) = \{\mathbb{0}, \mathbb{1}\}$ and where the transition relation δ is defined by

$$\delta = \begin{cases} (v_s, \mathbb{1}, incr) & \rightarrow (v_s + 1) \text{ if } v_s \in [0,1] \\ (v_s, \mathbb{1}, decr) & \rightarrow (v_s - 1) \text{ if } v_s \in [1,2] \\ (v_s, v_w, \epsilon) & \rightarrow (v_s) \end{cases}$$

3.3 Compositions

An important feature of a modeling language from a practical perspective is the possibility to define hierarchical systems. This allows for example to substitute a subsystem with a more detailed or more abstract one one without changing the global system. In order to describe systems as hierarchies of components, we define in this section three composition operators, namely *parallel composition*, *flow connections* and *synchronization*.

3.3.1 Parallel composition

The parallel composition of two SMA we define here is an adaptation of the free product (or interleaving composition [CGP99]). The parallel composition of two SMA produces a SMA in which each transition is a transition originating from one of the two SMA.

Definition 3.7 (**Parallel composition of SMA**)**.** The parallel composition of the n SMA $(\mathcal{A}_1 \parallel \mathcal{A}_2 \parallel ... \parallel \mathcal{A}_n)$ whose alphabets and sets of variables are disjoints is the SMA $\mathcal{A} = (S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, \delta, \sigma, W, \Gamma)$, where

$$S = \bigcup_{i=1}^{n} S_i, \qquad F^{in} = \bigcup_{i=1}^{n} F_i^{in},$$
$$F^{out} = \bigcup_{i=1}^{n} F_i^{out}, \qquad \Sigma = \bigcup_{i=1}^{n} \Sigma_i,$$

and where $W(e) = W_{A_i}(e)$, and $\Gamma(e) = \Gamma_{A_i}(e)$ for the unique A_i where $e_i \in \Sigma_i$, and where $\mathrm{dom}(v) = \mathrm{dom}_i(v)$ for the unique A_i where $v \in S_i \cup F_i^{in} \cup F_i^{out}$ and for any configuration $s = \langle s_1, ..., s_n\rangle$ of $\mathrm{dom}(S)$, for any configuration $in = \langle in_1, ..., in_m\rangle$ of $\mathrm{dom}(F^{in})$, we define for any $e_i \in \Sigma_i$

$$\begin{aligned} \delta(\langle s, in, e\rangle &= \{\langle s_1, ..., s_{i-1}, v, s_{i+1}, ..., s_n\rangle \mid v \in \delta_i \langle s_i, i_i, e\rangle\}, \\ \sigma(\langle s, in\rangle &= \langle \sigma_1 (s_1, in_1), ..., \sigma_n (s_n, in_n)\rangle . \end{aligned}$$

At the BioRica node level, parallel composition can be performed by glueing together all the clauses of the nodes we wish to compose. To avoid any name conflict (and to force the alphabet to be disjoints), we prefix each event and variable of the resulting SMA by a unique name.

Example 3.5. This example illustrate the result of a parallel composition performed at the syntax level by the `altatools` [Poi00]. Consider the BioRica system

```
node counter
    state
        s :[0,2];
    flow
        working:BOOL;
    event
        incr ,decr;
    trans
        working ⊢ incr  →s:=s+1;
        working ⊢ decr  →s:=s−1;
edon

node System
    sub
        c1,c2:counter;
edon
```

in which a system is defined by the parallel composition of two nodes *counter*. The syntactical composition produces the following program

```
node System
 flow
  c1.working : bool;
  c2.working : bool;
 state
  c1.s  :  [0,2];
  c2.s  :  [0,2];
 event
  '<c1.e,c2.e>',
  '<c1.e,c2.decr>',
  '<c1.e,c2.incr>',
  '<c1.decr,c2.e>',
  '<c1.incr,c2.e>';
 trans
     𝟙
   ⊢ '<c1.e,c2.e>' →;
    c1.working
  ⊢ '<c1.incr,c2.e>' →
  c1.s:=((c1.s) + (1));

    c1.working
  ⊢ '<c1.decr,c2.e>' →
  c1.s:=((c1.s) − (1));

    c2.working
  ⊢ '<c1.e,c2.incr>' →
  c2.s:=((c2.s) + (1));

    c2.working
  ⊢ '<c1.e,c2.decr>' →
  c2.s:=((c2.s) − (1));

edon
```

3.3.2 Flow connections

Flow connection are defined to model sharing of variable value between two nodes, by connecting an output flow variable to an input flow variable. This imply that in any configuration of the resulting SMA, the input variable value is always equal to the output variable value. This process is illustrated in fig. 3.2.

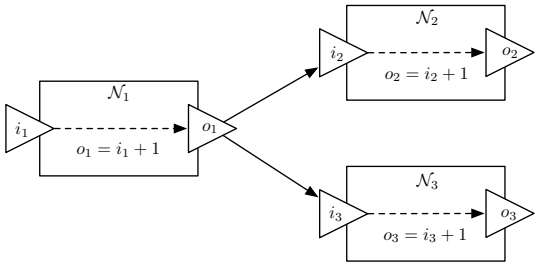


Figure 3.2: Illustration of connections between three BioRica nodes. Input variables i_2 and i_3 are made always equal to the output variable o_1. Furthermore, in each of the nodes $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, an output variable value is dependent of the input variable.

In order for a connection to be valid, it must introduce no loop. Consider for example the connection between the three nodes depicted in fig. 3.2 and suppose that we want to add a connection between o_3 and i_1. All configurations of the resulting system will be a solution of the system of equation

$$o_1 = i_1 + 1 \wedge i_2 = o_1 \wedge i_3 = o_1 \wedge o_2 = i_2 + 1 \wedge o_3 = i_3 + 1 \wedge i_1 = o_3,$$

admitting no solutions. In order to avoid these problematic connections, we adapt the notion of causality between variables from [Rau02].

Causality

A variable o is said to be causally independent of a variable i if the value of i cannot be used to determine the value of o. In other words, the variable i can be constant while the variable o ranges over its full domain. More formally we have:

Definition 3.8 (**Causality between input and output variables**)**.** Consider two variables $i \in F^{in}, o \in F^{out}$ of a given SMA. The variables i and o are said to be causally independent if, for any valuation v in $\mathrm{dom}(S)$ and for any valuation v_{in} in $\mathrm{dom}(F^{in})$ we have

$$\forall x \in \mathrm{dom}(i), \sigma(v, v_{in})(o) = \sigma(v, v_{in}[x/i])(o)$$

Semantics

For variables that are not causally dependent, we can define the connection composition operation in the following way.

Definition 3.9 (**Connection between causally independent variables**)**.** Let $\mathcal{B}$ be a BioRica node, A connection $\mathcal{I} = (i_1, ..., i_n; o)$ is a list of flow variables, s.t. for every input variables $i_k \in F^{in}$, i_k and $o \in F^{out}$ are causally independent.

Once a connection is established from an output to an input variable, it is possible to completely remove the input variable from the resulting system. However, since we restrict the guards of macro transitions to be formulas whose variables can only be input and state variables, it is not possible to simply replace each occurrence of an input variable by the connected output variable. However, since an output variable must be constrained by a term over state or input variables, it is possible to replace each occurrence of an input variable by this term.

Let $\mathcal{A} = (S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, \delta, \sigma, W, \Gamma)$ be an SMA, let $o \in F^{out}$ be an output variable and $i_1, \ldots, i_n \in Fin$ be a sequence of input variables such that o is causally independent of $i_1, \ldots, i_n \in Fin$.

The new set of input variables F^{in}_* is defined as $F^{in}_* = F^{in} - \{i_1, \ldots, i_n\}$. Then let $\vec{x} \in \mathrm{dom}()S)$ be a valuation of state variables and let $\vec{i} \in \mathrm{dom}(F^{in}_*)$ be a valuation of the new set of input variables, then the *updated valuation* $\vec{i}_{S,i_1,\ldots,i_n}$ is defined by

$$I_{S,i_1,\ldots,i_n}(v) = \begin{cases} I(v) \text{ if } v \in F^{in}_* \\ \sigma(S, I')(o) \text{ otherwise} \end{cases}$$

where I' is any extension of I into a valuation of $\mathrm{dom}(F^{in})$. Since the variables are assumed as being causally independent, $\sigma(S, I')(o)$ is the same, no matter how I' is built [Rau02]. We can now build the SMA $\mathcal{A} \propto \mathcal{I}$ implementing the connection $\mathcal{I}$.

Definition 3.10 (**Application of a connection**). The application $\mathcal{A} \propto \mathcal{I}$ of a connection $\mathcal{I} = (i_1, ..., i_n; o)$ on a SMA $\mathcal{A} = \langle S,\ F^{in},\ F^{out},\ \text{dom},\ \Sigma,\ \delta,\ \sigma,\ W,\ \Gamma\rangle$ is the SMA $\mathcal{A} \propto \mathcal{I} = (S, F^{in}_*, F^{out}, dom_*, \Sigma, \delta_*, \sigma_*, W, \Gamma)$ where dom_*, σ_* and δ_* are defined as follows:

- $\forall x \in \{S \cup F^{in}_* \cup F^{out}\}, dom_*(x) = \text{dom}(x)$
- For all valuation $\vec{v} \in \text{dom}(S)$, for all valuation $\vec{i} \in \text{dom}(F^{in}_*)$, for any event $e \in \Sigma$,

$$\delta_*(\vec{v}, \vec{i}, e) = \sigma(\vec{v}, \vec{i}_{S,i_1,...,i_n})$$

- For all valuations $\vec{x} \in \text{dom}(()S)$, for all valuations $\vec{i} \in \text{dom}(F^{in}_*)$,

$$\sigma_*(\vec{x}, \vec{i}) = \sigma_(\vec{x}, \vec{i}_{S,i_1,...,i_n})$$

Although implanting a connection modifies the set of variables of a node, the order in which successive connections are applied does not change the resulting SMA.

Proposition 3.2. *Let $\mathcal{A}$ be an SMA, let I, I' be connections $\mathcal{I} = (i_1, ..., i_n; o), \mathcal{I}' = (i'_1, ..., i'_n; o')$ such that $\mathcal{I}, \mathcal{I}'$ are valid connections for $\mathcal{A}$, I' is a valid connection for $\mathcal{A} \propto I$, and such that I is a valid connection for $\mathcal{A} \propto I'$ we have,*

$$(\mathcal{A} \propto \mathcal{I}) \propto \mathcal{I}' = (\mathcal{A} \propto \mathcal{I}') \propto \mathcal{I}.$$

A connection can be used to import the value of one variable into another node. Previous definition used assertions and substitutions to apply a connection on a SMA. As a result, in the transition system of the SMA with connection, states violating the assertion are forbidden. This approach is the one advocated in the original AltaRica formalism, which did not distinguished between assertions and connections, and is only feasible when analysis of a system is preceded by a flattening operation. However, in the context of hierarchical simulation of a BioRica system, which avoid completely the flattening operation, the semantics of connections must be interpreted differently. Instead, we present here a sequential interpretation, in which connections are interpreted functionally as propagation of the value of output variables in the hierarchy in order to update the value of input variables. We first determine the order in which such updates must be made.

Let $\mathcal{C}_1 = (S_1, F^{in}_1, F^{out}_1, \Sigma_1, A_1, M_1, W_1, \Gamma_1)$ and $\mathcal{C}_2 = (S_2, F^{in}_2, F^{out}_2, \Sigma_2, A_2, M_2, W_2, \Gamma_2)$ be two BioRica nodes, and suppose that we wish to connect $[\![C_1]\!]$ and $[\![C_2]\!]$ by using the connection $\mathcal{I}$. We suppose that this connection is valid and that it verifies the causality independence property. Let V be the set of variables of the two nodes, that is

$$V = S_1 \cup F^{in}_1 \cup F^{out}_1 \cup S_2 \cup F^{in}_2 \cup F^{out}_2.$$

And consider now the binary relation $\triangleright$ over V defined by

$$x \triangleright y \Leftrightarrow \begin{cases} \text{either} & \exists i \in \{1, ..., n\}, \mathcal{I}_i = \langle v_1, ..., y, ...v_j; x\rangle \\ \text{or} & x \in \text{vars}(A_1(y)) \cap F^{in}_1 \\ \text{or} & x \in \text{vars}(A_2(y)) \cap F^{in}_2. \end{cases}$$

We have the following property.

Property 3.3 (**Absence of cycle in the connections**). *If I is valid then the directed graph $G = \langle V, \triangleright \rangle$ is bipartite and acyclic.*

Proof. We suppose that I is valid. The fact that the graph is bipartite is straightforward from the definition of a connection. We can have $x \triangleright y$ only if one is an input variable and the other an output variable.

We will show that the graph is acyclic by contraposition. Suppose that $V, \triangleright$ admits a cycle and let $a_1, \ldots, a_n, a_1$ be an elementary cycle in G. Without loss of generality, we will suppose that a_1 is an input flow variables and that all the terms appearing in the assertions A_1 and A_2 do not use any arithmetic operation. In other words, the connections are simply equality between one output and one input variable. Since this cycle is elementary, no variable appears twice in the prefix $a_1, \ldots, a_n$. The semantics of connections imply that every configuration in the state space of the underlying transition system is solution of the system of equations

$$a_1 = a_2 \wedge a_3 = a_2 \wedge \cdots \wedge a_{n-1} = a_{n-2} \wedge a_{n-1} = a_n.$$

Consider some valuations $\vec{x} \in \mathrm{dom}(S), \vec{i} \in \mathrm{dom}(F^{in})$, and $c \in \mathrm{dom}(a_1)$, we have

$$\sigma(\vec{x}, \vec{i})(a_n) = \vec{i}(a_{n-1}) = \sigma(\vec{x}, \vec{i})(a_{n-1}) = ... = \vec{i}(a_1),$$

and thus a_1 and a_n are not causally independent. Hence, the connection is invalid. □

We can now use the previous relation $\triangleright$ to determine in which order variables must be updated when propagating flow variables values. Let $G = \langle V, \triangleright \rangle$ be the bipartite acyclic graph defined previously. The relation $\triangleright$ induces a partial order over flow variables. By linearizing G, we can build a total order $\prec$. Let $\prec$ be any total order built via linearization and let pre be an application that maps a variable $x \in V$ to a term in $\mathcal{T}^V$

(resp. minimal) element of any subset of V with respect to $\prec$. For any input variable i, let $\mathcal{I}_i$ be the output flow variable connected to i. In the case where i is not connected, we set $\mathcal{I}_i = i$. Finally, let $D = \mathrm{dom}(S \cup F^{in} \cup F^{out})$ be the set of possible valuations. We can now define the flow propagation function as follows.

Definition 3.11 (**Flow propagation function**). For any pair $(\vec{v}, V) \in (D \times \mathcal{P}(V))$ where $\vec{v}$ is a valuation in D and V is a set of variables, the *flow propagation function* fp maps $(\vec{v}, V)$ to a valuation in D and is recursively defined

$$fp(\vec{v}, V) = \begin{cases} \vec{v}'[\vec{v}'(I_{\hat{M}})/\hat{M}] & \text{if } \hat{M} \in F^{in} \\ \vec{v}'[[\![A(\hat{M})]\!]_{\vec{v}'}/\hat{M}] & \text{if } \hat{M} \in F^{out} \\ \vec{v} & \text{if } V = \emptyset. \end{cases}$$

where $\vec{v}' = fp\left(\vec{v}, V - \left\{\hat{M}\right\}\right)$, and where $[\![t]\!]_{\vec{v}}$ is the value of the term t when interpreted in the valuation $\vec{v}$ and where $\hat{M} = max_{\prec}(V)$ is the greatest element of the set V w.r.t. to the total order $\prec$.

Example 3.6. This example illustrates the result of applying the flow propagation function to update a valuation. Let $V = S \cup F^{in} \cup F^{out} = \{v_1, v_2\} \cup \{i_1, i_2\} \cup \{o_1, o_2\}$ be variables from a BioRica node $\mathcal{B}$ with the assertion A defined as

$$\begin{aligned} A(o_1) &= v_1 + i_1 \\ A(o_2) &= v_2 + i_2, \end{aligned}$$

and suppose we want to implement the connection $\mathcal{I} = \langle i_2, o_1 \rangle$, in order to yield a system with the connections $i_1 \rightarrow o_1 \rightarrow i_2 \rightarrow o_2$. One of the complete order $\prec$ over V (in fact for this example, it is the single) obtained by linearizing the relationship $\triangleright$ is

$$i_1 \prec o_1 \prec i_2 \prec o_2,$$

and thus we have for the valuation $\vec{v} = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : \bullet, o_1 : \bullet, o_2 : \bullet\}$ where $\bullet$ is any value,

$$fp\,(\vec{v}, \emptyset) = \vec{v} = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : \bullet, o_1 : \bullet, o_2 : \bullet\}$$

and $\vec{v}_1 = fp\,(\vec{v}, \{i_1\}) = \vec{v}[\vec{v}(i_1)/i_2] = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : \bullet, o_1 : \bullet, o_2 : \bullet\}$

and $\vec{v}_2 = fp\,(\vec{v}, \{i_1, o_1\}) = \vec{v}_1[11/o_1] = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : \bullet, o_1 : 11, o_2 : \bullet\}$

and $\vec{v}_3 = fp\,(\vec{v}, \{i_1, o_1, i_2\}) = \vec{v}_2[\vec{v}_2(o_1)/i_2] = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : 11, o_1 : 11, o_2 : \bullet\}$

and $\vec{v}_4 = fp\,(\vec{v}, \{i_1, o_1, i_2, o_2\}) = \vec{v}_3[18/o_2] = \{w_1 : 8, w_2 : 7, i_1 : 3, i_2 : 11, o_1 : 11, o_2 : 18\}$

We now state that it is sufficient to apply this function once to update all the functionally dependent variables.

Property 3.4. *The flux propagation function is idempotent when applied to the set of all variables of a SMA. That is, for any valuation $\vec{v} \in D$*

$$f_p(f_p(\vec{v}, V), V) = f_p(\vec{v}, V).$$

Finally, we have to relate the two semantics we have defined for the connections. Indeed, we have the following property.

Property 3.5. *Let $\mathcal{A}$ be a SMA, $A_I = A \propto (\mathcal{I}_1 ... \mathcal{I}_n)$ be the SMA obtained by implanting the connections $(\mathcal{I}_1 ... \mathcal{I}_n)$ as defined in definition 3.10. Let fp be the flow propagation function defined previously. We have for any valuation $\vec{v} \in \mathrm{dom}(S \cup F^{in}_*)$ of state of input flow variables,*

$$\sigma_*(\vec{v}) = f_P(\vec{v}, V)$$

where V is the set of variables $V = S \cup F^{in} \cup F^{out}$.

Example 3.7. This example shows the result of a flow connection performed at the syntactical level.

```
        node not_connected                          calc ;
2           state                              10   trans
              _s:BOOL;                                i_1 ⊢ calc →;
4           flow                               12   assert
              o_1:BOOL:o;                             o_1:=¬i_1;
6             o_2:BOOL:o;                      14     o_2:=¬_s;
              i_1:BOOL:i;                     edon
8           event
```

Once the connection between i_1 and o_1 is applied at the syntactical level, we obtain the following BioRica node.

```
1   node connected
        state
3           _s:BOOL;
        flow
5           o_1:BOOL:o;
            o_2:BOOL:o;
7       /* The input flow is removed */
        /*          i_1 :BOOL:i; */
9
        event
11          calc;
        trans
13
        /*          i_1 ⊢ calc →; */
15      /* Each occurrence of
        i_1 should be replaced by the output flow */
17
        /*          o_2 ⊢ calc →; */
19
        /* Still , output flow cannot appear in the guard */
21       => we replace it by the terms constraining it */

                ¬(_s) ⊢ calc →;
23
        assert
25          o_1:=¬(¬_s);
            o_2:=¬_s;
27      edon
```

3.3.3 Event synchronization

Synchronization semantics

Events in BioRica are considered to occur independently and therefore not simultaneously. In some cases, two distinct events occurring in two distinct nodes have to be simultaneous, or they model the same physical event but occurring in two different places. This leads to the notion of synchronization of events.

Synchronization between events is a mechanism to force the simultaneous occurrence of two separated events. The synchronization of two events modifying the same set of variables would lead to an incoherent transition. In order to avoid such synchronization, we define events' incompatibility and restrict the definition of synchronization to compatible events.

Definition 3.12 (**Incompatible events**). Let $v, v_{in} \in \mathrm{dom}(S) \times \mathrm{dom}(F^{in})$ and v a valuation $v = \langle v, v_{in} \rangle$, we say that the events $e_1, e_2 \in \Sigma$ are incompatible w.r.t. the valuation v if

$$\exists v' \in \delta(v,_{v_{in}}, e_1), \exists v'' \in \delta(v, v_{in}, e_2), \exists s \in S, v_s \neq v'_s \neq v''_s.$$

Definition 3.13 (**Synchronization vector for $\mathcal{A}$**). A *synchronization vector for* $\mathcal{A}$ is a list $\langle e_1, ..., e_n \rangle$, where for any $i, j \in \{0...n\}$, we have $e_i, e_j \in \Sigma$, $\Gamma_C(e_i) = \sharp$, and e_i, e_j are compatible.

The semantics of synchronization is given by the following definition.

Definition 3.14 (**Synchronization**). The SMA $\mathcal{A} = \langle S,\ F^{in},\ F^{out},\ \mathrm{dom},\ \Sigma,\ \delta,\ \sigma,\ W,\ \Gamma \rangle$ synchronized with $\vec{e} = \langle e_1, ..., e_n \rangle$ is the SMA $A \mid \vec{e} = \langle S,\ F^{in},\ F^{out},\ \mathrm{dom}, \Sigma_s,\ \delta_s, \sigma, W, \Gamma_s \rangle$ defined by

- $\Sigma_s = \Sigma - \vec{e} \cup \dot{e}$, where $\dot{e} = (e_1 \cdot e_2 \cdot \ ... \ \cdot e_n)$ is a new label built with the concatenation of the synchronized events
- For any v, v_{in}, e resp. in $\mathrm{dom}(S), \mathrm{dom}(F^{in}), \Sigma_S$, we define $\delta_S : \mathrm{dom}(S) \times \mathrm{dom}(F^{in}) \times \Sigma_S$ by

$$\delta_S(v, v_{in}, e) = \begin{cases} \delta(v, v_{in}, e) & \text{if } e \neq \dot{e} \\ \delta(\delta(...\delta(v, v_{in}, e_n), ...)e_2)e_1) & \text{otherwise.} \end{cases}$$

In the case where synchronization is applied on events having stochastic weights, the weight of the resulting synchronized event is arbitrarily set as follows.

Definition 3.15 (**Weight of a synchronized event**)**.** Let $\vec{e} = \langle e_1, ..., e_n \rangle$ be a synchronization vector, and let $\dot{e} = (e_1 \cdot e_2 \cdot \ ... \ \cdot e_n)$ be the label of the new event denoting the synchronization of $(e_1, e_2, \ ... \ , e_n)$ in $(A \mid \vec{e})$, we arbitrarily set the weight of $\dot{e}$ by

$$W(\dot{e}) = \begin{cases} \sharp & \text{if } \exists i \in \{1, ..., k\} \mid W(e_i) = \sharp \\ \prod\limits_{e_i \in v} W(e_i) & \text{otherwise.} \end{cases}$$

3.4 BioRica systems syntax and semantics

Now that we defined the possible composition operators, we can now define how the operations and node semantics put together results in are hierarchical systems, called BioRica systems.

Definition 3.16 (**BioRica systems**)**.** A BioRica system is a tuple $\mathcal{S} = \big\langle (A_1 ... A_n), (\vec{S}_1 ... \vec{S}_n), (C_1 ... C_n) \big\rangle$ where every A_i is either a node or a system whose sets of variables and alphabets are disjoints, every C_i is a connection, and every $\vec{S}_i$ is a synchronization vector.

Contrary to the general AltaRica formalism [AGPR00], the controller of a BioRica system is reduced to a set of connections between flow variables. The semantics of a BioRica system is finally defined by applying the operations defined in the previous sections.

Definition 3.17 (**Semantics of a BioRica system**)**.** The semantics $[\![\mathcal{S}]\!]$ of a BioRica system $\mathcal{S} = \big\langle (A_1 ... A_n), (\vec{S}_1 ... \vec{S}_n), (C_1 ... C_n) \big\rangle$ is the SMA $[\![S]\!] = (S, F^{in}, F^{out}, \text{dom}, \Sigma, \delta, \sigma, P, \Gamma)$ defined by

$$[\![\mathcal{S}]\!] = (([\![A_1]\!] \parallel ... \parallel [\![A_n]\!]) \propto (C_1 ... C_n)) \mid (\vec{S}_1 ... \vec{S}_n).$$

By the previous constructions, we showed that we can define nodes and hierarchical systems in BioRica, and that those two elements admits an equivalent semantics. This imply that any property or construction valid for a non hierarchical node also apply to hierarchical systems. This property fulfills the requirements of a compositional semantics.

3.5 Concluding remarks

In this chapter, we defined the syntax, the transition semantics and the compositional semantics of the BioRica language. This approach is based on the AltaRica DataFlow language defined in [Rau02, Rau01].

By extending the AltaRica formalism to account for stochastic information, we defined the abstract syntax of BioRica nodes, and their semantics in terms of stochastic mode automata. Stochastic mode automata embed a number of interesting features that are borrowed from existing formalisms.

- They are state/events based formalism, like finite state machines and reactive language,
- They are data flow oriented, like block diagrams,
- They can represent under-specified systems by non determinism, like finite automata,

- They can be defined by using a high level declarative language, like AltaRica systems,
- They can represent hierarchical systems via composition operations,
- They account for stochastic labeling of events, like discrete event systems.

We gave two alternative definitions of flow connections. The first definition is functional and is based on a node transformation. The second definition is operational and is based on a flow propagation function. The first definition is meant to be used when working at the semantical level of biorica, while the second is meant to be used for the simulation of hierarchical biorica systems. We defined an event synchronization mechanism, and which events are incompatible w.r.t. synchronization. We showed that we can verify events' compatibility either on the syntactical or semantical level. We showed that each of our composition operations can be applied either on the syntactical or on the semantical level.

Compared to AltaRica, variables and transitions in BioRica can only appear at the leaves of the hierarchy. Furthermore, assertions in BioRica are restricted to equality between an output variable and a term of state or input variables. As such, they are used to constraint to value of output flows. In AltaRica, assertions can be arbitrary quantifier free formulas and can be used to model complex dependencies between variables. Furthermore, the flow connections is specific to BioRica and the synchronization mechanism is defined as a binary operation on a SMA and a vector that yield a SMA. In AltaRica, these operations are applied on a transition system, and yields a transition system. Lastly, stochastic information can be associated with events in BioRica.

Finally in this chapter, we showed that this semantics is fully compositional, i.e. that a hierarchy of BioRica nodes can be flattened into a single node, and that this flattening operation preserves the transition semantics.

Chapter 4

Stochastic semantics

In this chapter, we introduce a transition system model that allows us to represent finite discrete state stochastic processes with arbitrary distributions. We call it *Stochastic Transition System* (hereafter STS) and is inspired by the Generalized Semi-Markov Processes[Whi80]. A STS is a transition system that have edges decorated with *timed events* and with a non deterministic transition relation.

A STS is a transition system extended with random timers. A random timer is simply a random variable that have a distribution specified by the stochastic labeling of the STS. To each enabled timed event is thus associated a timer whose value is a random variable denoting the inter event time, that is the duration in which the system must stays idle before this event fires. Once the event with the smallest inter-event time is fired, the system evolves instantaneously to a new state. Once the system reach a new state, a set of events are *newly enabled* and their timer are sampled in accordance to the probability distribution associated with the event.

In order to define the semantics of STS in terms of probabilities over possible executions, we determine the probability of a finite path as the joint distribution of the next state, next event and maximum sojourn time. This probability of finite paths formalize of the notion of *race condition* between continuous and discrete random variables.

Furthermore, the STS may be non deterministic transition systems for two reasons. First, given one state and one label, the transition relation may indicate more than one possible successors. Second, since we allow timers that are discrete random variables, the probability that two timer expire at the same time is not zero.

The transition system model we define in this chapter is based on the definitions of probabilistic transitions systems that dealt with discrete probabilities (e.g.[Seg96, VGSST95, LS89, PZ93, Var85]) and on the model of López et al. based on kripke structure representing Semi-Markov chains [LHK01]. In the case of continuous and discrete probabilities, our transition system is similar to the stochastic automaton of [DKB98], at the notable difference that we associate to STS a finite semantics.

Furthermore, we adapt the notion of *scheduler* [Seg96, Var85, D'A99] to resolve the non-determinism in a probabilistic way.

4.1 Stochastic Transition System

In this section, we define *stochastic transition systems* (hereafter STS) that are transitions systems with additional event labeling carrying probabilistic information. This probabilistic information is two fold: first, delays for firing of events, second, instantaneous probability of choosing an event in case of competing events.

With this two fold probabilistic information, stochastic transition systems can represent discrete state formalisms with both untimed and timed events. This generalizes both discrete Markov chains and continuous time Markov chains [Ros96]. Furthermore, we do not put any restrictions on the kind of probability distribution that can be associated with a transition, and thus STS are transition systems which stochastic semantics is in the semi Markov processes class.

Stochastic transition systems are obtained by unfolding the probabilistic information of SMA 3.5 and are defined here below.

Definition 4.1 (**Stochastic Transition System**)**.** A Stochastic Transition System (hereafter STS) is a tuple $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$where :

- Q is a finite set of states,
- $\longrightarrow \subseteq (Q \times \Sigma \times Q)$ is the set of labeled edges,
- Σ is a finite alphabet containing the letter ϵ,
- $W : \Sigma \nrightarrow \mathbb{N}_{>0}$ is the weight function,
- $\Gamma : Q \times \Sigma \nrightarrow \mathbb{F}^+$ is the clock distribution function.

Notice that in the previous definition, the transition relation $\longrightarrow$ is non deterministic. This implies that for a given state, multiple successor states are accessible via the same labeled transition. Furthermore, as we will detail in section 4.3, even if the transition relation $\longrightarrow$ is deterministic, the stochastic information may not be sufficient to decide on a single successor state. We illustrate this non determinism and the semantics of STS in the following example.

For any event $e \in \Sigma$, we say that e is *active in a state* q if and only if there exists a state q' such that $q \xrightarrow{e} q'$. Otherwise, we say that e is *inactive in* q. We will denote by $\mathrm{En}(q)$ the set of events that are active in the state q.

4.1.1 Stochastic Mode Automata Semantics

In the chapter Chap. 3, we have defined the semantics of BioRica nodes in terms of SMA. An SMA is a partial unfolding of a BioRica node where information about variables types is preserved. To compute the probability of executions of a SMA, such an information is no longer useful, and thus we unfold completely an SMA in terms of STS. This unfolding captures the stochastic information contained in the SMA by mapping a probabilistic distribution to each state-label pair of the STS.

Definition 4.2 (**Stochastic Mode Automata Semantics**)**.** The semantics $[\![\mathcal{A}]\!]$ of a stochastic mode automata $\mathcal{A} = \langle S, F^{in}, F^{out}, \mathrm{dom}, \Sigma, \delta, \sigma, W, \Gamma \rangle$ is the Stochastic Transition system $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$where $\langle Q, \longrightarrow, \Sigma \rangle$ is the labeled transition system associated with $\mathcal{A}$ (see definition 3.4).

4.2 Underlying stochastic process

The underlying stochastic process of a STS records the state of the STS as it evolves over continuous time. We define here the stochastic process in terms of a general state space Markov chain, that is a Markov chain taking values in an uncountably infinite set. [Dur05]. To prepare for this characterization, we first give a brief introduction to general state space Markov chains based on the presentation of [Haa02] and [Whi80].

4.2.1 General state space Markov chains

Consider a stochastic process where the future evolution of the process only depends on the past and present only through the current state. When the process evolves in discrete time and takes values in a finite of countably infinite space Ω, then the process is called a Discrete Time Markov Chain (see preliminaries for a discussion of DTMCs chap. 2). Whenever the DTMC is time homogeneous, that is that the probabilities of transitions do not depend on the time, then the DTMC is characterized by an initial distribution and a transition matrix M such that $M(i, j)$ denotes the probability starting in state i that the process next hit the state j.

When the state space Ω is uncountably infinite, the probability that the process hits a specified element of Ω typically is equal to 0, and a characterization in terms of transition matrix is not useful. The generalization of DTMCs to the uncountable case relies on the notion of *transition kernel*. A transition kernel P is an application that maps a state z and a set A to the probability that starting in state z, the chain then hits a state in A. Given an initial distribution and a transition kernel, we can define a stochastic process as follows.

Definition 4.3 (**General state-space Markov chain**)**.** A general state space Markov chain with initial distribution μ and transition kernel P taking values in Ω is a discrete time stochastic process $\{Z_n\}_{n\geq 0}$ where

$$P_\mu\{Z_0 \in A\} = \mu(A)$$

and

$$P_\mu\{Z_{n+1} \in A \mid Z_n, \ldots, Z_0\} = P(Z_n, A) \text{ almost surely}$$

for $n \geq 0$ and $A \subseteq \Omega$.

Typically, given a finite or countably finite set S of discrete state, the state space Ω has a product form $S \times \mathbb{R}^k$ for some $k \geq 1$ and have $A \in \mathcal{A}$ where the collection $\mathcal{A}$ of subsets of Ω is defined as all the sets

$$A = \{s\} \times [0, a_1] \times [0, a_2] \times \ldots \times [0, a_k]$$

where $s \in S$ and $(a_i)_{0\leq i\leq k} \in \mathbb{R}^+$.

Then, the *finite dimensional distributions* of the chain are computed by

$$\begin{aligned} & P_\mu\{Z_0 \in A_0, Z_1 \in A_1, \ldots, Z_n \in A_n\} \\ = & \int_{A_0} \mu(dz_0) \int_{A_1} P(z_0, dz1) \cdots \int_{A_{n-1}} P(z_{n-2}, dz_{n-1}) P(z_{n-1}, A_n) \end{aligned}$$

for $n \geq 0$ and $A_i \subseteq \Omega$.

The use of a transition kernel simplifies the description of discrete time stochastic processes that do not satisfy any memorylessness property. Indeed, the definition of a transition kernel is sufficiently generic to encode a non memory less process into a memory less process: It is sufficient to encode in an *augmented state* via *supplementary variables* the complete information that is necessary for the further behavior. The a general state-space Markov chain process is defined over the set of augmented states. In other words, by using additional variables ranging over infinite state space, a non Markov process can be converted in a Markov process.

In the case of discrete event systems, a calendar is required to determine a successor and this calendar is updated after each discrete transition. In order to model discrete event systems where such calendar queues are required, Matthes [Mat62] defined the class of *generalized semi-Markov processes* (GSMP). A GSMP is a stochastic process evolving via events over a finite or countably finite state space. In a GSMP, the destination and duration of each transition depends on which of the several possible events associated with the current state occurs first. GSMP are stochastic processes with memory, and in order to define a transition kernel (originally called a Markov Kernel), a general state space is defined over the possible discrete state of the system and the possible calendar values. The operations performed on the calendar in order to sample new delays, select the minimum, update the system global etc. are all encoded in a transition kernel. This is the approach followed e.g. by [Haa02] for *stochastic petri nets*, by Cassandras [CL08] and Zimmermann [Zim08] for *stochastic timed automata* (although both definitions differs). These approach are inherited by the original work of Whitt [Whi80] and surely by Matthes.[2]

With regards to probabilistic systems evolving through discrete transitions, the Markov property allows immediate derivation of an infinite state space DTMC. Indeed, for models satisfying the Markov property (and thus for general state space Markov chains), a small step semantics is immediate: Given an augmented state z, the probabilities of the successors after one step are immediately given by application of the transition kernel.

Still, as Asmussen and Glynn point out in [AG07],

> The transition structures of [GSMPs] are messy to write down. Nevertheless, the very fact that GSMPs can be viewed as Markov processes is conceptually important from a simulation standpoint, since this means that any methodology developed for Markov processes on a general state space applies (at least in principle) to discrete-event simulations.

However, although approaches based on transition kernel adequately define the semantics of GSMP (and thus of discrete event systems with constant clock rate [CL08]), this approach has not yet yielded automatic computation of the probability of a path. To this end, we give in this chapter an alternate definition of STS, not based on transition kernels but on the probability of finite paths decorated with timing information.

4.2.2 Paths and sojourn paths

Basically, given an initial configuration or *initial state*, a path of a STS is a path obtained by traversing the transition relation. We further consider paths decorated with timing information on transitions. This additional information represent the sojourn time in the successive states visited during the traversal following the path.

[2]The original Matthes' article published in 1967 could not have been translated.

Definition 4.4 (Path, sojourn path of a STS). Let $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ be a STS. An *path* of $\mathcal{S}$ is a finite sequence $w = (q_0, e_0, q_1, \ldots, q_{n-1}, e_{n-1}, q_n)$ such that for all $i \geq 0$, we have $q_i \xrightarrow{e_i} q_{i+1}$. A *sojourn path* of $\mathcal{S}$ is a sequence $\omega = (q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_n)$ such that for all $i \geq 0$, we have $q_i \xrightarrow{e_i} q_{i+1}$ and $t_i \in \mathbb{R}^+ \cup \{\infty\}$. The length $\text{card}(\omega)$ of a sojourn path ω is the number of transitions of ω, for instance we have here $\text{card}(\omega) = n$.

We will denote by $\omega \mid i$ the prefix of ω of length $\text{card}(\omega \mid i) = i$ and by $\omega \mid -i$ the prefix of ω of length $\text{card}(\omega \mid -i) = \text{card}(\omega) - i$. Finally for two sojourn path ω_1 and ω_2 such that the last state of ω_1 is the first state of ω_2, we will denote by $\omega_1.\omega_2$ the concatenation of the two sequences with the the first and last state merged.

Sojourn paths are a finite representation of an infinite number of execution of a STS. Let $\omega = (q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_n)$ be a sojourn path of a STS $\mathcal{S}$, the set $[\![\omega]\!]$ of executions of $\mathcal{S}$ that are represented by the sojourn path ω is the set defined by

$$[\![\omega]\!] = \left\{ w = q_0, (e_0, t'_0), q_1, (e_1, t'_1), \ldots, q_{n-1}, (e_{n-1}, t'_{n-1}), q_n \mid \forall 0 \leq i \leq n, t'_i \leq t_i \right\}.$$

In words, it is the set consisting of all executions whose sojourn time are less than the corresponding sojourn time of the sojourn path.

The distinction between sojourn paths and executions is essential with regards to measure theory and how to characterize the stochastic process underlying a STS. Indeed, since the random variables T_i representing the sojourn in a state before the event e_i fire can be a continuous r.v., the probability of the specific execution $q_i \xrightarrow{e_i, t} q_{i+1}$ will be zero almost surely for any value of t. Therefore we are not interested in measuring the probability of a specific execution but the probability of all the executions represented by a sojourn path. We formalize this approach by defining a stochastic process whose finite realizations are exactly the executions of a STS.

4.2.3 Paths as realizations of a stochastic process

We now define in stochastic process theory terms the relationships between STS, the underlying stochastic process and sojourn path.

Given a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$, we define the underlying stochastic process of $\mathcal{S}$ as a discrete time stochastic process $P = \langle Q_n, T_n, E_n \rangle_{n \in \mathbb{N}}$, where

- Q_n is a r.v. taking values in Q called *the state of the process at the nth step*,
- T_n is a r.v. taking values in R^+ called *the sojourn time of the process in the nth state after n steps*,
- E_n is a r.v. taking values in Σ called *the event label of the transition from the nth state to the $(n+1)$th state.*

We first impose that the finite realizations of P are exactly the executions $\mathcal{S}$. That is, if for some $(q, e, q') \in Q \times Q \times \Sigma$, we have $q \not\xrightarrow{e} q'$ then for any $t \in \mathbb{R}^+ \cup \{+\infty\}$, for any $n \geq 0$, we set $\mathbf{P}(Q_{n+1} = q' \mid Q_n = q, E_n = e, T_n \leq t) = 0$.

Consider now a sojourn path ω of length n in the STS $\mathcal{S}$, that is

$$\omega = (q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_n)$$

then the nth finite dimensional measure of P is used to compute the probability of ω (in the space of P) as

$$\mathbf{P}_P(Q_0 = q_0, E_0 = e_0, T_0 \leq t_0, Q_1 = q_1, E_1 = e_1, T_1 \leq t_1, \ldots, Q_n = q_n, E_n = e_n, T_n \leq t_n).$$

We define the probability $\mathbf{P}(\omega)$ of a sojourn path in STS $\mathcal{S}$ as

$$\mathbf{P}_P(\omega) = \mathbf{P}_{\mathcal{S}}(\omega).$$

That is, the semantics of $\mathcal{S}$ is the law of the stochastic process P. With this approach, we have a direct correspondence between the set of executions satisfying a sojourn path and the finite dimensional probability measures of the stochastic process $\langle Q_n, T_n, E_n \rangle$.

4.3 Finite dimensional measures of the underlying stochastic process

4.3.1 Overview of the method

In the definitions of the previous subsection (Sec. 4.2.3), we merely specified the stochastic process P without giving any method to build it (or equivalently to build its finite dimensional measure). Although the stochastic process may be built directly with a transition kernel (e.g. see [Haa02] for a completely defined GSSMC over markings of stochastic petri nets), we motivate here our alternative approach on a small STS example. Consider a three state STS.

Suppose that $\Gamma(e)$ is absolutely continuous for any $e \in \Sigma$, and consider the variable time advance simulation algorithm[She93] given in Alg. 1. This simulation algorithm is a mean to generate executions of the STS $\mathcal{S}$, or equivalently, finite realizations of the underlying stochastic process.

The algorithm Alg. 1 can be used to generate a set R of sample realizations of the underlying stochastic process of $\mathcal{S}$ and thus can be used to estimate the probability of a sojourn path ω as the ratio $\mathbf{P}_{\mathcal{S}}(\omega) = \text{card}(\{r \in R \mid r \in [\![\omega]\!]\})/\,\text{card}(R)$.

Furthermore in algorithm Alg. 1, given a probability distribution $\mathcal{P}$, the function $Sample(\mathcal{P})$ returns a value x such that $\mathbf{P}(X \leq x) = \mathbf{P}(Sample(\mathcal{P}) \leq x)$ where $X \sim \mathcal{P}$. Repeated calls to $Sample(\mathbf{P})$ are supposed to be (probabilistically) independent. In other words, the sequence $(\Gamma(\mathbf{P}), \Gamma(\mathbf{P}), \ldots)$ is a sequence of independent and identically distributed r.v. $(X_0, X_1, \ldots)$ such that $X_i \sim \mathcal{P}$. Suppose now that with Alg. 1, we generate the realization $X \xrightarrow{a,2.2} X \xrightarrow{b,0.2} Y \xrightarrow{c,1.09} Z$. During the execution of the algorithm that yielded this realization, the sequence of calls to $Sample$ were

$$(Sample(\Gamma(a)), Sample(\Gamma(b)), Sample(\Gamma(a)), Sample(\Gamma(c)))$$

and this call sequence returned the values $(2.2, 2.4, 9.2, 1.09)$.

Suppose now that we modify the function $Sample$ such that, for a given distribution, it always returns the same sequence of values. Then, since the algorithm is deterministic and fully determined by the values returned by successive calls to $Sample$, this algorithm will always return the same realization. That is, then if we modify $Sample$ such that

- $Sample(\Gamma(a)) = (2.2, 2.2, \ldots)$,
- $Sample(\Gamma(b)) = (2.4, 2.4, \ldots)$,

Algorithm 1 Variable time advance simulation algorithm for the STS $\mathcal{S}$. The variable *Clock* is a dictionary like data structure.

