Machine Learning and Statistical Verification for Security

Dimitri Antakly

▶ **To cite this version:**

Dimitri Antakly. Machine Learning and Statistical Verification for Security. Machine Learning [cs.LG]. Université de Nantes (UN), FRA., 2020. English. NNT : . tel-02891862

HAL Id: tel-02891862
https://hal.science/tel-02891862

Submitted on 20 Jul 2020

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

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

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE NANTES
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Dimitri Antakly »

« Apprentissage et Vérification Statistique pour la Sécurité »

Thèse présentée et soutenue à « Nantes », le 02 Juillet 2020
Unité de recherche : LS2N
Thèse N° :

Rapporteurs avant soutenance :

Thomas Schiex	Directeur de Recherche à INRAE Toulouse
Karim Tabia	Maître de Conférences (HDR) à l'Université d'Artois, Lens

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du Jury ne comprend que les membres présents

Président :	Céline Rouveirol	Professeure des Universités à l'Université Paris 13
Examinateurs :	Céline Rouveirol	Professeure des Universités à l'Université Paris 13
	Nathalie Bertrand	Chargée de Recherche (HDR) à Inria Rennes Bretagne-Atlantique
	Thomas Schiex	Directeur de Recherche à INRAE Toulouse
	Karim Tabia	Maître de Conférences (HDR) à l'Université d'Artois, Lens
Dir. de thèse :	Philippe Leray	Professeur des Universités à l'Université de Nantes
Co-dir. de thèse :	Benoît Delahaye	Maître de Conférences à l'Université de Nantes

Invité(s) :

Christophe Sabet	Responsable Financement et Partenariats R&D chez GFI informatique

I have learned silence from the talkative, toleration from the intolerant, and kindness from the unkind; yet strange, I am ungrateful to these teachers.

-Gebran Khalil Gebran

ACKNOWLEDGEMENT

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

Firstly, I would like to express my sincere gratitude to my advisors Benoît Delahaye and Philippe Leray for the continuous support throughout my PhD study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the times of research and writing of this thesis. I could not have imagined having better advisors and mentors for my PhD.

I gratefully acknowledge the funding received towards my PhD from the collaboration with GFI informatique group. Thanks to all my managers and colleagues especially Thierry Vrignaud, Martial Renaud, Christophe Sabet and Arnaud Grall for their encouragement and expertise.

My deep appreciation goes out to the LS2N laboratory and research team members of the DuKe team: Pierre-Hugues Joalland, Mathilde Monvoisin and Philipp Behrendt. Their excellent work during the implementation phase has made an invaluable contribution towards my PhD. I am also grateful to all the Aelos team members especially: Christian Attiogbe, Eva Ran Bao, Pascal André and Fawzi Azzi for the friendly moments and the warmth they extended to me during my time in the lab.

A thank you from the heart to every member of my family, especially my mom whose unconditional love made me believe that I can achieve the impossible. My dad, a strong man who raised me well. My little sister Dina who I wish to have by my side and protect until forever and my uncle Michel who has been like my second father during my adventure in Nantes. I would also like to send a heartfelt thank you for my childhood beloved friends, my second family: Jad, Mario, Mounir, Lilo and Jano for all the memories and good times that kept me going during all these years.

And last but not least, a HUGE thank you for my "Nantaise" family (the list is too long), you guys made it feel like home. I am grateful to every person who made the journey better, easier, more fun and memorable. Thank you Nantes for being so great. Thank you Mic for constantly having faith in me, thank you Fiche for teaching me great values, thank you Zlatex for always being there despite the distance, thank you Flou "the comrade of the road!", thank you Besse for teaching me not to settle for normal, thank you Kryzo for helping me reach for the stars, thank you Georgie for your big heart, thank you Monsieur Keeen for the role modeling, thank you Ace for all the laughs, thank you Saloume for your care and thank you Georges Challita for being awesome.