Require: $MAX \in \mathbb{N}$
  $epoch := 0, state := X, result := (X)$
  $E := \left\{ e \in \Sigma \mid \exists q' \in Q, state \xrightarrow{e} q' \right\}$
  for all $e \in E$ **do**
    $clock(e) := Sample(\Gamma(e))$
  end for
  while card$(result) < MAX \wedge state \neq Z$ **do**
    $event' := argmin(clock), epoch' := min(clock)$
    $E' := E - \{event'\}$
    $state' := \left\{ q' \in Q \mid state \xrightarrow{e} q' \right\}$
    $result = result.(epoch' - epoch, event').(state')$
    for all $e \in E'$ **do**
      $clock'(e) := clock(e)$
    end for
    $E := \left\{ e \in \Sigma \mid \exists q', state' \xrightarrow{e} q' \right\}$
    for all $e \in E - E'$ **do**
      $clock'(e) := epoch' + Sample(\Gamma(e))$
    end for
    $E := E', epoch := epoch', state := state', clock' := clock$
  end while
  return *result*

- $Sample(\Gamma(c)) = (1.09, 1.09, \dots)$,

then the algorithm will always return the realization $X \xrightarrow{a,2.2} X \xrightarrow{b,0.2} Y \xrightarrow{c,1.09} Z$. In other words, the simulation algorithm is deterministic and probabilistic.

Consider now the inverse problem, given a sojourn path $\omega = (X, a, 2.2, b, 0.2, Y, c, 1.09, Z)$, what are the possible sequence of calls to *Sample* and their returned values that yields a realization in $[\![\omega]\!]$? Given the fact that the first state of the system is X, the algorithm will always start with the call sequence $(Sample(\Gamma(a)), Sample(\Gamma(b))$. Let the sequence of r.v. (A_0, B_0) represents the values return by *Sample*. We know that if $\{A_0 \leq B_0 \wedge A_0 \leq 2.2\}$, then the system will step from X to X via the event a before the epoch 2.2. Once the system hits the state X again, a new value is sampled via a call to $Sample(\Gamma(a))$. Let A_1 be a r.v. representing the value returned by this call. We know that if $\{B_0 \leq A_0 + A_1 \wedge B_0 \leq 2.2 + 0.2\}$, then the system will choose the event b and will step out of the state X after sojourning in X for less than 0.2 time units. Once the system hits the state Y, a call to $Sample(\Gamma(c))$ is made and the system will step out of state Y in less than 1.09 t.u. if $C_0 \leq 1.09$. Finally, the realization r returned by the algorithm is in $[\![\omega]\!]$ if and only if

$$\{A_0 \leq B_0 \wedge A_0 \leq 2.2 \wedge B_0 \leq A_0 + A_1 \wedge B_0 \leq 2.2 + 0.2 \wedge C_0 \leq 1.09\}$$

which is equivalent to the set $D =$

$$\{(0 \leq A_0 \leq 2.2) \wedge (A_0 \leq B_0 \leq A_0 + 0.2) \wedge (B_0 \leq A_1 \leq +\infty) \wedge (0 \leq C_0 \leq 1.09)\}.$$

Now given that $\{A_0, A_1, B_0, C_0\}$ are independent, given that $A_i \sim \Gamma(a), B_i \sim \Gamma(b), C_i \sim \Gamma(c)$, we have

$$\begin{aligned}
&\mathbf{P}_P(\omega) = \mathbf{P}_{\mathcal{S}}(\omega) = \mathbf{P}_{\mathbb{R}^4}(D) \\
&= \int_D f_A BAC(a_0, b_0, a_1, c_0) \\
&= \int_D f_A(a_0) \cdot f_B(b_0) \cdot f_A(a_1) \cdot f_C(c_0) \\
&= \left(\int_0^{2.2} f_A(a_0) \int_{a_0}^{a_0+0.2} f_B(b_0) \int_{b_0}^{+\infty} f_A(a_1) da_1 db_0 da_0 \right) \cdot \int_0^{1.09} f_C(c_0) dc_0.
\end{aligned}$$

where f_A, f_B, f_C are resp. the probability density functions of $\Gamma(a), \Gamma(b)$ and $\Gamma(c)$.

The rest of this section generalizes this construction. To this end, we start with the simplest STS we can think of: finite state discrete time Markov Chains. We then gradually introduce more complicated cases as follows. The subclasses of STS that we consider are:

1. STS_1 are STS that have a DTMC like semantics
2. STS_2 are non-deterministic STS
3. STS_3 are STS that have a single state with timed transitions whose delay follows a continuous distribution
4. STS_4 relax the continuous restriction of delays of STS_3
5. STS_5 are STS that have only one regeneration state
6. STS_6 are STS without restrictions.

4.3.2 Probability of a sojourn path in STS_1: Accounting for immediate transitions

The first subclass STS_1 of STS that we consider are STS where the stochastic delay associated with every event is a delay following a degenerate distribution. We will show that STS_1 is exactly the class of DTMC.

Definition 4.5 (STS_1)**.** A STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ is in STS_1 if

1. for every event $e \in \Sigma$, we have $supp(\Gamma(e)) = \{0\}$,
2. for every event $e \in \Sigma$, the weight $W(e)$ is defined,
3. the transition relation $\rightarrow$ is deterministic, that is for any state q, any event e, the set of states $\left\{ q' \mid q \xrightarrow{e} q' \right\}$ is either empty or a singleton.

The probability of a sojourn path ω depends on the events that are competing at each successive state (q_i). Let $\mathrm{En}(q)$ be the subset of events of Σ that are enabled in q. That is, $\mathrm{En}(q) = \left\{ e \mid \exists q' \in Q, q \xrightarrow{e} q' \right\}$.

Definition 4.6 (Measure $\mathbf{P}_1$). For a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ in STS_1, for any sojourn path $\omega = (q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_n)$ of length greater or equal than 1, the probability $\mathbf{P}_1(\omega)$ is

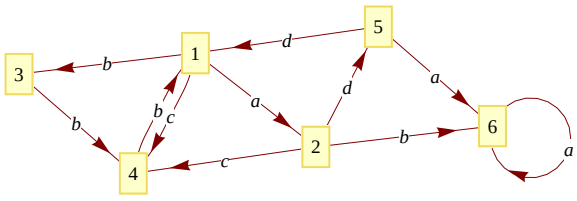
$$\mathbf{P}_1(\omega) = \prod_{e_i \in \omega} \frac{W(e_i)}{\sum_{e \in \mathrm{En}(q_i)} W(e)}.$$

If ω has a length of 0, we set $\mathbf{P}_1(\omega) = 1$.

Simulation

Generating sample timed paths from a STS in STS_1 by simulation fundamentally consists of selecting the successor of a state by accounting for the weight of competing events. We illustrate the simulation algorithm on the STS defined by the following code.

```
Σ = {a, b, c, d};
W = {a → 5, b → 2, c → 1, d → 4};
Q = {1, 2, 3, 4, 5, 6};
Γ = {a → 0, b → 0, c → 0, d → 0};
S = {
    {1 → 2, a}, {1 → 3, b}, {1 → 4, c},
    {2 → 6, b}, {2 → 5, d}, {2 → 4, c},
    {3 → 4, b},
    {5 → 6, a}, {5 → 1, d}, {4 → 1, b},
    {6 → 6, a}
   };
GraphPlot[S, DirectedEdges → True, VertexLabeling → True]
```



In order to select the successor of the current state q, we map each of the event that are enabled in q to a normalized weight and then use this normalized weight as the probability of selecting this event in a non uniform discrete distribution. Given a function `Choice` to perform non uniform discrete sampling, computation of the successor state is illustrated by the following code.

```
In[7]:= EnabledIn[s_] := Select[S, #[[1, 1]] == s &][[All, 2]]
        TransitionFrom[s_, e_] := Select[S, MatchQ[#, {s → _, e}] &][[1]]

In[9]:= EnabledIn[1]
Out[9]= {a, b, c}

In[10]:= TransitionFrom[1, b]
Out[10]= {1 → 3, b}

In[15]:= Step[s_] := Module[{enabled, totalWeight, nextE},
           enabled = EnabledIn[s];
           totalWeight = Total[enabled /. W];
           nextE = RandomChoice[enabled /. W / totalWeight → enabled];
           Return[TransitionFrom[s, nextE][[1, 2]]]
         ]
         NestList[Step, 1, 4]
         NestList[Step, 1, 4]
         NestList[Step, 1, 4]
Out[16]= {1, 2, 5, 6, 6}

Out[17]= {1, 3, 4, 1, 2}

Out[18]= {1, 2, 4, 1, 2}
```

The function `Step` can be nested n times on an initial state to generate samples executions of length $n+1$. We use this function to estimate the probability of the sojourn path $\omega = (1, a, +\infty, 2, d, +\infty, 5, d, +\infty, 1, b, +\infty, 3)$. Compared to the exact probability $5/126$, after 10000 samples, the difference between the estimate and the exact value is approximately 3.10^{-4}.

```
In[95]:= samples = Table[NestList[Step, 1, 4], {10 000}];
         ω[p_] := p == {1, 2, 5, 1, 3}
         Length[Select[samples, ω]]
                  Length[samples]

Out[97]= 429
         10 000
```

```
In[98]:=      a /. W                d /. W                d /. W          b /. W
         ------------- * ------------- * ------------- * -------------
         Total[Σs[1] /. W]   Total[Σs[2] /. W]   Total[Σs[5] /. W]   Total[Σs[1] /. W]

Out[98]= 5
         126
```

```
In[99]:= N[Norm[5/126 - 429/10 000]]

Out[99]= 0.00321746
```

Properties of STS in STS_1

We show here that our stochastic semantics is sound with regard to the usual interpretation of discrete-time Markov chains. Let $\mathcal{M} = \langle Q, M\rangle$ be a DTMC with n states (see sec. 2.5). The STS interpretation of $\mathcal{M}$ is the STS_1 $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma\rangle$ built with

1. $\Sigma = \{e_{ij} | 1 \leq i \leq n, 1 \leq j \leq n\}$,
2. $\Gamma(e) = \delta_0$,
3. $W(e_{ij}) = m_{ij}$,
4. $q_i \xrightarrow{e_{ij}} q_j$ iff $m_{ij} \neq 0$.

The idea behind the construction is to map each possible transition of $\mathcal{M}$ to an event in $\mathcal{S}$. We now show that both process are probabilistically equal.

Property 4.1 (Soundness of $\mathbf{P}_1$). *For this DTMC $\mathcal{M}$ and this STS $\mathcal{S}$, for any finite path $\omega = q_0 q_1 \dots q_n$ of $\mathcal{M}$ such that $\mathbf{P}_{\mathcal{M}}(\omega) > 0$ we have*

$$\mathbf{P}_{\mathcal{M}}(\omega) = \mathbf{P}_1(q_0 \xrightarrow{e_{01}} q_1 \xrightarrow{e_{12},0} q_2 \dots \xrightarrow{e_{n-1n},0} q_n)$$

Similarly, for any sojourn path ω of $\mathcal{S}$ such that $\mathbf{P}_1(\omega) > 0$, then there exists a corresponding path ω' in $\mathcal{M}$ such that $\mathbf{P}_1(\omega) = \mathbf{P}_{\mathcal{M}}(\omega)$.

Proof. For a path of length 1, both measure yields a probability of 1. Suppose that for a path ω of $\mathcal{M}$ that has a length $k-1$, we build a path ω' of $\mathcal{S}$ as stated in the property, we have

$\mathbf{P}_{\mathcal{M}}(\omega) = \mathbf{P}_1(\omega')$. Let q_k be the kth state of ω. We have $\mathbf{P}_{\mathcal{M}}(\omega \cdot q_k) = \mathbf{P}_{\mathcal{M}}(\omega) * m_{k-1,k}$, while on the other hand we have

$$\mathbf{P}_1(\omega' \xrightarrow{e_{k-1,k}} q_k) = \mathbf{P}_1(\omega') \times \frac{W(e_{k-1,k})}{\sum_{e \in \mathrm{En}(q_{k-1})} W(e)}.$$

By construction, $\mathrm{En}(q_{k-1}) = \{e_{k-1,j} \mid m_{k-1,j} \neq 0\}$. Furthermore, we know by definition of a DTMC that $\sum_{j \in Q} m_{k-1,j} = 1$, therefore $W(e_{k-1,k}) / \left(\sum_{e \in \mathrm{En}(q_{k-1})} W(e)\right) = W(e_{k-1,k}) = m_{k-1,k}$, and thus $\mathbf{P}_{\mathcal{M}}(\omega \cdot q_k) = \mathbf{P}_1(\omega' \xrightarrow{e_{k-1,k}} q_k)$. □

4.3.3 Probability of a sojourn path of STS_2: Accounting for non determinism

The probabilistic labeling is not always enough to completely determine the next state of an execution. In the case where the execution of a STS ends in such a non deterministic state, we need to rely on an additional mechanism to resolve conflicts in competing events. To this end, we define an additional variable weighting mechanism: weight schedulers. Weight schedulers are required to compute the probability of a path in a STS with non-deterministic transition relation or whose weight labeling is not defined for each event. More formally the class STS_2 is defined as follows.

Definition 4.7 (STS_2)**.** A STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ is in STS_2 if for every event $e \in \Sigma$, we have $supp(\Gamma(e)) = 0$.

In order to compute the probability of a sojourn path of a STS in STS_2, we require a *weight scheduler* that assign a weight to every enabled events in a given state.

Definition 4.8 (**Weight scheduler**)**.** Let $\mathcal{S}$ be a STS. A weight scheduler $\mathcal{W}$ for $\mathcal{S}$ is an application from $\{(q_i, e_i)_{i \in \mathbb{N}}\}^{\mathbb{N}}$ to $(\Sigma \times Q)^{\mathbb{N}}$ that maps a path $q_0 \xrightarrow{e_0} q_1 \xrightarrow{*} q_n$ of $\mathcal{S}$ to an application (W) that maps a pair of label and state to a (integer) weight value.

Let $w = q_0 \xrightarrow{*} q_{n-1}$ be a path of $\mathcal{S}$ such that there exists at least one outgoing transition $q_{n-1} \xrightarrow{e} q_n$ from the state q_{n-1} (i.e. it is not a deadlock state). A weight scheduler is said to be valid with regards to $\mathcal{S}$if for every path w, we have

- $\mathcal{W}(\omega, e, q_n) = 0$ if $q_{n-1} \not\xrightarrow{e} q_n$,
- $\mathcal{W}(\omega, e, q_n) = W(e)$ if $W(e)$ is defined for every e in $\mathrm{En}(q_{n-1})$,
- There exists at least a pair (e, q_n) of label and state such that $\mathcal{W}(\omega, e, q_n) \neq 0$.

In the following, we will only consider valid schedulers. We can now extend the definition of $\mathbf{P}_1$ to account for the non-deterministic choices.

Definition 4.9 (**Measure $\mathbf{P}_2$**)**.** For a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ in STS_2, for a weight scheduler $\mathcal{W}$, for any sojourn path $\omega = (q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_n)$ of length greater or equal than 1, the probability $\mathbf{P}_{2\mathcal{W}}(\omega)$ is

$$\mathbf{P}_{2\mathcal{W}}(\omega) = \prod_{q_i \in \omega} \frac{\mathcal{W}(\omega \mid i, e_i, q_{i+1})}{\sum_{e \in \mathrm{En}(q_i)} \sum_{\left\{q' \mid q \xrightarrow{e} q'\right\}} \mathcal{W}(\omega \mid i, e, q')},$$

If card($omega$) $= 0$, we set $\mathbf{P}_{2\mathcal{W}}(\omega) = 1$.

Simulation in STS_2

To generate sample execution of a STS in STS_2, we select the successor of a state according to both the weights of competing events and the weights returned by a given weight scheduler.

```
In[28]:= Step[ω_, 𝒲_] := Module[{sfin, enabled, newWeights, totalWeight, nextE},
           sfin = ω[[-1, 2]];
           enabled = EnabledIn[sfin];
           newWeights = 𝒲[ω, #] & /@ enabled;
           totalWeight = Total[newWeights];
           nextE = RandomChoice[newWeights/totalWeight → enabled];
           Return[Append[ω, {sfin, TransitionFrom[sfin, nextE][[1, 2]], nextE}]]
         ]

         𝒲1[ω_, σ_] := If[W[σ] =!= #, W[σ], 1]
         Step1[ω_] := Step[ω, 𝒲1]

         Nest[Step1, {{1, 1, "init"}}, 5]
         Nest[Step1, {{1, 1, "init"}}, 5]
         Nest[Step1, {{1, 1, "init"}}, 5]

Out[31]= {{1, 1, init}, {1, 2, a}, {2, 6, b}, {6, 6, a}, {6, 6, a}, {6, 6, a}}

Out[32]= {{1, 1, init}, {1, 3, b}, {3, 4, b}, {4, 1, b}, {1, 2, a}, {2, 6, b}}

Out[33]= {{1, 1, init}, {1, 2, a}, {2, 6, b}, {6, 6, a}, {6, 6, a}, {6, 6, a}}
```

With a sample size of 10000, we compute the most and less probable paths.

```
In[34]:= samples = Table[Nest[Step1, {{1, 1, "init"}}, 10], {10 000}];

In[35]:= ωlow = SortBy[Tally[samples], #[[2]] &][[2, 1]]
         ωhigh = SortBy[Tally[samples], #[[2]] &][[-1, 1]]

Out[35]= {{1, 1, init}, {1, 2, a}, {2, 4, c}, {4, 1, b}, {1, 2, a},
          {2, 4, c}, {4, 1, b}, {1, 2, a}, {2, 4, c}, {4, 1, b}, {1, 4, c}}

Out[36]= {{1, 1, init}, {1, 2, a}, {2, 6, b}, {6, 6, a}, {6, 6, a},
          {6, 6, a}, {6, 6, a}, {6, 6, a}, {6, 6, a}, {6, 6, a}}

In[37]:= {Length[Select[samples, # == ωlow &]] / Length[samples],
          Length[Select[samples, # == ωhigh &]] / Length[samples]}
         N[%]

Out[37]= {1/10 000, 639/5000}

Out[38]= {0.0001, 0.1278}
```

The probability of path can be computed exactly, since it only involves multiplication of rational numbers.

```
In[39]:= Prob2[ω_, 𝒲_] := Module[{},
           MapIndexed[
            Module[{ωi, prefix, qi, σi, conflicting, currentWeights, totalWeight},
              ωi = ω[[First[#2]]];
              prefix = ω[[ ;; First[#2]]];
              σi = ωi[[3]];
              qi = ωi[[1]];
              conflicting = EnabledIn[qi];
              currentWeights = 𝒲[prefix, #] & /@ conflicting;
              totalWeight = Total[currentWeights];
              𝒲[prefix, σi] / totalWeight
            ] &, ω]
          ]

In[40]:= {Times @@ (Prob2[ωlow[[2 ;;]], 𝒲1]), Times @@ (Prob2[ωhigh[[2 ;;]], 𝒲1])}
         N[%]

Out[40]= {1/16 384, 1/8}

Out[41]= {0.0000610352, 0.125}
```

As we saw, this probability depends on the weight scheduler. For example, we can modify

the previous model by using a weight scheduler that disables the event d whenever it has been fired more than 2 times.

```
In[42]:= 𝒲2[ω_, σ_] := If[W[σ] =!= #, W[σ],
           With[{dcount = (d /. (#[[1]] → #[[2]] & /@ Tally[ω[[2 ;;, 3]]] ))},
            If[dcount =!= d ∧ dcount ≥ 2 ∧ ω[[-1, 1]] == 2, 0, 100]]]

         {Times @@ (Prob2[ωlow[[2 ;;]], 𝒲2]), Times @@ (Prob2[ωhigh[[2 ;;]], 𝒲2])}
         N[%]
```

Out[43]= $\left\{\frac{1}{279\,738\,112}, \frac{1}{206}\right\}$

Out[44]= $\{3.57477 \times 10^{-9}, 0.00485437\}$

```
In[45]:= Step2[ω_] := Step[ω, 𝒲2]
         samples = Table[Nest[Step2, {{1, 1, "init"}}, 10], {10 000}];
         {Length[Select[samples, # == ωlow &]]
                        Length[samples]
          ,
          Length[Select[samples, # == ωhigh &]]
                        Length[samples]
         }
         N[%]
```

Out[47]= $\left\{0, \frac{59}{10\,000}\right\}$

Out[48]= {0., 0.0059}

Properties of STS in STS_2

We consider first the problem of characterizing the STS that are deterministic. For such STS, the probability of a sojourn path is independent of a weight scheduler.

Definition 4.10 (Deterministic STS). A STS $\mathcal{S}$ in STS_2 is said to be deterministic if for every weight schedulers $\mathcal{W}$ and $\mathcal{W}'$, for every sojourn path ω, we have $\mathbf{P}_2(\omega, \mathcal{W}) = \mathbf{P}_2(\omega, \mathcal{W}')$.

Let $\mathcal{B}$ be a BioRica node and $\mathcal{S} = [\![\mathcal{B}]\!]$ be the underlying STS. We can decide if $\mathcal{S}$ is in STS_1 and thus if it is deterministic by a syntactical check on $\mathcal{B}$.

Property 4.2 (Characterization of deterministic nodes). *The STS $\mathcal{S} = [\![\mathcal{B}]\!]$ is in STS_1 if its set of input flow is $F^{in} = \emptyset$ and if for every event e in Σ, the weight $W(e)$ is defined, the delay $\Gamma(e, q) = \delta_0$ for any configuration q, and there is only one macro-transition labeled with e.*

Note that this condition is sufficient but not necessary. Consider for example a BioRica node that have two macro-transitions m and m' sharing the same label. If $[\![Pre(m_t) \wedge Post(m_t)]\!] \cap [\![Pre(m'_t) \wedge Post(m'_t)]\!] = \emptyset$ then both macro-transitions are never enabled in the same configuration, hence the transition relation is deterministic and $\mathcal{S}$ is in STS_1.

Property 4.3 (Equivalence between $\mathbf{P}_1$ and $\mathbf{P}_2$). *For a STS in STS_1, for any sojourn path ω, for any weight scheduler $\mathcal{W}$, we have $\mathbf{P}_{2\mathcal{W}}(\omega) = \mathbf{P}_1(\omega)$.*

Proof. Let $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ be a STS in STS_1. By definition, $\mathcal{S}$ is deterministic, and W is defined for all the events in Σ. Let $\mathcal{W}$ and $\mathcal{W}'$ be two weight schedulers. Since W is defined over Σ, we always have $\mathcal{W}(\omega, e, q) = \mathcal{W}'(\omega, e, q) = W(e)$ for any state q and for any sojourn path ω. Thus, $\mathbf{P}_1(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega) = \mathbf{P}_{2\mathcal{W}'}(\omega)$. □

For the following STS classes, the results are meant to be interpreted with regards to a weight scheduler.

4.3.4 Probability of a sojourn path of STS_3: Accounting for one step timed transitions with continuous delays

We consider here the probability of making a timed transition. Let $\mathcal{S}$ be a STS and $\mathcal{W}$ a weight scheduler for $\mathcal{S}$. We consider in this subsection the subclass of STS that have a single state with timed transitions whose delay follows a continuous distribution. More formally, we define the class STS_3 as follows.

Definition 4.11 (A). STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ is in STS_3

1. if there exists a unique, distinguished state $q_T \in Q$ such that for every event e in $\mathrm{En}(q_T)$, $\Gamma(e)$ is a continuous distribution (see 2.4 for the definition),
2. and if for every other state $q' \neq q_T$ in Q, for every event e in $\mathrm{En}(q')$, we have $\Gamma(e) = \delta_0$.

Consider a sojourn path $\omega = q_0, (e_0, t_0), q_1, (e_1, t_1), \ldots, q_{n-1}, (e_{n-1}, t_{n-1}), q_T$ of $\mathcal{S} \in STS_3$ where q_T do not appear except as the final state. Since q_T is the only state where timed transitions are enabled, the probability $\mathbf{P}_3(\omega)$ is

$$\mathbf{P}_3(\omega) = \mathbf{P}_2(q_0, (e_0, 0), q_1, (e_1, 0), \ldots, q_{n-1}, (e_{n-1}, 0), q_T).$$

In other words, we reach q_T within 0 time unit almost surely with the probability given by the underlying discrete time Markov chain.

However the the next transition $q_T \xrightarrow{e,t} q'$ implies that t will be non zero almost surely. Since the probability distribution of t depends on the event e that wins the race condition in q_T, we will compute the probability that a given event e wins the race condition in q_T in less that t time units. Let E be a r.v. taking values in Σ and representing the event winning the race condition, and let T be a r.v. taking values in $\mathbb{R}$ and representing the sojourn time in q_T.

In the case where a realization of E is not enough to uniquely determine the next state q' (e.g. for a non-deterministic STS), we suppose that a weight scheduler will elect the next state *after* the race between $\mathrm{En}(q_T)$ is over. Therefore, we characterize first the distribution of the random vector $\langle E, T \rangle$.

By definition of STS_3 every the events $\mathrm{En}(q_T)$ are labelled with an absolutely continuous distribution, thus the outcome of the race condition is fundamentally described by the probability that a specific event e of $\mathrm{En}(q_T)$ wins the race condition in less than time t when competing with events in $\mathrm{En}(q_t)$, i.e. we have to compute $\mathbf{P}(\langle E = e, T \leq t \rangle)$ for e and t given. We will establish that

$$\mathbf{P}(\langle E = e, T \leq t \rangle) = \int_0^t \left(\prod_{e' \in \mathrm{En}(q_T) - \{e\}} (1 - F_{e'}(t)) \right) * f_e(t) dt \tag{4.1}$$

Proof. The proof is carried out explicitly for three events $\text{En}(q_T) = \{e_1, e_2, e_3\}$, but the generalization is easy. Let $\{X, Y, Z\}$ be the r.v. denoting the delays of $\{e_1, e_2, e_3\}$ respectively. Each r.v. X, Y, Z follows the distribution $\Gamma(e_1), \Gamma(e_2)$ and $\Gamma(e_3)$ respectively. We first formalize the meaning of the probabilistic event "the event e_1 wins before time t". By using conditional probabilities, we observe that

$$\begin{aligned}
&\mathbf{P}(E = e_1, T \leq t \mid X \leq t, X < Y, X < Z) = 1,\\
&\mathbf{P}(E = e_1, T \leq t \mid X \leq t, X > Y \vee X > Z) = 0,\\
&\mathbf{P}(E = e_1, T \leq t \mid X \leq t, X = Y, X < Z) = a,\\
&\mathbf{P}(E = e_1, T \leq t \mid X \leq t, X = Z, X < Y) = b,\\
&\mathbf{P}(E = e_1, T \leq t \mid X \leq t, X = Z = Y) = c
\end{aligned}$$

where $a = \frac{\mathcal{W}(e_1)}{\mathcal{W}(e_1)+\mathcal{W}(e_2)}, b = \frac{\mathcal{W}(e_1)}{\mathcal{W}(e_1)+\mathcal{W}(e_3)}, c = \frac{\mathcal{W}(e_1)}{\mathcal{W}(e_1)+\mathcal{W}(e_2)+\mathcal{W}(e_3)}$ are real-valued constants.

Since the collection of sets $\{X < Y, X < Z\}$, $\{X = Y, X < Z\}$, $\{X = Z, X < Y\}$,$\{X = Y = Z\}$ and $\{X > Y \vee X > Z\}$ forms a partition of $\mathbb{R}^3$, we can apply the law of total probability and hence

$$\begin{aligned}
\mathbf{P}(E = e_1, T \leq t) \quad &= \mathbf{P}(X \leq t, X < Y, X < Z)\\
&+a \times \mathbf{P}(X \leq t, X = Y, X < Z)\\
&+b \times \mathbf{P}(X \leq t, X = Z, X < Y)\\
&+c \times \mathbf{P}(X \leq t, X = Z = Y).
\end{aligned}$$

However since

$$\mathbf{P}(X = Y) = \int\int_{\{x=y\}} f_X(x) f_Y(y) dxdy = \int_{-\infty}^{+\infty} f_X(x) \int_x^x f_Y(x) dx = 0,$$

and therefore $\mathbf{P}(X = Y) = \mathbf{P}(X = Z) = \mathbf{P}(X = Y = Z) = 0$, we have

$$\mathbf{P}(E = e_1, T \leq t) = \mathbf{P}(X \leq t, X < Y, X < Z).$$

This probability can be expressed as follows

$$\begin{aligned}
\mathbf{P}(X \leq t, X < Y, X < Z) &= \int\int\int_{\{x \leq t, x < y, x < z\}} f_{XYZ}(x, y, z) dxdydz\\
&= \int\int\int_{\{x \leq t, x < y, x < z\}} f_X(x) f_Y(y) f_Z(z) dxdydz\\
&= \int\int\int_{\{x \leq t, x < y\}} f_X(x) f_Y(y) \int_x^{+\infty} f_Z(z) dzdydx\\
&= \int\int_{\{x \leq t, x < y\}} f_X(x) f_Y(y) \times (1 - F_Z(x)) dydx\\
&= \int_{\{x \leq t\}} f_X(x) \times (1 - F_Y(x)) \times (1 - F_Z(x)) dx\\
&= \int_0^t (1 - F_Y(x)) \times (1 - F_Z(x)) \times f_X(x) dx.
\end{aligned}$$

The steps are similar for $\mathbf{P}(Y \leq t, Y < X, X < Z), \mathbf{P}(Z \leq t, Z < X, Z < Y)$. In fact, one can obtain the corresponding probabilities by performing substitutions in the last formula. □

Property 4.4 (**Marginals, Exit rate**). *We have for any* $e \in \mathrm{En}(q_T)$

$$\mathbf{P}(E = e) = \int_{-\infty}^{+\infty} \left(\prod_{e' \in \Sigma_s} (1 - F_{e'}(t)) \right) * f_e(t)dt, \tag{4.2}$$

and

$$\mathbf{P}(T \leq t) = 1 - \left(\prod_{e \in \Sigma_s} (1 - F_e(t)) \right). \tag{4.3}$$

Proof. The proof of equation (4.2) is immediate by the definition of the marginal. As for equation (4.3), we have by the definition of the marginal and by equation (4.1):

$$\begin{aligned} \mathbf{P}(T \leq t) &= \sum_{e \in \mathrm{En}(q_T)} \mathbf{P}(E = e, T \leq t) \\ &= \int_0^t (1 - F_Y(x)) \times (1 - F_Z(x)) \times f_X(x) + (1 - F_X(x)) \times (1 - F_Z(x)) \times f_Y(x) \\ &\qquad + (1 - F_X(x)) \times (1 - F_Y(x)) \times f_z(x)dx. \end{aligned}$$

Consider the substitutions: $u(x) \leftarrow (1 - F_X(x)), v(x) \leftarrow (1 - F_Y(x)), w(x) \leftarrow (1 - F_Z(x))$. We have $u'(x) = -f_X(x), v'(x) = -f_Y(y)$, and $w'(x) = -f_Z(x)$. By performing this substitution in the last formula, we obtain

$$\mathbf{P}(T \leq t) = -1 \cdot \int_0^t v(x) \times w(x) \times u'(x) + u(x) \times w(x) \times v'(x) + u(x) \times v(x) \times w'(x)dx. \tag{4.4}$$

Since $(u(x) \times v(x) \times w(x))' = v(x) \times w(x) \times u'(x) + u(x) \times w(x) \times v'(x) + u(x) \times v(x) \times w'(x)$, we can integrate equation (4.4) and thus

$$\mathbf{P}(T \leq t) = \lim_{b \to -\infty} \Big(u(b) \times v(b) \times w(b) - u(x) \times v(x) \times w(x) \Big).$$

Since $\lim_{b \to -\infty} u(b) \times v(b) \times w(b) = 1$, we have

$$\mathbf{P}(T \leq t) = 1 - (1 - F_X(x)) \times (1 - F_Y(x)) \times (1 - F_Z(x)). \tag{4.5}$$

One could also obtain the same result without considering equation (4.3). Indeed, the complementary probabilistic event of $\{T \leq t\}$, is the probabilistic event $\{T > t\} = \{\min\{X, Y, Z\} > t\}$. In order to realize this event, we must have all the delays strictly greater than t. More formally, we have

$$\begin{aligned} 1 - \mathbf{P}(T \leq t) &= \mathbf{P}(\text{"all delays are greater than } t\text{"}) \\ &= \mathbf{P}(X > t, Y > t, Z > t) = \mathbf{P}(X > t) \times \mathbf{P}(Y > t) \times \mathbf{P}(Z > t) \\ &= (1 - F_X(t)) \times (1 - F_Y(t)) \times (1 - F_Z(t)), \end{aligned}$$

and thus $\mathbf{P}(T \leq t) = 1 - (1 - F_X(t)) \times (1 - F_Y(t)) \times (1 - F_Z(t))$. □

We can now use this probability to define the probability of a sojourn path of STS_3. To this end, given a sojourn path ω in which q_T appears more than once, we decompose ω into probabilistically independent sojourn sub paths. More formally, for any sojourn path

ω in which the distinguished state q_T appears, the *decomposition of ω in probabilistically independent sub paths* is the sequence of sojourn paths $(\omega_i)_{i\leq k}$ such that

$$\omega = \omega_0.(q_T,(e_0,t_0),q_{T1}).\omega_1.(q_T,(e_1,t_1),q_{T2}).\ldots.(q_T,(e_{k-1},t_{k-1}),q_{Tk}).\omega_k.$$

Note that by definition of the concatenation operation (see Def. 4.4), the sojourn path ω_i must start with the state q_{Ti}. Furthermore, the length of ω_i can be zero, and in the case where q_T does not appear in ω, the decomposition of ω is not defined. Finally, if q_T is the last state but one of ω, then the decomposition of ω is ω itself.

Decomposition of a sojourn path ω into probabilistically independent sub paths allows us to decompose the computation of $\mathbf{P}(\omega)$. However in the case where the STS $\mathcal{S}$ is not deterministic, the outcome of a weight scheduler $\mathcal{W}$ *is* dependent on the full prefix. That is, we cannot assume for any decomposition $\omega = \omega_1.\omega_2$ that

$$\mathbf{P}_{2\mathcal{W}}(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega_1) \times \mathbf{P}_{2\mathcal{W}}(\omega_2),$$

and thus cannot decompose *a priori* the probability of ω by reusing $\mathbf{P}_2$ on sub paths. However, consider for example a sojourn path $\omega = (X,(e_0,0),q_T,(e_1,t_1),Y)$ of a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma\rangle$ with the non deterministic transition relation defined by

$$\left\{X \xrightarrow{e_0} qT, X \xrightarrow{f} X, qT \xrightarrow{e_1} Y, qT \xrightarrow{e_1} Z, qT \xrightarrow{j} X\right\}$$

We have by definition

$$\begin{aligned}
\mathbf{P}_{2\mathcal{W}}(\omega) &= \prod_{q_i\in\omega} \frac{\mathcal{W}(\omega \mid i, e_i, q_{i+1})}{\sum_{e\in\mathrm{En}(q_i)}\sum_{\left\{q' \mid q\xrightarrow{e} q'\right\}} \mathcal{W}(\omega \mid i, e, q')} \\
&= \frac{\mathcal{W}((X), e_0, qT)}{\mathcal{W}((X), f, X) + \mathcal{W}((X), e_0, Y)} \\
&\quad \times \frac{\mathcal{W}((X,e_0,qT), e_1, Y}{\mathcal{W}((X,e_0,qT),e_1,Y) + \mathcal{W}((X,e_0,qT),e_1,Z) + \mathcal{W}((X,e_0,qT),j,X)}.
\end{aligned}$$

In order to express $\mathbf{P}_3(\omega)$ in terms of $\mathbf{P}_{2\mathcal{W}}(\omega)$, we have to correct $\mathbf{P}_{2\mathcal{W}}(\omega)$ to account for the outcome of the race condition in qT between $\{e_1, j\}$ that elected e_1 as the winner. That is, we have to apply the weight scheduler not to select between all the transitions of the events $\{e_1, j\}$ but solely between the transitions that are labeled with the winning event e_1. Thus we have

$$\begin{aligned}
\mathbf{P}_3(\omega) &= \mathbf{P}_{2\mathcal{W}}(\omega) \times \mathbf{P}(E = e_1, T \leq t_1) \\
&\quad \times \frac{\mathcal{W}((X,e_0,qT),e_1,Y) + \mathcal{W}((X,e_0,qT),e_1,Z) + \mathcal{W}((X,e_0,qT),j,X)}{\mathcal{W}((X,e_0,qT),e_1,Y) + \mathcal{W}((X,e_0,qT),e_1,Z)}
\end{aligned}$$

Definition 4.12 (Measure $\mathbf{P}_3$). For a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma\rangle$ in STS_3, for a weight scheduler $\mathcal{W}$, for any sojourn path ω of length greater or equal than 1 with the decomposition

$(\omega_i)_{i\leq k}$ then we define

$$\mathbf{P}_3(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega)\times \prod_{0\leq i\leq k} \mathbf{P}(\langle E = e_i, T \leq t_i\rangle) \times \frac{\sum_{e\in \mathrm{En}(q_T)} \sum_{\left\{q' | q_T \xrightarrow{e} q'\right\}} \mathcal{W}(\omega \mid i, e, q')}{\sum_{\left\{q' | q_T \xrightarrow{e_i} q'\right\}} \mathcal{W}(\omega \mid i, e_i, q')}$$

where

$$\mathbf{P}(\langle E = e_i, T \leq t_i\rangle) = \int_0^{t_i} \left(\prod_{e'\in \mathrm{En}(q_T)-\{e_i\}} (1 - F_{e'}(t)) \right) * f_{e_i}(t)dt.$$

For sojourn paths where q_T does not appear (and thus that does not admit a decomposition), we define

$$\mathbf{P}_3(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega).$$

Property 4.5 (Relation with $\mathbf{P}_1$). *For a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma\rangle$ with a deterministic transition relation and for which each event e enabled in a state $q \neq q_T$, the weight $W(e)$ is defined, then we have for any sojourn path ω,*

$$\mathbf{P}_3(\omega) = \mathbf{P}_1(\omega) \times \prod_{0\leq i\leq k} \mathbf{P}(\langle E = e_i, T \leq t_i\rangle).$$

Numerical validation We verify the formulas 4.12 and 4.1 by using numerical integration and sampling. Let X, Y and Z be three continuous r.v. whose distributions are $X \sim N(12, 1), Y \sim U([8, 13])$, and $Z \sim \mathrm{Exp}(1/12)$. We first sample realizations of the r.v. $\langle X, Y, Z\rangle$ and then compare the numerical approximation of the integral 4.1 with its empirical estimator. Comparison is made for values of t in the range $[7, 14]$ with a step of 0.5. For the three integrals, we get a total error less than 10^{-2}. This is done with the following program.

In[1]:=
```
X = NormalDistribution[12, 1];
Y = UniformDistribution[{8, 13}];
Z = ExponentialDistribution[1 / 12];
```

In[4]:=
```
rvSamples = Table[RandomReal /@ {X, Y, Z}, {100 000}];
```

In[5]:=
```
XisSmallestAndLessThanT[t_] :=
  Select[rvSamples, Ordering[#][[1]] == 1 ∧ #[[1]] ≤ t &];
YisSmallestAndLessThanT[t_] :=
  Select[rvSamples, Ordering[#][[1]] == 2 ∧ #[[2]] ≤ t &];
ZisSmallestAndLessThanT[t_] :=
  Select[rvSamples, Ordering[#][[1]] == 3 ∧ #[[3]] ≤ t &];

XWins[t_] := NIntegrate[
  (1 - CDF[Y, x]) * (1 - CDF[Z, x]) * PDF[X, x], {x, 0, t}]
YWins[t_] := NIntegrate[
  (1 - CDF[X, y]) * (1 - CDF[Z, y]) * PDF[Y, y], {y, 0, t}]
ZWins[t_] := NIntegrate[
  (1 - CDF[Y, z]) * (1 - CDF[X, z]) * PDF[Z, z], {z, 0, t}]

totalErrorOnX = Total[Table[Norm[
    XWins[t] - Length[XisSmallestAndLessThanT[t]] / Length[rvSamples]], {t, 7, 14, 0.5}]]
totalErrorOnY = Total[Table[Norm[
    YWins[t] - Length[YisSmallestAndLessThanT[t]] / Length[rvSamples]], {t, 7, 14, 0.5}]]
totalErrorOnZ = Total[Table[Norm[
    ZWins[t] - Length[ZisSmallestAndLessThanT[t]] / Length[rvSamples]], {t, 7, 14, 0.5}]]
```

Out[11]= 0.011851

Out[12]= 0.0234978

Out[13]= 0.0200618

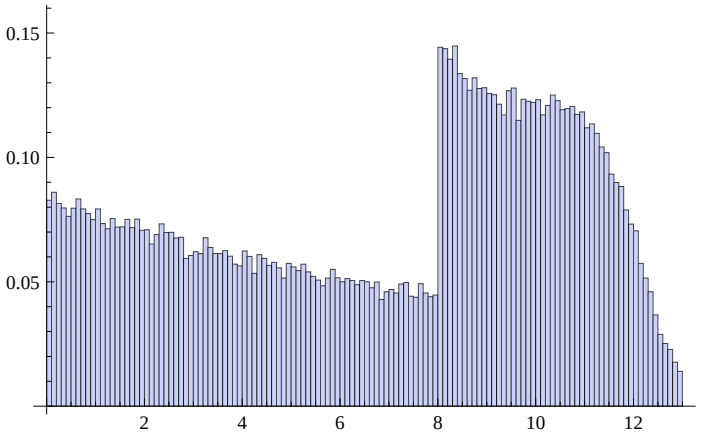
In order to compare the results of the marginals, we follow as similar scheme of sampling and numerical integration.

```
In[19]:= XisSmallest = Select[rvSamples, Ordering[#][[1]] == 1 &];
         YisSmallest = Select[rvSamples, Ordering[#][[1]] == 2 &];
         ZisSmallest = Select[rvSamples, Ordering[#][[1]] == 3 &];

In[22]:= Norm[Length[XisSmallest] / Length[rvSamples] - XWins[+∞]]
         Norm[Length[YisSmallest] / Length[rvSamples] - YWins[+∞]]
         Norm[Length[ZisSmallest] / Length[rvSamples] - ZWins[+∞]]

Out[22]= 0.00182082

Out[23]= 0.00299586

Out[24]= 0.00117505
```

The distribution of the exit rate can also be obtained by mapping each realization of the random vector $\langle X, Y, Z \rangle$ to its minimum and then bin the results. Here, we binned the minimum in bins of length 0.1 and obtained the following distribution.

```
In[35]:= quantum = 0.1;
         Histogram[Min /@ rvSamples, {quantum}, "ProbabilityDensity"]
```

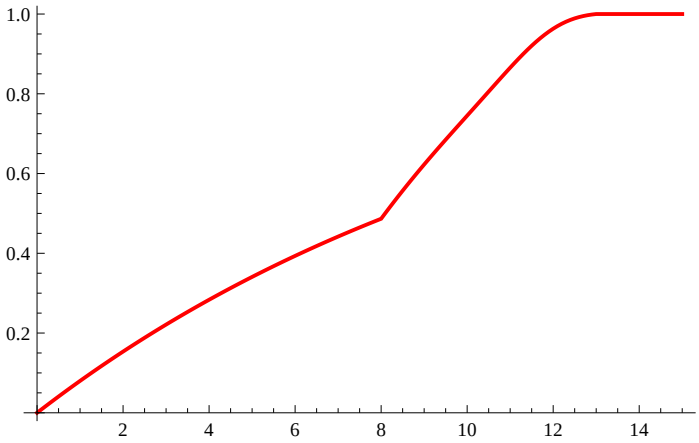
Out[36]=



The theoretical CDF of the minimum of the three r.v. is given by the marginal $\mathbf{P}(T \leq t)$. In order to compare this theoretical distribution to the previous empirical PDF, we perform a discrete derivation of successive values of the theoretical CDF obtained with a step of 0.1. We obtain the following PDF.

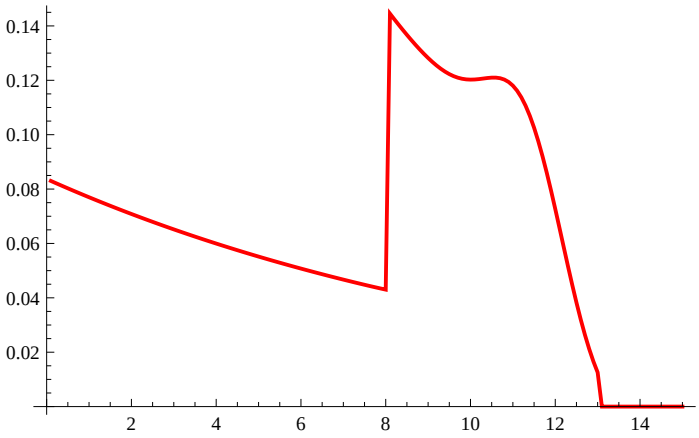
In[27]:=
```
ExitRate[t_] := 1 - (1 - CDF[X, t]) * (1 - CDF[Y, t]) * (1 - CDF[Z, t])
quantum = 0.1;
ExitRateCDF = Table[{t, ExitRate[t]}, {t, 0, 15, quantum}];
ListLinePlot[ExitRateCDF, PlotStyle → {Red, Thick}]
```

Out[30]=



In[31]:=
```
ExitRatePDF = Transpose[{
    ExitRateCDF[[2 ;;, 1]],
    Differences[ExitRateCDF[[All, 2]]] * 1 / quantum}
  ];
ListLinePlot[ExitRatePDF, PlotStyle → {Red, Thick}]
```

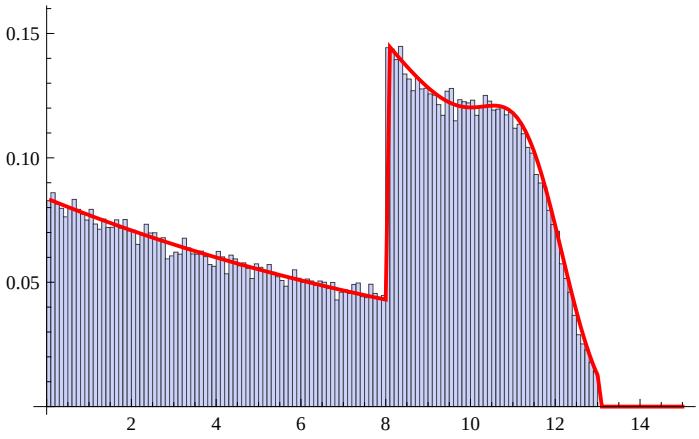
Out[32]=



And we finally compare the theoretical and empirical distributions by plotting them on the same graph. On this graph, the red line denote the value of the PDF of the theoretical exit rate, while the histogram bars represent the empirical PDF of the exit rate.

```
In[34]:= Show[
          Histogram[Min /@ rvSamples, {quantum}, "ProbabilityDensity"],
          ListLinePlot[ExitRatePDF, PlotStyle → {Red, Thick}]
         ]
```

Out[34]=

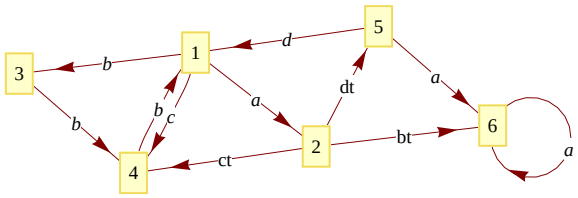


Simulation of STS in STS_3

Simulation of a STS in STS_3 consists of running the simulation according to the algorithm given for a STS in STS_2 up to arriving at the state q_T. Once in q_T, we sample a realization of the random vector corresponding to the delays of the events in $\mathrm{En}(q_T)$. The next transition is then chosen by selecting the event scheduled at the smallest epoch. We modify the previous STS by adding three timed events bt, ct and dt enabled in the state 2.

In[37]:=
```
Σ = {a, b, c, d, dt, bt, ct};
W[σ_] := Switch[σ, "init", 1, a, 1, b, 2, c, 1, d, 5]
Q = {1, 2, 3, 4, 5, 6};
Γ[σ_] := Switch[σ, "init", 0, a, 0, b, 0, c, 0, d, 0,
  dt, NormalDistribution[12, 1],
  bt, ExponentialDistribution[1 / 12],
  ct, UniformDistribution[{8, 13}]]

S = {
   {1 → 2, a}, {1 → 3, b}, {1 → 4, c},
   {2 → 6, bt}, {2 → 5, dt}, {2 → 4, ct},
   {3 → 4, b},
   {5 → 6, a}, {5 → 1, d}, {4 → 1, b},
   {6 → 6, a}
  };
GraphPlot[S, DirectedEdges → True, VertexLabeling → True]
```

Out[42]=



Simulation of such STS is carried out as follows. Once the set of enabled events in the current state is computed, we associate each event to a sample from the delays distribution. From this sample list, we select the set of events that are associated with the smallest delay, then in the case of ties (i.e. in STS_3, for delays following a Dirac distribution), we apply the weight scheduler $\mathcal{W}_1$ to elect the winning event.

In[43]:=
```
EnabledIn[s_] := Select[S, #[[1, 1]] == s &][[All, 2]]
TransitionFrom[s_, e_] := Select[S, MatchQ[#, {s → _, e}] &][[1]]
```

In[45]:=
```
Sample[f_] :=
 If[Head[f] === Integer, f,
  RandomReal[f]
 ]
```

In[46]:=
```
Step[ω_, 𝒲_] := Module[{
   sfin, enabled, newWeights, totalWeight, nextE,
   clocks, winners, nextT, winnerLabels, currentT},
  sfin = ω[[-1, 2]];
  currentT = ω[[-1, 4]];
  enabled = EnabledIn[sfin];
  clocks = (#1 -> Sample[Γ[#1]]) & /@ enabled;
  clocks = GatherBy[clocks, #[[2]] &];
  winners = SortBy[clocks, #[[1, 2]] &][[1]];
  {nextT, winnerLabels} = {winners[[1, 2]], winners[[All, 1]]};
  nextE = If[Length[winnerLabels] == 1, winnerLabels[[1]],
    newWeights = 𝒲[ω, #] & /@ winnerLabels;
    totalWeight = Total[newWeights];
    RandomChoice[newWeights/totalWeight → winnerLabels]
   ];

  Return[Append[ω, {sfin, TransitionFrom[sfin, nextE][[1, 2]], nextE,
     currentT + nextT}]]
 ]
𝒲1[ω_, σ_] := If[W[σ] =!= #, W[σ], 1]
Step1[ω_] := Step[ω, 𝒲1]
```

In[49]:=
```
Nest[Step1, {{1, 1, "init", 0}}, 20]
Nest[Step1, {{1, 1, "init", 0}}, 20]
Nest[Step1, {{1, 1, "init", 0}}, 20]
```

Out[49]=
```
{{1, 1, init, 0}, {1, 3, b, 0}, {3, 4, b, 0}, {4, 1, b, 0}, {1, 4, c, 0},
 {4, 1, b, 0}, {1, 2, a, 0}, {2, 4, ct, 9.88599}, {4, 1, b, 9.88599},
 {1, 3, b, 9.88599}, {3, 4, b, 9.88599}, {4, 1, b, 9.88599},
 {1, 3, b, 9.88599}, {3, 4, b, 9.88599}, {4, 1, b, 9.88599},
 {1, 2, a, 9.88599}, {2, 6, bt, 11.7483}, {6, 6, a, 11.7483},
 {6, 6, a, 11.7483}, {6, 6, a, 11.7483}, {6, 6, a, 11.7483}}
```

Out[50]=
```
{{1, 1, init, 0}, {1, 4, c, 0}, {4, 1, b, 0}, {1, 3, b, 0},
 {3, 4, b, 0}, {4, 1, b, 0}, {1, 4, c, 0}, {4, 1, b, 0},
 {1, 3, b, 0}, {3, 4, b, 0}, {4, 1, b, 0}, {1, 4, c, 0},
 {4, 1, b, 0}, {1, 4, c, 0}, {4, 1, b, 0}, {1, 3, b, 0},
 {3, 4, b, 0}, {4, 1, b, 0}, {1, 4, c, 0}, {4, 1, b, 0}, {1, 2, a, 0}}
```

Out[51]=
```
{{1, 1, init, 0}, {1, 3, b, 0}, {3, 4, b, 0}, {4, 1, b, 0}, {1, 3, b, 0},
 {3, 4, b, 0}, {4, 1, b, 0}, {1, 2, a, 0}, {2, 6, bt, 5.3873},
 {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873},
 {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873},
 {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873}, {6, 6, a, 5.3873}}
```

Suppose that we want to compute $\mathbf{P}_3(\omega)$ for ω being one of $\{(1 \xrightarrow{a,0} 2 \xrightarrow{dt,10} 5 \xrightarrow{a,0} 6), (1 \xrightarrow{a,0} 2 \xrightarrow{bt,6} 6 \xrightarrow{a,0} 6), (1 \xrightarrow{a,0} 2 \xrightarrow{ct,20} 5 \xrightarrow{a,0} 6), (1 \xrightarrow{a,0} 2 \xrightarrow{bt,14} 5 \xrightarrow{a,0} 6)\}$ The empirical estimators of the probabilities of these sojourn paths is computed as follows.

In[52]:=
```
samples = Table[Nest[Step1, {{1, 1, "init", 0}}, 3], {50 000}];
```

In[71]:=
```
Untime[w_] := w[[All, 1 ;; 3]]

IsInω[s_] := Untime[s] ==
   {{1, 1, "init"}, {1, 2, a}, {2, 5, dt}, {5, 6, a}} ∧ s[[3, 4]] ≤ 10

IsInω2[s_] := Untime[s] ==
   {{1, 1, "init"}, {1, 2, a}, {2, 6, bt}, {6, 6, a}} ∧ s[[3, 4]] ≤ 6

IsInω3[s_] := Untime[s] ==
   {{1, 1, "init"}, {1, 2, a}, {2, 4, ct}, {4, 1, b}} ∧ s[[3, 4]] ≤ 20

IsInω4[s_] := Untime[s] ==
   {{1, 1, "init"}, {1, 2, a}, {2, 6, bt}, {6, 6, a}} ∧ s[[3, 4]] ≤ 14
```

In[76]:=
```
emp = Length[Select[samples, #]] & /@ {IsInω, IsInω2, IsInω3, IsInω4}
                Length[samples]
N[%]
```

Out[76]= $\left\{\frac{7}{25\,000}, \frac{61}{625}, \frac{2161}{25\,000}, \frac{359}{2500}\right\}$

Out[77]= {0.00028, 0.0976, 0.08644, 0.1436}

We compare the empirical estimators with the theoretical probability.

In[60]:=
```
Prob2[ω_, 𝒲_] := Module[{},
  MapIndexed[
   Module[{ωi, prefix, qi, σi, conflicting, currentWeights, totalWeight},
     ωi = ω[[First[#2]]];
     prefix = ω[[ ;; First[#2]]];
     σi = ωi[[3]];
     qi = ωi[[1]];
     conflicting = EnabledIn[qi];
     currentWeights = 𝒲[prefix, #] & /@ conflicting;
     totalWeight = Total[currentWeights];
     𝒲[prefix, σi]
      totalWeight
    ] &, ω]
  ]
```

In[61]:=
```
ProbDWinsBeforeT[t_] :=
 NIntegrate[(1 - CDF[Γ[bt], d]) * (1 - CDF[Γ[ct], d]) * PDF[Γ[dt], d], {d, 0, t}]
ProbBWinsBeforeT[t_] :=
 NIntegrate[(1 - CDF[Γ[dt], b]) * (1 - CDF[Γ[ct], b]) * PDF[Γ[bt], b], {b, 0, t}]
ProbCWinsBeforeT[t_] :=
 NIntegrate[(1 - CDF[Γ[dt], c]) * (1 - CDF[Γ[bt], c]) * PDF[Γ[ct], c], {c, 0, t}]
```

In[67]:=
```
theo = {
  Times @@ Prob2[{{1, 2, a}}, 𝒲1] * ProbDWinsBeforeT[10] *
   Times @@ Prob2[{{5, 6, a}}, 𝒲1],
 Times @@ Prob2[{{1, 2, a}}, 𝒲1] * ProbBWinsBeforeT[6] *
   Times @@ Prob2[{{6, 6, a}}, 𝒲1],
  Times @@ Prob2[{{1, 2, a}}, 𝒲1] * ProbCWinsBeforeT[20] *
   Times @@ Prob2[{{4, 1, b}}, 𝒲1],
  Times @@ Prob2[{{1, 2, a}}, 𝒲1] * ProbBWinsBeforeT[12] *
   Times @@ Prob2[{{6, 6, a}}, 𝒲1]
 }
```

Out[67]=
```
{0.000287621, 0.0983673, 0.0851965, 0.143219}
```

In[70]:=
```
theo - emp
```

Out[70]=
$$\{7.62069 \times 10^{-6}, 0.000767335, -0.00124353, -0.000380768\}$$

4.3.5 Probability of a sojourn path of STS_4: Accounting for one step timed transitions with general delays

We now remove the continuous restrictions on the r.v.. In this case, since the distributions $\Gamma(e)$ can be either continuous or discrete, the probability of two epochs being equal may be non null. More formally, we define the class STS_4 as follows.

Definition 4.13 (A). STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ is in STS_4 if there exists a unique, distinguished state $q_T \in Q$ such that for every other state $q' \neq q_T$ in Q, for every event e in $\mathrm{En}(q')$, we have $\Gamma(e) = \delta_0$ and for every event e in $\mathrm{En}(q), \Gamma(e)$ is defined.

Following similar arguments as for the case of STS_3, the probability measure $\mathbf{P}_4$ is fundamentally defined by the outcome of a race condition between events competing in $\mathrm{En}(q_T)$. This implies that we have to compute $\mathbf{P}(\langle E = e, T \leq t \rangle)$ for e and t given. We will establish that

$$\mathbf{P}(\langle E = e, T \leq t \rangle) = \sum_{\mathcal{E} \in \mathcal{P}(\mathrm{En}(q_T) - \{e\})} \left(w_{e,e'} \times \int_0^t \left(\prod_{e' \in \mathrm{En}(q_T) - \mathcal{E} - \{e\}} (1 - F_{e'}(x)) \times \prod_{e' \in \mathcal{E} \cup e} f_{e'}(x) \right) dx \right). \tag{4.6}$$

where $w_{e,\mathcal{E}}$ is a weight value dependent on the conflict between e and the set $\mathcal{E}$.

Proof. The proof is again carried out explicitly for three events $\mathrm{En}(q_T) = \{e_1, e_2, e_3\}$. The beginning of the proof follows the same principle as before, and thus we start with the equation (4.2), that is

$$\begin{aligned} \mathbf{P}(E = e_1, T \leq t) = \mathbf{P}(X \leq t, X < Y, X < Z) \\ + a \times \mathbf{P}(X \leq t, X = Y, X < Z) \\ + b \times \mathbf{P}(X \leq t, X = Z, X < Y) \\ + c \times \mathbf{P}(X \leq t, X = Z = Y). \end{aligned} \tag{4.7}$$

We will prove that each of the four terms of the sum in equation (4.7) corresponds to a term of the sum of the equation (4.7). Let $\mathcal{E}$ be one of the subsets of $\mathcal{P}(\mathrm{En}(q_T) - \{e_1\}) = \mathcal{P}(\{e_2, e_3\}) = \{\emptyset, \{e_2\}, \{e_3\}, \{e_2, e_3\}\}$.

By equation (4.1), we can compute that the first term of (4.2) is

$$\mathbf{P}(X \leq t, X < Y, X < Z) = \int_0^t (1 - F_Y(x)) \times (1 - F_Z(x)) \times f_X(x) dx. \tag{4.8}$$

Consider the terms of the sum in equation (4.7). When $\mathcal{E} = \emptyset$ we have $\mathrm{En}(q_T) - \mathcal{E} - \{e_1\} = \{e_2, e_3\}$ and $\mathrm{En}(q_T) \cup \{e_1\} = \{e_1\}$. Hence the term of the sum corresponding to $\mathcal{E} = \emptyset$ yields equation (4.8).

Consider now the second term of equation (4.2). We have

$$\mathbf{P}(X \leq t, X = Y, X \leq Z) = \int_0^t (1 - F_Z(x)) \times f_Y(x) \times f_X(x) dx.$$

In the case of $\mathcal{E} = \{e_2\}$, we have $\mathrm{En}(q_T) - \mathcal{E} - \{e_1\} = \{e_3\}$ and $\mathrm{En}(q_T) \cup \{e_1\} = \{e_1, e_2\}$. Therefore, the second term of the sum (4.7) yields equation (4.9).

Similarly,

$$\mathbf{P}(X \leq t, X = Y = Z) = \int_0^t f_Z(x) \times f_Y(x) \times f_X(x)dt \tag{4.9}$$

is the last term of the sum (4.7) and of the sum (4.7). □

Property 4.6 (**Exit rate**). *In the case where the r.v. denoting the delays of the events in* $\mathrm{En}(q_T)$ *can be either continuous or discrete, we have the marginal distribution*

$$\mathbf{P}(T \leq t) = 1 - \left(\prod_{e \in \Sigma_s} (1 - F_e(t)) \right). \tag{4.10}$$

This marginal distribution does not depends on the weight function W *and hence does not depend on a weight scheduler* $\mathcal{W}$*.*

Proof. (Sketch) Since we built a mapping between each terms of equation (4.2) and (4.7), we will work on (4.2). In the case of three variables $\{X, Y, Z\}$, notice that the terms $\mathbf{P}(X \leq t, X = Z = Y), \mathbf{P}(Y \leq t, X = Y = Z), \mathbf{P}(Z \leq t, X = Y = Z)$ of (4.2) are measures of the same set. These terms appears with different weights in the sum $\mathbf{P}(T \leq t) = \sum_{e \in \mathrm{En}(q_T)} \mathbf{P}(E = e, T \leq t)$. These weights sums to 1. Therefore we can factorize these three terms, and sum the weights to 1. Hence, W does not appear in the marginal distribution. □

Simarly as the case for $\mathbf{P}_3$, we decompose a sojourn path ω into probabilistically independent sub paths $(\omega_i)_{i \leq k}$ and reuse $\mathbf{P}_2$ whenever possible.

Definition 4.14 (**Measure** $\mathbf{P}_4$). For a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ in STS_4, for a weight scheduler $\mathcal{W}$, for any sojourn path ω of length greater or equal than 1 with the decomposition $(\omega_i)_{i \leq k}$ then we define

$$\mathbf{P}_4(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega) \times \prod_{0 \leq i \leq k} \mathbf{P}(\langle E = e_i, T \leq t_i \rangle) \times \frac{\sum_{e \in \mathrm{En}(q_T)} \sum_{\left\{ q' | q_T \xrightarrow{e} q' \right\}} \mathcal{W}(\omega \mid i, e, q')}{\mathcal{W}(\omega \mid i, e_i, q_{Ti})}$$

where

$$\mathbf{P}(\langle E = e_i, T \leq t_i \rangle) = \sum_{\mathcal{E} \in \mathcal{P}(\mathrm{En}(q_T) - \{e_i\})} \left(w_{e_i, \mathcal{E}} \times \int_0^{t_i} \left(\prod_{e' \in \mathrm{En}(q_T) - \mathcal{E} - \{e_i\}} (1 - F_{e'}(x)) \times \prod_{e' \in \mathcal{E} \cup e_i} f_{e'}(x) \right) dx \right).$$

where

$$w_{e_i, \mathcal{E}} = \frac{\mathcal{W}(\omega \mid i, e_i, q_{Ti})}{\sum_{e' \in \mathcal{E} \cup \{e\}} \sum_{\left\{ q' | q_T \xrightarrow{e'} q' \right\}} \mathcal{W}(\omega \mid i, e, q')}.$$

For the border cases where ω has a length 0, we set $\mathbf{P}_4(\omega) = 1$.

Properties of STS in STS_4

We can relate the measure $\mathbf{P}_4$ with the measure $\mathbf{P}_2$ and $\mathbf{P}_3$ as follows.

Property 4.7 (Relation with $\mathbf{P}_2$ and $\mathbf{P}_3$). *For a STS $\mathcal{S}$, for a weight scheduler $\mathcal{W}$, if $\mathcal{S}$ is in STS_2, then for any sojourn path ω, we have $\mathbf{P}_{2,\mathcal{W}}(\omega) = \mathbf{P}_4(\omega)$. If $\mathcal{S}$ is in STS_3, then for any sojourn path ω, we have $\mathbf{P}_3(\omega) = \mathbf{P}_4(\omega)$.*

Proof. Let $\mathcal{S}$ be a STS in STS_2, by definition, $\Gamma(e) = \delta_0$ for $e \in \Sigma$. Let q_T be any state of $\mathcal{S}$, we have $\mathcal{S}$ in STS_4. Without loss of generality, assume that ω is a sojourn path where q_T is the one last state but one, and that it ends with $(q_T, (e_1, t_1), q_{T1}$. Thus, the decomposition of ω is ω itself. We have by definition Def. 4.14

$$\mathbf{P}_4(\omega) = \mathbf{P}_2(\omega) \times \mathbf{P}(E = e_1, T \leq t_1) \times \frac{\sum_{e \in \mathrm{En}(q_T)} \sum_{\left\{q' | q_T \xrightarrow{e} q'\right\}} \mathcal{W}(\omega, e, q')}{\mathcal{W}(\omega, e_1, q_{T1})}$$

with

$$\mathbf{P}(\langle E = e_1, T \leq t_i \rangle) = \tag{4.11}$$
$$\sum_{\mathcal{E} \in \mathcal{P}(\mathrm{En}(q_T) - \{e_1\})} \left(w_{e_1,\mathcal{E}} \times \int_0^{t_1} \left(\prod_{e' \in \mathrm{En}(q_T) - \mathcal{E} - \{e_1\}} (1 - F_{e'}(x)) \times \prod_{e' \in \mathcal{E} \cup e_1} f_{e'}(x) \right) dx \right).$$

with

$$w_{e_1,\mathcal{E}} = \frac{\mathcal{W}(\omega, e_1, q_{T1})}{\sum_{e' \in \mathcal{E} \cup \{e_1\}} \sum_{\left\{q' | q_T \xrightarrow{e'} q'\right\}} \mathcal{W}(\omega, e, q')}.$$

Since $\Gamma(e) = \delta_0$ for each $e \in \Sigma$, $\mathbf{P}(X < Y) = 0$ and $\mathbf{P}(X = Y, X \leq t) = 1$ for X and X being two r.v. distributed as δ_0. Thus any term in the sum of the equation Eq. 4.12 where $\mathcal{E} \neq \mathrm{En}(q_T) - e_1$ is null, and thus equation Eq. 4.12 reduces to $w_{e_1,\mathrm{En}(()q_T)-e_i}$. We have

$$w_{e_1,\mathrm{En}(q_T)-e_i} = \frac{\mathcal{W}(\omega, e_1, q_{T1})}{\sum_{e' \in \mathrm{En}(q_T)} \sum_{\left\{q' | q_T \xrightarrow{e'} q'\right\}} \mathcal{W}(\omega, e', q')}. \tag{4.12}$$

Hence, we have

$$\begin{aligned} \mathbf{P}_4(\omega) &= \mathbf{P}_{2\mathcal{W}}(\omega) \times \frac{\mathcal{W}(\omega, e_1, q_{T1})}{\sum_{e' \mathrm{En}(q_T)} \sum_{\left\{q' | q_T \xrightarrow{e'} q'\right\}} \mathcal{W}(\omega, e, q')} \times \frac{\sum_{e \in \mathrm{En}(q_T)} \sum_{\left\{q' | q_T \xrightarrow{e} q'\right\}} \mathcal{W}(\omega, e, q')}{\mathcal{W}(\omega, e_1, q_{T1})} \\ &= \mathbf{P}_{2\mathcal{W}}(\omega). \end{aligned}$$

The case for $\mathbf{P}_3(\omega) = \mathbf{P}_4(\omega)$ follows similar arguments: The probability $\mathbf{P}(X = Y)$ for two r.v. with continuous distribution is equal to $\mathbf{P}(X = Y) = 0$, thus terms appearing in the sum $\mathbf{P}(\langle E = e, T \leq t \rangle)$ reduces to 0 except for the case when $\mathcal{E} = \emptyset$, and thus we have

$$\begin{aligned} w_{e_1,\emptyset} &= \frac{\mathcal{W}(\omega, e_1, q_{T1})}{\sum_{e' \in \{e_1\}} \sum_{\left\{q' | q_T \xrightarrow{e'} q'\right\}} \mathcal{W}(\omega, e, q')} \\ &= \frac{\mathcal{W}(\omega, e_1, q_{T1})}{\sum_{\left\{q' | q_T \xrightarrow{e_1} q'\right\}} \mathcal{W}(\omega, e_1, q')}, \end{aligned}$$

and thus

$$\mathbf{P}_4(\omega) = \mathbf{P}_{2\mathcal{W}}(\omega) \quad \times \quad \int_0^{t_1} \left(\prod_{e' \in \mathrm{En}(q_T) - \{e_1\}} (1 - F_{e'}(t)) \right) * f_{e_1}(t) dt$$
$$\times \quad \frac{\cancel{\mathcal{W}(\omega, e_1, q_{T1})}}{\sum_{\left\{ q' | q_T \xrightarrow{e_1} q' \right\}} \mathcal{W}(\omega, e_1, q')} \times \frac{\sum_{e \in \mathrm{En}(q_T)} \sum_{\left\{ q' | q_T \xrightarrow{e} q' \right\}} \mathcal{W}(\omega, e, q')}{\cancel{\mathcal{W}(\omega, e_1, q_{T1})}} = \mathbf{P}_3(\omega).$$

□

Property 4.8 (**Instantaneous weighted transitions**). *Let $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ be a STS, $\mathcal{W}$ be a weight scheduler for $\mathcal{S}$, and ω be a sojourn path of $\mathcal{S}$ ending in a state q. If the set $\mathrm{En}(q)$ of events enabled in q contains an event $e \in \Sigma$ such that $\Gamma(e) = \delta_0$, then for every $e' \in \mathrm{En}(q)$ where $\mathbf{P}(X = 0) = 0$ for a random variable $X \sim \Gamma(e')$, we have $\mathbf{P}(E = e') = 0$.*

Proof. Immediate by application of the marginal 4.4. For a random variable $X \sim \delta_0$, we have for any $t \geq 0$, $F_X(t) = 1$, and thus $\mathbf{P}(E = e) = 0$ for any delay distribution $\Gamma(e)$ such that $0 \notin supp(\Gamma(e))$. □

Numerical validation To validate numerically the definition 4.14, we merely have to validate the formula (4.7). To this end, we compute the probability that a random variable T that is distributed along a discrete uniform distribution over $[5, 14]$ wins a race against three random variables $\{X, Y, U\}$ such that X is normally distributed, Y is distributed along a (continuous) uniform distribution over $[8, 13]$ and U is distributed along a poisson distribution with rate 25. In order to resolve ties, we have $W(T) = 5$ and $W(U) = 7$. When comparing the theoretical value with an empirical estimator over 30000, we obtain a total error of less than 0.03.

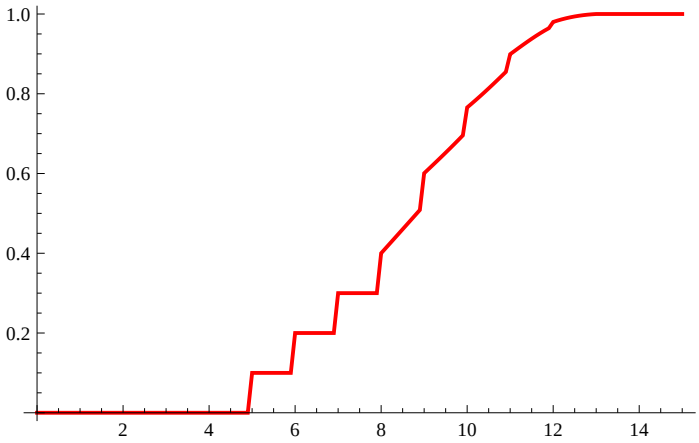
```
In[56]:= X = NormalDistribution[12, 1];
         Y = UniformDistribution[{8, 13}];
         T = DiscreteUniformDistribution[{5, 14}];
         U = PoissonDistribution[25];

In[23]:= rvSamples = Table[Join[RandomReal /@ {X, Y}, RandomInteger /@ {T, U}],
           {30 000}];

In[60]:= wT = 5;
         wU = 7;

In[62]:= ElectWinner[s_] := Module[{winnerIdx, winnerLbl},
           winnerIdx = Ordering[s][[1]];
           winnerLbl = winnerIdx /. {1 → "X", 2 → "Y", 3 → "T", 4 → "U"};
           winnerLbl = If[winnerLbl == "T" ⋀ s[[3]] == s[[4]],
             RandomChoice[{wT, wU} → {"T", "U"}], winnerLbl];
           {winnerLbl, s[[winnerIdx]]}
          ]

In[63]:= winners = ElectWinner /@ rvSamples;

In[64]:= XisSmallestAndLessThanT[t_] :=
           Select[winners, #[[1]] == "X" ⋀ #[[2]] ≤ t &];
         YisSmallestAndLessThanT[t_] :=
           Select[winners, #[[1]] == "Y" ⋀ #[[2]] ≤ t &];
         TisSmallestAndLessThanT[t_] :=
           Select[winners, #[[1]] == "T" ⋀ #[[2]] ≤ t &];
         UisSmallestAndLessThanT[t_] := Select[winners, #[[1]] == "U" ⋀ #[[2]] ≤ t &];

In[68]:= TWins[t_] := N[
           Sum[(1 - CDF[X, tv]) * (1 - CDF[Y, tv]) * (1 - CDF[U, tv]) * PDF[T, tv],
             {tv, 0, t}]
            + Sum[(1 - CDF[X, tv]) * (1 - CDF[Y, tv]) * PDF[U, tv] * PDF[T, tv],
              {tv, 0, t}] * wT/(wU + wT)
          ]

In[71]:= Total[Table[Norm[TWins[t] - Length[TisSmallestAndLessThanT[t]]/Length[rvSamples]],
           {t, 2, 14, 0.5}]]

Out[71]= 0.0232144
```

Similarly to the numerical validation performed for STS_3, we compute the theoretical marginal distribution as follows.

In[50]:=
```
ExitRate[t_] :=
 1 - (1 - CDF[X, t]) * (1 - CDF[Y, t]) * (1 - CDF[T, t]) * (1 - CDF[U, t])
```

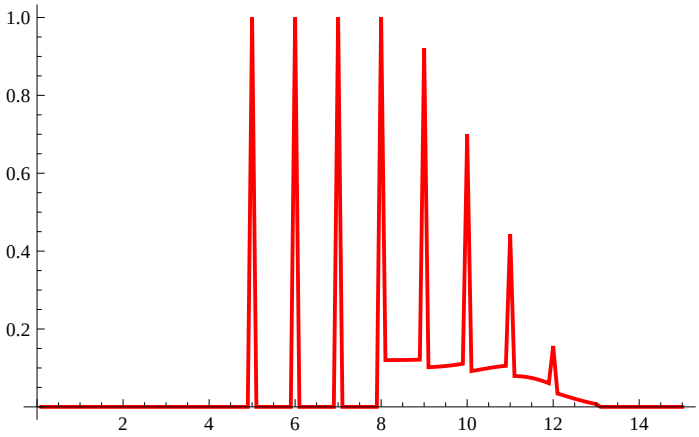
In[51]:=
```
ExitRateCDF = Table[{t, ExitRate[t]}, {t, 0, 15, quantum}];
ListLinePlot[ExitRateCDF, PlotStyle → {Red, Thick}]
```

Out[52]=



In[53]:=
```
ExitRatePDF = Transpose[{
    ExitRateCDF[[2 ;;, 1]],
    Differences[ExitRateCDF[[All, 2]]] * 1 / quantum}
  ];
ListLinePlot[ExitRatePDF, PlotStyle → {Red, Thick}, PlotRange → All]
```

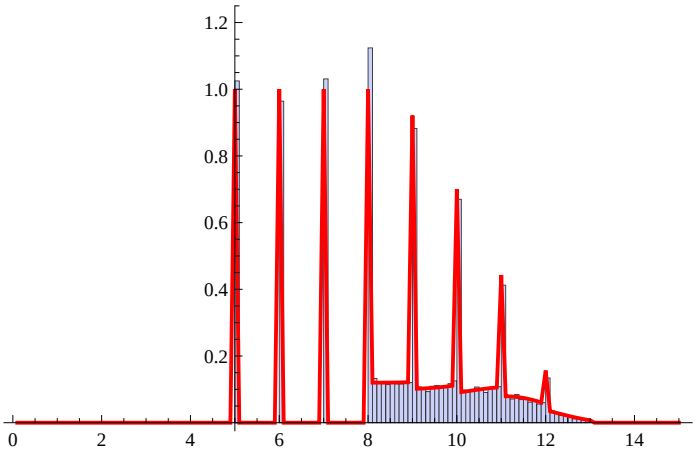
Out[54]=



And we can compare the theoretical marginal distributions with the empirical one by plotting both on the same plot.

In[55]:=
```
Show[
 Histogram[Min /@ rvSamples, {quantum}, "ProbabilityDensity",
  PlotRange → All],
 ListLinePlot[ExitRatePDF, PlotStyle → {Red, Thick}, PlotRange → All]
]
```

Out[55]=



4.3.6 Probability of a sojourn path in STS_5: Accounting for elapsed time since regeneration

We suppose that we want to compute the probability of a sojourn path $(Q, a, t_0, R, b, t_1, S)$. In the initial state of this path, the process is at the global epoch 0, the process hits Q a first time and the timers of events enabled in Q are sampled accordingly to the probability laws defined by Γ. Suppose that we have $\mathrm{En}(()Q) = \{a, b, c\}$. At the epoch where the process hits the state Q, we say that the state Q regenerates the events $\{a, b, c\}$ (or equivalently that Q is regenerative for the events a, b, c). Let A, B, C be three r.v. representing the timers of the events a, b, c at the epoch 0.

The probability of the prefix sojourn path (Q, a, t_0, R) is given by $\mathbf{P}_4$. Once in R, the probability of taking the transition $R \xrightarrow{b} S$ while sojourning in R less than t_1 time units depends on the outcome of the race condition between the events $\{b, c\}$. However, contrary to the race condition in Q, we do not resample the timers of these events and thus the race condition in R must account for time that was elapsed since the epoch where the timers of b and c were sampled. This implies that the outcome of the race condition in R is dependent on the outcome of the race condition in Q, and more precisely of the sojourn time in Q.

Suppose that in Q, the sampled clocks were $\langle A : a, B : b, C : c\rangle$. The sojourn time in the state Q is thus a t.u., and the epoch of arrival in state R is also a. In order for the event b to wins the second race with a sojourn time in the state R less than t_1, we must have $b < t_1 + a \wedge b < c$. This implies that the probability of the sojourn path $(Q, a, t_0, R, b, t_1, S)$ is given by

$$\mathbf{P}(A < t_0, A < B, A < C, B < t_1 + A, B < C) = \mathbf{P}(A < t_0, A < B < t_1 + A, B < C),$$

and since all these (probabilistic) events are dependent, this probability cannot be decomposed in a product of probabilities. However the subset of R^3 satisfying the inequalities $a < t_0 \wedge a < b < t_1 + a \wedge b < c$ is a c, b, a-simple domain, and we can thus evaluate this probability by repeated integration as follows

$$\begin{aligned} P(A < t_0, A < B < t_1 + A, B < C) &= \int_{\{a<t_0, a<b<t_1+a, b<c\}} f_A(a) \cdot f_B(b) \cdot f_C(c) \\ &= \int_0^{t_0} f_A(a) \int_a^{t_1+a} f_B(b) \int_b^{+\infty} f_C(c) dc db da \\ &= \int_0^{t_0} f_A(a) \int_a^{t_1+a} f_B(b) * (1 - F_C(b))\, db da. \end{aligned}$$

We now generalize the previous construction and formally define the class STS_5 that are STS_5 where all the timers of the timed events are always regenerated at the same epoch. We will then express a sojourn path as a conjunction of constraints over n r.v. variables, where n is the number of events. We will then show that the region of $\mathbb{R}^n$ satisfying this conjunction is a simple domain of $\mathbb{R}^n$ and thus that we can use repeated integration to measure its probability.

In order to compute the probability that the sojourn time in a state q is less that an threshold value t when taking a transition labeled by the event e, we must account for the time elapsed since the regeneration of the clock of the event e. This regeneration happens when the event e became enabled for the first time when reaching a state. More formally, we have :

Definition 4.15 (**Events regenerated by a transition**). Let $q \xrightarrow{e} q'$ be a transition of a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$. The events regenerated by this transition are

$$Reg(q \xrightarrow{e} q') = (En(q') - En(q)) \cup (En(q') \cap \{e\})$$

And thus, we can define the class STS_5 as follows.

Definition 4.16 (STS_5). Let $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ be a STS. It is in STS_5 if for any transition $q \xrightarrow{e} q'$ we have either $Reg(q \xrightarrow{e} q') = \Sigma_T$ or we have $Reg(q \xrightarrow{e} q') \cup \Sigma_T = \emptyset$, where $\Sigma_T = \{e \in \Sigma \mid \Gamma(e) \neq \delta_0\}$ is the set of timed events in $\mathcal{S}$.

Regenerative steps corresponds to a probabilistic reset in the underlying stochastic process. In other words, once a regenerative step is taken, the future state of the stochastic process does not depends on what happened before the regenerative step. In order to define the finite probability of a sojourn path, we will first assume that the sojourn path starts with a state where timed events are enabled, then remove this restriction and show that we can decompose a sojourn path into probabilistically independent sub-paths where each sub-path starts with a state where timed events are enabled.

Definition 4.17 (**Non regenerative sojourn paths**). Let $\omega = (q_0, (e_i, t_i, q_i)_{i \leq n})$ be a sojourn path of a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ in STS_5. We say that ω is not regenerative if we have $En(q_0) \cap \Sigma_T \neq \emptyset$ and if $Reg(q_i \xrightarrow{e_i} q_{i+1} = \emptyset$ for all $i \geq 0$.

We first characterize the probability of non regenerative paths. To this end, we first have :

Lemma 4.9. *If ω is not regenerative, then*

- *no event in ω appears more than once,*
- $\{e_i\} \subseteq En(q_0)$,
- $En(q_{i+1}) \subset En(q_i)$.

Since no event appears more than once in ω and since all the clocks of the (timed) events in ω are sampled at the same state and at the same epoch, we can consider a single r.v. $C(e)$ denoting the value of the clock of the event e. We can already model the fact that the event e has an activity delay following $\Gamma(e)$ by setting $C(e) \sim \Gamma(e)$.

Let q_i be a state appearing in ω and let the r.v. $Epoch(q_i)$ denotes the epoch when the process reach the state q_i. The state q_i is reached after firing the event e_{i-1} during the transition $q_{i-1} \xrightarrow{e_{i-1}} q_i$. Since the clock $C(e_{i-1})$ is sampled only once at epoch 0, we have $Epoch(q_i) = C(e_{i-1})$.

In any state q_i of ω, the sojourn time T_i must be less that t_i. This sojourn time in the state q_i is the r.v. $Epoch(q_{i+1}) - Epoch(q_i)$. Since $Epoch(q_i) = C(e_{i-1})$, the sojourn time T_i is $T_i = C(e_i) - C(e_{i-1})$ for $i > 1$. For the first sojourn time T_0, we have $T_0 = C(e_0)$.

Finally, in order for an event e_i to win the race condition between the competing events in $\mathrm{En}(q_i)$, we must have $\bigwedge_{e \in \mathrm{En}(q_i)} C(e_i) \leq C(e)$.

Thus we have $\mathbf{P}(\omega) =$

$$\mathbf{P}(Q_0 = q_0, E_0 = e_0, T_0 \leq t_0, Q_1 = q_1, E_1 = e_1, T_1 \leq t_1, \ldots, Q_n = q_n, E_n = e_n, T_n \leq t_n) =$$

$$\mathbf{P}(C(e_0) \leq t_0 \wedge \bigwedge_{e \in En(q_0)} C(e_0) < C(e) \wedge \bigwedge_{i \leq n} C(e_i) - C(e_{i-1}) \leq t_i \wedge \bigwedge_{e \in En(q_i)} C(e_i) \leq C(e)).$$

In order to compute this probability via integration, we express it as a $n+1$-dimensional region of $\mathbb{R}_+^{n+1}$ with axes labeled in order by $c_0, \ldots, c_n$. Since every events in ω appears only once, let $ind(e)$ be the index value i such that $ind(e) = i$ if $e = e_i$. Let R be the region defined by the inequalities $R = \left\{ c_0 \leq t_0 \wedge \bigwedge_{e \in En(q_0)} c_0 \leq c_{ind(e)} \wedge \bigwedge_{i \leq n} c_i - c_{i-1} \wedge \bigwedge_{e \in En(q_i)} c_i \leq c_{ind(e)} \right\}$. Since all the clocks r.v. C_i are independent, we have

$$\mathbf{P}(\omega) = \int_{\langle c_0, c_1, \ldots, c_n \rangle \in R} \prod_{0 \leq i \leq n} f_i(c_i), \tag{4.13}$$

where f_i is the probability density function of $\Gamma(e_i)$.

Property 4.10 (**Simplicity of** R)**.** *The region R is equal to the $(c_n, \ldots, c_0)$-simple domain D represented by the inequalities*

$$D = \{c_0 \leq t_0 \wedge c_0 \leq c_1 \leq c_0 + t_1 \wedge \cdots \wedge c_{n-1} \leq c_n \leq c_{n-1} + t_n\}.$$

Proof. Since $\bigwedge_{i \leq n} c_i - c_{i-1} \leq t_i$ is one of the terms in the inequalities of R, we have $R \subseteq D$. Let $\langle c_0, \ldots, c_n \rangle$ be an element of D. By transitivity, we know that for every index i, we have $c_i \leq c_j$ if $j > i$. Since we have $En(q_{i+1}) \subset En(q_i)$, we have $ind(e) > i$ for $e \in En(q_i)$. Thus, we have $c_i < c_{ind(e)}$ for all $e \in En(q_i)$. □

We now restrict the following formulas to the case where all r.v. are absolutely continuous. Since R and D are the same subset of $\mathbb{R}_+^{n+1}$, since D is $(c_n, \ldots, c_0)$-simple, we can repeatedly integrate the right hand side of 4.13 and thus we have $\mathbf{P}(\omega) =$

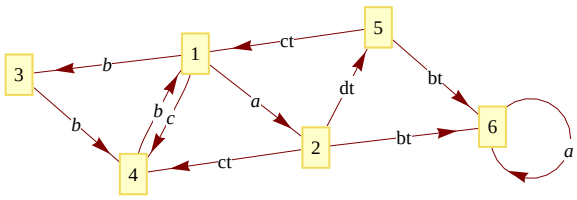
$$\int_0^{t_0} \left(f_0(x_0) \cdot \int_{x_0}^{x_0+t_1} \left(f_1(x_1) \cdot \int_{x_1}^{x_1+t_2} \left(f_2(x_2) \cdots \int_{x_{n-1}}^{x_{n-1}+t_n} f_n(x_n) dx_n dx_{n-1} \ldots \right) dx_2 \right) dx_1 \right) dx_0. \tag{4.14}$$

We now remove the restriction on ω and consider paths with multiple regeneration points. Let ω be a sojourn path in a STS $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$. The decomposition of ω into probabilistically independent sub paths $\omega = \omega_1 \omega_2 \omega_3 \ldots \omega_n$ is the sequence of sub paths $(\omega_i)_{i \leq n}$ where ω_i ends with a transition $q \xrightarrow{e} q'$ such that $Reg(q \xrightarrow{e} q') = \Sigma_T$. Since all clocks are regenerated at the end of a sub-path ω_i, two consecutive sub paths are probabilistically independent and any sub-path ω_i is not regenerative.