TABLE OF CONTENTS

INTRODUCTION

Since the creation of the Internet in the early 90's, people and societies opinions regarding it have been permanently evolving. Today we live in a world that is completely dependent on the Internet, it is a part of everyone's life. Therefore, the use of machines, computers, smart phones and connected devices is exponentially growing year after year. Researchers and sociologists have even identified four types of Internet, the Internet of content (Google, Wikipedia, etc.), the Internet of people (social media), the Internet of things (clouds, connected objects, etc.) and the Internet of places (mobility, Google maps, etc.) [ZH09; Sel+09; BG14]. As a consequence, connected machines became the most targeted entry point for malicious agents. All connected machines communicate via huge networks and servers that are governed by complicated systems. The more the systems become complicated the more the fraudulent exploit techniques become sophisticated.

In most of nowadays real life jobs or applications, employees, workers, consumers and users are using connected devices. Hence, they are prone to not only external exploits but also internal misuse that can become dangerous. Consequently, it is important to monitor the behavior of users and workers in their environment to ensure a "secure" work flow. For instance, monitoring the behavior of truck drivers can allow the supervisor to verify if they are getting enough rest time while on route. Computer scientists have been putting a lot of effort in researches in order to build a safe perimeter in which connected devices can be used [Wur+16; Abo+15]. In this work, we use behavior analytics in order to identify a normal behavior from a dangerous one while rating its level of dangerousness. The key of establishing an adequate behavior analytic is a set of faithful data recorded from the concerned system. In 2020 we generate in ten minutes more data than the entire data that is recorded throughout history until 2003. Thus, clearly we have no problems with generating data nor storing it, hence the problem is the capability of extracting useful information from this huge amount of data. This is where data mining techniques [Van11; Van14] come into play. In this thesis we do not directly address this problem, however we emphasize on the importance of faithful data

because we use data-driven techniques.

This thesis was achieved in collaboration with GFI informatique group[1]. GFI is an international group that mainly proposes computer science solutions and expertise, touching a wide variety of domains (banking, road safety, supervision, cloud, AI solutions and many more). The Center of Innovation and Expertise (CIE) at GFI directs many research and development projects (R&D). The main expectations of GFI about this thesis are to explore behavior analytics including how it can be applied in security or supervision, and to use innovative techniques in order to be distinguished from the available market solutions.

GFI informatique group was actively involved in this work in order to contribute into building new algorithms for detecting malicious behavior in software systems. Therefore, GFI's vision was to evaluate the dangerousness of a behavior by comparing it to a *nominal behavior* or a reference behavior. GFI motivated the use of Graphical Models to represent behavior in order to avoid any black-box solution and have full control of the process at all times. In this thesis, we present the approach and formalisms that were used in this context.

In order to build a secure access to data in a real world system and to ensure its safeness from any upcoming potential threat one should learn the dependencies and behavior of the different components of the system, identify malicious ones and act at the right moment to intercept them. Thus, we can start to see the importance of model-based Machine Learning that allows to build a model of the concerned system based on data or some prior knowledge about the system. Many types of modeling formalisms exist in the literature, each developed for its own purpose. Some are better tailored for the verification of given properties or hypothesis, others for learning behaviors and dependencies; in this work we are interested in all of these aspects. Probabilistic finite automaton, for example, were used in modeling and verification of known or desired behaviors [Mao+11; SL94; Rab63]. Petri Nets [Pet77; Pet81] were used in modeling and verification of several parallel tasks as well as in Process Mining [Van14]. Hidden Markov models were widely used in speech and image recognition, as well as in evaluating the quality of discovered processes [RVV08; KA98; VM98]. Probabilistic graphical

1. Website: gfi.world

models were used for machine learning and the representation of dependencies between the different variables of a system [MR02].

Each of these formalisms has its own advantages and disadvantages. Nonetheless, all of the formalisms cited above and the ones that are in the same family have an insufficiency with regards to our main objective: the discretisation of time. Instead, we adopt a continuous time approach in this work since it represents some advantages that will be discussed in the following, from a security point of view. The discretisation of time can be described as a representational bias in the learning of these formalisms. Indeed, every two consecutive events, there is a time step but there is no quantitative measure of time that shows the actual elapsed time between events, the delay before any event occurs in the beginning of a process nor the delay at the end of the process. Thus, from a security point of view, it is better to use continuous time modeling formalisms that allow knowing more precisely when to act and not only what action to take; for example when predicting a system failure or forecasting future user tendencies.