Simulation of STS in STS_5

We consider a STS with non regenerative paths of length 2.

```
Σ = {a, b, c, d, dt, bt, ct};
W[σ_] := Switch[σ, "init", 1, a, 10, b, 2, c, 1, d, 5]
Q = {1, 2, 3, 4, 5, 6};
Γ[σ_] := Switch[σ, "init", 0, a, 0, b, 0, c, 0, d, 0,
  bt, NormalDistribution[12, 3],
  ct, UniformDistribution[{9, 12}],
  dt, NormalDistribution[8, 1]
 ]

S = {
    {1 → 2, a}, {1 → 3, b}, {1 → 4, c},
    {2 → 6, bt}, {2 → 5, dt}, {2 → 4, ct},
    {3 → 4, b},
    {5 → 6, bt}, {5 → 1, ct}, {4 → 1, b},
    {6 → 6, a}
   };
GraphPlot[S, DirectedEdges → True, VertexLabeling → True]
```



In order to simulate this STS, we implement the variable time advance algorithm [She93] that was described in section sec. 4.3.1. To this end, we first define a function `Regenerate(q,e,q')` that returns the set of event that should be regenerated after the transition $q \xrightarrow{e} q'$. Note that for STS in STS_5, this function always return one of the sets: all the timed events, a set of untimed events, the empty set.

```
In[52]:= SetDiff[X_, Y_] := Complement[X, Intersection[X, Y]]

In[62]:= Regenerate[q_, e_, qp_] :=
          Union[SetDiff[EnabledIn[qp], EnabledIn[q]],
           Intersection[EnabledIn[qp], {e}]]

         Step[{ω_, clocks_}, 𝒲_] := Module[{
            sfin, enabled, newWeights, totalWeight, nextE,
            minclocks, winners, nextT, nextS, winnerLabels, currentT,
            nextClock, toRemove},
           sfin = ω[[-1, 2]];
           currentT = ω[[-1, 4]];
           If[Length[clocks] == 0, Return[{ω, {}}]];
           minclocks = GatherBy[clocks, #[[2]] &];
           winners = SortBy[minclocks, #[[1, 2]] &][[1]];
           {nextT, winnerLabels} = {winners[[1, 2]], winners[[All, 1]]};
           nextE = If[Length[winnerLabels] == 1, winnerLabels[[1]],
             newWeights = 𝒲[ω, #] & /@ winnerLabels;
             totalWeight = Total[newWeights];
             RandomChoice[newWeights/totalWeight → winnerLabels]
            ];
           nextS = TransitionFrom[sfin, nextE][[1, 2]];
           toRemove = SetDiff[EnabledIn[sfin], EnabledIn[nextS]];
           nextClock = Union[
             Select[clocks, #[[1]] =!= nextE ⋀ Not[MemberQ[toRemove, #[[1]]]] &],
             (#1 → nextT + Sample[Γ[#1]]) & /@ Regenerate[sfin, nextE, nextS]
            ];
          Return[{Append[ω, {sfin, nextS, nextE, nextT}], nextClock}]
          ]
         𝒲1[ω_, σ_] := If[W[σ] =!= #, W[σ], 1]
         Step1[{ω_, clock_}] := Step[{ω, clock}, 𝒲1]

In[66]:= ω = {{1, 1, "init", 0}}
         nextClock = (#1 → Sample[Γ[#1]]) & /@ EnabledIn[ω[[-1, 2]]]
         Step1[{ω, nextClock}]

Out[66]= {{1, 1, init, 0}}

Out[67]= {a → 0, b → 0, c → 0}

Out[68]= {{{1, 1, init, 0}, {1, 2, a, 0}}, {bt → 15.7947, ct → 9.44422, dt → 7.44469}}
```

We now compare the exact probability (as given by Eq. (4.14)) with an empirical estimation. The empirical estimation of the path

$$1 \xrightarrow{a,0} 2 \xrightarrow{dt,d1} 5 \xrightarrow{ct,d2} 1 \xrightarrow{a,0} 2 \xrightarrow{dt,d1} 5 \xrightarrow{ct,d2} 1 \xrightarrow{a,0} 2 \xrightarrow{dt,d1} 5$$

is given by the following code.

```
In[96]:= IsInL1TimedWithDeadlines[s_, d1_, d2_] :=
          Untime[s] == exPath && VectorLessEqual[Time[s], {0, d1, d2, 0, d1, d2, 0, d1}]

In[97]:= N[Length[Select[samples, IsInL1TimedWithDeadlines[#, 8, 4] &]] / Length[samples]]

Out[97]= 0.01645
```

while the theoretical value is given by

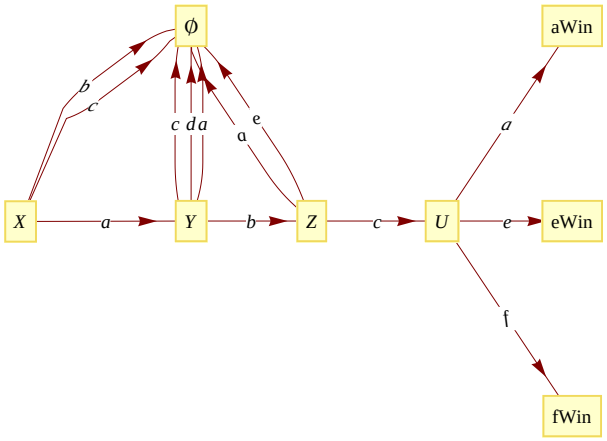
```
In[92]:= d1 = 8;
         d2 = 4;

In[94]:= 10 / 13 *
          NIntegrate[(1 - CDF[Γ[bt], c]) * PDF[Γ[ct], c] * PDF[Γ[dt], d],
           {d, 0, d1}, {c, d, d + d2}] *
          10 / 13 *
          NIntegrate[(1 - CDF[Γ[bt], c]) * PDF[Γ[ct], c] * PDF[Γ[dt], d],
           {d, 0, d1}, {c, d, d + d2}] *
          NIntegrate[(1 - CDF[Γ[bt], d]) * (1 - CDF[Γ[ct], d]) * PDF[Γ[dt], d],
           {d, 0, d1}]

Out[94]= 0.0195693
```

4.3.7 Probability of a sojourn path in STS_6: General case

In the general case, we must account for regeneration of clocks that do not happen at the same epoch. This implies that in a given state, the clocks of competing events account for elapsed time since regeneration (like in STS_5) but also for a delay between regeneration epochs. We illustrate this on the following example.

Example 4.1.

```
In[25]:= S = {
           {X → Y, a}, {X → ∅, b}, {X → ∅, c},
           {Y → Z, b}, {Y → ∅, c}, {Y → ∅, d}, {Y → ∅, a},
           {Z → U, c}, {Z → ∅, a}, {Z → ∅, e},
           {U → fWin, f}, {U → eWin, e}, {U → aWin, a}
          };
         Γ[σ_] := Switch[σ,
           a, NormalDistribution[15, 3],
           b, NormalDistribution[24, 3],
           c, UniformDistribution[{15, 18}],
           d, NormalDistribution[8, 1],
           e, NormalDistribution[10, 2],
           f, ExponentialDistribution[1 / 3]
          ]
         TreePlot[S, Left, X, DirectedEdges → True, VertexLabeling → True]
```



We will follow the clocks and regenerations as we step trough the path $X \xrightarrow{a} Y \xrightarrow{b} Z \xrightarrow{c} U \xrightarrow{e} eWin$ in this STS. As we did for STS_5, we will express this sojourn path as a conjunction of constraints over n clock variables, show that this conjunction yields a simple domain of $\mathbb{R}^n$ and use repeated integration to measure the probability of this domain.

Starting from state X, the initial clock structure is $\langle A : a, B : b, C : c\rangle$. Suppose that a is the minimal epoch, the first step in S is thus $X \xrightarrow{a} Y$. Once in Y, the global epoch is set to a, the clocks of the events $\{a, d\}$ are regenerated and thus the new clock structure is $\langle A : a + a', B : b, C : c, D : d + a\rangle$. Suppose now that the minimal epoch in the clock structure is b. After the transition $Y \xrightarrow{b} Z$, the global epoch is set to b, and since the set of newly enabled events is $\{e\}$, we update the clock structure and obtain $\langle A : a + a', C : c, E : e + b\rangle$. If c is the

minimal epoch, after the transition $Z \xrightarrow{c} U$, the global epoch is set to c, and the clock structure is updated to $\langle A : a + a', E : e + b, F : f + c\rangle$. For this example, the race condition in the state U involves three events whose delays are r.v. following a sum of independent and non identical r.v.. Furthermore, in the state U, we must account for the elapsed time since regeneration and (equivalently) the races already lost, i.e. we have $a + a' > b \wedge a + a' > c \wedge e + b > c$.

In order to compute the probability of the sojourn path

$$\omega = X \xrightarrow{a,tX} Y \xrightarrow{b,tY} Z \xrightarrow{c,tZ} U \xrightarrow{e,tU} eWin$$

where tX, tY, tZ, tU are real valued constants, we have to determine the regeneration epoch of each event racing in each state of ω as well as all the races that has been lost. Consider for example the sojourn time T_4 in the last state. The sojourn path specify that $T_4 \leq tU$. The sojourn r.v. T_4 is equal to $Epoch(eWin) - Epoch(U)$. The r.v. $Epoch(eWin)$ is defined as the sum $C_e + Epoch(\text{last regeneration of } e)$ where $C_e \sim \Gamma(e)$. Since the event e has been regenerated after the transition $Y \xrightarrow{b} Z$, we have $Epoch(\text{last regeneration of } e) = Epoch(Z)$. The r.v. $Epoch(Z)$ is defined as $Epoch(Z) = C_b - Epoch(\text{last regeneration of } b)$. Since b has been regenerated when reaching the initial state X, we have $Epoch(\text{last regeneration of } b) = Epoch(X) = 0$, and thus $Epoch(Z) = C_b$ and thus $Epoch(eWin) = C_e + C_b$. Similarly, we have

$$Epoch(U) = C_c + Epoch(\text{last regeneration of } c) = C_c + Epoch(X) = C_c$$

and thus we have $\{T_4 \leq tU\} = \{C_e + C_b - C_c < tU\}$. Furthermore, in the state U, the r.v. representing the epoch of firing the event e is $C_e + C_b$. For the event a, it is $C_a + Epoch(\text{last regeneration of } a)$, with $Epoch(\text{last regeneration of } a) = Epoch(Y) = C'_a$ and thus the epoch of firing the event a is $C_a + C'_a$. Finally, for the event f, the epoch of firing is $C_f + Epoch(U) = C_f + C_c$. This implies that in the state U, the sojourn time is less than tU and the event e wins if

$$C_b + C_e < C_a + C'_a \wedge C_b + C_e < C_c + C_f \wedge C_b + C_e - C_c < tU.$$

which can be restated as the following a, b, c, e, d, f, a'-simple domain

$$C_e < tU - C_b + C_c \wedge C_f > C_b - C_c + C_e \wedge a' > -C_a + C_b + C_e$$

where each C_l is a r.v. following $\Gamma(l)$.

By performing similar computation, the event c wins in state Z in less than tZ t.u. if

$$C_c < C_b + tZ \wedge C_e > C_c - C_b \wedge C'_a > C_c - C_a.$$

In state Y, the event b wins in less than tY t.u. if

$$C_b < C_a + tZ \wedge C_c > C_b \wedge C_d > C_b - C_a \wedge C'_a > C_b - C_a.$$

In state X, the event a wins in less than tX t.u. if

$$C_a < tX \wedge C_b > C_a \wedge C_c > C_a.$$

Finally, the probability $\mathbf{P}(\omega)$ of the sojourn path ω is

$$\begin{aligned}\mathbf{P}(\quad & 0 < C_a < tX \wedge C_a < C_a < C_a + tY \wedge C_a < C_c < C_a + tZ \wedge C_c - C_a < C_e < -C_a + C_c + tU \\ & \wedge C_d > C_a - C_a \wedge C_f > C_a - C_c + C_e \wedge C'_a > -C_a + C_a + C_e).\end{aligned}$$

We now generalize this construction. Given a sojourn path ω, we want to express the constraints over Q, E, T present in ω by constraints over C_e and t which are resp. r.v. denoting the value of the remaining activity delay of some event e and real valued constants.

Since STS_6 are the general case, an event may be regenerated or appear multiple times in ω. Let Σ_ω be the *multiset* over Σ defined by $\Sigma_\omega = Reg(q_i \xrightarrow{e_i} q_{i+1})_{i \leq n}$. An event e has multiplicity k if its timer is regenerated k times during a realization going trough the states of ω. All these regenerated timers are different independent and identically distributed r.v. $C_i(e)$ following $\Gamma(e)$. Thus, even if n events appears in ω, its probability will involve $\text{card}(\Sigma_\omega)$ different random variables. In order to avoid explicitly stating which r.v. represent the timer of the ith regeneration of e, we impose without loss of generality that all the multiplicities in Σ_ω are 1. If it is not the case and since all the regeneration of a timer are independent and identically distributed, we can augment Σ with unique labels $e_1, e_2, \ldots$ and update Γ and W accordingly by setting $\Gamma(e_i) = \Gamma(e), W(e_i) = W(e)$. We will then denote by $C(e)$ *the timer random variable* associated with the event e and by χ the set of the timer r.v. of all the events in Σ_ω. Note that with the assumption that Σ_ω has a multiplicity of 1 for each event, we have $\text{card}(\chi) = \text{card}(\omega)$.

Given a sojourn path ω and an event e, the last regenerative state $Reg_\omega(e)$ is the last state of ω where the event e is regenerated. That is, $Reg_\omega(e)$ maps an event $e \in \Sigma_\omega$ to a state $q \in \omega$ and is defined by

$$Reg_\omega(e) = q_i, \text{ s.t } e \in Reg(q_i \xrightarrow{e_i} q_{i+1}), \text{ and s.t. } \forall j > i, e \notin Reg(q_j \xrightarrow{e_j} q_{j+1}),$$

where $Reg(q \xrightarrow{e} q')$ is the function defined in Def. 4.15.

The r.v. $Epoch_\omega(q_i)$ representing the epoch at which the state q_i is reached in the path ω is a linear combination of timer random variables. The function $Epoch_\omega(q_i)$ thus map a state $q_i \in \omega$ to a term over χ, and is defined recursively by

$$Epoch_\omega(q_i) = \begin{cases} 0 & \text{if } |\omega| = 0 \\ C(e) + Epoch_{\omega|i}(Reg_{\omega|i-1}(e)) & \text{otherwise} \end{cases}$$

where e is the event in Σ_ω for which we have $q_{i-1} \xrightarrow{e} q_i$ in ω. In the definition of $Epoch_\omega(q_i)$, the term $Epoch_{\omega|i}(Reg_{\omega|i-1}(e))$ is a linear combination of timer r.v. representing the epoch of regeneration of the event e.

By using $Epoch_\omega(q_{i+1})$ we can describe the sojourn time in the state q_i as being the difference between the two successive epoch. More formally, the r.v. $Sojourn_\omega(q_i)$ representing the sojourn time in state q_i is a linear combination of timer r.v. and is defined by

$$Sojourn_\omega(q_i) = Epoch_\omega(q_{i+1}) - Epoch_\omega(q_i).$$

Finally, the epoch at which a given event e fires can also be expressed in terms of timer random variables. For any event e in ω, we define the r.v. $FiringEpoch_\omega(e)$ as

$$FiringEpoch_\omega(e) = C(e) + Epoch_{\omega|-1}(Reg_\omega(e)).$$

Both functions $Sojourn$ and $Epoch$ maps a state visited in ω to a linear combination of timer random variables, while the function $FiringEpoch$ maps an event fired in ω to a linear combination of timer random variables. We now use these three functions to express the probabilistic event $\{e$ wins the race in state q_n before time $t_n\}$ as constraints over the realizations of the timer random variables.

In the last state q_n of ω, the sojourn time of the process is bounded by t_n. Therefore, we already must impose that

$$Sojourn_\omega(q_n) \leq t_n.$$

Futhermore, for the event e_n to win the race condition in state q_n, we must have

$$\bigwedge_{e' \in En(q_n), e' \neq e_n} FiringEpoch_\omega(e_n) \leq FiringEpoch_\omega(e).$$

Then, for any sojourn path ω, we must have

$$\bigwedge_{e_i \in \omega} \left(Sojourn_{\omega|i}(q_i) \leq t_i \wedge \bigwedge_{e \in En(q_i), e' \neq e_i} \left(FiringEpoch_{\omega|i}(e_i) \leq FiringEpoch_{\omega|i}(e) \right) \right). \tag{4.15}$$

However the region of $\mathbb{R}^{\mathrm{card}(\chi)}$ satisfying this inequality is not simple and thus not amenable to measure with integration. Still, we can remark that Σ_ω can be partitioned in the three partitions:

- $Winners(\omega) = \{e \mid e \in \omega\}$,
- $Delayed(\omega) = \{e \mid e \notin \omega, e \in En(q_n)\}$,
- $Loosers(\omega) = \{e \mid e \notin \omega, e \notin En(q_n)\}$,

where $Winners$ represent the set of events that won a race condition, $Delayed$ the set of events that did not win any race condition but are still enabled and may thus win a future race condition, and where $Loosers$ is the set of event that lost all race conditions and were deactivated in some state visited by ω. For any event $e_i \in Winners$, we know that in state q_i the event must fire before the deadline t_i, t_i being relative to the time at which the process reached q_i. Consider again the STS of the example Ex. 4.1, the sojourn path $\omega = X \xrightarrow{a,tX} Y \xrightarrow{b,tY} Z \xrightarrow{c,tZ} U \xrightarrow{e,tU} eWin$, and particularly the three transitions $Y \xrightarrow{b} Z \xrightarrow{c} U \xrightarrow{e} eWin$. In the following, we omit the subscript ω. The sojourn in Z is given by

$$\begin{aligned} Sojourn(Z) = \quad & Epoch(U) - Epoch(Z) \\ & C(c) + Epoch(Reg(c)) - C(b) - Epoch(Reg(b)), \end{aligned}$$

and the term

$$\begin{aligned} C(c) - Sojourn(Z) = \quad & C(c) - \big(C(c) + Epoch(Reg(c)) - C(b) - Epoch(Reg(b))\big) \\ & C(c) - C(c) - Epoch(Reg(c)) + C(b) + Epoch(Reg(b)) \\ & C(b) + Epoch(Reg(b)) - Epoch(Reg(C)) \end{aligned}$$

represent the firing epoch of b relatively to the regeneration of c. Since the system reached the state Z exactly at the firing epoch of b, we must have

$$\begin{aligned} & C(b) + Epoch(Reg(b)) - Epoch(Reg(C)) \leq C(c) \leq C(b) + Epoch(Reg(b)) - Epoch(Reg(C)) + t_Z \\ \Leftrightarrow \quad & C(c) - Sojourn(Z) \leq C(c) \leq C(c) - Sojourn(Z) + t_Z. \end{aligned}$$

More generally, for any event e_i such that $q_i \xrightarrow{e_i,t_i} q_{i+1} \in \omega$, we have

$$C(e_i) - Sojourn(q_i) \leq C(e_i) \leq C(e_i) - Sojourn(q_i) + t_i,$$

where the term $C(e_i) - Sojourn(q_i)$ does not involve $C(e_i)$ and is a linear combination of timer r.v. of the events that won the i first race.

For a delayed event $e \in Delayed$, we know that it lost all n race conditions, and thus that for every event $e_i \in \omega$, we have $FiringEpoch_{\omega|i(e)} \geq FiringEpoch_{\omega|i}(e_i)$. However, for any i, we have $FiringEpoch_{\omega|i}(e_i) \leq FiringEpoch_{\omega|i+1}(e_{i+1})$, and thus it is sufficient to impose for any delayed event $e \in Delayed$ that

$$FiringEpoch_{\omega}(e) \geq FiringEpoch_{\omega}(e_n).$$

Finally, for any event $e \in Loosers$, let $Last(e) = q_i$ be the state such that $q_i \xrightarrow{e_i} q_{i+1}$ and $e \in En(q_i)$ and $e \notin En(q_{i+1})$. In words, $Last(e)$ is the last state where e was active, and is the state where it lost its last race. We must have

$$FiringEpoch_{\omega|i}(e) \geq FiringEpoch_{\omega|i}(e_i).$$

Now let D_ω be the set of inequality over timer r.v. defined by

$$\begin{aligned} D_\omega = & \bigwedge_{e_i \in Winners(\omega)} \big(C(e_i) - Sojourn(q_i) \leq C(e_i) \leq C(e_i) - Sojourn(q_i) + t_i\big) \wedge \\ & \bigwedge_{e \in Delayed(\omega)} \big(FiringEpoch_{\omega}(e) \geq FiringEpoch_{\omega}(e_n)\big) \wedge \\ & \bigwedge_{e \in Loosers(\omega), q_i = Last(e)} \big(FiringEpoch_{\omega|i}(e) \geq FiringEpoch_{\omega|i}(e_i)\big). \end{aligned} \tag{4.16}$$

Recall that we are given a rewriting such that all events appears with multiplicity one in Σ_ω. Then the sequence $\prec = c_{e_0}, c_{e_1}, \ldots, c_{e_n}, (c_e)_{e \in \Sigma_\omega, e \notin \omega}$ where we enumerate first the events appearing in ω in order, then all the events regenerated in ω in any fixed order is a complete order over Σ_ω. Label the axes of $\mathbb{R}^{\mathrm{card}(\Sigma_\omega)}$ by $\prec$, then for a sojourn path ω, we have the following property.

Property 4.11 (Simplicity of D_ω). *The region $\{D_\omega\} \subseteq \mathbb{R}^{\mathrm{card}(\chi)}$ is a $\succ$-simple domain.*

Proof. Consider in order the axes of $\prec$ and recall that each axis is associated to an unique event e, and thus to a unique timer r.v. $C(e)$. Let e be an event.

If $e \in Winners(\omega)$, then there exists i s.t. $e = e_i$. Let j be the index of i in $\prec$, then the range in D_ω of the variable c_j associated with the r.v. $C(e_i)$ is constrained by

$$C(e_i) - Sojourn(q_i) \leq C(e_i) \leq C(e_i) - Sojourn(q_i) + t_i.$$

Since $Sojourn(q_i)$ is a linear combination over some timer r.v. in $\{C(e_k)\}$ such that $k \leq i$ and where $C(e_i)$ appears only once, the term $C(e_i) - Sojourn(q_i)$ is a linear combination of timer r.v. in $\{C(e_k)\}$, and thus the variable c_j is bounded by variables of lower index.

If $e \in Delayed$, for the index j of e in $\prec$, the variable c_j is constrained by

$$FiringEpoch_{\omega}(e) \geq FiringEpoch_{\omega}(e_n)$$

and since by construction j is greater than the index of the axis associated with e_n, then the variable c_j is only dependent by variables of lower index.

Finally, if $e \in Loosers$, for the index j of e in $\prec$, the variable c_j is constrained by

$$FiringEpoch_{\omega|i}(e) \geq FiringEpoch_{\omega|i}(e_i)$$

for some (event) index i. Since by construction j is greater than the index of the axis associated with e_i, then the variable c_j bounded by variables of lower index.

Since for any axis of $\mathbb{R}^{\text{card}(\chi)}$ appearing as a variable in $\{D_\omega\}$ we have lower and upper constraints that are dependendent of variables of lower index w.r.t to the order $\prec$, D is a $\succ$-simple domain. □

We can now express the probability of a sojourn path in terms of a probability over clock r.v..

Definition 4.18 (Probability of a sojourn path). Let $\mathcal{S} = \langle Q, \longrightarrow, \Sigma, W, \Gamma \rangle$ be a STS where all delay distributions in $\Gamma(\Sigma)$ are continuous and where $\rightarrow$ is deterministic, then for any sojourn path ω of $\mathcal{S}$, the probability of ω is defined by

$$\mathbf{P}_{\mathcal{W}}(\omega) = \mathbf{P}(\bigwedge_{e_i \in \omega} e_i \text{ wins the race in state } q_i \text{ before time } t_i).$$

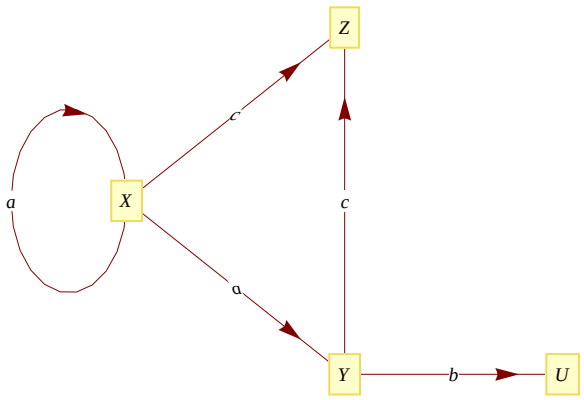
Let Σ_ω be a set of r.v. $\{C_e\}$ such that $C_e \sim \Gamma(e)$, then we have

$$\mathbf{P}(\bigwedge_{e_i \in \omega} e_i \text{ wins the race in state } q_i \text{ before time } t_i) = \int_D \prod_{\{c_{e_i} | C_{e_i} \in \Sigma_\omega\}} f_{e_i}(c_{e_i})$$

where D_ω is the $(c_{e_{(n-i)}})_{i \leq n}$-simple domain with $n = \text{card}(\Sigma_\Omega)$ defined in Eq. 4.16

General distributions We now consider that the delays of events in the STS can follow any kind of probabilistic distribution with positive support. As it was the case for $\mathbf{P}_4$, we need to consider all the possible ties and weight them by using a weight scheduler. We will fully express the probability of a sojourn path on an example.

```
S = {
    {X → X, a}, {X → Y, a}, {X → Z, c},
    {Y → U, b}, {Y → Z, c}
  };
TreePlot[S, Left, X, DirectedEdges → True, VertexLabeling → True]
```



Consider the sojourn path $\omega = X \xrightarrow{a,tX} Y \xrightarrow{b,tY} U$ in S. By applying the definition 4.18, the clock constraints associated with ω representing the clock values for which b wins the second

race condition before tU is $D_\omega = \{0 \leq a < 8 \wedge 0 \leq b < 12 \wedge c \geq a + b\}$. In the case where the clock r.v. can be either discrete or continuous, we have to account for all the possible ties. To this end, we decompose D_ω into disjoint subsets by considering that any inequality of the form $a \leq b$ is equivalent to the disjunction $(a < b) \vee (a = b)$. Therefore we have the decomposition of D_ω into 2^3 disjunct domains,

$$\begin{aligned} D_\omega &= \{a = 0 \wedge b = 0 \wedge c = a + b\} \\ &\cup \{a = 0 \wedge b = 0 \wedge c > a + b\} \\ &\cup \{a = 0 \wedge 0 < b < 12 \wedge c > a + b\} \\ &\cup \{a = 0 \wedge 0 < b < 12 \wedge c = a + b\} \\ &\cup \{0 < a < 8 \wedge b = 0 \wedge c = a + b\} \\ &\cup \{0 < a < 8 \wedge b = 0 \wedge c > a + b\} \\ &\cup \{0 < a < 8 \wedge 0 < b < 12 \wedge c = a + b\} \\ &\cup \{0 < a < 8 \wedge 0 < b < 12 \wedge c > a + b\}. \end{aligned}$$

Each of these sub-domains are simple domain and can be measured with formulas similar to the one in definition Def. 4.18.

To each of this sub-domain, we associate a *tie sequence*. A tie sequence $\{\Sigma_q\}_{q\in\omega}$ is a sequence of subset of Σ indexed by the successive states in the sojourn path. Two events are in the same subset if their associated firing epoch are equals. In other words, each element of a tie sequence $\{\Sigma_q\}$ associated to a simple domain D_i is a subset of the events $En(q)$ in q for which the clock constraints in D_i are not sufficient to determine an unique successor state.

Consider the first sub-domain $D_1 = \{a = 0 \wedge b = 0 \wedge c = a + b\}$. The firing epoch of the event a is 0, the firing epoch of the event c is 0, the firing epoch of the event b is 0. Therefore, this sub-domain is mapped to the tie sequence $(\{a, c\}_X, \{b, c\}_Y)$. For the sub-domain $D_4 = \{a = 0 \wedge 0 < b < 12 \wedge c = a + b\}$, we know that the firing epoch of the event a is 0, the firing epoch of the event b is the r.v. b, the firing epoch of the event c is the r.v. b. Therefore we have the tie sequence $(\{\emptyset\}_X, \{b, c\}_Y)$. The other tie sequences are given in the following table.

D_1	$\{a = 0 \wedge b = 0 \wedge c = a + b\}$	$(\{a, c\}_X, \{b, c\}_Y)$
D_2	$\{a = 0 \wedge b = 0 \wedge c > a + b\}$	$(\{\emptyset\}_X, \{\emptyset\}_Y)$
D_3	$\{a = 0 \wedge 0 < b < 12 \wedge c > a + b\}$	$(\{\emptyset\}_X, \{\emptyset\}_Y)$
D_4	$\{a = 0 \wedge 0 < b < 12 \wedge c = a + b\}$	$(\{\emptyset\}_X, \{b, c\}_Y)$
D_5	$\{0 < a < 8 \wedge b = 0 \wedge c = a + b\}$	$(\{a, c\}_X, \{\emptyset\}_Y)$
D_6	$\{0 < a < 8 \wedge b = 0 \wedge c > a + b\}$	$(\{\emptyset\}_X, \{\emptyset\}_Y)$
D_7	$\{0 < a < 8 \wedge 0 < b < 12 \wedge c = a + b\}$	$(\{\emptyset\}_X, \{b, c\}_Y)$
D_8	$\{0 < a < 8 \wedge 0 < b < 12 \wedge c > a + b\}$	$(\{\emptyset\}_X, \{\emptyset\}_Y)$

We now map each tie sequence to the appropriate call to the weight scheduler. Since for the first sub-domain, the events a and c tie in the state X, the weight scheduler is applied to associate a weight to all the conflicting transitions in X. Similarly, the events b and c tie in the state Y, and the tie is resolved with a weight scheduler. Therefore For the sojourn path

$\omega = X \xrightarrow{a,tX} Y \xrightarrow{b,tY} U$, for a weight scheduler $\mathcal{W}$, we associate the weight

$$W_{\mathcal{W}}(\omega, D_1) = \frac{\mathcal{W}(\omega|0, X \xrightarrow{a} Y)}{\mathcal{W}(\omega|0, X \xrightarrow{a} Y) + \mathcal{W}(\omega|0, X \xrightarrow{a} X) + \mathcal{W}(\omega|0, X \xrightarrow{c} Z)} \times \frac{\mathcal{W}(\omega|1, Y \xrightarrow{b} U)}{\mathcal{W}(\omega|1, Y \xrightarrow{b} U) + \mathcal{W}(\omega|1, Y \xrightarrow{c} Z)},$$

to the first sub domain. We finally have for the sojourn path ω and for general distributions

$$\mathbf{P}_6(\omega) = \sum_{D_i \in \mathcal{D}} W_{\mathcal{W}}(\omega, D_i) * \mathbf{P}(D_i)$$

where $\mathcal{D}$ is the decomposition of D into disjoint domains D_i previously illustrated.

Notice that in the case where $\Gamma(a) = \Gamma(b) = \Gamma(c) = \delta_0$, the only sub-domain that has a non zero measure is the sub domain 1 and hence we have $\mathbf{P}_{\mathcal{W}2}(\omega) = \mathbf{P}_6(\omega)$.

Numerical validation

We consider again the STS $\mathcal{S}$ of the example Ex. 4.1. Given a sojourn path ω of $\mathcal{S}$, we can derive the constraints over clocks with the functions *Epoch* and *Sojourn*.

In[117]:=
```
EnabledIn[s_] := Select[S, #[[1, 1]] == s &][[All, 2]]
TransitionFrom[s_, e_] := Select[S, MatchQ[#, {s → _, e}] &][[1]]
SetDiff[X_, Y_] := Complement[X, Intersection[X, Y]]

Regenerate[q_, e_, qp_] :=
 Union[SetDiff[EnabledIn[qp], EnabledIn[q]],
  Intersection[EnabledIn[qp], {e}]]

LastRegenerativeState[p_, e_] :=
 With[{ω = Append[Reverse[p], {"", "", 0, p[[1, 1]]}]},
  Select[ω, MemberQ[Regenerate[#[[1]], #[[2]], #[[4]]], e] &][[1, -1]]]

Epoch[p_, q_] := Module[{i, ev, found},
  If[Length[p] == 0, Return[0]];
  found = Position[Reverse[p], {_, _, _, q}];
  If[Length[found] == 0, Return[0]];
  i = Length[p] + 1 - found[[1, 1]];
  If[i == 1, Return[0]];
  ev = p[[i, 2]];
  ev + Epoch[p[[ ;; i]], LastRegenerativeState[p[[ ;; i - 1]], ev]]
 ]

Sojourn[p_, q_] := Module[{i, tgt},
  If[Length[p] == 0, Return[0]];
  i = Length[p] + 1 - Position[Reverse[p], {q, _, _, _}][[1, 1]];
  If[i == 1, Return[0]];
  tgt = p[[i, -1]];
  Epoch[p, tgt] - Epoch[p[[ ;; i]], q]
 ]

ClockRVInLastStateOfp[p_, e_] :=
 e + Epoch[p[[ ;; -2]], LastRegenerativeState[p, e]]

ConditionForWinningInLastStateOfP[p_, e_] := Module[{en, ϕ, clockRvs},
  en = EnabledIn[p[[-1, 1]]];
  clockRvs = (# → ClockRVInLastStateOfp[p, #]) & /@ en;
  ϕ = And @@ (#[[1]] < #[[2]] & /@ Tuples[{{e}, Complement[en, {e}]}]) /.
    clockRvs
 ]

EventWinsBeforeDeadlineInLastStateOfP[ω_, e_] :=
 ConditionForWinningInLastStateOfP[ω, e] ⋀
  Sojourn[ω, ω[[-1, 1]]] < ω[[-1, 3]]
```

We will compute the probability of the sojourn path defined as ω in the following code.

In[208]:=
```
TX = 9; TY = 8; TZ = 5; TU = 2;
ω = {{"", "", 0, X}, {X, a, TX, Y}, {Y, b, TY, Z}, {Z, c, TZ, U}, {U, e, TU, eWin}}
```

Out[209]=
```
{{, , 0, X}, {X, a, 9, Y}, {Y, b, 8, Z}, {Z, c, 5, U}, {U, e, 2, eWin}}
```

We can derive constraints over all the clocks that are regenerated during an execution in ω. Basically, we derive first constraints for each transition in ω. For a given transition $q \xrightarrow{e} q'$, the winning clock C_e must be smaller than all the clocks of the events in $En(q)$.

```
In[137]:= EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 2]], a] /. (2 a → a + ap)
          EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 3]], b] /. (2 a → a + ap)
          EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 4]], c] /. (2 a → a + ap)
          EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 5]], e] /. (2 a → a + ap)

Out[137]= a < b && a < c && a < 9

Out[138]= b < a + ap && b < c && b < a + d && -a + b < 8

Out[139]= c < a + ap && c < b + e && -b + c < 5

Out[140]= b + e < a + ap && b + e < c + f && b - c + e < 2
```

Since in ω, the clock of the event a is regenerated twice (once at the beginning and once when the system transition from X to Y), we use two r.v. a and ap to differentiate the two clocks. These constraints are exactly the one obtained in the example Ex. 4.1. We can now build a constraint over all the clocks regenerated in ω, i.e. the conjunction of the one step constraint and reduce it to a simple domain.

```
In[143]:= And[
            And @@ ((0 < #) & /@ Σ) ⋀ 0 < ap,
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 2]], a],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 3]], b],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 4]], c],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 5]], e]
          ] /. (2 a → a + ap)

Out[143]= 0 < a && 0 < b && 0 < c && 0 < d && 0 < e && 0 < f && 0 < ap && a < b &&
           a < c && a < 9 && b < a + ap && b < c && b < a + d && -a + b < 8 && c < a + ap &&
           c < b + e && -b + c < 5 && b + e < a + ap && b + e < c + f && b - c + e < 2

In[144]:= CylindricalDecomposition[%, {a, b, c, e, d, f, ap}]

Out[144]= 0 < a < 9 && a < b < 8 + a && b < c < 5 + b &&
           -b + c < e < 2 - b + c && d > -a + b && f > b - c + e && ap > -a + b + e
```

We now compare empirical estimators of $\mathbf{P}(\omega)$ with the exact probability as defined in definition Def. 4.18. Empirical estimators are based on the same step function than the one used to simulate STS in STS_5. The most frequent paths of $\mathcal{S}$ are obtained by tallying with un-timed paths.

```
In[155]:= samples = Table[
            Nest[Step1, {ω, (#1 → Sample[Γ[#1]]) & /@ EnabledIn[X]}, 4]
             [[1, 2 ;;]], {20 000}];
          SortBy[Tally[Untime /@ samples], #[[2]] & // MatrixForm
```

Out[156]//MatrixForm=

{{X, Y, a}, {Y, ∅, d}}	6
{{X, ∅, c}}	19
{{X, Y, a}, {Y, Z, b}, {Z, U, c}, {U, eWin, e}}	145
{{X, Y, a}, {Y, Z, b}, {Z, U, c}, {U, fWin, f}}	2989
{{X, ∅, b}}	3733
{{X, Y, a}, {Y, ∅, c}}	4891
{{X, Y, a}, {Y, Z, b}, {Z, U, c}, {U, aWin, a}}	8217

The real probability is computed via integration. We first consider the probability of the prefix $X \xrightarrow{a,12} Y$ of ω. The realizations of $\mathcal{S}$ that are in $[\![\omega]\!]$ are selected with a predicate $IsIn\omega$ over realizations. Then the empirical estimator of $\mathbf{P}(\omega)$ is $\mathrm{card}(())IsIn\omega(samples))/\mathrm{card}(())samples)$.

```
In[167]:= TX = 12;
          deadlines = {TX};
          ω = {{"", "", 0, X}, {X, a, TX, Y}};

In[170]:= IsInω[realization_] :=
           If[Length[realization] < Length[ω], False,
            With[{s = TimedRunWithEpochsToSojourn[realization][[ ;; Length[ω] - 1]]},
             Untime[s] === ω[[2 ;;, {1, 4, 2}]] ⋀
              VectorLessEqual[Time[s], deadlines]
            ]
           ]

In[171]:= N[ Length[Select[samples, IsInω]] / Length[samples] ]

Out[171]= 0.80395
```

We now derive from this prefix the set of constraints over the 3 clock random variables.

```
In[172]:= And[
            And @@ ((0 < #) & /@ {a, b, c}) ⋀
             EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 2]], a]
           ] /. (2 a → a + ap)
          ineqs = CylindricalDecomposition[%, {a, b, c, ap, e, d, f}]

Out[172]= 0 < a && 0 < b && 0 < c && a < b && a < c && a < 12

Out[173]= 0 < a < 12 && b > a && c > a
```

Which are constraints of the same kind than the one in STS_3. Since these constraints define a simple domain, we can perform repeated integration to obtain the real probability $\mathbf{P}(X \xrightarrow{a,12} Y)$ and compare it with the estimator.

```
Out[173]= 0 < a < 12 && b > a && c > a
```

```
In[174]:= NIntegrate[
           PDF[Γ[a], av] * PDF[Γ[b], bv] * PDF[Γ[c], cv],
           {av, 0, 12}, {bv, av, 100}, {cv, av, 100},
           Method → Automatic
          ]
          Abs[% - N[Length[Select[samples, IsInω]] / Length[samples]]]
Out[174]= 0.804729

Out[175]= 0.000779215
```

Computing the probability of the prefix $X \xrightarrow{a,12} Y \xrightarrow{b,8} Z$ follows the same method. The estimator and domain are first computed.

```
In[176]:= TX = 12; TY = 8;
          deadlines = {TX, TY};
          ω = {{"", "", 0, X}, {X, a, TX, Y}, {Y, b, TY, Z}};

In[179]:= N[Length[Select[samples, IsInω]] / Length[samples]]
Out[179]= 0.56335

In[182]:= And[
            And @@ ((0 < #) & /@ {a, b, c, d}) ∧ 0 < ap,
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 2]], a],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 3]], b]
          ] /. (2 a → a + ap);
          ineqs = CylindricalDecomposition[%, {a, b, c, ap, e, d, f}]
Out[183]= 0 < a < 12 && a < b < 8 + a && c > b && ap > -a + b && d > -a + b
```

The probability of this domain is given by

$$\int_{\{a<12 \wedge a<b<a+8 \wedge c>b \wedge ap>b-a \wedge d>b-a\}} f_A(a) * f_B(b) * f_C(c) * f_D(d) * f_A(ap),$$

however since we have $c > b, ap > b-a$ and $d > b-a$ we can rewrite this integration by using cumulative distribution functions as follows

$$\int_0^{12} \int_a^{a+8} f_A(a) \cdot f_B(b) \cdot (1 - F_A(b-a)) \cdot (1 - F_C(bv)) \cdot (1 - F_D(b-a)).$$

We can now compute the error between the empirical estimator and the real probability of $\mathbf{P}(X \xrightarrow{a,12} Y \xrightarrow{b,8} Z)$.

```
In[184]:= NIntegrate[
           PDF[Γ[a], av] * PDF[Γ[b], bv] * (1 - CDF[Γ[c], bv]) * (1 - PDF[Γ[a], bv - av]) *
            (1 - PDF[Γ[d], bv - av]),
           {av, 0, TX}, {bv, av, av + TY},
           Method → Automatic
          ]

Out[184]= 0.563575

In[185]:= Abs[% - N[Length[Select[samples, IsInω]] / Length[samples]]]

Out[185]= 0.00022491
```

We consider now a longer sojourn path $\omega = X \xrightarrow{a,12} Y \xrightarrow{b,8} Z \xrightarrow{c,5} U \xrightarrow{a,5} aWin$. We first compute the constraints over the clocks random variables.

```
In[204]:= And[
            And @@ ((0 < #) & /@ Σ) ⋀ 0 < ap,
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 2]], a],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 3]], b],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 4]], c],
            EventWinsBeforeDeadlineInLastStateOfP[ω[[ ;; 5]], a]
           ] /. (2 a → a + ap);
          ineqs = CylindricalDecomposition[%, {a, b, c, ap, e, d, f}]

Out[205]= 0 < a < 12 && a < b < 8 + a && b < c < 5 + b &&
           -a + c < ap < 5 - a + c && e > a + ap - b && d > -a + b && f > a + ap - c
```

Even if on domain of this kind, the numerical integration converges too slowly or the numerical error is too large, we do get the same value as the empirical estimator.

```
In[197]:= NIntegrate[
           PDF[Γ[a], av] * PDF[Γ[b], bv] * PDF[Γ[c], cv] * PDF[Γ[a], apv] *
            (1 - CDF[Γ[e], av + apv - bv]) *
            (1 - CDF[Γ[f], av + apv - cv]) *
            (1 - CDF[Γ[d], -av + bv]),
           {av, 0, TX}, {bv, av, TY + av}, {cv, bv, TZ + bv}, {apv, -av + cv, -av + cv + TU}
          ]

          NIntegrate::slwcon :
            Numerical integration converging too slowly; suspect one of the following: singularity,
                  value of the integration is 0, highly oscillatory
                  integrand, or WorkingPrecision too small. >>

Out[197]= 0.109661
```

However, we can also approximate this value by using a rejection sampling scheme[RC04], which yields here the same probability for a sample of 50000.

```
In[208]:= 1/50 000 Tally[
            Table[ineqs /. Join[(# -> RandomReal[Γ[#]]) & /@ Σ,
                {ap → RandomReal[Γ[a]]}], {50 000}]][[2, 2]]

Out[208]= 663/6250

In[209]:= N[%]

Out[209]= 0.10608
```

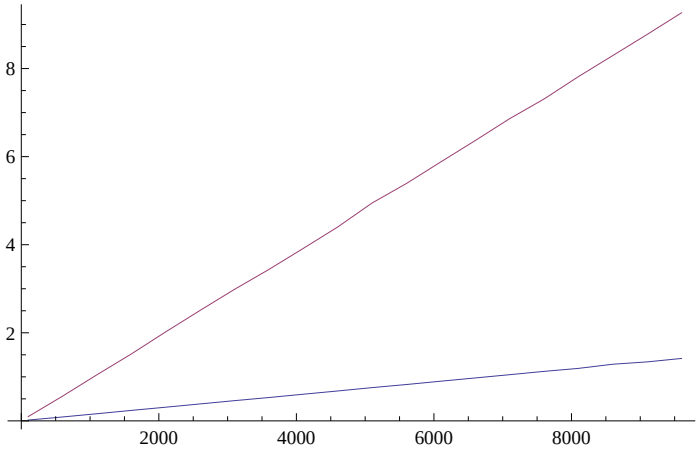
Note that the rejection sampling approach is order of magnitude faster than the simulation.