To explore the dynamics of a wide variety of systems behavior based on collected event streams, there exist many advanced continuous time modeling formalisms: for instance, continuous time Bayesian networks [NSK02], Markov jump processes [RT13], Poisson networks [RGH05] and graphical event models (GEMs) [GMX11]. In this work we are particularly interested in *Recursive Timescale Graphical Models* (RTGEMs) [GM16] a sub-family of GEMs, that present advantages compared to the other formalisms. In particular, they are designed to universally approximate any smooth, non-explosive, stationary, multivariate temporal marked point process [DV07].

Appropriate learning and verification techniques should be adapted for the type of formalism that we wish to construct. Standard model checking, for example, is used as a verification method [BK08]. It has been applied to many formalisms, but to the best of our knowledge, never adapted to RTGEMs. Another valid solution for verification are approximation methods, such as Statistical Model Checking (SMC) [LDB10], which is an efficient technique based on simulations and statistical results. SMC has been successfully applied to graphical probabilistic models such as dynamic Bayesian networks (DBNs) in [Lan]. Therefore, SMC could be adapted to RTGEMs. When we consider

simulation-based techniques (such as SMC), one could think of using them directly on the original data and not on the sampled data from a learned model. Nonetheless in some cases, in order to perform precise formal verification (in particular for SMC) we need to artificially generate more *traces* of data that typically represent the same "source" of the original data. Moreover, the real data we collect may contain rare events that we are interested in analyzing, thus possibly the data is hiding different scenarios. As a consequence, it is important to learn a probabilistic model and sample data from it, in order to apply simulation-based techniques like SMC for verification and to avoid ignoring (completely or partially) certain scenarios. The latter is achieved by using techniques like *importance sampling* that can be used on a model (not the original data) to increase the frequency of appearance of rare events. However, another concern emerges from a model-based approach, that is the model's *quality* with regards to the original data. As a consequence, we rely on a modeling formalism whose quality (with regards to the input data) can be measured.

The main objectives of this work are first to build links between the graphical model-based learning field and the formal verification field, in order to benefit from the advantages of both for security assessments. The second objective is to seek a model that is at the same time representative of the input data and safe from a security point of view. We are not only interested in evaluating the fitness of the model using standard scoring techniques but also in its suitability from a security point of view. In particular, it is likely that the learned model does not satisfy given security properties. Hence, we propose a strategy where we choose to learn the "fittest" RTGEM (the one that is the most representative of the data) while controlling its complexity by penalizing it. If our security standards are not verified on the learned model, we also propose a search methodology to find another *close* model that satisfies them. To do so, an appropriate model-based strategy is proposed and a distance measure is introduced in order to compare two RTGEMs. By comparing the "fittest" model and the one found in its neighborhood (if one exists) that satisfies the security standard, we are giving an insight about the dangerousness of the detected (learned) behavior. Our approach is generic with respect to the verification procedure and the notion of distance between models. For the sake of completeness, the strategy we propose is then tested on synthetic data. The outline of this thesis is given below.

Thesis Outline

This thesis is composed of four chapters, each one is summarized in what follows. The first chapter contains state of the art formalisms and techniques that inspired this work. The second chapter contains necessary preliminaries for this thesis and the last two chapters consist of the main contribution of this work.

In the first chapter, we establish a review on the state of the art about commonly used discrete time formalisms. In particular we introduce, transition systems, Markov models, Dynamic Bayesian Networks, Petri Nets and Probabilistic Automata. We formally recall the definition of each presented formalism, and discuss its advantages and disadvantages. We also present a synthesis about the expressive power, the evaluation and the verification of each presented formalism. By proceeding as such, we notice that the discretisation of time is actually a problem from a security point of view, and that we should adopt a continuous time formalism in order to attain our objectives. Furthermore, we formally define the type of properties that are later on used for defining the security properties mentioned above. Finally, we discuss two formal verification techniques: Model Checking and Statistical Model Checking, as well as their advantages and disadvantages. We also justify the choice of Statistical Model Checking as the verification technique used in the rest of this work.