In[210]:=
```
res = Table[
  {sampleSize,
   Timing[
     Table[ineqs /. Join[(# -> RandomReal[Γ[#]]) & /@ Σ,
          {ap → RandomReal[Γ[a]]}], {sampleSize}];][[1]],
   Timing[
     Table[Nest[Step1, {{{X, X, "init", 0}},
          (#1 → Sample[Γ[#1]]) & /@ EnabledIn[X]}, 4], {sampleSize}];][[
    1]]
  },
  {sampleSize, 100, 10 000, 500}]
```

Out[210]=
```
{{100, 0.017953, 0.096681}, {600, 0.088656, 0.56145},
 {1100, 0.162167, 1.04673}, {1600, 0.23777, 1.51933},
 {2100, 0.310356, 2.02132}, {2600, 0.384909, 2.50856},
 {3100, 0.459148, 2.98657}, {3600, 0.530181, 3.43778},
 {4100, 0.605146, 3.91626}, {4600, 0.678839, 4.40132},
 {5100, 0.755065, 4.94655}, {5600, 0.825433, 5.38994},
 {6100, 0.900914, 5.8806}, {6600, 0.974816, 6.36457},
 {7100, 1.04842, 6.8615}, {7600, 1.1241, 7.30769},
 {8100, 1.19152, 7.8162}, {8600, 1.28615, 8.29468},
 {9100, 1.33933, 8.77442}, {9600, 1.41744, 9.26613}}
```

In[211]:=
```
ListLinePlot[{res[[All, {1, 2}]], res[[All, {1, 3}]]}]
```

Out[211]=



4.4 Concluding remarks and related work

In this chapter, we have defined stochastic transition systems, a general model of transition systems with probabilistic behavior. We gave a semantics of stochastic mode automata (cf Chap. 3) in terms of STS. In order to account for the non determinism implied either by the transition relation or by the use of discrete distributions, we defined *weight schedulers* and described how the probability of sojourn paths are dependent of weight schedulers.

Although the stochastic behavior of our model may be interpreted in terms of general Semi-Markov processes, we chose a different characterization in terms of probabilities of sojourn paths. Contrary to the usual definition of the stochastic process underlying a GSMP by using

transition kernels, our semantics is adapted to compute probabilities of finite paths, and we showed how to represent these probabilities in terms of inequalities over random variables. Once the probability of a sojourn path has been reduced to such inequalities, we showed that it can be computed either by (numerical or symbolic) integration or by rejection sampling.

Note that when we are only interested in the probability of a given path, there is no need to build the full transition system. Indeed, as the semantics of SMA with pre and post-conditions enable us to compute the set of enabled events of the successive state of a sojourn path, we do not need additional information than the sequence of states described by a sojourn path.

In contrast to existing approaches, the most similar probabilistic transition system is the *stochastic automaton* of Zimmermann [Zim08] which, in turn, is very similar to the (similarly named but different) *stochastic automaton* of Cassandras [CL08]. The notable difference with their approach is that we explicitly allow non determinism and account for it in the semantics, while Zimmermann and Cassandras assume that the system is deterministic. Furthermore, they both give the semantics of stochastic automaton in terms of continuous space stochastic process by the use of a transition kernel, and thus completely discard the possibility of computing simple probabilities. Similarly, the formalism of *generalized stochastic petri nets* described in details in [KBD+94] and [Haa02] is also based on the same discrete event scheme and is defined with a semantics in terms of continuous-space stochastic process. In general, the definition of semantics in terms of continuous state stochastic process basically encode in a transition kernel the advance time simulation algorithm of [She93].

Concerning *analysis* of systems that admits a general Semi-Markov process semantics, the authors of [LHK01] devised a model checking[CES86] algorithm to verify properties expressed in CSL [BHHK00, ASSB96] against *semi-Markov chains* (SMC). Semi-Markov chains (well described in [Cin75, Kul95]) are extensions of the continuous time Markov chains (cf Chap. 2) in which the sojourn time in a state is determined by a general continuous distributions. Thus, similarly as in STS and in GSMP, the second Markovian property that states that the elapsed sojourn time is irrelevant does not hold for SMC. However, once a transition is fired (and thus that the sojourn timed has elapsed), there is no past state memory and this corresponds in our setting of STS to a regeneration of all the clocks. Kwiatkowska et al. [KNSS00] refined the approach based on the region graph [ACD93] that is classical in verification of timed automata [AD94]. Although the region graph approach was successfully adapted to timed automata with discrete probabilities [KNSS02], the extension to continuous probability involved a subdivision of the region graph into subintervals of equal size, thus leading to 1) an algorithm infeasible in practice and 2) an approximation of the probabilities.

Another approach presented in [YS05] is based on approximations of general distributions by using phase type distributions[Neu81, Neu75]. Phase type distributions are generalization of the exponential distribution that allows memory dependence in the form of *phases*, and once a general distribution has been approximated by a phase type distribution, states are added in the transition systems to denote the elapsed phase of the distribution. The resulting continuous time process satisfies the Markov property and can thus be analyzed with the same techniques as CTMCs.

Part III

Analysis

Chapter 5

Dynamic analysis

A model description in BioRica is both functional and quantitative. In order to evaluate the impact of the stochastic delays on measures such as probabilistic reachability or response time, the model is analyzed using the computations described in the previous chapter. However, since arbitrary distributions are allowed, analysis can only be performed on restricted cases, e.g. models with a small state space [CL08] or models where all delays follow an exponential distribution (see [Par02a] for an overview).

For the models that are too complex to be analyzed we take a more general approach by using discrete event simulation. In discrete event simulation state changes take place at discrete points in time, although the time is considered to be continuous. The simulation algorithm that we introduce for discrete event is a variant of a variable time-advance procedure [She93]. A variable time advance procedure is a simulation algorithm in which the simulation logical time is advanced at each step to the time of the next scheduled event, thus skipping intervening time.

One advantage of using simulation is that there is no requirement to generate the stochastic transition system. Therefore, simulation do not suffer from the main problem of analytical techniques: state explosion problem. To perform analysis by using simulation, simulation runs (also called sample paths) are generated, and are analyzed to determine the measures of interest by using statistical estimators.

This chapter is organized as follows. We first review a general discrete event simulation scheme then describe how BioRica clauses are translated to C++ code. We then consider three important points of the discrete event simulation of hierarchical systems, namely how to adapt the next time advance algorithm to a tree, then consider the specificities of non determinism and randomness for hierarchical systems. We finally present some languages extensions that can be used to incorporate in a generated simulator constructions not declarable in (pure) BioRica.

5.1 Overview of discrete event simulation

The simulation of a BioRica system is based on a discrete event simulation scheme. A discrete event system functions in the following way: at the time t it is in a state during some time interval Δ_t. After this delay of Δ_t time unit, an atomic event happens and changes the state of the system immediately. In other words, it is a system where transitions are instantaneous, are driven by event, and their occurrence date are real values. It is important to note that

contrary to differential system (for example), the state of the system does not change between two subsequent events.

We consider our system at a time t, and in a state s that gives a valuation to each variable of the system. These values are used to determine the set of fireable events by evaluating the truth value of all of the guards associated with all of the events of the system. Then, the stochastic labeling of the system is used in order to associate to each fireable event a random occurrence date, thus yielding a *schedule* for the system at time t and state s.

Such a schedule is a function associating to each event a real number representing the waiting time before this event will fire. Note that this duration can be undefined or infinite for impossible events (for which the guard constraint doesn't hold) or that this duration can be zero for immediate events, which is always the case for events without stochastic labeling.

The next event to fire will be one of the events that have the minimal waiting time t_m. We consider for the moment that only a single event should fire at time $t + t_m$. If all the waiting times are infinite, we consider the system to be in its final state and deadlocked, therefore the simulation is finished. If not, we fire the event with minimal waiting time t_m and the system state is updated according to the assignments of the event. Afterwards, the waiting time of all the events that have not been fired is decreased by t_m, thus giving them a chance to be the next fireable event.

This discrete stepping evolution is very close to the one described by [Gly89]. However, our models can take into the account untimed events and durations following discrete probabilistic distributions. This added expressivity introduces non deterministic behavior such as when multiple events with discrete probabilistic distributed timing should fire at the same time[3]. In these situations, the specification of the model doesn't provide enough information to be able to infer which of the events should be fired first, and so every possible order of events has to be taken into the account to simulate the possible behaviors.

5.2 Code generation

A compiler from BioRica to C++ has been developed in order to automatically generate a stand-alone discrete event simulator from a BioRica system description. The simulators implementing a discrete event simulation scheme of a BioRica system consists of two parts: a compiler and a simulation framework. The output of simulation is meant to be analyzed separately by feeding simulation results into trace analysis scripts. The software architecture of BioRica is depicted in Fig. 5.1.

The first set of classes concerns the implementation of the simulation algorithms, data structures, and pseudo-random number generators. These classes are generic and are linked statically to the system specific classes during the C++ linking phase. These classes and algorithms are detailed in section sec. 5.3.

The second set of C++ classes is made of all the classes that are specific to the BioRica system under consideration. These C++ classes are automatically generated by a compiler from a BioRica system description. The BioRica to C++ compiler uses a data structure representing the abstract syntax of a BioRica node and BioRica systems that was detailed in chapter 3.

The semantical checks performed before this compilation phase verify the conformity of the

[3]Notice that having multiple event scheduled at the same could not happen in previous discrete event formalism that only considered continuous distributions

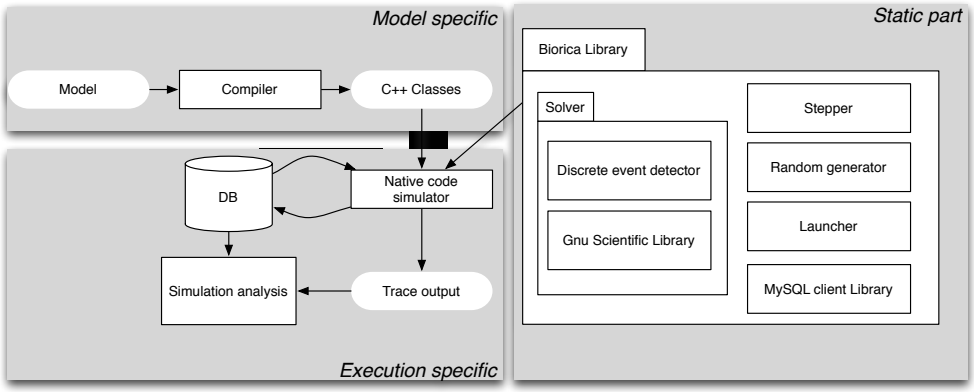


Figure 5.1: BioRica software architecture.

declaration and use of each state variable, event label, and subnode. Importantly, there is no need to build the transition relation, to flatten the domains of the variables, or to apply the composition operators.

5.2.1 Overview of the generated code

Each node of the BioRica system is analyzed by a BioRica compiler. This compiler generates the source code of a C++ class for each node. In these classes, the composition operations are described by object aggregation. One of the objective of this code generation procedure is to code readable by a person, that can be thus further modified if including construction not expressible in BioRica is required. Consider the following simple BioRica node,

```
node simple
  state
    x,y :[0,100];
  event
    𝟙 ⊢ incr_x →x:=x+1;
    𝟙 ⊢ incr_y →y:=y+1;
edon
```

The interface of the associated generated C++ class is the following.

```
class simple:public Node{

 private:
  string* name;
  Node* parent;
  int x;
  int y;
  bool incr_x_is_synched;
  bool incr_y_is_synched;

 public:
  simple(char* a_name,
         Node* parent=NULL);
  virtual ¬simple();


  void incr_x();
```

```
  bool incr_x_eval();
  int incr_x_weight();
  float incr_x_delay();
  std::vector<int>* incr_x_parameters();
  void incr_x_set_synched(bool s);


  void incr_y();
  bool incr_y_eval();
  int incr_y_weight();
  float incr_y_delay();
  std::vector<int>* incr_y_parameters();
  void incr_y_set_synched(bool s);

  void
  storeAllPossibleTrans(vector<TransitionTuple*>
                        *store);


  void display_variables() const;
  friend ostream&
    operator<<(ostream& os, const simple& c);
  void display(ostream&) const;
  void display_hierarchy(ostream& out) const;
  std::string* get_name() const;


  static bool in_x_domain(const int val);
  static bool in_y_domain(const int val);

  void update_flows();
};
```

In order to simulate the previous node, it is dynamically linked to a simulation framework implementing the variable time-advance advance algorithm. The generated simulator can then be executed on any POSIX compatible platform to obtain simulation traces describing the state of the system and the system transitions, such as the following, describing the state of the system at each step, until a deadlock is reached.

```
#>time a.cpt   b.cpt   c.cpt
0       0       0       0
#?(s:3)
#!a.incr
0       1       0       0
#?(s:2)
#!c.incr
0       1       0       1
#!b.incr
0       1       1       1
#Deadlock
```

5.2.2 BioRica clauses to C++ code

Each node declaration in the BioRica system is compiled into a class that inherits from the abstract class *Node*. This class contains the following clauses.

Variables Each variable is compiled into a int instance variable and a domain method. The domain method receives an integer value as input and returns a boolean value indicating if the given value is in the domain of the variable. For flow variables, read or write accessor methods are generated according to the direction of the flow.

Events Each event is compiled into four methods: one representing the weight label, one representing the stochastic delay label, a synchronization setter, and a parameter accessor.

The method representing the weight label returns the integer weight associated with the event. The method representing the stochastic delay is an arithmetic expression which terms are state variables of the instance and function calls to one of the *Distribution* class. This method returns a floating point number denoting the inter-event time. The synchronization setter sets a boolean variable indicating that this event is synchronized in an upper node. This method is called after for each synchronized events after the instantiation of a node. The parameter accessor returns an `std::vector` of integer numbers representing the value of all the state variables appearing in the terms in the stochastic delay. This vector is used to detect whenever a change in one of the values of the state variable should trigger a resampling of the delay associated with this event.

Macro-transitions Each macro-transition is compiled into two methods, one corresponding to the assignments, the other to the conditions under which the transition is enabled.

For the assignment method the assignments in the BioRica node are translated to C++ assignments whose right hand side are arithmetic expressions over variables of the instance. To simulate the semantics of a parallel assignment, values of state variables are copied into temporary local variables at the beginning of the method call.

The method representing conditions that enable an event is built with a conjunction of the pre- and post- conditions. The post-conditions are translated into a boolean formula by substituting the assignment expressions into the domain formula. For each variable modified by the assignment, this expression is used to verify that its next value lies in the domain the variable.

Hierarchies The hierarchy of the system is mapped to a tree data structure. The root of the tree is the root of the BioRica system hierarchy. Each node of the tree corresponds to an instance of a node in the BioRica hierarchy. Each instance uses a `std::vector<Node*>` to store pointers to its sub-nodes instances.

Flow variables Flow connections are implemented by using the flow propagating function3.11. This function is computed at translation time by linearizing the partial order represented by the graph of connections. During simulation, each node is responsible of updating the flow variables of its sub-nodes.

Active event computation Computation of the set of possible events follows a tree visitor pattern. An `std::vector` is transmitted recursively to each node of the tree. Each node is

responsible for evaluating if one of its events admits a transition that is possible in the current state. Each enabled event is inserted into the `std::vector`.

Synchronized events One particularity we must account for concerns event synchronization. The synchronization of two events is specified in a system node. In this system node, multiple instances of the same node may have different synchronization declarations. For example, two events may be synchronized only in one of the instances of the node, as in the following example.

```
1  node simple
     [...]
3  edon

5  node sys
     sub
7      a,b,c:simple;
     sync
9      <a.incr_x,b.incr_y>;
   edon
```

We see that out of the three instances of the node simple, only two are synchronized. In such cases, a single class is generated for the node *simple*, and a boolean instance variable indicating which event of which instance is synchronized is set at the instantiation by the node *sys*.

To determine if a synchronized event is enabled in the current state, the system node checks if all the events in the synchronization vector are enabled. If all the events of the vector admit a transition which pre- and post- conditions are valid in the current state, the synchronized event is inserted in the vector of possible event.

The C++ class associated with a BioRica node is self contained, and the same class can be used either separately or incorporated in a system. This approach makes it possible to generate a single class that is independent of the system in which it is used; and thus a library of pre-compiled models can be built.

5.3 Next event time advance simulation algorithm

A simulation algorithm for a discrete event system computes a sequence of states, events, and events occurrence date. Contrary to the path semantics described in the chapter 4, here we provide an operational approach where the complete information needed to determine the next state is updated at each step of the simulation. Although a BioRica system is equivalent to a BioRica node by flattening, the discrete event simulation scheme we use is based on a tree-traversal algorithm and can thus be used to simulate a BioRica system or a BioRica node indiscriminately.

The variable time-advance simulation scheme informally described in section 5.1 is given by the following algorithm.

This algorithm relies on a visitor pattern, who visits each node in the system tree with the following function

At each step of the next event time advance algorithm 2, the update of the list of enabled events is delegated to the sub-nodes of the system. Consequently the flow of information is top-down and mirror the BioRica system hierarchy.

Algorithm 2 Variable time-advance stepping algorithm. At each iteration in the inner loop, the set of possible events is updated, and new delays are associated to newly enabled events.

Require: $MAXTIME \in \mathbb{R}^+$; $MAXROUND \in \mathbb{N}$;
Require: $Schedule = \emptyset$
Require: *topNode* a BioRica system or a BioRica node
  while $t \leq MAXTIME$ *and* $r \leq MAXROUND$ **do**
    for all $e \in Schedule$ **do**
      Remove if guard doesn't hold
      Remove if post-conditions doesn't hold
      Remove if parameter were modified
    end for
    $schedule \leftarrow possibleTransitions(topNode)$
    if $length(schedule) = 0$ **then**
      end while
    end if
    sort schedule by scheduled date
    $t \leftarrow min_{e \in schedule} \{date(e)\}$
    $r \leftarrow r + 1$
    $Firing \leftarrow \{e \in schedule \mid date(e) = t\}$
    $firing \leftarrow pick(Firing)$
    update state variables accordingly to $firing$
    $schedule \leftarrow schedule - \{firing\}$
    update flows
  end while

5.4 Randomness

Usually in C++, pseudo random number generation (hereafter RNG) is done via the standard library. The generator implemented in this library relies on a global variable called *seed* denoting the initial state of the generator. The successive states of the random number generator are determined by this initial seed.

However, having a global shared RNG is not well suited for the simulation of hierarchical non deterministic systems.

Indeed consider for example a node with stochastic transitions and an input flow i that is not connected. The values of the input flow variable can be chosen randomly after each step of the system. Suppose that there is only one global RNG, the state of which is determined by a single global seed. Varying this seed will modify the behavior of the node, the values assigned to i as well as the random delay associated with each event. Thus it is not possible to vary the behavior of the input flow independently of the behavior of the node and of the random delays associated with each event.

In order to be able to vary independently these behaviors, the random number generation is implemented via a random generator class. Each instance of this class is completely independent and has a private seed. At minimum, three RNGs are instantiated: one for the environment, one for the system and one for the delays.

For systems composed of a large number of nodes, it can be useful to make independent the randomness of particular set of nodes. In such cases, more than three RNG can be

Algorithm 3 Algorithm to compute the set of possible transitions in a hierarchy. Called as "possibleTransitions" in Alg. 2

Require: *node*, a BioRica node
Require: $schedule = \emptyset$
  for all $e \in event(node)$ **do**
    $t' \leftarrow sampleRandomDelay(e, params(e))$
    if $guard(e)$ *and* $post\_condition(e)$ **then**
      $schedule \leftarrow schedule \bigcup \{\langle e, t', params(e)\rangle\}$
    end if
  end for
  for all $n \in subnodes(node)$ **do**
    $schedule \leftarrow schedule \bigcup possibleTransitions(n)$
  end for
  return *schedule*

instantiated. In particular, it is possible to have a RNG associated with each node of the system, and thus the randomness of each node is independent.

This separation of randomness is implemented in the simulation framework via an object oriented random number generation library implementing the Mersenne twister algorithm [MN98].

5.5 Non determinism

Non determinism is useful at the design phase for under specifying a system. For example, the information about the frequency (hence probability) of an event outcome might be unavailable at early steps of modeling. To evaluate measures such as probabilistic reachability or response time of non deterministic systems, we reuse the notion of scheduler [Seg95, Var85, D'A99].

A scheduler is an additional model used to resolve non determinism. In each state where the system is in a non deterministic state, the choice between all the possible next states is delegated to the scheduler. As a consequence, the results of simulation should be considered with respect to a given scheduler.

In the chapter 4, we defined a scheduler as a function returning the next transition based on the history of the system. This functional definition of a scheduler is used for the static analysis of the stochastic transition semantics of a BioRica model. In the context of discrete event simulation, schedulers are used to resolve non determinism on the fly. Although the definition of stochastic transition systems 4 considered a single characterization of non determinism, we consider for simulation two kind of under-specification. The first kind is the non determinism inherent to the environment. This under-specification is due to dangling input flows and represent an under-specified context for the system. The second kind is the non determinism inherent to the system. This under-specification arises in some state when the system description do not contain enough information to uniquely determine the next transition.

For simulation purpose, these schedulers can be implemented separately. The interface of a scheduler is defined in a Scheduler virtual class. This interface is reduced to a virtual method. The signature of this method follows the functional definition given in chapter 4. Three classes inherit from this virtual class. The first class is a generic scheduler used by default. This

scheduler selects the next state by assigning equal probabilities to all the possible successors. The second class specifies the interface of an environment scheduler. This scheduler resolves the non determinism relative to the input flows, and assigns a value to each input flow variable after each step of the system. Finally, the third scheduler is a system scheduler and it decides between multiple events and transitions which one will be the next. This scheduler is a friend class of BioRica nodes and can thus access all state variables of the system.

We illustrate the advantages for two separate schedulers on the following example. Consider the following BioRica node `Cell`.

```
node Cell
    state
        alcohol :[0,100];
        alive :BOOL;
    init
        alive :=𝟙;
    flow
        temperature :[0,100]: i ;
    event
        ferment,die;
    trans
        alive ∧ temperature > 50 ⊢ die →alive:=𝟘;
        alive ∧ alcohol > 10 ⊢ die → alive:=𝟘;

        alive ∧ temperature > 15 ⊢ ferment →alcohol:=alcohol+1;
        alive ⊢ ferment →;
    extern
        Law<ferment>:Exponential{1};
edon
```

The node `Cell` models a fermenting cell. Depending on the temperature of the medium, this cell can either undergo a fermentation process that produces one unit of alcohol or can stay in an idling phase. Moreover, whenever the temperature or the alcohol level are above limit values, the cell undergoes necrosis and is considered as dead. This node has two different source of non determinism. On one hand, the temperature is a dangling input flow, on the other hand, each time the temperature is above `15`, the outcome of the `ferment` event can either be the production of alcohol (line 16) or be void (line 17). Although the `die` event is also non deterministic since the two guards of the two transitions can be satisfied at the same time, the assignment of those transitions are identical and thus there is no need to distinguish the two transitions.

By default, the same scheduler is used for updating the value of the temperature flow variable and for choosing which transition to apply whenever the event `ferment` is the next event. The selection mechanism of this default scheduler is based on a random choice between uniformly distributed outcomes. This selection is performed at two point of the simulation, during the update of the flow variables and whenever the `ferment` event is the winning event. By default, the following C++ code is generated for this non deterministic resolution[4].

```
% \begin{source_cpp}
// [...]
void Cell::ferment(){
    if(this→alive ∧(this→temperature > 15) ∧in_alcohol_domain(this→alcohol+ 1))
        possible++;

    if(this→alive ∧(this→temperature≤ 25))
        possible++;

    //Ask for the system scheduler to choose which transition to follow
```

[4]The generated code has been slightly simplified for readability purpose

```
        choosed=Scheduler::general_instance()→choose(possible);
        if (choosed==1){
                //Real fermentation
                int local_alcohol=alcohol;
                alcohol=(((local_alcohol) + (1)));
                return;
        }else if (choosed == 2){
                //Stay idle
                return;

        }else{
                abort();
        }
}

// [...]
void Cell::update_flows(){
        //Ask for the default environment scheduler to update the temperature
        //By default return a sample from an uniform distribution
        this→temperature=Scheduler::general_instance()→choose(100);
}
```

A typical simulation runs is as follows.

```
#>seed:2773155
#?(e:100→36)
#>time alcohol alive    temperature
0               0       1       36
#!ferment
#Real fermentation
#?(e:100→50)
0.117368        1       1       50
#!ferment
#Real fermentation
#?(e:100→5)
1.15054 2       1       5
#!ferment
#Real fermentation
#?(e:100→8)
1.94106 3       1       8
#!ferment
#Stay idle
#?(e:100→14)
2.64544 3       1       14
#!ferment
#Real fermentation
#?(e:100→75)
3.66174 4       1       75
#!die
#?(e:100→79)
3.66174 4       0       79
#Deadlock
```

The mechanism for solving the non determinism resolution mechanism can be made more specific in order to model a less random evolution of the temperature. For example, the following environment scheduler models a strictly increasing temperature.

```
// Increasing temp scheduler

class IncreasingTemp:public EnvironmentScheduler{
        public:
                int setFlow(int maxVal);
                IncreasingTemp(int seed):lastTemp(0),EnvironmentScheduler(seed) {};
        private:
                int lastTemp;
};
 [...]
```

```
12 extern int env_seed;

14 int IncreasingTemp::setFlow(int maxVal){
       int choosed = rg→IRandom(this→lastTemp,this→lastTemp+5);
16         if (choosed>maxVal){
                   choosed=maxVal;
18         }
           this→lastTemp=choosed;
20         return choosed;
   }
```

When simulated, the same model but with this new environment scheduler produces the following output.

```
1  #>seed:1831963
   #>env_seed:725031083
3  #>sys_seed:775198903
   #>time alcohol alive    temperature
5  0       0       1       2
   #!ferment
7  #?(s:1→1)
   #Real fermentation
9  0.503281        1       1       5
   #!ferment
11 #?(s:1→1)
   #Real fermentation
13 0.693249        2       1       7
   #!ferment
15 #?(s:1→1)
   #Real fermentation
17 2.57803 3       1       7
   #!ferment
19 #?(s:1→1)
   #Real fermentation
21 2.78362 4       1       9
   #!ferment
23 #?(s:1→1)
   #Real fermentation
25 3.33836 5       1       10
   #!ferment
27 #?(s:1→1)
   #Real fermentation
29 4.18309 6       1       10
   #!ferment
31 #?(s:1→1)
   #Real fermentation
33 4.48096 7       1       15
   #!ferment
35 #?(s:1→1)
   #Real fermentation
37 5.37614 8       1       18
   #!ferment
39 #?(s:2→1)
   #Real fermentation
41 5.5048  9       1       18
   #!ferment
43 #?(s:2→2)
   #Fake fermentation
45 5.88346 9       1       20
   #!ferment
47 #?(s:2→1)
   #Real fermentation
49 6.05959 10      1       22
   #!ferment
51 #?(s:2→1)
   #Real fermentation
53 7.19132 11      1       25
   #!die
55 #?(s:1→1)
   7.19132 11      0       25
57 #Deadlock
```

With this model and scheduler, the average lifespan (i.e. time before triggering the "die" event) is 10.89. We can now add a system scheduler to the model. The following scheduler implements a strategy where the production of alcohol is rarified as the amount of alcohol increase.

```
1  class StrategicProduction:public SystemReferee{
       public:
3          virtual int choose(Cell *node, int n);
```

```
                StrategicProduction(int seed):SystemReferee(seed){};
};
 [...]
int StrategicProduction::choose(Cell *node, int n){
        int choosed = Referee::choose((int)node→alcohol+1);
        if (choosed>node→alcohol){
                choosed=1;
        }else{
                choosed=2;
        }
        return choosed;
}
```

With this non uniform system scheduler, the average lifespan is 20.12. The complete source code of this example is available in the appendix in sec. 8.

The scheduler mechanism and non determinism can be used to model arbitrary complex computations outside of the model. However, as the previous example illustrated, schedulers are eventually implemented in C++. This implies that they are inherently finite state (with bounded memory). Hence, they can always be specified as a BioRica node and thus the scheduler mechanism does not add any expressivity to the language. Thus, the distinction between what should be specified in the model and what should be specified in schedulers is based on the purpose of the model. System schedulers can be used to force the behavior of a model to mimic the observed behavior of a biological process when all the biological mechanisms underlying the process are not known. In order to obtain a gross model prototype of a biological process, the scheduler mechanism can be used to build a model with partial biological knowledge: known biological mechanisms are specified in the model, while imposed mechanisms are specified in the schedulers.

5.6 Language extensions

Escaped code The definition of BioRica nodes (see chapter 3) restricts atoms in the terms of formulas to be either integer constants or variables. This restriction permits the unfolding of a BioRica system in terms of a finite transition system.

On the theoretical level, any computation over variables with finite domains (e.g. *int* or *float*) can be represented as a finite state machine. It is thus possible to represent the numerical algorithm to compute a mathematical function (e.g. *sin* or *cos*) in a finite state machine (and hence in a BioRica node). However, such a representation will dramatically increase the number of states of the final transition system.

In order to ease the description of models relying on mathematical functions, the compiler is parametrized and can generate code using either integer or floating point values for the state variables. Furthermore, we implemented a back quote mechanism that makes it possible for BioRica nodes to contain expressions that should not be interpreted as BioRica constructions. These expressions are eventually inserted verbatim in the generated code. This mechanism permits BioRica nodes to contain calls to user provided C++ functions. These external functions can be evaluated in guards, used in assignments, as weight labels, and as stochastic delays.

Hybrid systems and external computations Physical and biological systems may be modeled as evolving continuously over time, thus leading to continuous systems, most of the time modeled by differential equations. Events may happen and change discretely the dynamics of the continuous system. A model containing both continuous dynamics and discrete transitions is called a hybrid system [ACH+95].

A typical example is the bouncing ball. The motion of a bouncing ball is characterized by two variables: height and velocity direction. The ball moves continuously between bounces, whereas discrete changes happen at bounce times and invert the direction of the velocity. The motion of the ball is modeled by equations relating height, velocity direction and acceleration caused by the gravitational force. The bounce of the ball is modeled by a discrete event, the assignment of which uses the elasticity coefficient to compute the new value of the velocity vector. This discrete event is triggered whenever the height of the ball is low enough to be considered as hitting the ground.

By definition, a BioRica model is inherently discrete and thus can not model the dynamic of the ball between bounces. For simulation purposes, we can use the non determinism and flow connection constructions to simulate a hybrid model. The (discrete) BioRica node is reduced to the discrete transitions of the hybrid bouncing ball model.

```
1  #define g 9.81
   #define e 0.9
3  node BouncingBall
       state
5          v :[0,1]
       flow
7          h :[0,100]: i
       event
9          bounce
       transitions
11         h≤ 0.1 ⊢ bounce →v:=v∗−e
       init
13         v=1
       law
15         bounce: Diff(h'=v,v'=−g,v)
   edon
```

Note that in this model, the current height of the ball is completely non specified, and considered as evolving non deterministically. When this model is simulated, the height of the ball is transmitted to the node using an environment scheduler.

In fact, an external simulator (or CAS in this case) is responsible for transmitting the correct value of the height variable by integrating the differential equations $height' = v, v' = g$. This external simulator is called at each step of the system when evaluating the delay before the next bounce. This delay is computed by integrating the differential equations up to the point where the h becomes less than 0.1. In other words, given initial conditions specified by the value of the velocity, we can associate a delay corresponding to the date at which the height becomes less than 0.1 with the bounce event. After this delay expires, the bounce event is fired, and the new value for the velocity parameter is sent to the external solver via the call to the `Diff` law.

Note that the BioRica node `BouncingBall` is perfectly valid and that it can be simulated without integrating the differential equation. In this case, the value associated with the height variable evolves randomly, and the delays between bounces are arbitrary values.

5.7 Discussion and concluding remarks

In this chapter, we have shown how a system specified in the BioRica language can be compiled into a discrete event simulator.

The advantages of using the BioRica language are similar to other domain specific languages [VDV00]. In particular, having a high level language for model description results in faster model prototyping. Indeed, there is no need to implement the simulation algorithm for each new model or after each iteration of the modeling process. This has to be contrasted with using non specialized computer algebra systems (e.g. Mathematica, Matlab) for model simulation.

Usually, increasing the abstraction of a formalism implies a decrease in performance. However, since with BioRica the model is not interpreted but compiled into C++ code, the whole abstraction layer represented by the BioRica constructions is removed. This has to be contrasted with the approach based on interpreted languages used for example by Copasi or E-Cell [TIS+03, THT+99, Men93, HSG+06].

Having a high level language and a generator for executable simulators, the whole life cycle of model development is accelerated. To this end, we clearly separate in BioRica what is specific to the simulation (coming from BioRica semantics) and what is specific to the model. The parts specific to the simulation are implemented in our simulation framework, while the parts specific to the model are generated automatically.

Since we generate human readable C++ code, generated models can be manually modified to account for constructions not present in the core BioRica language. For example, we have shown here that by using a simple back quote mechanism, existing mathematical functions can be used in a discrete BioRica model. Furthermore, we have shown in this chapter that the flexibility of non determinism makes it possible to represent a hybrid system in BioRica.

Chapter 6

Representation of continuous systems by discretization

This chapter explores approximations of differential equations (and more generally of systems whose state space is continuous) where time discretization is replaced by a quantization of the state variables. We will first explore classical quantization techniques used for the simulation of differential equations by means of discrete event systems. We will see that the stochastic labeling of the BioRica node associated with a continuous system can be further used to describe continuous systems with random perturbations. We then see an alternative approach, where the continuous part of a system is simulated separately by using a numerical integrator. Finally, we describe an approach based on abstraction, and show how we can represent an arbitrary trajectory of a continuous as a simple stochastic system, that can later be composed with existing BioRica nodes.

We then introduce *Qualitative Transition Systems* (QTS) and define their probabilistic semantics. A novel abstraction operation is defined in section 3 with the goal of building QTSs from simulation results. We then show in section 4 that when constructing a QTS from an ODE, the QTS construction can be made independent of the numerical integration scheme. In section 5, we show that trajectory comparison using QTS can be made more resistant to noise by detecting points of interest (extremums and inflection) through the construction of a piecewise linear approximation (PLA). In section 6, we validate our approach on models from literature.

6.1 Quantization techniques

We show here how ordinary differential equation systems can be described in BioRica nodes and can have randomized perturbation or random parameter switches added to them. Instead of trying to determine the value that the system state can take at any point in time (as in numerical integration), we rather try to determine at what time some continuous state variable will deviate from its current value by more than a threshold value Δ. Hence, we wish to compute the smallest time step h, such that $x(t_k + h) = x(t_k) \pm \Delta$. This approach allows us to consider that an ODE system is potentially an **h**ybrid system, that is, a system whose internal state flows continuously while having discrete jumps. The discrete jump is modeled by constrained events while the internal state is kept hidden in a timing function, which outputs the time delay before a constrained event will happen.

6.1.1 Discretization of numerical integration

As we stated earlier, one of the main goal of the BioRica Formalism was to allow easy incorporation of continuous dynamics into discrete models. Such a translation relies on a discretization of the continuous system, following the methods to solve and simulate these systems using numerical algorithms.

Numerical integration overview

An ODE system can be seen as the system

$$\begin{aligned} \frac{d[x_1]}{dt} &= f_1(x_1, x_2, \ldots, x_n) \\ \frac{d[x_2]}{dt} &= f_2(x_1, x_2, \ldots, x_n) \\ &\cdots \\ \frac{d[x_n]}{dt} &= f_n(x_1, x_2, \ldots, x_n) \end{aligned}$$

where x_i are state variables while t is called the control variable and usually represents the time. Each f_i is a $\mathbb{R}^n \to \mathbb{R}$ function, whose values are fully determined by the n x_i variables.

In the most simpler scheme, the numerical solving of such a system can be seen as a function $s(\overrightarrow{x}, t, h)$, from $\mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, where $\overrightarrow{x}$ denotes the values of the x_i at time t in a given simulation and h is a real variable denoting the *stepsize* of the algorithm. Then, $s(\overrightarrow{x}, t, h) = \overrightarrow{x'}$ is the real vector denoting the value of the n variables at time $t + h$. With such an approach, a complete simulation of an ODE system from time t_0 to t_n can be seen as the generation of the real vector list

$$\begin{aligned} [(t_0, \overrightarrow{x_0}); & \\ (t_1, \overrightarrow{x_1}) &= (t_0 + h, s(\overrightarrow{x_0}, t_0, h)); \\ &\cdots \\ (t_n, \overrightarrow{x_{n/h}}) &= (t_(n-h) + h, s(\overrightarrow{x_{n/h-1}}, t_(n-h), h))] \end{aligned}$$

Here, the smaller the step size, the more accurate the solution is, while being more computationally costly, thus this simple method allows some control over the computation time.

However, such a simple method is inappropriate when, for a given fixed time step h, the solution is roughly affine on any interval of length greater than h, and less straight on some intervals of length lesser than h. To overcome such limitations, *adaptive methods* are intensively used when solving ODEs. These methods implements some estimation of the error made over an interval, thus providing sufficient information to extend (resp. reduce) the step size when the error is less (resp. greater) than a fixed threshold. An adaptive stepsize function is a function $s_a(\overrightarrow{x}, t, \epsilon) = (\overrightarrow{x'}, h)$ from $R^n \times \mathbb{R} \times \mathbb{R}$ to $\mathbb{R}^n \times \mathbb{R}$, where the new parameter ϵ is the error threshold and where h is the computed time step. When simulating an ODE, the

resulting data points can be seen as the list

$$\begin{aligned}
&[(t_0, \overrightarrow{x_0});\\
&(t_1, \overrightarrow{x_1}) &&= (t_0 + s_a(\overrightarrow{x_0}, t_0, \epsilon)_h, s_a(\overrightarrow{x_0}, t_0, \epsilon)_{\overrightarrow{x}});\\
&\quad ; \ldots ;\\
&(t_n, \overrightarrow{x_k}) &&= (t_k + s_a(\overrightarrow{x_{k-1}}, t_{k-1}, \epsilon)_h\\
&&&\quad , \quad s_a(\overrightarrow{x_{k-1}}, t_{k-1}\epsilon)_{\overrightarrow{x_{k-1}}}
\end{aligned}$$

where k is the optimal number of steps given the accuracy threshold ϵ.

Discrete event representation of a real-vector valued function

From now on, we'll see the discretization problem as a translation problem. In other words, given a list of real-valued tuple

$$[(t_0, \overrightarrow{x_0}, (t_n, \overrightarrow{x_n}]$$

where the t_i are time valuation and the $\overrightarrow{x_i}$ are components of $\mathbb{R}^k$, we need to compute a set of discrete event E such that the discrete simulation of a model with events E will show similar dynamics.

The first step of the translation is to convert the real-valued state variables $\overrightarrow{x_i}$ into integer valued ones by classical fixed point conversion by truncating float variables to integer ones, building the list

$$[(t_0, \overrightarrow{y_0} = \lfloor \overrightarrow{x_0} * H \rfloor), \ldots, (t_n, \overrightarrow{y_n} = \lfloor \overrightarrow{x_n} * H \rfloor)]$$

where H is the desired scale factor and each y_i is an integer vector.

As we can see, between t_i and t_{i+1}, we need to generate $n = \lfloor x_{i+1} * H \rfloor - \lfloor x_{i-1} * H \rfloor$ events, uniformly spaced on a real interval of length $h = t_{i+1} - t_i$. For this example, let's assume that n is positive. Thus, the system will generate the timed trace $(t_i + h/n, inc_x); (t_i + h/n * 2, inc_x); \ldots; (t_i + h/n * n, inc_x)$. This approach can be generalized to case when n is negative by also considering the macro transitions $true| - dec_x - > x := x - 1$.

For this, we interpolate the difference between a state and it's successor to build the list

$$[(td_0, \overrightarrow{z_0}), \ldots, (td_m, \overrightarrow{z_m})]$$

such that

$$\begin{aligned}
\forall i \in \{0, \ldots, m\}\, , \|\overrightarrow{z_{i+1}} - \overrightarrow{z_i}\|_\infty = 1\\
\forall i \in \{0, \ldots, n\}\, , \exists j \in \{0, \ldots, m\}\, , (t_i, \overrightarrow{y_i}) = (t_j, \overrightarrow{z_j})
\end{aligned}$$

This list can be built by adapting the Bresenham line drawing algorithm in a vectorial space. Let n be the maximum variation on a time interval of length h, and j the index of the corresponding variable. By considering a discrete line of length n, we can place each events for the jth variable on each step of the line. For the other variables, who by construction needs less events, we distribute their events uniformly on this line.

Then, we translate this integer-values evolution into discrete timed event, by considering timed incrementation or decrementation macro transitions for each variable, such as

$$true| - inc_x - > x := x + 1;$$

, with a delay function defined by

$$d(v, \overrightarrow{x}, [(td_k, \overrightarrow{z_k}), \ldots, (td_m, \overrightarrow{z_m})]) =$$
$$\begin{cases} td_k & \text{if } (\overrightarrow{z_k} - \overrightarrow{x})_v = 1 \\ \infty \end{cases}$$

who try to schedule non deterministically an increment for each variable at the same time.

Still, by considering the ODE $\frac{d[x]}{dt} = 10 * x$, we can see that the algebraic solution $x(t) = e^{10*t} + K$ (where K is a constant depending on the initial conditions) is a rapidly growing function. Therefore, by considering a scale factor of 10^3, between $t = 1$ and $t = 2$, we have to generate approx. 10^{12} events, which is clearly inefficient. In fact, we can consider a set of accelerated k-iteration of our base incrementation event, as for example $\mathbb{1} \xrightarrow{inc\_x\_10} x := x + 10; \ldots; \mathbb{1} \xrightarrow{inc\_x\_10^k} x := x + 10^k$ thus allowing the incrementation to adapt large scale functions. Semantically, the largest possible value for the model constant k could be computed by considerations on the discrete behavior of the model, so as to preserve reachability properties when accelerating transitions built from a continuous function. This question is actually one research point when considering general accelerations schemes for this class of process.

A node can only try to schedule an event, without any guarantee nor feedback on the effective occurrence of the event. In fact between the scheduling time and the date at which the event should fire, this delay function can be called any number of time or it can have a previously scheduled event cancelled, due to the occurrence of other events in the system.