In the second chapter, we introduce basic prerequisites, definitions and notations that are used in the rest of the work. We present and formally define marked point processes and conditional intensity models. Most importantly, we describe the type of data considered for this study. Furthermore, we introduce and formally define the different families of Graphical Event Models, their learning procedures and limitations. The main objective of this chapter is to justify the choice of Recursive Timescale Graphical Event Models as a formal modeling technique, by balancing out their benefits and drawbacks, and demonstrating their easy manipulability.

In the third chapter, we build the foundation of our contribution. We explicitly state the problem and we formalize it. We propose a solution to the problem via a generic model-based strategy presented in the form of an algorithm. We detail the strategy that is proposed for learning and verification for security assessments. We also define a

distance measure between graphical event models . Our strategy consists in learning the model that is the most representative of the underlying system. It also consists in checking if the learned model satisfies a given security property. If not, a search for a *close* model that verifies the given security property is performed. Finally, we adapt a distance measure that is computed between the two models to see how far the *fittest* model is from verifying the property.

In the last chapter, we build tests to evaluate the performance of each step in the proposed algorithm. We show the performances of the learning and sampling of RTGEMs, as well as the application of SMC on this formalism. Furthermore, we build a pipeline of experiments in order to show that what we are proposing actually fulfills the starting goals that we have set. We provide evidence that if we have data coming from a secure system, the learned RTGEM will also be secure with regards of the security property that we propose, and vice versa. We experimentally show that the neighborhood exploration technique in order to find a *close* model that satisfies the property (in case the learned model does not satisfy it) actually works. Finally, we show how in practice, we can evaluate the dangerousness of a learned model in comparison to a nominal behavior.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we introduce Model Checking and Statistical Model Checking (SMC) as formal model based verification techniques, in order to give an insight about how one can formally verify certain properties (or queries) on models and what the limitations are.

Also in this chapter, we present the state of the art concerning some discrete time probabilistic and graphical probabilistic formalisms, as well as their functionality and expressive power. This will help in understanding the importance of model based machine learning and the variety of models that can be learned. In addition, we describe verification methods related to each formalism as well as some model evaluation techniques. We give a brief illustration of model based learning in Figure 1.1, in which we display possible prerequisites, e.g. assumptions, data, prior knowledge and others, that could take part in model learning . In Figure 1.1, we also show a possible process of improving the model, with regards to certain criteria (that are also discussed in this chapter), before reaching the target model that best answers the initial prerequisites.

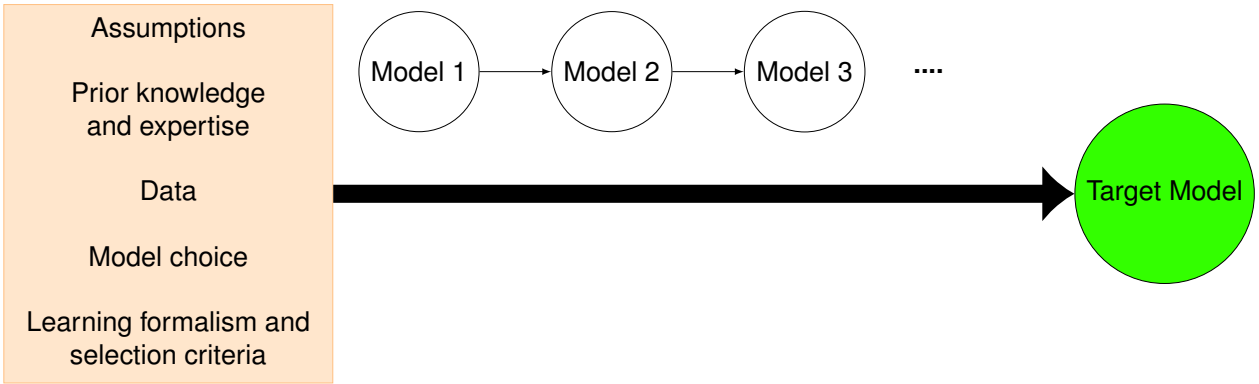


Figure 1.1 – Example of a model based learning process.

In what follows, we start by defining a family of models called transition systems in

order to introduce the concept of a model and its formal definition. We also define the concepts of atomic propositions and traces, which are important prerequisites to this chapter.

1.1 A preliminary model definition

A transition system is usually represented as a directed graph, with states as vertices and transitions as edges.

Definition 1.1.1 *[BK08] A transition system $\mathcal{S}$ is a tuple (S, I, R) such that*

- *S is a set of states;*
- *$I \subseteq S$ is a set of initial states;*
- *$R \subseteq S \times S$ is a transition relation.*

A transition system is finite if S is finite and infinite otherwise. An execution of a transition system is a sequence of states of the form $s_0, s_1, ...$ such that $\forall i, (s_i, s_{i+1}) \in R$. An execution can be finite or infinite. It is finite when we stop at a certain state s_n with n the number of steps (or transitions), and infinite otherwise.

In practice, transition systems are equipped with atomic propositions AP (usually a set of labels) that are associated with each state by the following relation: $V : S \to 2^{AP}$. In other words, each state of the system carries the characteristics of the associated atomic proposition(s).

Definition 1.1.2 *A trace of a transition system $\mathcal{S}$ defined over a set of atomic propositions AP, is the projection on AP of an execution of the transition system. A trace is finite if the corresponding execution is finite and infinite otherwise. A finite trace is formally written as the sequence $a_0, a_1 ..., a_n$, with $a_i \in 2^{AP}$ such that there exists an execution $s_0, ..., s_n$ with $\forall 0 \leq i \leq n, V(s_i) = a_i$, where i represents a step in the trace and n the total number of steps.*

The concepts introduced above are illustrated in the following example.

Example 1.1.1 *In Figure 1.2, an example of a Kripke structure [GCP99] is shown, which is a labeled transition system $K = (S, I, R, V)$ defined over $AP = \{q, p\}$, a set of atomic propositions, as follows*

- $S = \{s_1, s_2, s_3\}$ *is a finite set of states;*
- $I = \{s_1\}$ *is a finite set of initial states;*
- $R = \{(s_1, s_2), (s_2, s_1), (s_2, s_3), (s_3, s_3)\}$ *is the transition relation;*
- $V = \{(s_1, \{p, q\}), (s_2, \{q\}), (s_3, \{p\})\}$ *is the function associating each state to a set of atomic propositions.*

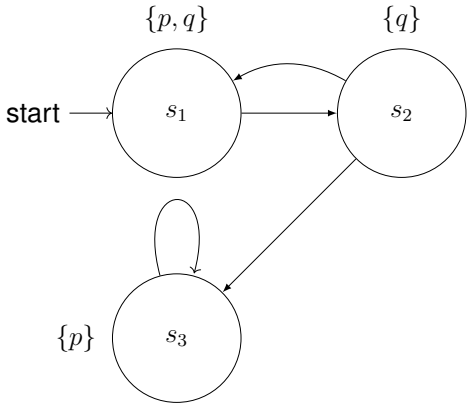


Figure 1.2 – An example of a Kripke structure transition system.

In order to generate a trace from this transition system, we consider a possible finite execution: $s_1, s_2, s_1, s_2, s_3, s_3$ *which will correspondingly generate a trace, that is in this case a succession of the atomic propositions:* $\{p, q\}, \{q\}, \{p, q\}, \{q\}, \{p\}, \{p\}$.

With this in mind, we introduce Model Checking along with properties specifications and Statistical Model Checking in the next section.

1.2 Model Checking

Given a finite-state model of a system and a formal property, Model Checking is a technique that systematically checks whether this property holds for all behaviors of that model [BK08]. Model checking is a "brute-force" method that analyses the entire space of possible scenarios in regard to a certain query (or property); the form and type of these queries will be elaborated later on. In other words, assuming that the model is already created and the property to be checked is already formalized, a model checker (the unit that performs model checking) checks if all possible behaviors of the model satisfy the property.

Depending on the nature of the property, when one stumbles upon a behavior that does not verify the property, this information is used as counter example of why the property is not satisfied, thus providing diagnostic information. Furthermore, many advantages of Model Checking can be cited, such as the wide range of real life applications (that can be modeled) on which it can be applied without a lot of prior knowledge or expertise.

Many limitations or weaknesses of Model Checking can be identified, such as scalability issues with large models in addition to the fact that it is more appropriate for control applications. Thus, Model Checking is less suitable for any data driven system that cannot be easily modeled [AK86].