As such, we need to associate to our delay function a control function to determine if the event did fire or not, if it is still scheduled or cancelled, and accordingly advance in the next discrete value list by popping the front value. Furthermore, this control function must take in account external modifications to the node variables, who can occur in any other event. Our control function is defined by the following algorithm

Require: $t \in \mathbb{R}$; $r \in \mathbb{N}$; $\overrightarrow{x} \in \mathbb{N}^n$; $\overrightarrow{dx} \in \mathbb{N}^n$; $lr \in \mathbb{N}$
Require: $L = [(td_k, \overrightarrow{z_k}), \ldots, (td_m, \overrightarrow{z_m})] \in (\mathbb{R} \times \mathbb{R}^n)^m$
  if $\|\overrightarrow{x} - \overrightarrow{z_{k\,\text{inf}}}\|_\infty$=0 && $t = t_k$ **then**
    {We've attained our goal, we can advance one step}
    pop top val
  else if $lr \leq r$ **then**
    {Some events were fired since last call}
    if $(td_k - t, \overrightarrow{z_k} - \overrightarrow{x})$ is a successor of $\overrightarrow{dx}$ **then**
      {At least one variable was updated by our scheduled events}
      $dx \leftarrow \overrightarrow{tk} - \overrightarrow{x}$
    else
      {External modification happened}
      {Recompute discrete value list}
      $\overrightarrow{nearestState} \leftarrow fixedStepper(td_k, t, \overrightarrow{z_k})$
      $\overrightarrow{perturbedState} \leftarrow \overrightarrow{nearestState} - \overrightarrow{z_k} - \overrightarrow{x}$
      $L \leftarrow adaptativeStepper(t, \overrightarrow{perturbedState})$
    end if
  end if
  $lr \leftarrow r$

$\delta \leftarrow tk - t$
$dx \leftarrow \overrightarrow{tk} - \overrightarrow{x}$
return front(L)
where we use the $\mathbb{R} \times \mathbb{N}^n \times \mathbb{N}^n$ predicate defined by

$$succ(\Delta_1, \overrightarrow{x}, \overrightarrow{x'}) \Leftrightarrow \begin{cases} & \overrightarrow{x} = \overrightarrow{x'} \\ or & \Delta_1 = 0 \\ & and\ \forall i \in \{1, ..., n\}, \\ & \overrightarrow{x}_i = 0\ or\ \overrightarrow{x_i} = \overrightarrow{x'_i} \end{cases}$$

who accounts for the non deterministic evolution of multiple variables at the same date. By considering $\overrightarrow{x'} = \langle 10, 10, 20 \rangle$ and $\Delta_1 = 0$, all the following vector are in the same equivalence class

$$\langle 0, 10, 20 \rangle ; \langle 10, 0, 20 \rangle ; \langle 10, 10, 0 \rangle ;$$
$$\langle 0, 0, 20 \rangle ; \langle 0, 10, 0 \rangle ; \langle 10, 0, 0 \rangle ; \ldots$$

Note that between two computed values of the adaptive step integrator, we schedule a sequence of events interpolating the evolution of the function between these two states. As such, if an external perturbation arose at time t_Δ between these two "ckeckpoints", the algorithm advances the continuous solution up to t_Δ to compute the vector *nearestState*, apply the perturbation to compute the vector *perturbedState* and compute the next checkpoint by taking in account the perturbation.

Furthermore, when simulating multiple ODE evolving in parallel, this algorithm computes exactly the numerical solution that could have been computed sequentially, but requires slightly more computation time when compared with traditional simulation techniques (a constant factor). Moreover, when simulating an hybrid system where discrete transition occurs and interrupt the continuous solver without modifying the continuous variables, this algorithm still computes exactly the same solution as the corresponding purely continuous system.

6.2 Incorporation of a numerical integrator

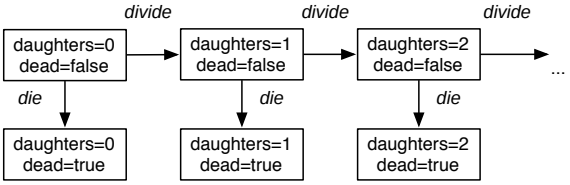


Figure 6.1: A cell division cycle modeled as a transition system built with two variables "daughters" and "dead" respectively denoting the number of daughters and the state of the cell (e.g. dead or alive). It is defined in BioRica by a node containing those variables and the constrained events: $dead = false \xrightarrow{divide} daughters := daughters+1$; $true \xrightarrow{die} dead := true$.

In the cell cycle example described in figure 6.1, the dynamics of the cycle can be described by variation of protein concentrations, described by using differential equations. Constrained

events triggered by crossing protein concentration threshold can describe what precisely happens at division time (mass is divided, certain proteins are transmitted to the bud, etc.). The visible part of this system, defined by choice of discrete variables in the topmost BioRica node, can be the number of division, the current phase of the cell division cycle, etc.

At a higher level, consider now a cell population system, where each cell is represented by a node having its dynamics described by an ODE system. Since ODE systems are inherently deterministic, once initial conditions and parameters are set, every cell will behave exactly like its neighbor, thus leading to an artificially synchronized population[MK01]. In BioRica, since ODE systems are translated in timing functions, we can easily add random perturbations to timing functions by adding to the numerical integration result an exponentially distributed random variable. Thus, each node can be slightly shifted in time while preserving the qualitative properties of its ODE system. Such a cell population can be described from a cell node in BioRica by using **c**omposition.

BioRica systems are hierarchical and modular descriptions: each node can use the most suitable modeling approach and may interact with nodes that use different modeling approaches. This allows for great flexibility in describing complex models. This unified view also allows a simple and efficient simulation scheme.

Multiscale integration. The BioRica framework can simulate systems having nodes whose underlying ODE systems use different time scales. This is done by assigning to each node a private numerical integrator, which can use a local step size and thus can be adapted to the node local configuration and ODE stiffness. The multi scale problem arise mostly when composing ODE nodes whose time scale range over different order of magnitudes.
Furthermore, the BioRica simulator can speed up simulation involving similar ODEs by using a memoization scheme. This is done by reusing previously stored ODE integrations when detecting that two ODE systems have reached the same trajectory or that an ODE system has reached an oscillatory state. This approach is mathematically sound since solutions to ODE systems enjoy a memory-less property. Basically, for a given set of parameters, the solution of an ODE system is completely determined by its initial conditions. This implies, among other properties, that once an ODE node reaches a state that was previously seen (either in the same node or in another node having the same ODE and parameters), then it will behave exactly in the same way. More formally, consider an ODE system of dimension n and its solution, the trajectory function f from $\mathbb{R}$ (time) to $\mathbb{R}^n$ (variables values). Let t and t' be two points in time, with $t < t'$. We can prove that whenever f takes the same value in two separate point of time, then it will take the same value for every corresponding successive point. That is, if there exists two real numbers t and t' such that we have $f(t) = f(t')$, then for any real number ϵ, we have $f(t + \epsilon) = f(t' + \epsilon)$.

6.3 Multi scale parallel oscillator performance study

The benchmark shows the efficiency of the BioRica solver relative to a classical pure ODE solver. BioRica results were compared to the results obtained with a C ODE solver built with the Gnu Scientific Library[Gal06].
The system used is a multi scale uncoupled oscillator built with multiple parallel composition of oscillatory "predator-prey" Lotka-Volterra systems (x'_i, y'_i). The amount of preys is classically given by the ODE $x'_i = p \cdot x_i - (p \cdot x_i \cdot y_i)$ while the amout of predators is described by $y'_i = (p \cdot y_i \cdot x_i) - (2p \cdot y_i)$. Each system (x_i, y_i) represents a "sub population" and has an

unique p parameter, ranging from 3.10^{-05} to 280, leading to a specific period that shorten as p increase. Since these sub population are not coupled, independant simulation of each sub population leads to the same results than the simulation of the whole system. The results shown in figure 6.2 shows that the overhead needed by the BioRica solver is lessened when simulating parallel systems. This advantage is further leveraged when exploiting the memoisation.

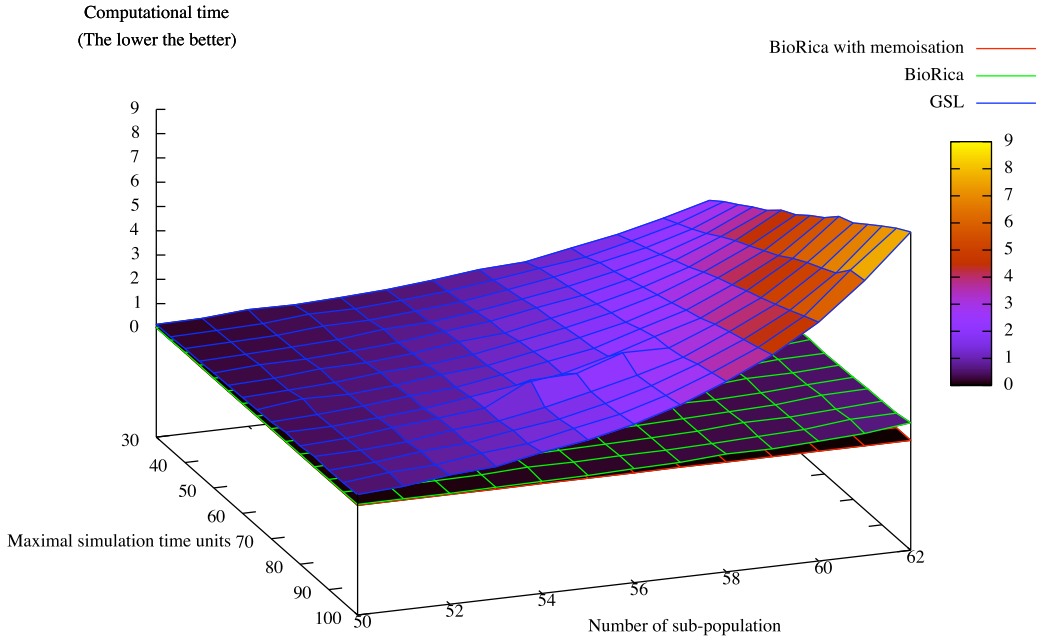


Figure 6.2: Comparison of computationnal time needed to simulate a multi scale uncoupled system between GSL and BioRica. In BioRica, simulation were performed with and without memoisation. For example, simulation from 0 to 100 t.u. of the coupled system with 124 ODEs (62 sub-populations) took 8.6s with the GSL while the equivalent system of 62 BioRica nodes took 0.7s without memoisation and 0.02s with memoisation.

6.4 Stochastic transition system representation of continuous trajectories

Quantitative models in Systems Biology depend on a large number of free parameters, whose values completely determine behavior of models. These parameters are often estimated by fitting the system to observed experimental measurements and data. The response of a model to parameter variation defines qualitative changes of the system's behavior. The influence of a given parameter can be estimated by varying it in a certain range. Some of these ranges produce similar system dynamics, making it possible to define general trends for trajectories of the system (e.g. oscillating behavior) in such parameter ranges. Such trends can be seen as a qualitative description of the system's dynamics within a parameter range. In this work, we define an automata-based formalism to formally describe the qualitative behavior of systems' dynamics. Qualitative behaviors are represented by finite transition systems whose states contain predicate valuation and whose transitions are labeled by probabilistic delays. Biochemical system' dynamics are automatically abstracted in terms of these qualitative transition systems by a random sampling of trajectories. Furthermore, we use graph theoretic tools to compare the resulting qualitative behaviors and to estimate those parameter ranges that yield similar behaviors. We validate this approach on published biochemical models and show that it enables rapid exploration of models' behavior, that is estimation of parameter ranges with a given behavior of interest and identification of some bifurcation points. Dynamic models in System Biology rely on kinetic parameters to represent the range of possible behaviors when enzymatic information is incomplete. Analysis of these parametrized models aims at the identification of parameter ranges yielding similar qualitative behaviors, or of parameter values yielding a given behavior of interest. Qualitative transient behavior can be successfully analyzed by model checking algorithms applied to models admitting a computable path semantics. However, in Systems Biology state explosion and negative decidability results limit the scope of model checking to a certain subset of models. Moreover, some published and curated Systems Biology models lack explicit semantics. Little can be assumed for these "black box" models, except the possibility of simulation. Mining these simulation results to identify parameter regions yielding similar behaviors is hindered by the size of the parameter space to explore, numerical artifacts and the lack of formal definition of what it means for simulation results to be similar. In this section, we propose the new formalism of *qualitative transition systems* for abstracting simulation results in terms of discrete objects that admit efficient similarity measures. Indeed, simulation results for ODEs are obtained using numerical integration schemes operating on floating point numbers. The resulting approximation is problematic for the identification of precise transient properties since transient properties of interest are mathematically by equality between real numbers (e.g. $f'(x) = 0$ is necessary for a local maximum) which is inconsistent in floating point arithmetic[Che94]. Consequently, even analysis of basic properties (like the detection of the first time a deterministic system is in a previously visited state) fail in practice due to this inconsistency. Furthermore, different integration schemes (n-th order, implicit/explicit) yield different and incomparable numerical approximations of the same trajectory. Although using normalized sampling and fixed precision decimal numbers seem to solve this problem, the multiplicity of time scales in ODEs show that this solution is not completely satisfying. For dynamic models admitting a computable path semantics, the impact of numerical artifacts is absent. Indeed, it is possible to compute a finite description of the set of trajectories of the model. Consequently, for these models,

model checking algorithms can decide if a logical representation of a behavior holds, and if not, can provide a counter example. Recently, a probabilistic model checking approach was successfully used to solve the inverse problem: given a logical representation of a transient behavior, return a parameter space in which any trajectory satisfies the specified behavior with sufficiently high probability[RBFS08]. For dynamic models suitable for model checking, the intuitive notion of "similar behavior" is thus fully formalized and generally decidable.

6.5 Qualitative Transition Systems

Given a set Σ, we denote by Σ^* the set of all (finite) *words* $s_0 \cdots s_k$ over Σ. A (finite) *timed word* over Σ is any word $W = (t_0, s_0) \cdots (t_k, s_k) \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ such that $t_i < t_{i+1}$ for $i \in [0, k)$. The nonnegative real numbers t_i are interpreted as the absolute *observation times* and the s_i are the *observed values*. We will focus in the paper on the particular case of $\Sigma = \mathbb{R}^n$, where observed values are vectors of reals. In this case, timed words are called (multivariate) *time series*, and are denoted $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$.

We define a *Qualitative Transition System* (QTS) in the following way.

Definition 6.1 (A). *qualitative transition system* is a tuple $\mathcal{A} = \langle Q, E, \mu, \sigma, w \rangle$ where Q is a finite set of set of qualitative *states*, $E \subseteq Q \times Q$ is a finite set of *transitions*, $\mu, \sigma : E \to \mathbb{R}$ are *mean* and *standard deviation* labelings, and $w : E \to \mathbb{N}$ is a *weight* labeling.

For any transition $e \in E$, $\mu(e)$ and $\sigma(e)$ are respectively interpreted as being the mean and the standard deviation of a normal distribution that is followed by a random variable called *sojourn time*. The weight labeling w induces probabilities for transitions. Formally, the *transition probability* labeling $p : E \to [0, 1]$ induced by w is defined by

$$p(q, q') = \frac{w(q, q')}{\sum_{(q,q'') \in E} w(q, q'')}.$$

A QTS is thus a transition system where each transition is labeled with the amount of time the system is idle before moving to another state. The delay between two state changes follows a parametrized normal distribution. This has to be contrasted with continuous Markov chains, where the sojourn time in a state must be exponentially distributed (see e.g. [KNP02] for a complete definition).

Suppose that a QTS is in the state q, and that there exists an outgoing transition $e = (q, q')$. The probability of moving from state q to state q' is $p(e)$, the transition probability of e. Suppose that the transition e is selected in favor of other outgoing transitions; the system will stay in the state q for a delay that is normally distributed with mean $\mu(e)$ and with standard deviation $\sigma(e)$. Let X be such a normally distributed random variable that denotes the sojourn time in the state q, and let F_X be its cumulative distribution function. The probability to move from q to q' between t_1 and t_2 time units is thus given by $F_X(t_2) - F_X(t_1)$. Contrary to the standard semantics of continuous time Markov chains, our semantics does not involve a race condition. That is, in a given state, the probability for the successor state is not conditioned by the delays but solely by the transition weights, similarly to a discrete time Markov Chain.

6.5.1 Timed semantics

Let $\mathcal{A} = \langle Q, E, \mu, \sigma, w \rangle$ be a QTS. An infinite sequence $q_0 \xrightarrow{\delta_0} q_1 \xrightarrow{\delta_1} q_2 \xrightarrow{\delta_2} \cdots$ with $q_i \in Q$, $(q_i, q_{i+1}) \in E$ and $\delta_i \in \mathbb{R}_{\geq 0}$ is called a *path* in A. Let $Path^A$ denote the set of all paths in A, and $Path^A(q)$ the set of paths in A starting at state q. The superscript A is omitted when understood. For a path π, let $\pi[i] = q_i$ denote the i^{th} state of π and $\delta(\pi, i) = \delta_i$ the time spent in $\pi[i]$.

A probability measure Pr on sets of paths in a QTS can be defined using the standard cylinder set construction [BKH99] as follows. Let $q_0, \ldots, q_k \in S$ with $(q_i, q_{i+1}) \in E$ and $I_0, \ldots, I_{k-1}$ nonempty intervals in $\mathbb{R}_{\geq 0}$. Then $C(q_0, I_0, \ldots, I_{k-1}, q_k)$ denotes the *cylinder set* consisting of all paths π of $Path(q_0)$ such that $\pi[i] = q_i$ and $\delta(\pi, i) \in I_i$ for all $i < k$. Let $F(Path)$ be the smallest σ-algebra on $Path$ which contains all cylinder sets $C(q, I_0, \ldots, I_{k-1}, q_k)$ where $q_0, \ldots, q_k$ range over all state sequences with $q = q_0, (q_i, q_{i+1}) \in E$, and $I_0, \ldots, I_{k-1}$ range over all sequences of non-empty intervals in $\mathbb{R}$. The probability measure Pr on $F(Path)$ is the unique measure defined by induction on k with $Pr(C(q_0)) = 1$ and for $k \geq 0$:

$$\begin{aligned} Pr(C(q_0, I_0, ..., q_k, I', q')) &= Pr(C(q_0, I_0, ..., q_k)) * L_p(q_k, q') * \\ &\quad \frac{1}{2}\left(\mathrm{Erf}\left[\frac{b - L_\mu(q_k, q')}{\sqrt{2}L_\sigma(q_k, q')}\right] - \mathrm{Erf}\left[\frac{a - L_\mu(q_k, q')}{\sqrt{2}L_\sigma(q_k, q')}\right]\right) \end{aligned}$$

where Erf is the Gauss error function, a is $\inf(I')$, b is $\sup(I')$. With this definition, a path corresponds to a sequence of bivariate random variables satisfying the properties of Markov renewal sequences.

Contrary to the standard semantics of continuous time Markov chains (see e.g. [KNP02] for a complete definition), our semantics does not involve a race condition. That is, in a given state, the probability for the successor state is fully conditioned by the transition weights and not by the delays.

6.6 Abstraction of a time series in terms of Qualitative Transition Systems

6.6.1 Abstraction of a time series in terms of timed words

In order to represent a real-valued trajectory as an abstract-valued trajectory, each concrete observation $(t, \vec{x})$ of a time series S is transformed into an abstract observation (t, a) where the observation time t is unchanged and a is an abstract value in a finite domain A called the *abstract domain*. The rationale behind abstraction is that two concrete observations that are transformed into the same abstract observation are assumed indistinguishable w.r.t. qualitative properties.

Formally, an *abstraction function* is any function $\alpha : \mathbb{R}^n \to A$ where A is a finite domain. For any time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ in $(\mathbb{R}_{\geq 0} \times \mathbb{R}^n)^*$, the abstraction of S is the timed word $\alpha(S) = (t_0, \alpha(\vec{x_0})) \cdots (t_k, \alpha(\vec{x_k}))$ in $(\mathbb{R}_{\geq 0} \times A)^*$. Note that abstraction functions may be combined by the cartesian product. In practice, it is often desirable to use multiple-arity abstraction functions that are defined on a fixed-width "window" of observations, i.e, functions $(\mathbb{R}^n)^{d+1} \to A$ where $d \in \mathbb{N}$ is the window width.

For such a function α, the abstraction of S would be defined as the timed word $\alpha(S) = (t_d, \alpha(\vec{x_0}, \ldots, \vec{x_d})) \cdots (t_k, \alpha(\vec{x_{k-d}}, \ldots, \vec{x_k}))$. Observe that $\alpha(S) = \alpha(S')$ where $S' = (t_d, (\vec{x_0}, \ldots, \vec{x_d})) \cdots (t_k, (\vec{x_{k-d}}, \ldots, \vec{x_k}))$.

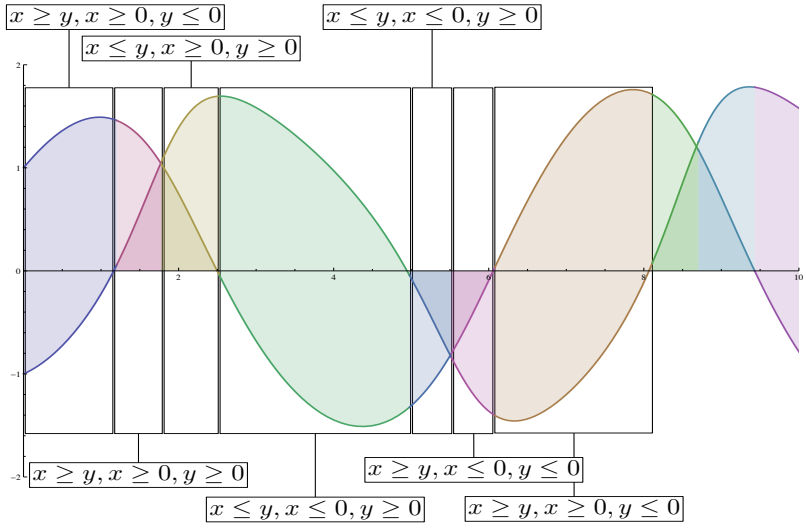


Figure 6.3: Decomposition of a limit cycle of a two variables (x, y) system. By considering the sign and rank of the variables, we map an abstract value (here denoted by a formula) to each point of the trajectory. Boxes in the figure encompass successive points that are mapped to the same abstract value. Successive points of the trajectory are collapsed whenever they have the same abstract value, that is whenever they have the same sign and rank. This trajectory can thus be abstracted as a seven state QTS.

is a time series over $\mathbb{R}^{nd}$. For simplicity, and without loss of generality, we only formalize our approach for unary abstraction functions (with zero width).

For example, the *sign* function can abstract a time series into a timed word over the domain domain $\{-, 0, +\}$. The *rank* function can abstract a timed word over the domain $\{1, ..., n!\}$ by mapping to each component of a vector its index in the corresponding sorted vector. For example, $sort(11, -2, 1, 2) = (-2, 1, 2, 11)$ therefore $rank(11, -2, 1, 2) = (4, 1, 2, 3)$. In the same way, the sign of the first (resp. second) derivative can distinguish between intervals where the time series is increasing (resp. rapidly increasing) or decreasing (resp. rapidly decreasing). Abstracting the value of first derivative (resp. second derivative) requires two points (resp. three points).

Since the abstract domain is finite, it is often the case that the abstract time series $\alpha(S)$ has successive observations that are equal. These repeated observations are removed by collapsing them. Formally, for any timed word $W = (t_0, a_0) \cdots (t_k, a_k)$, let $collapse(W)$ be the timed word $(t_{i_0}, a_{i_0}) \cdots (t_{i_h}, a_{i_h})$ where $i_0 < \cdots < i_h$ are such that $i_0 = 0$ and $a_{i_j} = a_{i_j+1} = \cdots = a_{i_{j+1}-1} \neq a_{i_{j+1}}$ for every $0 \leq j < h$. Observe that collapsing is idempotent: for any timed word W, it holds that $collapse(W) = collapse(collapse(W))$. The *reduced abstraction* of any time series S is then defined as the timed word $collapse(\alpha(S))$.

6.6.2 Abstraction of a timed word in terms of Qualitative Transition System

The abstraction of a time series in terms of timed words abstracts the value component of the time series. In order to adequately compare two timed words, we also need to abstract the time of observations. Consider a timed word $W = (t_0, a_0) \cdots (t_k, a_k)$ over an abstract domain A. To

qualitatively abstract this timed word, it is represented as a transition system by considering that for any $i \in [0, k)$, the pair $((t_i, a_i), (t_{i+1}, a_{i+1}))$ of successive abstract observations of W is induced by a timed transition $a_i \to a_{i+1}$ between two *states* of a transition system with a delay of $t_{i+1} - t_i$. We can then consider the set of all transitions between two given states. From such a set of transitions with identical source and target, we suppose that the delays are approximately normal, and thus estimate the mean and standard deviation of the supposed underlying normal distribution. In this way, the set of concrete transitions can be abstracted by a single stochastic transition in a qualitative transition system. Formally, a timed word is abstracted in terms of QTS with the following definition. For any finite subset $X \subseteq \mathbb{R}$, we denote by $\mathbf{E}[X]$ the *mean* of X and by $\mathbf{V}[X]$ its *variance*.

Definition 6.2 (T). he *QTS abstraction* of a timed word $W = (t_0, a_0) \cdots (t_k, a_k)$ over A is the qualitative transition system $\mathcal{A} = \langle Q, E, \mu, \sigma, w \rangle$ with

$$\begin{array}{rclcrcl} Q & = & \{a_i \mid 0 \leq i \leq k\} & & \mu(q,q') & = & \mathbf{E}[\Delta(q,q')] \\ E & = & \{(a_i, a_{i+1}) \mid 0 \leq i < k \wedge a_i \neq a_{i+1}\} & & \sigma(q,q') & = & \sqrt{\mathbf{V}[\Delta(q,q')]} \\ & & & & w(q,q') & = & |\Gamma(q,q')| \end{array}$$

where for any $(q, q') \in E$, $\Gamma(q, q')$ is the set of pairs (i, j) with $0 \leq i < j \leq k$ such that $a_i = q$, $a_j = q'$, and $a_{i-1} \neq a_i = a_{i+1} = \cdots = a_{j-1}$, and $\Delta(q, q')$ is the multiset defined by $\Delta(q, q') = \{t_j - t_i \mid (i, j) \in \Gamma(q, q')\}$.

Note that in the definition, the set $\Gamma(q, q')$ contains pairs of indices (i, j) such that all observations between i and j are removed by collapsing. Therefore, any two timed words W and W' over A satisfying $\mathit{collapse}(W) = \mathit{collapse}(W')$ have the same QTS abstraction.

6.7 Abstraction of the Transient Behavior of Deterministic Parametrized Models

Deterministic parametrized models, such as ODE systems, can exhibit different qualitative behaviors depending on the value of the parameters. When these systems admit a simulation algorithm (e.g. numerical integration), they generate time series. We show in this section that under assumptions concerning the simulation algorithms, the properties of interest of a given system are preserved by the abstraction in terms of qualitative transition systems.

Sampling independence

In the context of time series obtained by sampling, the definition of QTS obviously depends on the precision of the sampling. However, we show that for convex abstraction functions if the sampling is "precise enough" then the QTS obtained from any oversampling has the same transitions (but with more precise delay distributions). We first introduce additional notations. For any two vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$, we denote by $\overline{\vec{x}\,\vec{y}}$ the open line segment between $\vec{x}$ and $\vec{y}$, formally $\overline{\vec{x}\,\vec{y}} = \{\lambda\,\vec{x} + (1-\lambda)\,\vec{y} \mid \lambda \in \mathbb{R}, 0 < \lambda < 1\}$. A time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ is a *sampling* of a (partial) function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ if $\vec{x_i} = f(t_i)$ for every $i \in [0, k]$. Given a time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$, we denote by $\Lambda_S : [t_0, t_k] \to \mathbb{R}^n$ its linear interpolation, defined by: $\Lambda_S(t_i) = \vec{x_i}$ for each $0 \leq i \leq k$, and $\Lambda_S(t) = \frac{t - t_i}{t_{i+1} - t_i}\,\vec{x_{i+1}} + \frac{t_{i+1} - t}{t_{i+1} - t_i}\,\vec{x_i}$ for each $0 \leq i < k$ and $t_i < t < t_{i+1}$. Given an abstraction function $\alpha : \mathbb{R}^n \to A$, we say that α is *convex* if $\alpha^{-1}(a)$ is a convex subset of $\mathbb{R}^n$ for every $a \in A$.

We consider for the remainder of this section a convex abstraction function $\alpha : \mathbb{R}^n \to A$. When the sampling is precise enough, the linear interpolation Λ_S is often used in practice, in place of the "real trajectory". We formalize this notion of precision with respect to the abstraction function. A time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ is called *α-adequate* if $\alpha(\vec{x}) \in \{\alpha(\vec{x_i}), \alpha(\vec{x_{i+1}})\}$ for every $i \in [0, k)$ and $\vec{x} \in \overline{\vec{x_i}\,\vec{x_{i+1}}}$. It is called *$\alpha$-loose* otherwise. Intuitively, when S is α-adequate, the abstraction function α along $\overline{\vec{x_i}\,\vec{x_{i+1}}}$ is either constant, or is first equal to $\alpha(\vec{x_i})$ on $\overline{\vec{x_i}\,\vec{z}}$ and then equal to $\alpha(\vec{x_{i+1}})$ on $\overline{\vec{z}\,\vec{x_{i+1}}}$, for some $\vec{z} \in \overline{\vec{x_i}\,\vec{x_{i+1}}}$. Indeed, if $\alpha(\overline{\vec{x_i}\,\vec{x_{i+1}}}) \subseteq \{a, b\}$ with $a = \alpha(\vec{x_i})$ and $b = \alpha(\vec{x_{i+1}})$ being distinct, then, by convexity of α, the segment $\overline{\vec{x_i}\,\vec{x_{i+1}}}$ is partitioned into the two convex sub-segments $\overline{\vec{x_i}\,\vec{x_{i+1}}} \cap \alpha^{-1}(a)$ and $\overline{\vec{x_i}\,\vec{x_{i+1}}} \cap \alpha^{-1}(b)$, and $\vec{z}$ is at the boundary between these two sub-segments. Therefore, an α-adequate time series S captures all changes of α along its linear interpolation Λ_S. However, these changes are captured up to the precision $t_{i+1} - t_i$ of the sampling, which leads us to the following definition. The *α-fitting* of an α-adequate time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ is the abstract timed word $\widehat{S} = (\hat{t}_0, \hat{a}_0) \cdots (\hat{t}_k, \hat{a}_k)$ defined by $\hat{a}_i = \alpha(\vec{x_i})$, $\hat{t}_0 = t_0$, and

$$\hat{t}_{i+1} = \begin{cases} t_{i+1} & \text{if } \alpha(\vec{x_i}) = \alpha(\vec{x_{i+1}}) \\ \inf\{t \mid t \geq t_i \wedge \alpha(\Lambda_S(t)) = \alpha(\vec{x_{i+1}})\} & \text{otherwise.} \end{cases}$$

Proposition 6.1. *For any* α-adequate *time series* $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$*, the respective QTS abstractions* $\langle Q, E, \mu, \sigma, w\rangle$ *and* $\langle \widehat{Q}, \widehat{E}, \widehat{\mu}, \widehat{\sigma}, \widehat{w}\rangle$ *of* $\alpha(S)$ *and* $\widehat{S}$ *satisfy* $Q = \widehat{Q}$*,* $E = \widehat{E}$*,* $w = \widehat{w}$*. Moreover, letting* $\Delta = \max\{t_i - \hat{t}_i \mid 0 \leq i \leq k\}$*,* $|\mu(e) - \widehat{\mu}(e)| \leq \Delta$ *and* $\sigma^2(e) \leq \widehat{\sigma}^2(e) + 4\Delta^2 + 4\Delta\widehat{\sigma}(e)\sqrt{\widehat{w}(e)}$ *for every* $e \in E$*.*

Proof. Observe that $\alpha(S) = (t_0, a_0) \cdots (t_k, a_k)$ and $\widehat{S} = (\hat{t}_0, \hat{a}_0) \cdots (\hat{t}_k, \hat{a}_k)$ satisfy $a_i = \hat{a}_i = \alpha(\vec{x_i})$ for each $i \in [0, k]$. By Definition 6.2, it follows that $Q = \widehat{Q}$, $E = \widehat{E}$. Moreover, we also get $\Gamma(e) = \widehat{\Gamma}(e)$ for every $e \in E$, hence, $w = \widehat{w}$. Notice that $(t_i - \hat{t}_i) \geq 0$ for all $i \in [0, k]$. We derive that $|(t_j - t_i) - (\hat{t}_j - \hat{t}_i)| = |(t_j - \hat{t}_j) - (t_i - \hat{t}_i)| \leq \Delta$ for every $0 \leq i < j \leq k$, which entails that $|\mu(e) - \widehat{\mu}(e)| \leq \Delta$ and $\sigma^2(e) \leq \widehat{\sigma}^2(e) + 4\Delta^2 + 4\Delta\widehat{\sigma}(e)\sqrt{\widehat{w}(e)}$ for every $e \in E$. □

Note that in the above proposition, we have $\Delta \leq \max\{t_{i+1} - t_i \mid 0 \leq i < n\}$. Hence, the error on μ (w.r.t. to the α-fitted one) is bounded by the sampling period.

An *oversampling* of S is any sampling $U = (u_0, \vec{y_0}) \cdots (u_l, \vec{y_l})$ of Λ_S such that $t_0 \cdots t_k$ is a subsequence of $u_0 \cdots u_l$. It follows from the definitions that any oversampling of an α-adequate time series S is also α-adequate. Note that, informally, the α-fitting of an oversampling of S is an "abstract oversampling" of the α-fitting of S. According to Proposition 6.1, the QTS obtained by oversampling S is equal to the QTS obtained from S, except for the imprecision on μ and σ. This shows that for qualitative analysis there is little to be gained by oversampling: α-adequate time series are sufficient.

Periodic orbits detection

Oscillations are ubiquitous qualitative behaviors found in systems with a feedback loop. Although bifurcation analysis provides numerical methods to establish the presence of periodic orbits for ODEs, these methods cannot be applied to a general deterministic system such as an ODE with events. However, we show in this section that a QTS can be used efficiently to estimate the likelihood of a periodic orbit in a time series.

Under an adequate abstraction function, a QTS that abstracts the transient behavior of a system with a periodic orbit has cycles in its transition relation. Consider a QTS obtained by

applying the abstraction function α to an α-adequate time series S obtained by sampling a continuous function $f : t \to \mathbb{R}^n$. By definition, f admits an orbit if and only if there exists a time point t and a period π such that $f(t) = f(t + \pi)$. Furthermore, f admits a periodic and non constant orbit if and only if there exists an intermediate time step $t' < t + \pi$ such that $f(t') \neq f(t + \pi)$. Since S is adequately sampled for α, there exists at least three successive different values in $collapse(\alpha(S))$ and consequently the resulting QTS has at least a cycle of length 1.

Since equality between the real numbers and their floating point approximation is not coherent, detection of periodic orbits for a time series must rely on estimations. To find a periodic orbit in a time series S it is sufficient to find a period $\pi \in \mathbb{R}_{\geq 0}$ such that there exist two elements $(t_i, \vec{x_i}), (t_j, \vec{x_j}) \in S$ such that $(t_j, \vec{x_j}) \approx (t_i + \pi, \vec{x_i})$ for an adequate approximation relation $\approx$. However, for ODE systems integrated with an adaptive time step algorithm this scheme produces mainly false positives (successive integration steps in a quasi steady region of an ODE) and false negatives (regions with high variability).

The existence of a periodic orbit of period π also implies that for any value $k \in \mathbb{N}, f(t) = f(t + k * \pi)$. Thus, if the system reaches a periodic orbit at point l, then the nearest points (according to an euclidean distance on $\mathbb{R}^n$) of l contain points from all possible periods.

Therefore, we estimate the likelihood of a periodic orbit by considering a point $l = (t_l, \vec{x_l})$ of S that we suppose being in the periodic orbit, and a set of sample points P from S such that for any point $p' = (t_{p'}, \vec{x_{p'}})$ in $S - P$, for any point $p = (t_p, \vec{x_p})$ in P, we have $\mid x_{p'} - x_l \mid > \mid x_p - x_l \mid$. Less formally, P is a set containing the points that are the nearest to l w.r.t. the euclidean distance. The likelihood $\mathcal{L}((t_l, x_l), \pi, P)$ of π being the period of the orbit of x_l given a sample P of neighbors of x_l is then defined by

$$\mathcal{L}((t_l, x_l), \pi, P) = \left(\sum_{\delta \in \Delta} ([\delta] - \delta)^2 \right)^{-1}$$

with $\Delta = \{(t_p - t_l)/\pi \mid t_p \in P\}$ and $[\delta]$ being the integer part of δ.

Finding the period π that maximizes $\mathcal{L}$ is difficult in practice, since this function admits local maxima that are far from the global maximum. However, the sum of the mean of the longest cycle containing the last observation in a QTS provides a good initial guess of this period. (See the case study 6.9.2).

6.8 Accounting for noise by comparing critical points

Qualitative transition systems can capture the dynamics of a time series, even if the time series contains numerical errors that are only local. In the case of time series admitting global noise, abstraction functions that were adequate for a smooth time series may not be resistant to noise and can generate a QTS that inadequately captures the dynamics of noise. For example, abstracting with the sign of the first derivate can adequately detect oscillations[RBFS08] but fails for time series even with little noise. Although moving average can smooth a time series and seem to circumvent this problem, the size of the window must be fixed *a priori* and this approach is thus neither general nor adaptive.

We propose here an adaptive approach to capture the most important points w.r.t. the shape of a time series. The *critical points* of continuous function $f : \mathbb{R} \to \mathbb{R}$ are the set of points where $f'(x) = 0$. These are points where the function f either has a peak and changes direction (local or global extremum) or presents a curvature change (inflection points). In

both cases the shape of f changes around the point. We generalize this definition to time series in the following way.

Definition 6.3 (**T**)**.** he *critical point* of a time series $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ is the point $(t_c, \vec{x_c}) \in S$ maximizing the function $\Lambda(t_c, \vec{x_c}) = |\vec{x_c} - \vec{x_0}| + (t_c - t_0) * (\vec{x_k} - \vec{x_0})/(t_k - t_0)$.

The critical point of a time series is the point of maximal distance with the linear interpolation between the first and last points of the series. In a numerical context, this point is uniquely defined.

A critical point splits the time series in two time series. Since a critical point is also defined for these series, we can recursively approximate a time series by considering a piecewise function which is linear between critical points.

Definition 6.4 (**T**)**.** he *piecewise linear approximation of order i* (hereafter PLA) of a time series is the piecewise linear function on the intervals $I_0, ..., I_k$ where any interval I_j has a lower bound (resp. upper bound) corresponding to the location of the j^{th} (resp. $j+1$) critical point.

In order to compute the PLA of a time series, we define the piecewise linear interpolations of a set of points as the union of the linear interpolation between two successive points. The computation of the PLA of order i is then performed as follows.

PLA(S, i) returns a list of critical points of $S = (t_0, \vec{x_0}) \cdots (t_k, \vec{x_k})$ for each dimension of S.

1. For each dimension $d \in [0, \dim(\vec{x})]$,
   (a) Initialize the critical points with the first and the last point of S projected on the dimension $d : C_d \leftarrow \{(t_0, \pi_d(x_0)), (t_k, \pi_d(x_k))\}$.
   (b) While $\mid C_d \mid < i$,
      i. Compute the linear interpolation between each successive pair of C_d: Let $\Lambda_j(t) = \frac{t-t_j}{t_{j+1}-t_j} \vec{x_{j+1}} + \frac{t_{j+1}-t}{t_{j+1}-t_j} \vec{x_j}$,where $(t_j, \vec{x_j})$ and $(t_{j+1}, \vec{x_{j+1}})$ are two successive points in C_d
      ii. Build the piecewise linear interpolation: let $\Lambda(t) = \Lambda_j(t)$ for $t \in [t_j, t_{j+1}]$,
      iii. Let $(t_c, \vec{x_c}) = \text{argmax}\{\pi_d(x_c) - \Lambda(t_c) \mid (t_c, \vec{x_c}) \in S\}$,
      iv. Insert $(t_c, \pi_d(\vec{x_c}))$ in C_d s.t. C_d remains sorted w.r.t. the first component
2. Return $\{C_d \mid d \in [0, \dim(\vec{x})]\}$

The previous algorithm cannot append a point twice to the list of critical points. Indeed, once a point is appended to the list it becomes a bound of the piecewise linear interpolation that is used for determining the next critical point. Consequently, at this point the distance between the next piecewise linear interpolation and the time series is 0, and the distance can not be maximized. Note that this does not hold for the piecewise linear regression. Which implies that, for any unidimensional time series, the segmented linear regression of i intervals minimizing the residuals with the time series can be obtained by considering the critical points as bounds of the interval.

Examples of critical points

Critical points are highly related to the shape of the time series. Consider for example the sigmoid shape: a simple shape descriptor may simply specify that we start from a low plateau,

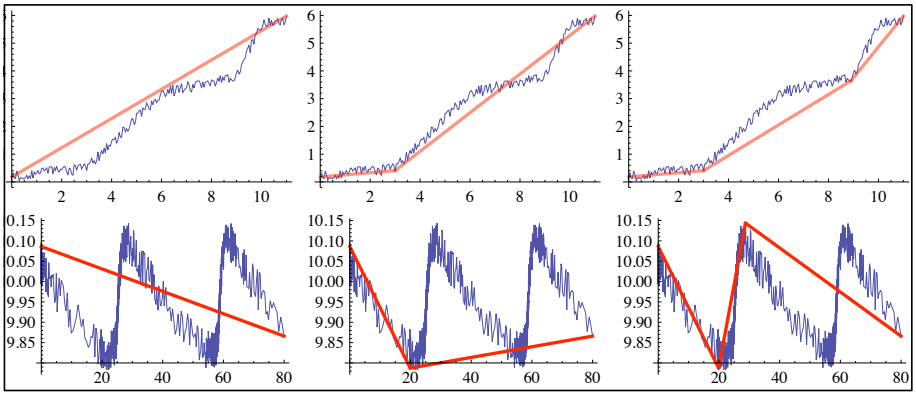


Figure 6.4: Example of piecewise linear approximation applied to two randomly generated noisy time series. The first of the three successive plots represents the time series while the last two plots represent the result of the first two iterations of the PLA algorithm. After two iterations, the PLA algorithm selects two points (as well as the first and last point) that are considered as being representative of the global shape of the time series.

follow an almost vertical increase before reaching another high plateau. Such a sigmoid shape exhibits two critical points, one at the end of the first (low) plateau, and one at the start of the second (high) plateau. Similarly, consider an oscillatory time series such as the one depicted in figure 6.4: its critical points contain the successive highest local maximum and the lowest local minimum.

6.9 Case Studies and experimental results

In this section we show how our approach can be used in practice by solving four problems related to qualitative behavior analysis. Although each problem and solution is illustrated on a specific model, the methods that are used are general-purpose. All the models used in this section were downloaded from the BioModels database[LNBB+06] in the SBML V2 L1 format[FHS+01], simulated using MathSBML[SHFD04] and were used without any modification. Simulations were performed on an Intel Core2 3,2GH personal computer and each algorithm was allowed to run for at most five minute. If parameter values are not specified in the case studies, it means that those provided in the SBML file were used.

6.9.1 Searching a trajectory with a given periodic orbit

The first model we consider is a model of the cell cycle based on the interactions between the cyclin dependent kinase cdc2 and cyclin [Tys91a]. The model is comprised of six variables and ten parameters. We consider the following problem. Given the representative trajectory and its associated parameters described in the original article (left in figure 6.5), what kind of similar trajectories can be found in the whole parameter space ?

Abstracting the behavior of the left figure with a rank abstraction function yields the QTS depicted in the right part of figure 6.5. Notice the highlighted non deterministic states. These states and transitions are due to numerical errors and happen while the system reaches its periodic orbit. Consequently, the weights of the outgoing highlighted transitions are 1 while

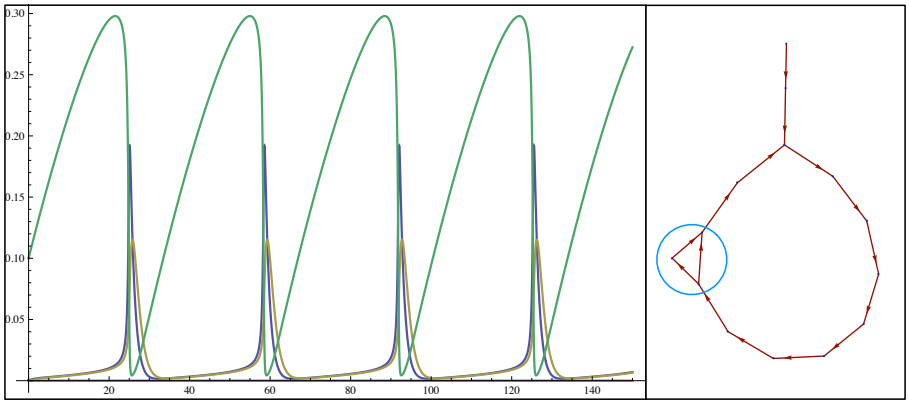


Figure 6.5: Dynamic behavior of the cell cycle model for default parameter values. **Left**: an example of trajectory obtained by numerical integration of the Tyson cell cycle model [Tys91a]. **Right**: abstraction of this trajectory in terms of QTS by using the rank function as the abstraction function (transition labels are omitted). The total standard deviation of this QTS is 0.07. The states and transitions highlighted by the circle correspond to stochastic transitions and represent numerical integration errors.

the incoming transitions are 17. All other transitions in the single cycle of the QTS have a weight of 18.