In the following, we define formal properties that can be verified using Model Checking and we show some examples.

1.2.1 Properties specifications

In this section, we introduce Linear Temporal Logic (LTL) as well as an illustrative example. Temporal logic is the set of any rule (or logic) reasoning about propositions in time, i.e. anything related to modal logics [Che80] combined with time. For example, the sentences "I am always late for work" or "I will eventually be late for work some day this week" are based on temporal logic. Linear Temporal Logic (LTL) in particular, introduced by Pnueli in [Pnu77], is a formalism for trace properties. This formalism is tailored to deal with reactive systems, whose correctness depends on their execution.

Atomic propositions (AP) represent a set of characteristics proper to each state of the model, on which one can build the syntax of LTL formulae. AP are the basic ingredients for LTL formulae, along with the Boolean connectors conjunction $\wedge$, negation $\neg$ and basic temporal connectors next $\bigcirc$ and until $\bigcup$. Formula $\bigcirc\varphi$ holds at the current step in a trace, if φ is satisfied in the next step. Formula $\varphi_1 \bigcup \varphi_2$ is satisfied at the current step in a trace, if φ_1 holds at all steps until φ_2 holds in a future step.

Formally, the syntax of LTL formulae is defined as follows:

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \bigcup \varphi_2$$

where $a \in AP$. From this syntax, one can obtain the full power of propositional logic by deriving other connectors, especially the disjunction $\vee$, the implication $\longrightarrow$ and the equivalence $\longleftrightarrow$ operators. Moreover, we define the eventually operator $\Diamond$ describing a property that will eventually hold in a trace and that is formally defined as: $\Diamond\varphi ::= true \bigcup \varphi$; and the always operator $\Box$ describing a property that holds globally in every step of a trace and that is formally defined as: $\Box\varphi ::= \neg\Diamond\neg\varphi$.

For a better understanding of the LTL syntax, we use the example below.

Example 1.2.1 (example 1.0.1 revisited) *Consider the Kripke structure from Example 1.0.1 given in Figure 1.2, let $\varphi_1 = \bigcirc q$ be the property to verify on the model. The property holds on every trace generated by the transition system, because the transition system always starts in s_1 and then transitions to s_2 (whose label is $\{q\}$) in the following step. The same applies for the property $\varphi_2 = \Box\Diamond p$, because p is occurring infinitely often in any generated infinite trace (thus its negation $\neg\Box\Diamond p$ is always false). In addition, the property $\varphi_3 = \varphi_2 \wedge \varphi_1$ is also always true. However, a property of the form $\varphi_4 = \Diamond\Box p$ is false in some generated traces, for instance the trace: $\{p,q\},\{q\},\{p,q\},\{q\},\{p,q\},\{q\},\{p,q\},\{q\}...$, where we don't reach the state s_3 in the execution.*

It is proven that the LTL-Model Checking problem is PSPACE complete [BK08] since we need to explore all the possible scenarios until we find a counter example (or not). Because it requires a complete representation of the model, this process suffers from state-space explosion, which represents the main limitation of LTL-Model Checking.

In the following, we briefly review other temporal logics as well as another type of formal verification: Statistical Model Checking (SMC).

1.2.2 Other temporal logics

Extensions and other families of Temporal logics exist in the literature, in particular LTL with past operators [Gab+80; LPZ85] and bounded LTL [Bie+03], both of which will

be combined and used later on, in the purpose of defining a new type of properties adequate to a certain family of models.

Other examples include the Computation Tree Logic (CTL) [CE81], a formalism used for expressing properties over computation trees (a view of branches of time) and an extension focusing on *qualitative* properties: Probabilistic CTL proposed in [HJ94]. Qualitative properties are suited for probabilistic models, they will be detailed in Chapter 3.

For now, we have introduced some ingredients and recipes for formal verification, that can be used in "classical" Model Checking. In the following, we introduce Statistical Model Checking (SMC).

1.2.3 Statistical Model Checking

We recall that Model Checking suffers from complexity issues when the models are large. Thus, the need for alternative formal verification techniques that scale up have emerged. An approximate yet effective simulation-based technique named Statistical Model Checking (SMC) is presented in what follows. One should note that SMC does not deal with the same problem as Model Checking: in order to avoid the scalability issue, SMC relies on simulations which only allows to *estimate* the probability of satisfaction for a property on the concerned model; when Model Checking instead computes the exact value.

Compared to Model Checking, Statistical Model Checking is easier to apply on certain models. SMC's complexity depends on the size of the property to be verified rather than the size of the model [LDB10]. Indeed, the core of SMC consists in executing simulations whose length depend on the size of the property.This is mainly why we use it on complex systems (complex models). This approach, applied on stochastic models (that will be introduced in the next section), does not compute the exact probability of satisfying a given property, but rather estimates this probability. At the same time SMC benefits from many advantages :

- Its application only requires that the system (model) is executable, i.e. that we can sample from it even if it is a very large model or a black box [You05a] (one should note that sampling can also be hard sometimes).

- It is applicable using properties that cannot be verified with classical Model Checking [CDL08]. For instance, in [LDB10] three cases of properties (the nested, the unbounded and the boolean combinations cases) are introduced whose satisfaction cannot be decided except by an estimation technique (SMC for instance).
- It is parallelizable, different independent samples can be retrieved at the same time for a faster execution.

For a detailed overview about the different aspects of this technique one can refer to [LDB10].

However, one should be aware of the limitations of SMC. As mentioned before, it is an approximation technique and can only provide probabilistic guarantees about the correctness of the algorithmic result. Furthermore, SMC relies on simulations so it is mostly restricted to linear properties (i.e. trace properties), expressed in LTL for instance. In addition, it only works on systems without any form of nondeterminism (purely probabilistic) for the samples to be faithful. Finally, keeping in mind that we are interested in model based model checking (particularly SMC), the main obstacle that we confront in practice is that for an accurate result the sample size must be huge, which can be very costly when handling certain types of models.

Let $\mathcal{S}$ be a stochastic system (regardless if it can be modeled using a certain formalism or not) and φ a property that needs to be verified. Statistical Model Checking consists in performing a series of simulation-based techniques that can answer two kinds of questions. The first kind are **qualitative** questions of the type: Is the probability that $\mathcal{S}$ satisfies φ greater or equal to a certain threshold? The second kind are **quantitative** questions of the type: What is the probability that $\mathcal{S}$ satisfies φ? In both approaches, the answer is given up to some correctness precision [Bas+10].

As previously mentioned, SMC is based on simulations, and each simulation is represented by a trace that has a binary outcome for satisfying the property: Yes or No. Therefore, we consider a discrete random variable $\mathcal{B}_i$ with a Bernoulli distribution of parameter p, that is associated to each trace and can take two values: 0 (if the property is not satisfied) and 1 (if the property is satisfied). Assume that $Pr[\mathcal{B}_i = 1] = p$ is the probability of satisfying the property and $Pr[\mathcal{B}_i = 0] = 1-p$ is the probability of violating

the property, SMC allows to either estimate p or decide if p is greater (or smaller) than a fixed value.

Qualitative approach. The original qualitative approaches are proposed in [You05b; SVA04], and are based on hypothesis testing. The idea is to test two hypothesis, $H_0 : p \leq \theta$ against $H_1 : p > \theta$, while bounding the probability of making an error. A Type-I error is when we accept H_0 while H_1 holds and a Type-II error is when we accept H_1 while H_0 holds. Therefore, we define two parameters α and β, with (α, β) the *"strength"* of the test, being the pair of bounding errors . An ideal performance of the test is when the Type-I error is equal to α and the Type-II error is equal to β, and both α and β are small. However, in practice it is impossible to ensure a low α and β simultaneously. In order to avoid this problem, an indifference region $[p_0, p_1]$ is defined. Let $p = Pr(\varphi)$ the probability of φ being satisfied on the system $\mathcal{S}$. In order to determine whether $p > \theta$ (qualitative property, θ being in $[p_0, p_1]$), one can test $H'_0 : p \leq p_0$ (instead of $H_0 : p \leq \theta$) against $H'_1 : p > p_1$ (instead of $H_1 : p > \theta$. However, the tuning of the indifference region in practice is constrained by the loss of precision regarding the estimation (when the size of the region increases) and by the fact that we cannot conclude about the result if p is inside $[p_0, p_1]$. Two hypothesis testing algorithms will be presented, Single Sampling Plan (SSP) and Sequential Probability Ratio Test (SPRT).