We obtained 500 random samples for the six parameters considered as being critical by the original author. For each parameter sample, we computed the trajectory, abstracted it in terms of QTS by applying the rank function and computed the Sorensen similarity index over the set of transitions to compare the sampled QTS with the representative QTS. Figure 6.6 depicts a subset of the results. Note that we chose parameter values exhibiting "similar" sustained oscillations, but of *different transient behaviors*. We then compared these results with the one obtained with a stochastic simulation algorithm. For each simulation result, we used the PLA algorithm to reduce each noisy trajectory to its 50 most critical points, and abstracted these points in terms of QTS by applying the rank function. Trajectories similar to the one simulated with numerical integration were found for comparable parameters values.

6.9.2 Estimating the period of orbits

The model of MAPK cascade from [Kho00] describes the effect of negative feedback and ultra-sensitivity on the emergence of oscillations. We investigated the dynamics of the period of the orbits under parameter changes. To estimate these periods, we considered the parameters $\{k_4, v_5\}$ as random variables following an uniform distribution over the intervals $[0, 1]$ and $[0, 0.1]$. For parameters' sample of size 500, we abstracted the corresponding time series in terms of QTS. These QTS were then reduced by removing transitions whose probability decreased as the simulation advanced. We then approximated the orbit's period with the sum of means of the transitions of the longest cycle of the QTS. This approximation was then used in a local maximization procedure to identify the exact period value maximizing the likelihood function. In our tests, providing this initial "educated guess" of the period value to the maximization procedure yielded the global maximum in 98% of cases.

We can see from the results (figure 6.7) that, for this parameter subspace, oscillating be-

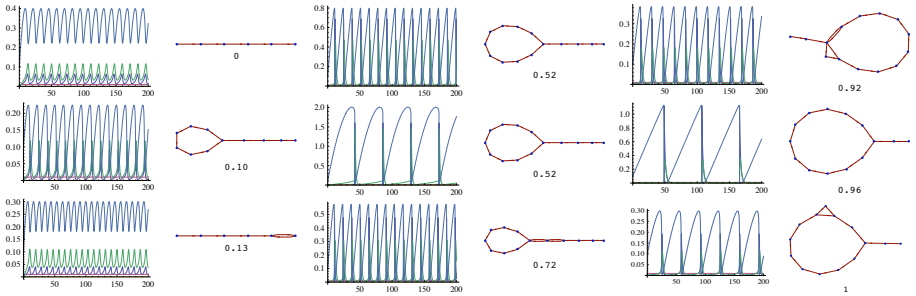


Figure 6.6: Trajectory comparison for the Tyson cell cycle model. For 500 randomly sampled parameter spaces, we abstracted the simulated trajectory in terms of QTS by using the rank function. Trajectories were clustered in 9 bins by measuring the Sorensen similarity index between each of the 500 QTS with the QTS of the figure 6.5. From each of the nine clusters, a representative trajectory is depicted here together with its QTS and similarity value. These trajectories are sorted (column wise, increasing) by their similarity value.

havior is very common and that the dynamics of the period does not exhibit abrupt changes.

6.9.3 Searching for any periodic orbits

We consider again the MAPK cascade model [Kho00] but with a more general objective. We consider the problem of detecting the possible oscillating behaviors and of computing the probability of finding an oscillating behavior in a larger parameter subspace. The parameters of interest are $\{k_3, k_4, k_7, k_8, V_5, V_6, V_9, V_{10}\}$ and are considered as random variables following an uniform probability over the interval $[0, 1] \subset \mathbb{R}$. We built a QTS as in previous sections. In this study, only periods in the range $[200, 5000]$ with a likelihood greater than 10 were considered genuine. Although all the resulting trajectories exhibit transient or limit cycle oscillations, they follow different transient dynamics. The four example trajectories of figure 6.8 show a subset of the possible dynamics: each of these time series admits a specific alternation of species at their maximum concentration. Multiple instances of each of these dynamics were successfully identified by applying the method from section 6.9.1. The number of samples needed before finding an oscillating behavior was 57 on average. For comparison, when k_3, k_4, k_7 and k_8 were sampled in the interval $[0, 0.1]$, the average number of samples needed dropped to 10.6.

6.9.4 Searching for given transient behavior in a parameter subspace

The extracellular signal regulated kinase (ERK) pathways plays a role in a hidden oncogenic positive feedback loop via a crosstalk with the Wnt pathway [KRKC07]. The pathological cases identified by the authors involve “an irreversible response leading to a sustained activation of both pathways”. Applying our QTS construction with random samples of the β-catenin synthetic rate (V_{12}) yields results depicted in figure 6.9.

This model involves 28 species, 58 parameters, and 2 discrete events. Applying the rank abstraction yields transition systems with a state space of 600 states on average out of the possible 28! state configurations.

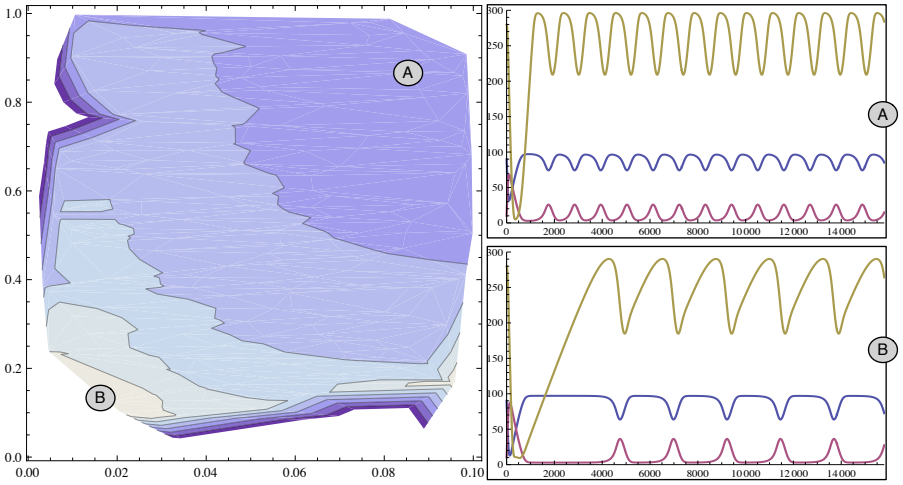


Figure 6.7: Oscillation period for the MAPK cascade model. **Left**: contour plot representing the period of oscillations. The bottom axis (resp. left axis) represents values of the $k4$ (resp. $v5$) parameter. The contour plot was built with 500 simulations with random parameters. Regions with comparable periods are represented by a region with uniform color. **Right**: Two example trajectories exhibiting oscillations of minimal and maximal period A:1094 time units and B:2236 time units.

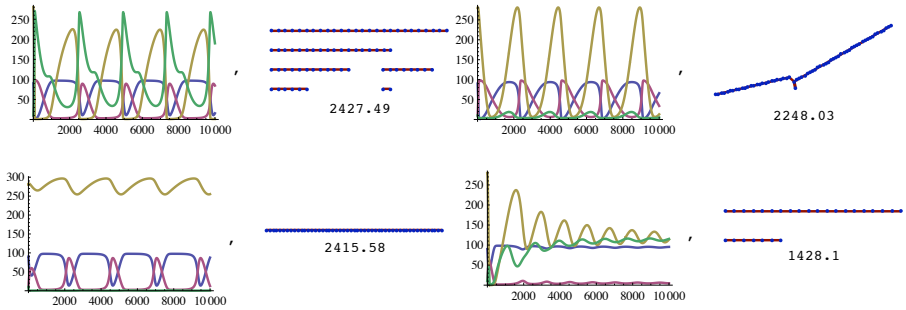


Figure 6.8: Example trajectories of the MAPK cascade exhibiting oscillating behavior found with a random sampling of ten parameters of the MAPK cascade model [Kho00]. On the right of each plot, the associated QTS reduced to its periodic form; under it, the period value with maximum likelihood.

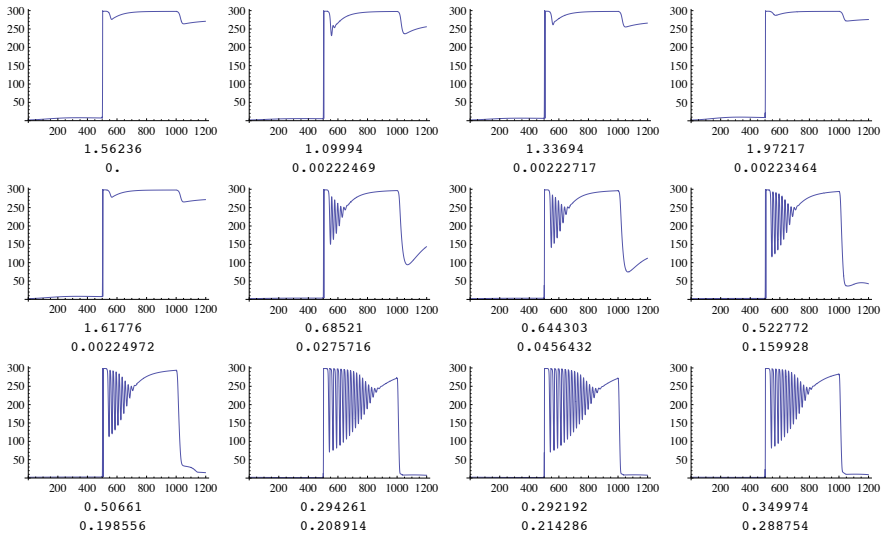


Figure 6.9: ERK Crosstalk simulation results. From top left to bottom right, simulation results are sorted by similarity with the non pathological case (reversible activation). The sampled value of v_{12} and the similarity index with the non pathological case are under each plot.

6.10 Concluding remarks

In this chapter, we have described two techniques that allows a continuous model described by ODE to be interpreted as a discrete event system. The first approach is similar to the quantization-based integration used in coupled DEVS models [CK05]. The second approach is based on the formalism of qualitative transition systems. A QTS is a transition system where each transition is labeled with the amount of time the system requires before moving to another state. The delay between two state changes follows a parametrized normal distribution. We have shown how QTS can be used to study qualitative properties of parametrized models. This is achieved by defining an appropriate abstraction function. By representing the characteristic qualitative features of a trajectory in an abstract domain that is countable, qualitative similarity can be detected by a simple equality test. The limits of our approach as compared to model checking is the lack of exhaustivity. This has to be counterbalanced by the fact that our method is applicable to a large panel of formalisms, even those lacking a precise semantics. Consequently, we can avoid any model transformation. Finally, our approach can be applied independently to the data and to the model.

Chapter 7

Case study: Inheritance of damaged proteins

In this chapter, we model a yeast population as a hierarchical model with dynamic instantiation. Yeasts mitosis imply a rejuvenation process by which the aged mother cell generates a daughter cells enjoying full replicative potential. Here we show how a hierarchical simulation tool that allow dynamical creation of cells can be used to expand a single cell model of protein damage to a cell population model. By explicitly tracking mother-daughter relations, this population model establish required conditions for exhibiting a rejuvenation effect consistent with experimental results.

Thorough knowledge of the model organism *S. cerevisiae* has fueled efforts in developing theories of cell ageing since the 1950s. Models of these theories aim to provide insight into the general biological processes of ageing, as well as to have predictive power for guiding experimental studies such as cell rejuvenation. Current efforts in *in silico* modeling are frustrated by the lack of efficient simulation tools that admit precise mathematical models at both cell and population levels simultaneously.

We developed a hierarchical simulation tool based on BioRica that allows dynamic creation of entities. We used it to expand a single-cell model of protein damage segregation to a cell population model that explicitly tracks mother-daughter relations. Large-scale exploration of the resulting tree of simulations established that daughters of older mothers show a rejuvenation effect, consistent with experimental results. The combination of a single-cell model and a simulation platform permitting parallel composition and dynamic node creation has proved to be an efficient tool for in silico exploration of cell behavior.

7.1 Introduction

A recurring challenge for *in silico* modeling of cell behavior is that hand-tuned, accurate models tend to be so focused in scope that it is difficult to repurpose them. *Hierarchical modeling* [AIK+03] is one way of combining specific models into networks. Effective use of hierarchical models requires both formal definitions of the semantics of such compositions, and efficient simulation tools for exploring the large space of complex behaviors. In this study, we propose the use of a hierarchical model to reduce the complexity of analyzing cell ageing phenomena such as cell rejuvenation. To this end, we extend a single-cell model of inheritance of protein damage to a structured population where mother-daughter relations are tracked.

Unlike most microorganisms or cell types, the yeast *Saccharomyces cerevisiae* undergoes asymmetrical cytokinesis, resulting in a large mother cell and a smaller daughter cell. The mother cells are characterized by a limited replicative potential accompanied by a progressive decline in functional capacities, including an increased generation time [SMG98]. Accumulation of oxidized proteins, a hallmark of ageing, has been shown to occur also during mother cell-specific ageing, starting during the first G1 phase of newborn cells [AGRN03]. Both asymmetric and symmetric division exist in different yeast species. In particular, *S. cerevisiae* is known to divide asymmetrically, although symmetrical division is observed in about 30% of cells at the end of their replicative lifespan [KJG94]. Another yeast model organism, *Schizosaccharomyces pombe*, divides symmetrically by fission (see [Nur94] for review). The following is a mathematical model we have developed to explain how the accumulation of damaged proteins influences fitness and ageing in yeast. In this paper we consider the two theoretically possible scenarios, namely asymmetrically and symmetrically dividing cells in different damaging environments. To explore any and all branches of the pedigree tree of a cell population, we will use a hierarchical model that allows us to track mother-daughter relations. We can therefore explore lineage-specific properties, such as the rejuvenation property.

7.2 From single cell to population model

Single cell model A minimal single-cell model of inheritance of damaged proteins can be formalized by the following three equations:

$$\frac{dP_{\text{int}}}{dt} = \frac{k_1}{k_s + P_{\text{int}} + P_{\text{dam}}} - k_2 P_{\text{int}} - k_3 P_{\text{int}} \tag{7.1}$$

$$\frac{dP_{\text{dam}}}{dt} = k_3 P_{\text{int}} - k_4 P_{\text{dam}} \tag{7.2}$$

$$\frac{dP}{dt} = \frac{k_1}{k_s + P_{\text{int}} + P_{\text{dam}}} - k_2 P_{\text{int}} - k_4 P_{\text{dam}} \tag{7.3}$$

The size of the cell is the sum (P) of intact (P_{int}) and damaged (P_{dam}) proteins, $P = P_{\text{int}} + P_{\text{dam}}$. We assume that cells grow until they have attained a critical cell size, P_{div}, which triggers a cell division. A cell may divide symmetrically (halving its mass) or asymmetrically, as defined by size coefficients s_{mother} and $s_{daughter}$. The proportion of damaged proteins that are transmitted to the daughter cell after division is defined by retention coefficient re. Protein temporal dynamics are determined by five rate constants k_1, k_2, k_3, k_4, and k_s. Protein production rate, k_1, has been adjusted by hand allowing for a steady state to be reached and has been assigned a final value of 10^7. We choose values of k_2 and k_4, the degradation rates of P_{int} and P_{dam}, respectively, so that $k_2 < k_4$. Degradation rates are computed using the half-life formula $t_{1/2} = \ln 2/k$, where k is the degradation rate; setting the half-life of intact proteins to be 1 time unit, $k_2 = \ln 2$. Since degradation of damaged proteins is faster, k_4 needs to be greater than k_2 and it has been set to $\ln 5$. To simulate different rates of conversion, k_3 has been given a range of values, from 0.1 to 2.3. The retention coefficient, re, has also range of values, simulating different proportions of damage proteins that are retained by progenitor. The maximum value for retention is $re = 1$, meaning that the mother keeps all damaged proteins, while for $re = 0$ the distribution of damage is simply proportional to the size of each cell. We simulated the equation systems for different types of divisions (symmetrical vs. asymmetrical) as described in Methods. In the case of symmetrical division, the size of

Table 7.1: Parameters of the single-cell model, their default values, and assumptions made in their estimation.

Parameter	Description	Values	Notes
P_{div}	division threshold	1500	
k_1	rate maximal protein production	10^7	
k_2	rate degradation of intact proteins	$\ln 2$	
k_3	rate of damaging of intact proteins	$[0.1, 2.3]$	
k_4	rate of degradation of damaged proteins	$\ln 5$	$k_4 > k_2$
k_s	half-saturation constant	1	
re	coefficient of retention damaged proteins	$[0, 1]$	
s_{mother}	size of progenitor after division	0.5 or 0.75	$s_{mother} + s_{daughter} = 1$
$s_{daughter}$	size of progeny after division	0.5 or 0.25	$s_{mother} + s_{daughter} = 1$

both progeny and progenitor is equal, so $s_{mother} = s_{daughter} = 0.5$. In asymmetrical division, cells in the next generation will have different sizes. We consider here the exemplary case that $s_{mother} = 0.75$ and $s_{daughter} = 0.25$. Finally, k_s is a half-saturation constant in the model, not used in this study. In the following study, s_{mother} and $s_{daughter}$ were given two pairs of values representing symmetric and asymmetric growth strategies, namely $\langle s_{mother}, s_{daughter} \rangle$ being $\langle 0.5, 0.5 \rangle$ or $\langle 0.75, 0.25 \rangle$.

The proteins distribution between the progenitor and the progeny after division is described by the following set of transition assignments: For progenitors:

$$P_{\text{int}} := P_{\text{int}} \cdot s_{mother} - P_{\text{dam}} \cdot re \cdot (1 - s_{mother}) \tag{7.4}$$

$$P_{\text{dam}} := P_{\text{dam}} \cdot (s_{mother} + re \cdot (1 - s_{mother})) \tag{7.5}$$

$$P := P_{\text{int}} \cdot s_{mother} + P_{\text{dam}} \cdot s_{mother} \tag{7.6}$$

For progeny:

$$P_{\text{int}} := P_{\text{int}} \cdot s_{daughter} + P_{\text{dam}} \cdot s_{daughter} \cdot re \tag{7.7}$$

$$P_{\text{dam}} := P_{\text{dam}} \cdot s_{daughter} \cdot (1 - re) \tag{7.8}$$

$$P := P_{\text{int}} \cdot s_{daughter} + P_{\text{dam}} \cdot s_{daughter} \tag{7.9}$$

where s_P is s_{mother} for the progenitor and $s_{daughter}$ for the progeny.

All the parameters of the model and their values in this study are synthesized in Table 7.1.

In previous work, simulations carried out with Mathematica allowed us to follow the fate of the progenitor and the progeny, separately, through a number of generations. We could thus draw a "mother-only lineage" and a "daughter-only lineage", whereby we would, after every division, follow respectively the next generation of mothers only, or the next generations of daughters only. These are external branches of the pedigree tree.

In this study, we use a hierarchical model that allows us to explore any and all branches of the pedigree tree, and precisely track mother-daughter relations. We can therefore explore lineage-specific properties, such as the rejuvenation property studied in this work.

Population Model Based on this single-cell model, we first define a hierarchical model of a structured population where complete mother-daughter relations are recorded, using the BioRica formalism.

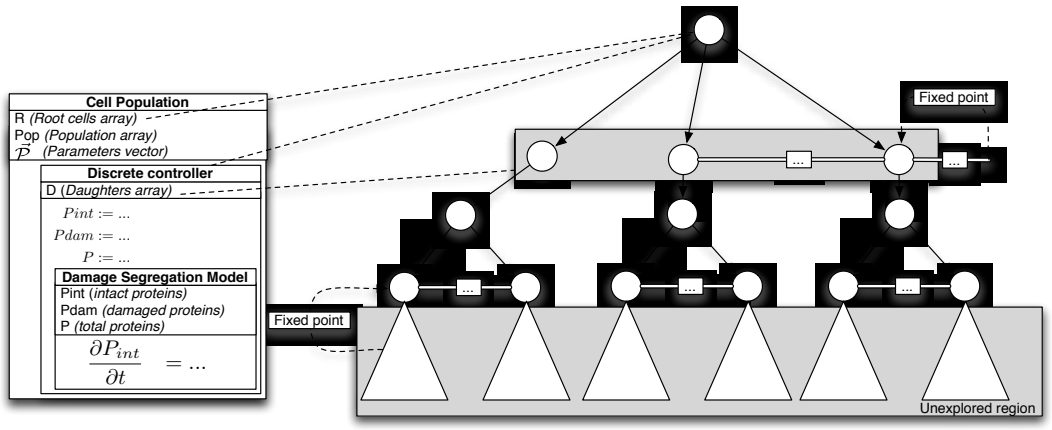


Figure 7.1: Three level hierarchical model, showing the discrete cell population and cell division controllers, and the continuous single-cell model. This model generates pedigree trees during simulation, instantiating new single-cell models for each cell division. Infinite width and depth are represented finitely by relaxing the tree constraints to permits loops from the leaves. These *fixed points* represents immortal cells or immortals lineages.

In this work each cell is encoded by a BioRica node that has a 2-level hierarchy: a discrete controller and a continuous system. The former determines the distribution of proteins at division time using the discrete transition assignments (4–6), while the latter determines the evolution of protein quantities during one cell cycle and is realized by the equations (1–3). More precisely, the discrete controller is encoded by a *BioRica node* (cf chap. 3) defining the discrete transitions between states. A *state* of a cell c^i is a tuple $\langle P^i_{\text{int}}, P^i_{\text{dam}}, D^i \rangle$, where P^i_{int} and P^i_{dam} are protein quantities as before, and D^i is a single dimension array of integers representing the identifiers of every daughter of c^i. A *transition* between states is a tuple $\langle G, e, A \rangle$, where G is a guard, e is an event, and A is a parallel assignment. For this model, the discrete transitions are atomic operations and consequently take zero time. In our case we have: for mitosis (event e), if the threshold of the cell size is attained $P_{\text{int}} = 1500$ (guard G), then create a new BioRica node c^j for the daughter of the current cell c^i, append c^j to the vector D^i, and perform the assignments (4–6) (assignments A of state variables). A second discrete event representing clonal senescence is triggered whenever protein production reaches zero, that is $\partial P_{\text{int}} < 0$.

The cell population is encoded by a BioRica node using the mechanism of parallel composition. This node contains the population array *Pop*, the root of the lineage tree R and the parameter vector $\vec{\mathcal{P}}$. Since our model focuses on the division strategy, we consider the growth medium as a non limiting factor, and consequently we do not account for cell to cell interactions. This absence of interaction is directly modeled by parallel composition of independently evolving cell nodes. For illustration see figure 7.1. The algorithmic challenges related to dealing with multiple time scales and event detection, and our solutions, are described in section 7.3.

7.3 Algorithm

We now describe our method for simulation of the cell population model (section 7.2), starting with an overview of the general simulation schema (algorithm 4) followed by a concrete specialization for damage segregation. The simulation schema for a given cell is by a hybrid algorithm that deals with continuous time and allows for discrete events that **roll back** (see figure 7.2) the time according to these discrete interruptions. Time advances optimally either by the maximal stepsize defined by an adaptive integration algorithm [PTVF07], or by discrete jumps defined by the minimal delay necessary for firing a discrete event. As shown in algorithm 4, the simulation advances in a loop that is interrupted when either the simulation time expires, or the *alive* flag indicates that this node has died in the current or previous state. The node evolves continuously by calling *advance_numerical_integration*, after which we check whether any guard G of some event $\langle G, e, A\rangle$ was satisfied. In which case a number of updates is performed: the time is set to the firing time of e, e is stored in the trace database, the current state S is set according to the algebraic equation A, and the numerical integrator is reset to take into the account the discontinuity.

Algorithm 4 General simulation schema

Require: current state S, current simulation time t, maximal simulation time $t_{\max}$

```
1:  S' = S
2:  while alive(S, S') = 1 and t < t_max do
3:     S' = S
4:     t, S = advance_numerical_integration()
5:     if e = discrete_events() then
6:        t = get_discrete_event_time()
7:        store_event(e)
8:        S = update(S, e)
9:        reset_numerical_integrator()
10:    end if
11:    store_state(S)
12: end while
```

As illustrated in figure 7.2, we assume that the step size proposed by the numerical integrator guarantees that the continuous function is linear between the current time t and the maximal step size. In this way the location of discrete events whose guards have been satisfied in this interval is reduced to computing the **first intersection** (see figure 7.2). It is the event e with the smallest firing time that is retained for the next discrete transition. After this transition the numerical integrator must restart from the point defined by A.

Correction of the stepping algorithm For asynchronous simulation of multi-agent hybrid systems, the correctness of a stepper algorithm concern mainly numerical stability and event detection [EK04], both being in general very difficult problems [SGB91]. For this specific model, numerical stability of the stepper described in algorithm 4 has been checked by evaluating the stiffness of the single cell ODE system by comparing the accumulated integration error between various order explicit and implicit methods. For the final implementation, the embedded Runge-Kutta-Fehlberg was retained, giving good tradeoff between efficiency and precision.

Failure of an event detection can be caused either when sub systems are coupled but not

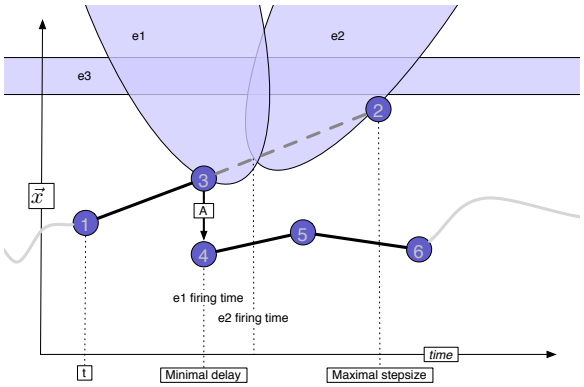


Figure 7.2: The numerical integrator advances between t (point 1) and the maximal stepsize (2). The guards of events e_1, e_2 are satisfied. The regions where these guards are satisfied are shaded. The firing time of e_1 (3) is used to reset the simulator after the discrete transition A (4).

correctly synchronized or when a guard is falsely assumed as monotonic between two successive simulation step. For the former, since the clonal senescence and the mitosis events only refer to the state and derivative of the currently integrated cell, the composed population model remains uncoupled and we do not need to synchronize any of the states.

Failure of an event detection can be caused when a guard is falsely assumed as monotonic between two successive simulation step. In BioRica, since the guard of an event can not refer to the current simulation time, detecting the occurrence of an event is reduced to an intersection test between an $n+1$ dimensional segment and a n dimensional region or polytope, where n is the number of state variables of the integrated node. More specifically for the guards of the cell population model, only intersection between the interpolated segment between two successive integration step and a downward closed region is used, and is performed by evaluating the guard at the end point.

Once the event detection is performed, exact location of the event is computed by numerically solving the equation built with the conjunction of the equation given by the linear interpolation and the guard logical formula.

Specialization The generic simulation algorithm 4 was specialized for the damage segregation study. In particular, *alive* and *update* had to be redefined in a specific way. The *alive* predicate verifies three conditions. First, the cell is checked for immortality, which is realized by fixed point detection. Second, we verify whether the cell is in the state of clonal senescence, by evaluating the two guards described in section 7.2. *Update* has the role of managing new cell creation. For the current cell c it updates its state variables, according to the algebraic equations (4-6 for progenitors), and its statistics (fitness, generation time, etc). It creates a new cell node (daughter of c) according to the equations (4-6 for progeny) and inserts it into the population array *Pop*.

Population simulation On top of this specific stepping algorithm, another algorithm drives the whole population simulation by selectively starting simulations for pending cells in *Pop*. Given a depth n, a root cell c and an extent value e, this algorithm first selects pending nodes required to get a complete binary pedigree tree of depth n rooted at cell c. Afterwards, e leftmost and e rightmost leaves are used as root cells in recursive calls of this algorithm with a decremented value of e. Fix points are detected by testing before simulation if a candidate

cell's initial values P_{int} and P_{dam} are equal to a previously simulated cell, in which case we get a pedigree graph by adding a loop edge.

Determining parameter values that exhibit optimal population fitness is based on averaging individual cell statistics (defined in section 7.5) to compute the mean fitness. In fact, this averaging maps a real value denoting the population fitness to each parameter vector. A coarse computation of this mapping is then built by varying the parameter vector using fixed step size. This coarse estimation is used to determine initial guess of optima position, that are then established by using Brent's Principal Axis[Bre02] method on the mapping.

7.4 Implementation

Simulation results are stored in a relational database. In the context of the damage segregation model, the requirement is to be able to store lineage trees. This implies a serialization step that flattens the tree topology in order to represent it as a vector of pairs of cells ids. These are then directly encoded as a database table, in relation with the table storing the model-specific parameters. The latter enables fixed point detection, as well as provides the possibility to pause and restart batches of simulation at any time and at any node during the exploration of the model.

A set of Python scripts performs simulation trace analysis, permitting in particular generation of graphical output using the graphviz package [EGK+01], interactive visualization with the Tulip software [Aub03], and statistical analysis with R and Mathematica.

7.5 Results

Initial calibration We assume that cells grow until they have attained a critical cell size, P_{div}, which triggers a cell division. A cell may divide symmetrically (halving its mass) or asymmetrically, as defined by size coefficients s_{mother} and $s_{daughter}$. The proportion of damaged proteins that are transmitted to the daughter cell after division is defined by retention coefficient re. To calibrate and validate the system, complete simulations were run to depth 4 in the pedigree tree for an exhaustive range of parameter values (Table 7.1). Rate constants k_1, k_2, and k_4 received fixed values, k_3 and re were given a range of values with small step sizes, and s_{mother} and were given $s_{daughter}$ two pairs of values representing symmetric and asymetric growth strategies. A total of 625 simulations were run summing to 9375 different initial conditions and parameters values. Sample results for pedigree tree are illustrated on figure 7.3. In previous results on the single-cell model, simulations carried out with Mathematica allowed us to follow the fate of progenitor and progeny through several generations. Back-to-back comparisons with these previous results were performed (ignoring pedigree) to validate the new simulator. Successful comparisons with a small number of experimental cell growth results were also performed (data not shown).

Simulation to depth 30. For each of the four scenarios studied here, a representative simulation was chosen by inspecting properties of the initial mother. From the whole parameter space, we selected simulations where the mother cell produces a number of daughters that is both finite and large enough (20-24 divisions depending on the case, since the average life span of wild type budding yeast is 24 divisions).

For each of these simulations, the pedigree tree was calculated up to depth 30, and for each cell in the tree we calculated five values: *initial damage* and *terminal damage* levels

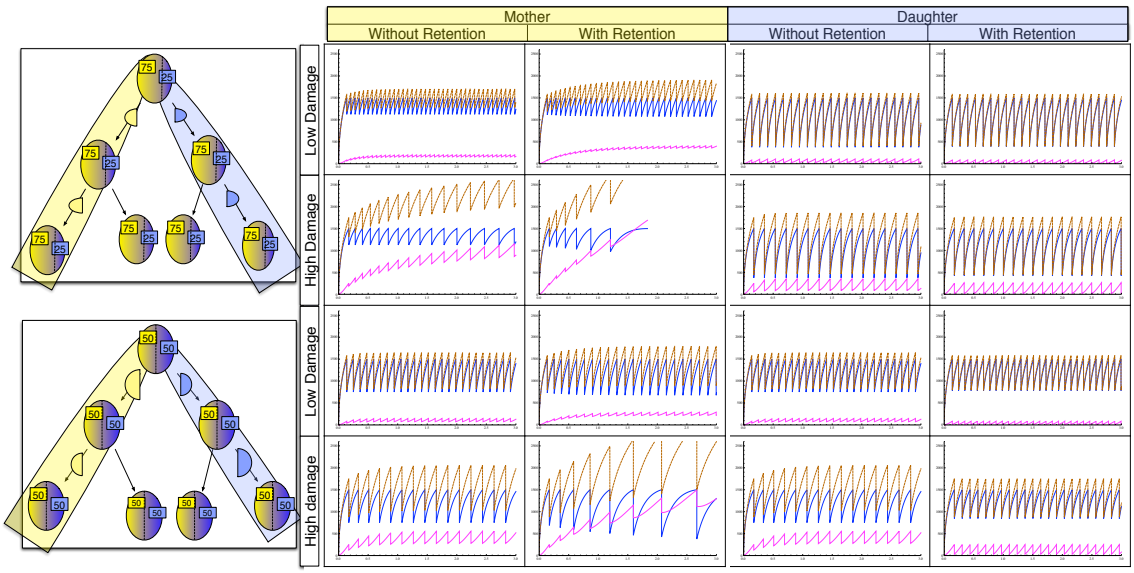


Figure 7.3: Sample pedigree tree results for asymmetrical (Top) and symmetrical (Bottom) division strategies. Pedigree tree (Left) showing mother-daughter relations; and simulation results (Right) showing single cell protein amounts over time: normal proteins (blue), damaged proteins (pink), total proteins (dashed). For example, in the asymmetrical case with high damage, from time zero, the amount of normal proteins (blue) in the mother eventually cross the division threshold at time approx. 0.15. At this time, the proteins repartition is approx. 250 damaged proteins for 1750 total proteins. Once division is triggered, the progeny separates, and a new simulation is started for the mother (resp. the daughter) with initial normal proteins set at $1500 \times 0.75 = 1125$ (resp. $1500 \times 0.25 = 375$) and damaged proteins set at $250 \times 0.75 = 187.5$ (resp. $250 \times 0.25 = 62.5$). Since in this case the mother accumulates damage over divisions (compare damaged proteins amount between time 0.15 and 3.0), it will eventually reach a senescence point after 27 divisions. Comparison of its life span with the life span of each of its daughter and the life span of each daughter of its daughters (as tracked by the upper pedigree tree) shows a rejuvenation effect.

(corresponding respectively to the amounts of damage P_{dam}at the beginning of cell cycle, and at the end of the cycle when division is about to occur), *generation time* (time between two divisions), *absolute date of birth* (in arbitrary time units, measured from the moment when mother starts its first division) and the *fitness* (defined as number of divisions during first time unit).

Parameter Exploration. Using parameter exploration (section 7.3) we identified sets of parameters that exhibited a given emerging high-level behavior, both at the single-cell and whole pedigree tree levels. For example, for the former we are interested in detecting cells that have a certain number of daughters (say, 24), and for the latter we are looking for parameters giving high rejuvenation value across the whole population. These two values are computed by a trace simulation analysis script.

Thus, for each of scenarios studied here, a representative simulation was chosen by inspecting properties of the initial mother. From the whole parameter space, we selected simulations where the mother cell produces a number of daughters that is both finite and large enough (20-24 divisions depending on the case, since the average life span of wild type budding yeast is 24 divisions). For each of these simulations, the pedigree tree was calculated up to depth 30, and for each cell in the tree we calculated five values: *initial damage* and *terminal damage* levels (corresponding respectively to the amounts of damage P_{dam}at the beginning of cell cycle, and at the end of the cycle when division is about to occur), *generation time* (time between two divisions), *absolute date of birth* (in arbitrary time units, measured from the moment when mother starts its first division) and the *fitness* (defined as number of divisions during first time unit).

Model analysis. The hierarchical model we have defined explicitly tracks mother-daughter relations in pedigree trees of simulations. This allows us to study lineage-specific properties, which are properties associated with connected subgraphs of the pedigree tree. Pedigree trees and typical simulation results are shown in Figure 7.3.

In the pedigree tree, a given mother cell generates a series of daughter cells; these siblings are ordered in time, and the younger a sibling, the older the mother at the time of division. We observe in simulation results that younger siblings have higher damage, consistent with inheritance from an older mother that has accumulated more damaged proteins, and these younger siblings are thus born "prematurely old." This increase in damage accumulation is reflected in the decrease of fitness values, shown in the first level of Figure 7.4.

Extending this analysis one level further in the pedigree tree shows, expectedly, that daughters born early to the same mother have low damage, and their daughters have normal fitness. Daughters born late to the same mother have high damage and lower fitness, but remarkably, in simulations with either retention or with asymmetric division, their own daughters are born with lower damage and higher fitness. This increase in fitness in the second generation is a *rejuvenation effect*, in part explaining how populations maintain viability over time despite inheritance of protein damage.

The testable hypothesis is thus that there exists a mechanism for retention of damaged proteins during cell division, that attenuates the accumulation of such proteins in descendants, and that a combination of the precise value of the corresponding retention coefficient (re) and the asymmetry coefficients (s_{mother} and $s_{daughter}$) in the model determines the scale of the rejuvenation effect.

These predictions are consistent with *in vivo* experimental results reported in the literature: Kennedy, et al. [KJG94] report that daughter cells of an old mother cell are born prematurely old, with lower replicative potential, but that the daughters of these daughters have normal

life spans. It should therefore be possible to experimentally estimate the value of the retention coefficient through indirect measures of cell lifespan and replicative potential.

As a control, we also inspected the case of no retention and symmetric division. We would not expect a rejuvenation effect, since inheritance of damaged proteins should be proportional in both mother and daughter cells, and indeed this is what is observed. We conclude that the hierarchical model is an accurate representation of protein damage inheritance.

We then investigated changes in generation time for a given mother cell as a function of cell age. In the case of retention and symmetric division, we observe in simulation results an exponential increase in generation time. This prediction is consistent with experimental observations of Egilmez et al. [EJ89].

In the case of no retention and asymmetric division, we observe a *linear* increase in generation time. Intuitively, without retention the inheritance of damaged protein increases linearly with age (equations 5 and 8), and generation time is linearly dependant on the proportion of intact to total proteins (equation 1). This prediction is consistent with experimental results for a ΔSir2 mutant (Nyström, personal communication).

Finally, in simulations we observe that fitness and viability are sensitive to precise values of k_3, the rate by which proteins are damaged (see figure 7.5). This provides a series of testable hypotheses that could be investigated experimentally in different damaging environments, such as oxidative damage or radiation damage.

7.6 Conclusions

Although purely continuous systems such as ODEs have long been used for quantitative modeling and simulation of biological systems (for example [Men97]) and are commonly thought to be powerful enough, they do not suffice for highly structured models where emerging properties result from dynamic changes to the model.

For this study, the BioRica hybrid formalism and the related framework proved to be powerful enough to model, simulate and analyze the rejuvenation property of a hierarchical damage segregation process, by extending an existing continuous cell model to a population model. Since our hybrid formalism allows a BioRica node to describe and import an ODE system, we maintain the low computational cost and biological soundness of ODEs.

Hybrid simulation in BioRica scheme enjoys the traditional advantages of numerical integration, since the computational overhead for the hybrid stepper is proportional to the number of discrete events in the model; thus hybrid simulation of a purely continuous model is as fast as integrating it. For hybrid models mixing both continuous and discrete dynamics, our clear semantics permits concise description and reproducibility of simulation results in other simulation frameworks. For example, while the division strategy could be described in a continuous model by adjusting sigmoid functions, it is more naturally described by algebraic equations and their description in the model ought to be kept algebraic. The resulting gain in clarity has been observed elsewhere, for example in the complete cell cycle model of [CCCN+04]. While most existing simulation tools admit a programming interface that allows for the modeler to simulate discrete events, the lack of a precise semantics renders the simulation predictions questionable and merely reproducible, since allowing such discrete events in a model has semantics issues[5]. Indeed, two discrete events can be enabled at the same time, but nothing defines whether in such cases the simulator should fire neither event, both events,

[5]See for example http://www.sys-bio.org/sbwWiki/compare/themysterysolved

or some random choice; and different strategies imply radically different simulation results.

In BioRica, we use the mathematical definition of non-determinism already used in discrete formalisms, thus giving any BioRica models a precise mathematical and unambiguous semantics. Furthermore, while not explicitly used in this study, BioRica leverages and extends the compositional operators initially defined in the AltaRica languages family [AGPR00] to allow for parallel, partially synchronous and data sharing compositions of hybrid, stochastic, multi-models and external abstract processes. Furthermore, since such compositions are mathematically defined in BioRica, we can exactly identify subclasses of programs admitting modern model analysis such as model checking, compositional reasoning, functional module decomposition and automatic simplification; all of which were spotted as grand challenges for modeling and simulation in system biology [SHK+06]. For compatibility with other systems biology software, BioRica imports SBML files through libSBML [HFS+03]. In addition to SBML support, BioRica exports the model as software independent C++ code, that can be compiled on any POSIX compliant system. This approach allows initial model prototyping in user friendly workbenches such as xCellerator [SLMW03], followed by use of optimized command line simulators for large scale analyses.

More specifically for population studies, since discrete variables and dynamic node creation are allowed in BioRica, our cell model can explicitly track a dynamic mother-daughter relationship. A realistic population model needs such a dynamic topology. Even when restricting ourselves to the biologically realistic case of dying cells, the number of daughters that any cell can have is *a priori* unbounded; thus, simply replicating the ODE equations to get a continuous population model as in [HMR02] is not scalable. Furthermore, when simulations were carried up to depth 30, approximately 2^{30} cells were evolving in parallel, adding up to a 2^{32}-variable differential system that is untractable using a classical ODE approach. Instead, our population model clearly separates each cell behavior from the population by using hierarchical composition, and uses this modularity to provide a hierarchical simulation scheme, thus ensuring that each individual cell continuous part will be integrated with the most efficient step size.

Furthermore, the properties of parallel composition render study of population model with up to 2^{30} individuals still partially tractable by our scheme since we can linearize this population tree to simulate each cell independently. This approach is efficient since the cost of simulating a population is linearly proportional to the cost of simulating an individual, while flat and unstructured models have a quadratic complexity [EK04]. Finally, since we use discrete variables to track the mother-daughter relationships, we can directly estimate the rejuvenation effects, which would otherwise be buried in a flat and unstructured model.

Large scale exploration to detect the rejuvenation effect required a tree coverage that is out of reach of naive exploration algorithms such as breadth-first or depth-first. In fact, neither the population tree width nor its height are bounded, and thus these algorithms do not terminate. An *ad hoc* exploration algorithm partially solves this problem by alternating evaluation of first born daughters and evaluation of late born daughters, but does not provide the required coverage to detect significant rejuvenation. However, substantial acceleration is provided by the fix point detection scheme encoded in our tree visitor pattern, whose soundness is ensured by the deterministic nature of a cell behavior. In the continuous model, initial values of $P_{\text{int}}$and $P_{\text{dam}}$for given parameter values entirely determine a unique single cell behavior.

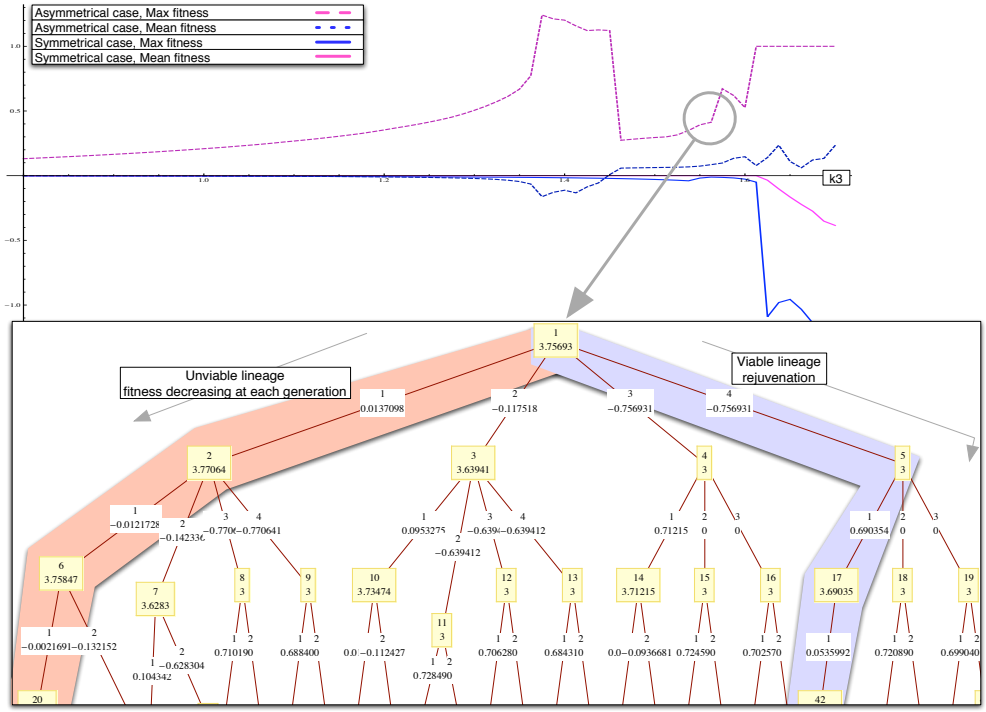


Figure 7.4: Sample parameter exploration result showing the high sensitivity and non linearity of the rejuvenation effect w.r.t. precise values of the damage rate k_3. Only some ranges of the parameter value (approx. between 1.53 and 1.59) exhibits an increase of both the maximal and the mean fitness difference between every mother and daughter of a population. **Top**: Estimation of the maximal and mean rejuvenation amount for damage rate ranging from 1.2 to 1.7 for the asymmetrical and symmetrical case. The higher the values, the fitter some cells of the lineage are compared to their direct mother. **Bottom:** Close up of a lineage tree used to compute one point of the previous rejuvenation plot. Each node (yellow box) is labeled with an numeric id and the floating-point fitness value of the cell, and each edge (white label) is labeled with the index of the daughter relatively to its mother and labeled with the difference of fitness between daughter and mother. For such a given tree, we can compute the mean and max value of the edges, which is represented as one point on the rejuvenation plot. The rejuvenation effect in young daughters of old mothers (right blue colored branch) is consistent with the experimental results of [KJG94].

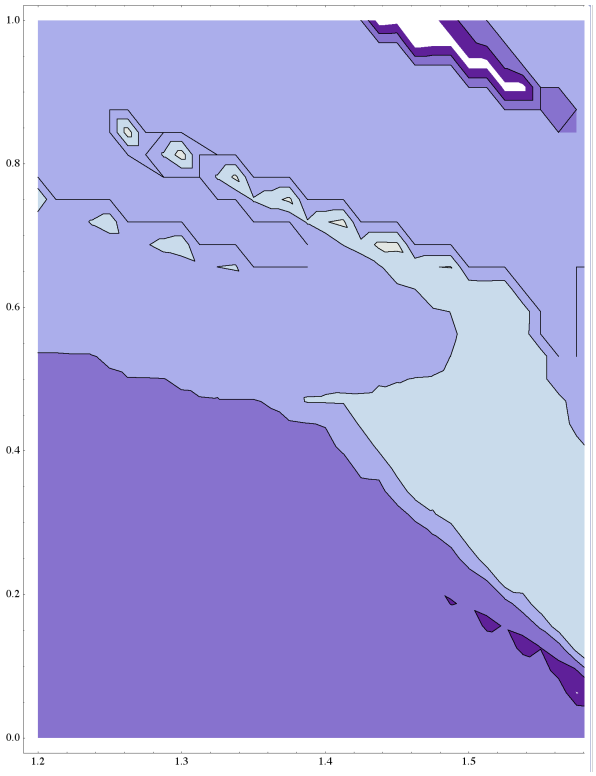


Figure 7.5: Contour plot of mean of cell fitness from one generation level to the next (viability) over a pedigree tree, for a series of simulations in the symmetry case. X axis: rate of damage of intact proteins ($k3$), Y axis: coeff. of retention (re). The lighter the color, the higher the mean, except for white regions, where the value is negative. This contour plot represents structured simulation results for approx. $5,120,000$ cells.

Chapter 8

Concluding remarks and future research directions

In this thesis, we have discussed automata based models for the description of biological processes modeled as hierarchical systems with stochastic timing.

To summarize, we have presented the BioRica language, its stochastic semantics and a translation scheme that generates stand-alone simulators. This framework is able to capture a range of existing time and state driven formalisms, while providing clear ways to combine disparate units into a bigger view. The associated simulation toolkit provides easy of use and computationally efficient simulations, able to cope with multi-scale and multi-models systems by the use of hierarchical modeling. In this way, the best paradigm for a given model can be chosen by the experimentalist who can thus follow a problem-specific approach and be assured that his model will remain usable in higher-order ones, even when those are described by a radically different method. The hope is that modeling can become more local and thus increasingly accessible to non specialists, and that stronger coupling between experimentation and modeling becomes more prevalent.

More precisely, we took as a starting point for BioRica the work done on the AltaRica specification language, and specifically that on the AltaRica Dataflow subset, and we extended its syntax and transition system semantics in order to account for probabilistic descriptions.

We considered the problem of analyzing the dynamics of such models. For this purpose, we defined stochastic transition systems that are transition systems with probabilistic labels. We sketched how the semantics of an STS could be given in terms of a stochastic process by using continuous state space process. However, in order to provide analytical results, we presented an alternative semantics based on probabilities of a sojourn path. One interesting point of this semantics is that the probability of a path can be computed without exploring the state space and that this probability is expressed as linear combinations of random variables. This implies that computations with numerical values are only performed at the very end if needed.

Furthermore, we have considered the problem of simulating hierarchical discrete event systems. To this end, we presented an implementation of a simulator generator from BioRica systems to C++ programs. Given a hierarchical system described in BioRica, we have indeed implemented a tool to compile this system into a set of C++ classes that are ultimately linked with a hierarchical discrete event simulator. By following this approach, simulation of a BioRica system avoids completely the costly process of flattening and state unfolding. Furthermore, by providing a compilation into an expressive target programming language (here

C++) extending a BioRica system with model specific constructions is reduced to inserting escaped expressions in a BioRica model. While we did not detail this approach here, the current implementation of the BioRica compiler is distributed with a Matlab and Mathematica bridge. This extension enables the use of the advanced numerical and symbolic algorithms present in computer algebra systems during simulation, either to perform complex computations in the model or to speed up numerical approximations.

We have then considered the problem of incorporating continuous models described by Ordinary Differential Equations into a BioRica system. We considered three separate complementary approaches: quantization, qualitative abstraction and flow variables with synchronizations. The first approach is classical in the domain of Discrete Event Systems and is based on a regular quantization of the state space and uses the expressivity of the delay mechanism to encode the numerical integrator in a BioRica node.

The second approach is based on an automated analysis of the transient behavior of the continuous system. Given a range of possible qualitative values represented as abstraction functions, we compute from the time series generated by numerical integrators a stochastic transition system. To this end, we introduced the model of *qualitative transition systems* (QTS). A QTS is a transition system where each transition is labeled with the amount of time the system requires before moving to another state. The delay between two state changes follows a parametrized normal distribution. We have shown how QTS can be used to study qualitative properties of parametrized models. This is achieved by defining an appropriate abstraction function. By representing the characteristic qualitative features of a trajectory in an abstract domain that is countable, qualitative similarity can be detected by a simple equality test. We have shown that the soundness of this approach depends on the adequacy of sampling with respect to the abstraction function. In particular, we have shown that for convex abstraction functions, if the sampling is "precise enough", then the QTS obtained from any oversampling has the same transitions. Finally, we applied this approach to some well known models. QTS were used to explore the parameter space and to detect uniform behaviors (oscillations etc.). The limits of our approach as compared to model checking is the lack of exhaustivity. This has to be counterbalanced by the fact that our method is applicable to a large panel of formalisms, even those lacking a precise semantics. Consequently, we can avoid any model transformation. Finally, our approach can be applied independently to the data and to the model.

Finally, the third approach of incorporating continuous systems into a discrete formalism is based on the flow variables and synchronization mechanism. This approach has been used in the last chapter of this thesis, where we applied our methodology to study a model of the process of cell ageing in a population of yeast. To this end, we considered a single cell model based on ODEs, built a "hybrid system" by composing it with a discrete controller modeled in BioRica, and then built a population model by parallel composition. The dynamics of the population (death and birth) is modeled by a BioRica node, while an external integrator solves the differential system of individual cells. The "hybrid model" is then built by using the mechanisms of flow composition and of synchronization. In this case of yeast population, the proposed model integrates the fact that a cell does not interact with the population except at birth, death and when it gives birth to a daughter cell. Furthermore, these interactions are unilateral (i.e. there is no feedback from the population to the cell) and thus, simulation of a population evolving in parallel and successive simulations of individuals are equivalent. We used this property to analyze parts of the population for which biological data were available. Had we limited ourselves to non hierarchical ODE systems, the same simulation would have

required solving a continuous system of approximately 2^30 variables.

Biological interpretation of the simulation results concerns mostly damage segregation strategy for yeasts population. The uneven distribution of damaged proteins may seem intuitive for organisms displaying markedly asymmetrical cytokinesis such as budding yeast; less so when there is no apparent distinction between the two sister cells. However, Stewart et al. [SMPT05] demonstrated that *E.Coli* cells enriched with old pole material displayed a longer generation time than their new pole enriched siblings. This result indicates that asymmetry is also present in systems dividing by binary fission. Although the measured differences in generation times are small in absolute terms, such asymmetry may be of significance for the robustness and fitness of the population. Our model suggests that even very low retention coefficients, e.g. a scenario where one cell receives 58% and the other 42% of the damage, may have a great impact on the systems ability to escape clonal senescence, at least at moderate damage rates. As the damage rate increases, the damage retention, or size asymmetry, needs to be more pronounced to allow the survival of the population.

Furthermore, one common assumption when modeling asymmetrical division is that the establishment of age asymmetry is linked to damage segregation. However, such segregation of damage has, as far as we know, only been shown in the asymmetrically dividing budding yeast. One of the somewhat surprising predictions of the model presented here is that a system dividing symmetrically by size (binary fission) display a higher fitness if damage is segregated regardless of the damage accumulation rate. This suggests that damage retention may be more common than previously anticipated and prompted Marija Cvijovic and her colleagues to analyze the distribution of oxidatively carbonylated proteins during cytokinesis in the fission yeast S. pombe. Her results (not shown in this thesis) showed that *S.pombe* displays an uneven distribution of carbonylated proteins between siblings, which correlates with their longevity and fitness. This result can explain why growth in the presence of minor stressors results in a higher mortality of the siblings with more birth scars, i.e. the ones inheriting more damage. Even though partitioning of damaged proteins to the older sibling is not as pronounced as that of the budding yeast, it appears to be sufficient to entrust the younger sibling with a significantly longer replicative potential and shorter generation time. As stated above, our modeling approach predicts exactly this, i.e. that a small bias towards damage asymmetry has profound consequences on the population's fitness and propagation.

Future research directions

The growing complexity of the fields dealing with biology highlights the need for an unambiguous and fixed framework into which new experimental results can be fitted, turned into predictions, tested, and communicated. In the past, new problems have often led to adopting different paradigms, definitions, and simulation algorithms for each part of the system. This has made it difficult to combine existing and curated sub-models into a communicating system, making it even harder to model or comprehend higher level emerging properties.

On the methodological plan, having a unifying framework for multi-formalism models encourages us to develop new modeling methodology that can account for better integration between experimental data and modeling. The tools we provide in this thesis thus allows one to compose models originating from different mathematical formulation. Executable models, as defined by Fisher in [FH07] can coexists with biologically sound models by describing them in a single formalism of compositional, hierarchical, stochastic and non deterministic discrete

event systems. Such an an approach providing an unifying semantics and tools for continuous, discrete and stochastic models have received numerous attention recently in the field of Systems Biology. Indeed, having the possibility to build complex models by composing almost automatically existing models from the literature is very appealing. However, as we have discussed in the introduction of this thesis, a model is not inherently "true" or even "good", since it has been built with a specific purpose and thus a specific set of assumption and validation methods. However, the focus is still on the model and not on the methodology used to build it. Therefore, a general future plan of research would be to develop the set of tools required to build complex models by using the wealth of available experimental data. That is, instead of following the interpretation of modeling for Systems Biology that suggest that models should be of increased realism, we consider that model can be kept simple while a great deal of complexity has been solved in the conceptual modeling process. By delimiting models and by formalizing the assumptions and validation scheme used during the modeling process, existing model could be reused and readapted. Instead of incorporating as much biological knowledge and data as possible in a model, as illustrated by the various genome scale models of microbial organisms [RVSP03, FSP$^+$06, DHP04] or of simple multicellular organisms [Har03, Har05], an alternative approach would be to consider that the increased "realism" should be in the modeling *process*, and not necessarily in the model itself.

On more technical plans, the BioRica language is just a very first approach to applying a declarative specification language to model biological systems. In order to adapt the language and its hierarchical semantics to the needs of mathematical biologist, collaboration with some of the authors of the SBML language [HFS$^+$03] has actually started. The general plan is that SBML already provide a widely used syntax for flat continuous model description. However, SBML being a machine readable language, it is not *per se* a modeling language and it thus do not provide a semantics. This has not been problematic since SBML v2 is restricted to purely continuous models that admits an non ambiguous interpretation by using numerical integration methods. However, the next iteration of SBML, namely SBML v3, plan on incorporating syntactical construct to declare timed events and hierarchical compositions. This extension requires an non ambiguous interpretation, namely, a mathematical semantics and we wish to propose one semantics that is close to the one established in this manuscript. Importance of an unambiguous semantics has been illustrated by the actual interpretation of the (nearly finished) proposal in state of the art simulators that can handle events. A set of "compatible" simulators yielded incomparable simulation results for the same model.

Concerning stochastic transition systems, the current analysis that are possible (and thus on the subset of GSMP they represent) are hindered by the complexity implied by the non determinism. Although we limit ourselves to finite path (and thus by definition we do not suffer from any "state explosion" problems), the presence of discrete distributions associated with event time implies that we must account for an exponential number of different "ties" to compute the probability of a single path under a given scheduler. However, this conclusion may hold only because we considered general weight scheduler without restrictions, and specific sub classes of schedulers (like memory less scheduler for instance) may reduce this complexity. Furthermore, we did not exactly built a stochastic process but rather characterized its finite dimensional probabilities. Although the Kolmogorov extension theorem may apply immediately, the existence of limit state probabilities is more delicate and needs to be characterized. For instance, their existence is already not guaranteed for finite state semi-Markov chains, which are a subclass of STS. However, characterizing the conditions under which these limit state probabilities would open the possibility to evaluate steady state distributions. Addi-

tionally, an immediate step would be to adapt the scheme of [LHK01] to verify time bounded formula instead of the finite step based formula we proposed. Note however that the best algorithm in our knowledge to evaluate a time bounded formula on a semi-Markov chain has a worst state complexity of $O(N^4)$ where N is the number of state of the SMC [LHK01].

The areas of future research for qualitative transition systems can be declined on the both technical and practical plans. As for the former, we envision a more thorough study of similarity measures of QTS and how QTS similarity relates to language equivalence. As for the latter, we plan to develop clustering techniques in order to detect the resulting behavior similarity in an experimental context.

Concerning the model of cell ageing, when the total fitness (number of cells produced per time unit) of the systems is considered, damage retention may be a mixed blessing. For example, when considering a different cell-size organism like budding yeast, our model predict that damage retention will push the upper limits for how much damage the system can endure before entering clonal senescence but become a selective disadvantage at low damage production rates. This raises the question of whether the efficiency of damage segregation could be adjusted with changing environmental demands. Interestingly, damage segregation in budding yeast becomes more pronounced following increased oxidative stress, suggesting that this unicellular organism, indeed, enjoys the capacity to increase damage segregation upon conditions elevating such damage. In addition, unusually difficult growth conditions elicit a switch from a morphologically symmetrical to a more asymmetrical type of division in fission yeast indicating that also this organism display a dynamic ability to break up symmetry upon environmental demands. Finally, Cvijovic's experimental results and her interpretation with our model suggests the hypothesis that sibling-specific aging and rejuvenation in unicellular systems may have evolved as by-products of a strong selection for damage segregation during cytokinesis. A question of interest is whether such division of labor between cells undergoing division is retained also in multicellular organisms, for example during the generation of germ line cells or differentiation.

Code sample

C++ code

Specialized schedulers

```
/****** Cell.h ********/
class Cell:public Node{
private:
        string* name;
        Node* parent;
        friend class StrategicProduction;
         [...]
};

class IncreasingTemp:public Referee{
        public:
                int choose(int n);
                IncreasingTemp(int seed):lastTemp(0),Referee(seed) {};
        private:
                int lastTemp;
};


class StrategicProduction:public SystemReferee{
        public:
                virtual int choose(Cell *node, int n);
                StrategicProduction(int seed):SystemReferee(seed){};
};
/****** Cell.cpp ********/
#include "Cell.h"
extern int env_seed;
extern int sys_seed;

int IncreasingTemp::choose(int n){
    int choosed = rg->IRandom(this->lastTemp,this->lastTemp+5);
        if (choosed>n){
                choosed=n;
        }
        this->lastTemp=choosed;
        return choosed;
}

int StrategicProduction::choose(Cell *node, int n){
        int choosed = Referee::choose((int)node->alcohol+1);
```

```
40          if (choosed>node→alcohol){
                choosed=1;
42          }else{
                choosed=2;
44          }
            return choosed;
46  }

48  Referee *environment_scheduler=NULL;
    StrategicProduction *system_scheduler=NULL;
50
    Cell :: Cell(char* a_name, Node *a_parent){
52          environment_scheduler=new IncreasingTemp(env_seed);
            system_scheduler=new StrategicProduction(sys_seed);
54
            this→name=new string(a_name);
56          this→laws = new Cell_Laws();
            this→parent=a_parent;
58          /* initVal */
            alive=𝟙;
60          this→update_flows();
    }
62   [...]

64  void Cell::ferment(){
            int possible=0;
66          int choosed=-1;

68          if((((alive) ∧ (((temperature) > (15)))) ∧∧
                    in_alcohol_domain(((alcohol) + (1))) ∧∧ 𝟙))
70                  possible++;

72          if((((alive) ∧ (((temperature) ≤(25)))) ∧∧ 𝟙))
                    possible++;
74
            // Ask for the system referee to choose which transition to follow
76          // choosed=Referee :: system_instance ()→choose( possible );

78          // Delegate the choice to the systemscheduler
            choosed=system_scheduler→choose(this,possible);
80          if (choosed==1){
                        // Alcohol producing fermentation
82                      float local_alcohol=alcohol;
                        alcohol=(int)(((local_alcohol) + (1)));
84                      return;
            }else if (choosed == 2){
86                  //no production
                    return;
88
            }else{
90                  abort();
            }
92  }
```

```
94  [...]
    void Cell::update_flows(){
96          //Ask for the default environment referee to update the temperature
            //  this->temperature=Referee::environment_instance()->choose(100);
98
            //Ask for the specific environment_scheduler
100         this->temperature=environment_scheduler->choose(100);
    }
```

Bibliography

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and computation*, 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, TA Henzinger, P.H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[AG07] S. Asmussen and P.W. Glynn. *Stochastic simulation: Algorithms and analysis*. Springer Verlag, 2007.

[AGPR00] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40:109–124, 2000.

[AGRN03] H Aguilaniu, L Gustafsson, M Rigoulet, and T Nystrom. Asymmetric inheritance of oxidatively damaged proteins during cytokinesis. *Science*, 299(5613):1751–1753, 2003.

[AIK+03] R Alur, F Ivancic, J Kim, I Lee, and O Sokolsky. Generating embedded software from hierarchical hybrid models. In *LCTES*, pages 171–182, 2003.

[And08] David R. Anderson. *Model Based Inference in the Life Sciences: A Primer on Evidence*. Springer, 2008.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communiquants*. Masson, Paris, 1992.

[Arn94] A. Arnold. *Finite Transitions Systems*. Number 0-13-0929990-5 in Prentice Hall. Prentice Hall, 1994.

[ASSB96] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton. Verifying continuous-time markov chains. *Lecture Notes in Computer Science*, 1102:269–276, 1996.

[Aub03] D Auber. Tulip : A huge graph visualisation framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

[BBHT06] K. Burrage, PM Burrage, N. Hamilton, and T. Tian. Compute-intensive simulations for cellular models. *Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies*, page 79, 2006.

[BBHW07] Fred C. Boogerd, Frank J. Bruggeman, Jan-Hendrik S. Hofmeyr, and Hans V. Westerhoff. *Systems Biology: Philosophical Foundations*. Elsevier, first edition edition, 2007.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BCP06] R. Blossey, L. Cardelli, and A. Phillips. A compositional approach to the stochastic dynamics of gene networks. *Lecture Notes in Computer Science*, 3939:99–122, 2006.

[BHHK00] C. Baier, B.R. Haverkort, H. Hermanns, and J.P. Katoen. Model checking continuous-time markov chains by transient analysis. In *Proc. 12th International Conference on Computer Aided Verification*, volume 1855, pages 358–372. Springer, 2000.

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

[BR94] S. Brlek and A. Rauzy. Synchronization of constrained transition systems. In H. Hong, editor, *Proc. of the First International Symposium on Parallel Symbolic Computation*, pages 54–62. World Scientific Publishing, 1994.

[BRDJ+05] G. Batt, D. Ropers, H. De Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider. Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in escherichia coli. *Bioinformatics-Oxford*, 21(1):19, 2005.

[Bre02] R.P. Brent. *Algorithms for minimization without derivatives*. Dover Pubns, 2002.

[CC08] Claudio Cobelli and Ewart Carson. *Modeling in Physiology and Medicine*. Academic Press, 2008.

[CCB01] Ewart Carson, Claudio Cobelli, and Joseph Bronzino. *Modelling methodology for physiology and medicine*. Academic press, 2001.

[CCCN+04] KC Chen, L Calzone, A Csikasz-Nagy, FR Cross, B Novak, and JJ Tyson. Integrative analysis of cell cycle control in budding yeast. *Mol Biol Cell*, 15(8):3841–62, Aug 2004.

[CD89] L. E. Catagirone and R.L. Doutt. The history of the vedalia beetle importation to california and its impact on the development of biological control. *Annual Review of Entomology*, 34:1–16, 1989.

[CE81] E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, volume 131, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Transactions on Programmaing Languages and Systems*, 8-2:244–263, April 1986.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. Springer, 1999.

[CH02] P. Caley and J. Hone. Estimating the force of infection; mycobacterium bovis infection in feral ferrets mustela furo in new zealand. *Journal of Animal Ecology*, 71(1):44–54, 2002.

[Cha31] TC Chamberlin. The method of multiple working hypotheses. *The Journal of Geology*, 39(2):155–165, 1931.

[Che94] J Chesneaux. The equality relations in scientific computing. *Numerical Algorithms*, Jan 1994.

[Cin75] E. Cinlar. Introduction to stochastic processes. *Englewood Cliffs*, 1975.

[CK83] TR Chay and J. Keizer. Minimal model for membrane oscillations in the pancreatic beta-cell. *Biophysical Journal*, 42(2):181–189, 1983.

[CK05] F.E. Cellier and E. Kofman. *Continuous System Simulation*. Birkh ”auser, 2005.

[CL08] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2008.

[CSS+08] M. Cvijovic, H. Soueidan, D.J. Sherman, E. Klipp, and M. Nikolski. Exploratory simulation of cell ageing using hierarchical models. *Genome Informatics*, 21:114–125, 2008.

[D'A99] PR D'Argenio. *Algebras and automata for real-time systems*. PhD thesis, Department of Computer Science, University of Twente, 1999.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):253–291, 1997.

[DHP04] N.C. Duarte, M.J. Herrgård, and B.Ø. Palsson. Reconstruction and validation of saccharomyces cerevisiae ind750, a fully compartmentalized genome-scale metabolic model. *Genome Research*, 14(7):1298, 2004.

[DKB98] P.R. D'Argenio, J.P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In *Proceedings IFIP Working Conference on Programming Concepts and Methods, 22 pages. New York, USA. Chapman & Hall*. Citeseer, 1998.

[Dur05] R. Durrett. *Probability: Theory and Examples*. Thomson, 2005.

[Dym04] Clive L. Dym. *Principles of Mathematical Modeling, 2ed, 2004, Elsevier, 0122265513*. Elsevier, 2004.

[EGK+01] J Ellson, ER Gansner, E Koutsofios, SC North, and G Woodhull. Graphviz - open source graph drawing tools. In *Graph Drawing*, pages 483–484, 2001.

[EJ89] NK Egilmez and SM Jazwinski. Evidence for the involvement of a cytoplasmic factor in the aging of the yeast saccharomyces cerevisiae. *J Bacteriol*, 171(1), 37-42 1989.

[EK04] J Esposito and V Kumar. An asynchronous integration and event detection algorithm for simulating multi-agent hybrid systems. *ACM Transactions on Modeling and Computer Simulation*, Jan 2004.

[F+57] W. Feller et al. *An introduction to probability theory and its applications*. Wiley New York, 1957.

[FF98] Dominique Foata and Aimé Fuchs. *Calcul des probabilités*. Dunod, 2nd edition, 1998.

[FH07] J. Fisher and T.A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239, 2007.

[FHS+01] A Finney, M Hucka, H Sauro, J Doyle, and H Kitano. The systems biology markup language. *Mol. Biol. Cell*, Jan 2001.

[FMW+03] C.P. Fall, ES Marland, JM Wagner, JJ Tyson, and J. Sneyd. Computational cell biology. *Mathematical Medicine and Biology*, 20:131–133, 2003.

[For71] J.W. Forrester. *World dynamics*. Wright-Allen Press Cambridge, MA, 1971.

[FP93] L Fribourg and M Veloso Peixoto. Concurrent constraint automata. In *ILPS*, page 656, 1993.

[Fra04] Jean-Pierre Françoise. *Oscillations en biologie : Analyse qualitative et modèles*. Mathématiques et applications. Springer, 2004.

[FSP+06] A.M. Feist, J.C.M. Scholten, B.Ø. Palsson, F.J. Brockman, and T. Ideker. Modeling methanogenesis with a genome-scale metabolic reconstruction of methanosarcina barkeri. *Molecular systems biology*, 2(1), 2006.

[Gal06] M Galassi. *GNU scientific library : reference manual*. Network Theory, Bristol, second edition, 2006.

[GGH+01] Albert Goldbeter, Didier Gonze, Gerald Houart, Jean-Christophe Leloup, Jose Halloy, and Genevieve Dupont. From simple to complex oscillatory behavior in metabolic and genetic control networks. *Chaos*, 11(1):247–260, Mar 2001.

[GHLG03] Didier Gonze, José Halloy, Jean-Christophe Leloup, and Albert Goldbeter. Stochastic models for circadian rhythms: effect of molecular noise on periodic and chaotic behaviour. *C R Biol*, 326(2):189–203, Feb 2003.

[Gil77] D Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.

[Gly89] PW Glynn. A gsmp formalism for discrete event simulation. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with pvs. *Lecture Notes in Computer Science*, 1254:72–83, 1997.

[Gut05] Allan Gut. *Probability: A graduate course*. Springer, 2005.

[Haa02] P.J. Haas. *Stochastic petri nets: Modelling, stability, simulation*. Springer Verlag, 2002.

[Hae05] James W. Haefner. *Modeling Biological Systems: Principles and Applications*. Springer, 2nd edition, 2005.

[Hal89] M B Hallett. The unpredictability of cellular behavior: trivial or of fundamental importance to cell biology? *Perspect Biol Med*, 33(1):110–9, 1989.

[Har03] D. Harel. A grand challenge for computing: Towards full reactive modeling of a multi-cellular animal. *Bulletin of the EATCS*, 81:226–235, 2003.

[Har05] D. Harel. A turing-like test for biological modeling. *Nature Biotechnology*, 23(4):495–496, 2005.

[HFS+03] M Hucka, A Finney, HM Sauro, JC Bolouri, and H Kitano et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–31, 2003.

[HH39] A.L. Hodgkin and A.F. Huxley. Action potentials recorded from inside a nerve fibre. *Nature*, 144(3651):710–711, 1939.

[HH52] AL Hodgkin and AF Huxley. A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. *J. Physiol.(Lond.)*, 117:500–544, 1952.

[Hil01] B. Hille. *Ion channels of excitable membranes*. Sunderland, MA: Sinauer Associates, 3rd edition edition, 2001.

[HKKH91] T. Hunkapiller, RJ Kaiser, BF Koop, and L. Hood. Large-scale and automated DNA sequence determination. *Science*, 254(5028):59, 1991.

[HMR02] M Henson, D Muller, and M Reuss. Cell population modelling of yeast glycolytic oscillations. *Biochem J*, Jan 2002.

[HMU06] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-wesley, 2006.

[Hol78] CS Holling. The spruce-budworm/forest-management problem. *Adaptive environmental assessment and management. John Wiley, New York, New York, USA*, pages 143–182, 1978.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science*. Cambridge University Press, 2004.

[HSG+06] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. Copasi–a complex pathway simulator. *Bioinformatics*, 22(24):3067, 2006.

[Kar77a] W.J. Karplus. The place of systems ecology models in the spectrum of mathematical models. *New directions in the Analysis of Ecological Systems. Part-2*, 1977.

[Kar77b] W.J. Karplus. The spectrum of mathematical modeling and systems simulation. *ACM SIGSIM Simulation Digest*, 9(1):38, 1977.

[Kar83] W.J. Karplus. The spectrum of mathematical models. *Perspectives in Computing*, 3(2):4–13, 1983.

[KBD+94] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[KE03] George J. Klir and Doug Elias. *Architecture of systems problem solving*. Springer, 2nd edition, 2003.

[KHK+04] N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, J.A. Hubbard, and M.J. Stern. Formal modelling of c. elegans development: A scenario-based approach. *Modelling in molecular biology*, pages 151–173, 2004.

[Kho00] B N Kholodenko. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *Eur J Biochem*, 267(6):1583–1588, Mar 2000.

[Kit01] Hiroaki Kitano, editor. *Foundations of Systems Biology*. The MIT press, 2001.

[KJG94] BK Kennedy, NR Austriaco Jr, and L Guarente. Daughter cells of saccharomyces cerevisiae from old mothers display a reduced life span. *J Cell Biol*, 127(6 part 2):1985–93, 1994.

[KKM+08] N. Kam, H. Kugler, R. Marelly, L. Appleby, J. Fisher, A. Pnueli, D. Harel, M.J. Stern, and E.J.A. Hubbard. A scenario-based approach to modeling development: A prototype model of c. elegans vulval fate specification. *Developmental Biology*, 323(1):1–5, 2008.

[KLH+07] E. Klipp, W. Liebermeister, A. Helbig, A. Kowald, and J. Schaber. Systems biology standards–the community speaks. *Nature biotechnology*, 25(4):390, 2007.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. *Lecture Notes in Computer Science*, 2324:200–204, 2002.

[KNSS00] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. *Lecture Notes in Computer Science*, 1877:123–??, 2000.

[KNSS02] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[KRKC07] D Kim, O Rath, W Kolch, and K Cho. A hidden oncogenic positive feedback loop caused by crosstalk between wnt and erk pathways. *Oncogene*, Jan 2007.

[KS98] James Keener and James Sneyd. *Mathematical Physiology*. Mathematical Biology. Springer, 1998.

[KSK66] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[Kul95] V.G. Kulkarni. *Modeling and analysis of stochastic systems*. Chapman & Hall/CRC, 1995.

[LC03] Daniel Lascar and René Cori. *Logique mathématique, tome 1 : Calcul propositionnel, algèbres de Boole, calcul des prédicats (Paperback)*. Dunod, 2003.

[Lev66] R Levins. The strategy of model building in population biology. In E. Sober, editor, *Conceptual issues in evolutionary biology*, pages 18–27. Cambridge, MA: MIT Press, 1966.

[LHK01] G Lopez, H Hermanns, and J Katoen. Beyond memoryless distributions: Model checking semi-markov chains. *Lecture notes in computer science*, Jan 2001.

[LK06a] Wolfram Liebermeister and Edda Klipp. Bringing metabolic networks to life: convenience rate law and thermodynamic constraints. *Theor Biol Med Model*, 3:41, 2006.

[LK06b] Wolfram Liebermeister and Edda Klipp. Bringing metabolic networks to life: integration of kinetic, metabolic, and proteomic data. *Theor Biol Med Model*, 3:42, 2006.

[LNBB+06] N. Le Novere, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, et al. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34(Database Issue):D689, 2006.

[LRR+02] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, et al. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298(5594):799, 2002.

[LS89] KG Larsen and A. Skou. Bisimulation through probabilistic testing (preliminary report). In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 344–352. ACM New York, NY, USA, 1989.

[LW00] D.J. Lockhart and E.A. Winzeler. Genomics, gene expression and dna arrays. *NATURE-LONDON-*, pages 827–836, 2000.

[MA99] H H McAdams and A Arkin. It's a noisy business! genetic regulation at the nanomolar scale. *Trends Genet*, 15(2):65–9, Feb 1999.

[Mat62] Klaus Matthes. Zur theorie der bedienungsprozesse. In Publishing House of the Czechoslovak Academy of Sciences, editor, *Transactions of the Third Prague Conference on Information Theory, Statistical Decision Functions, Random Processes*, June 1962.

[MED02] R. Mahadevan, J.S. Edwards, and F.J. Doyle. Dynamic flux balance analysis of diauxic growth in escherichia coli. *Biophysical Journal*, 83(3):1331–1340, 2002.

[Men93] P. Mendes. Gepasi: A software package for modeling the dynamics, steady states and control of biochemical and other systems, comput. *Applic. Biosci*, 9:563–571, 1993.

[Men97] P Mendes. Biochemistry by numbers: simulation of biochemical pathways with gepasi 3. *Trends Biochem. Sci.*, 22(9):361–3, 1997.

[Mil82] R. Milner. *A calculus of communicating systems*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1982.

[MK01] P. Mendes and D.B. Kell. MEG (Model Extender for Gepasi): a program for the modelling of complex, heterogeneous, cellular systems. *Bioinformatics*, 17(3):288, 2001.

[MN72] P Z Marmarelis and K Naka. White-noise analysis of a neuron chain: an application of the wiener theory. *Science*, 175(27):1276–8, Mar 1972.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[Mur03] J.D. Murray. *Mathematical biology*. Springer Verlag, 2003.

[Neu75] M.F. Neuts. Probability distributions of phase type. *Liber Amicorum Prof. Emeritus H. Florin*, pages 173–206, 1975.

[Neu81] M. F Neuts. Matrix geometric solutions in stochastic models, an algorithmic approach. *The Johns Hopkins University Press*, 1981.

[Nii99] I. Niiniluoto. Defending abduction. *Philosophy of Science*, pages 436–451, 1999.

[NN92] Hanne R. Nielson and Flemming Nielson. *Semantics with applications: a formal introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[Nur94] P Nurse. Fission yeast morphogenesis–posing the problems. *Mol Biol Cell.*, 5(6):613–616, June 1994.

[OS93] S.H. Orzack and E. Sober. A critical assessment of levins's the strategy of model building in population biology (1966). *Quarterly Review of Biology*, pages 533–546, 1993.

[Pag04] Claire Pagetti. *Extension temps réel d'Altarica*. PhD thesis, Ecole Centrale Nantes, 2004.

[Par02a] D.A. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[Par02b] David Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, School of Computer Science, University of Birmingham, 2002.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. report daimi fn-19. *Computer Science Department, Aarhus University*, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, 1977.

[Poi00] Gérald Point. *Altarica: Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, Ecole Doctorale de Mathématiques et Informatique, Bordeaux, 2000.

[Pri95] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578, 1995.

[PTVF02] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, February 2002.

[PTVF07] WH Press, SA Teukolsky, WT Vetterling, and BP Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, third edition, 2007.

[PZ93] A. Pnueli and L.D. Zuck. Probabilistic verification. *Information and computation*, 103(1):1–29, 1993.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in cæsar. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 195–220. Springer-Verlag, 1982.

[Rau01] Antoine Rauzy. The altarica dataflow syntax. Technical report, LaBRI, 2001.

[Rau02] Antoine Rauzy. Mode automata and their compilation in fault trees. *Reliability Engineering and System Safety*, 78:1–12, 2002.

[RBFS08] A. Rizk, G. Batt, F. Fages, and S. Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. *Lecture Notes in Computer Science*, 5307:251–268, 2008.

[RC04] C.P. Robert and G Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, second edition edition, 2004.

[Ros96] S.M. Ross. *Stochastic processes*. Wiley New York, 1996.

[Rud74] W. Rudin. *Real and complex analysis*. Series in Higher mathematics. McGraw Hill, 2nd edition, 1974.

[RVSP03] J.L. Reed, T.D. Vo, C.H. Schilling, and B.O. Palsson. An expanded genome-scale model of escherichia coli k-12 (ijr904 gsm/gpr). *Genome Biol*, 4(9):R54, 2003.

[Seg95] Roberto Segala. *Modeling and verification of randomized distributed real time systems*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1995.

[Seg96] R. Segala. Modeling and verification of randomized distributed real-time systems. 1996.

[SGB91] LF Shampine, I. Gladwell, and RW Brankin. Reliable solution of special event location problems for ODEs. *ACM Transactions on Mathematical Software (TOMS)*, 17(1):11–25, 1991.

[SGSB06] Ralf Steuer, Thilo Gross, Joachim Selbig, and Bernd Blasius. Structural kinetic modeling of metabolic networks. *Proc Natl Acad Sci U S A*, 103(32):11868–73, Aug 2006.

[She93] G.S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, 1993.

[SHFD04] B Shapiro, M Hucka, A Finney, and J Doyle. Mathsbml: a package for manipulating sbml-based biological models. *Bioinformatics*, Jan 2004.

[SHK+06] H.M. Sauro, D. Harel, M. Kwiatkowska, C.A. Shaffer, A.M. Uhrmacher, M. Hucka, P. Mendes, L. Stromback, and J.J. Tyson. Challenges for modeling and simulation methods in systems biology. In *Proceedings of the 38th conference on Winter simulation*, page 1730. Winter Simulation Conference, 2006.

[SLMW03] B Shapiro, A Levchenko, E Meyerowitz, and B Wold. Cellerator: extending a computer algebra system to include biochemical arrows for signal transduction simulations. *Bioinformatics*, Jan 2003.

[SMG98] DA Sinclair, K Mills, and L Guarente. Molecular mechanisms of yeast aging. *Trends Bioch. Sci.*, 23(4):131–4, April 1998.

[SMPT05] Eric J Stewart, Richard Madden, Gregory Paul, and François Taddei. Aging and death in an organism that reproduces by morphologically symmetric division. *PLoS Biol*, 3(2):e45, Feb 2005.

[SRT06] C.A. Shaffer, R. Randhawa, and J.J. Tyson. The role of composition and aggregation in modeling macromolecular regulatory networks. In *Proceedings of the 38th conference on Winter simulation*, page 1636. Winter Simulation Conference, 2006.

[SSN07] Hayssam Soueidan, David James Sherman, and Macha Nikolski. BioRica: A multi model description and simulation system. In *Foundations of Systems Biology and Engineering F0SBE*, pages 279–287, Allemagne, 2007.

[SSN09] Hayssam Soueidan, Grégoire Sutre, and Macha Nikolski. Qualitative Transition Systems for the Abstraction and Comparison of Transient Behavior in Parametrized Dynamic Models. In 21, editor, *Proc. 7th Int. Conf. on Computational Methods in Systems Biology (CMSB'09), Bologna, Italy, Aug.-Sep. 2009 Computational Methods in Systems Biology (CMSB'09)*, volume 5688 of *Lecture notes in BioInformatics*, pages 313–327, Bologna Italie, 2009. Springer Verlag.

[SZ96] D K Smetters and A Zador. Synaptic transmission: noisy synapses and noisy neurons. *Curr Biol*, 6(10):1217–8, Oct 1996.

[TCC+04] J.J. Tyson, Katherine C. Chen, Laurence Calzone, Attila Csikasz-Nagy, Frederick R. Cross, and Bela Novak. Integrative analysis of cell cycle control in budding yeast. *Mol. Biol. Cell*, 15(8):3841–3862, 2004.

[THT+99] M. Tomita, K. Hashimoto, K. Takahashi, TS Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, JC Venter, et al. E-cell: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72, 1999.

[TIS+03] K. Takahashi, N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama, and M. Tomita. E-cell 2: Multi-platform e-cell simulation system. *Bioinformatics*, 19(13):1727, 2003.

[TKA+00] J.J. Tyson, Chen KC, Csikasz-Nagy A, Gyorffy B, Val J, and Novak B. Kinetic analysis of a molecular model of the budding yeast cell cycle. *Mol Biol Cell*, pages 369–91, 2000.

[TKHT04] K. Takahashi, K. Kaizu, B. Hu, and M. Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20(4):538–546, 2004.

[Tys91a] J Tyson. Modeling the cell division cycle: cdc2 and cyclin interactions. *Proceedings of the National Academy of Sciences*, Jan 1991.

[Tys91b] J.J. Tyson. Modeling the cell division cycle: cdc2 and cyclin interactions. *PNAS*, 88(16):7328–7332, 1991.

[UDLK05] Adelinde M. Uhrmacher, Daniela Degenring, Jens Lemcke, and Mario Krahmer. Towards reusing model components in systems biology. In *Computational Methods in Systems Biology*, pages 192–206. Springer, 2005.

[UDZ05] A.M. Uhrmacher, D. Degenring, and B. Zeigler. Discrete event multi-level models for systems biology. In *Transactions on Computational Systems Biology*, volume 1, pages 66–69. Springer, 2005.

[UGC+00] P. Uetz, L. Giot, G. Cagney, T.A. Mansfield, R.S. Judson, J.R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, et al. A comprehensive analysis of protein–protein interactions in saccharomyces cerevisiae. *Nature*, 403(6770):623–627, 2000.

[Var85] Vardi. Automatic verification of probabilistic concurrent finite state programs. 26^{th} *Annual Symposium on Foundations of Computer Science, Portland, Oregon*, pages 327–338, 1985.

[vDMMO00] G. von Dassow, E. Meir, E.M. Munro, and G.M. Odell. The segment polarity network is a robust developmental module. *Nature*, 406(6792):188–192, 2000.

[VDV00] A. Van Deursen and J. Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[VGSST95] R.J. Van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[VKBL02] José M G Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proc Natl Acad Sci U S A*, 99(9):5988–92, Apr 2002.

[Vol31] V. Volterra. Variations and fluctuations of the number of individuals in animal species living together. *Animal Ecology*, 1931.

[Web79] Jackson Webster. Hierarchical organization of ecosystems. In E Halfon, editor, *Theoretical Systems Ecology*, pages 119–129. Academic Press, London, 1979.

[Whi80] Ward Whitt. Continuity of generalized semi-markov processes. *Mathematics of Operations Research*, 5(4):494–501, November 1980.

[Wil06] Darren J. Wilkinson. *Stochastic modelling for systems bioglogy*. Chapmand & Hall/CRC, 2006.

[Win93] G. Winskel. *The formal semantics of programming languages: an introduction*. The MIT Press, 1993.

[WR08] Michael Weisberg and Kenneth Reisman. The robust volterra principle. *The philosophy of science*, 2008.

[WRK00] J A White, J T Rubinstein, and A R Kay. Channel noise in neurons. *Trends Neurosci*, 23(3):131–7, Mar 2000.

[Yat78] F E Yates. Complexity and the limits to knowledge. *Am J Physiol*, 235(5):R201–4, Nov 1978.

[YS05] H.L.S. Younes and R.G. Simmons. Solving generalized semi-markov decision processes using continuous phase-type distributions. In AAAI Press, editor, *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 742–747, 2005.

[Zim08] Armin Zimmermann. *Stochastic Discrete Event Systems: Modeling, Evaluation, Applications*. Springer, 2008.

[ZK06] Z. Zi and E. Klipp. Sbml-pet: a systems biology markup language-based parameter estimation tool. *Bioinformatics*, 22(21):2704, 2006.