Single Sampling Plan consists in specifying a constant c and a number of simulations n to test whether $\sum_{i=1}^{n} b_i > c$, where b_i is the outcome of the Bernoulli random variable, in order to see which hypothesis is accepted. Thus, the hardest part in this algorithm is to compute values for the pair (n, c) with respect to the pair (α, β) and the indifference region. The number n increases when we minimize the size of the indifference region and the parameters α and β. An optimization algorithm was proposed in [You05b] in order to determine a pair (n, c) where n is minimal.

Sequential Probability Ratio Test is an approach proposed by Wald [Wal45], consisting in choosing two values A and B ($A > B$) ensuring the strength of the test and that are computed using α, β and the indifference region. This approach takes into account the observations made so far and tests whether the test result may change if we continue the sampling; if we are sure that it does not change then we can stop sampling. If m is the number of already performed observations, the test is based on the following

metric :

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i | p = p_1)}{Pr(B_i = b_i | p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}$$

with p_0, p_1, B_i as defined above, p the variable that is used to indicate which hypothesis is accepted, and $d_m = \sum_{i=1}^{m} b_i$. It is shown in [You05b] that we can accept H_1 after m samples if $\frac{p_{1m}}{p_{0m}} \geq A$ and H_0 if $\frac{p_{1m}}{p_{0m}} \leq B$. Using this approach we can reduce the number of simulations in certain scenarios compared to the Single Sampling Plan approach (where the number of samples is fixed to n), and avoid doing $n - m$ useless samples. However, when the value of the ratio $\frac{p_{1m}}{p_{0m}}$ is always varying inside the indifference region ($[A, B]$) it is very hard to converge using the Sequential Probability Ratio Test, making the first approach more advantageous.

Quantitative approach. The purpose of this type of SMC is to compute the probability p (or an estimate) for $\mathcal{S}$ to satisfy φ. In the works of Peyronnet et al. [Hér+04; Lap+07], a procedure based on the Chernoff-Hoeffding bound [Hoe94], was presented in which an estimate p' is calculated, given a precision δ, such that $|p' - p| \leq \delta$ and $Pr(|p' - p| > \delta) < \varepsilon$ with ε the probability of making an error. As a consequence, the result $p' \geq \theta + \delta$ means that the system $\mathcal{S}$ satisfies φ with a probability higher than θ and with a confidence of $1 - \varepsilon$. This has been applied in several works using different simulation techniques, in particular Monte Carlo simulations and estimates, e.g. in [JSD17].

In the next section, we present some probabilistic formalisms with discrete times as well as their specifications and proper use.

1.3 Probabilistic Graphical Formalisms with Discrete Time

In this section, we introduce four state of the art formalisms as different definitions of what we have already called a transition system. Graphical probabilistic models will be concisely presented, as well as their respective expressiveness, evaluation and verification. The objective of this section is to show that the existing modeling formalisms are means to different ends, i.e. a formalism exists because it is meant to be used in a certain manner, in certain applications and that a universal formalism has not been

found yet, to the best of our knowledge.

1.3.1 Discrete Time Markov Models

In this subsection, a family of formalisms obeying to the *Markovian* property are introduced, and the latter is also explained. We begin with one of the most basic yet very used models, discrete time Markov Chains (MC for simplicity) [KS76]. This type of Markov models is defined below and an example follows.

Definition 1.3.1 *A Markov Chain (MC) is a tuple* (Q, q_0, M)*, where*

- Q *is a set of states;*
- $q_0 \in Q$ *is an initial state;*
- $M : (Q \times Q) \longrightarrow [0, 1]$ *is a transition matrix, with:* $\forall q_1 \in Q, \sum_{q_2 \in Q} M(q_1, q_2) = 1$*.*

This model is a stochastic process that evolves through time, changing from a state to another in a probabilistic way. A Markov chain is finite if Q is finite and infinite otherwise. An example is shown in Figure 1.3, where we illustrate a finite MC with three states and six transitions.

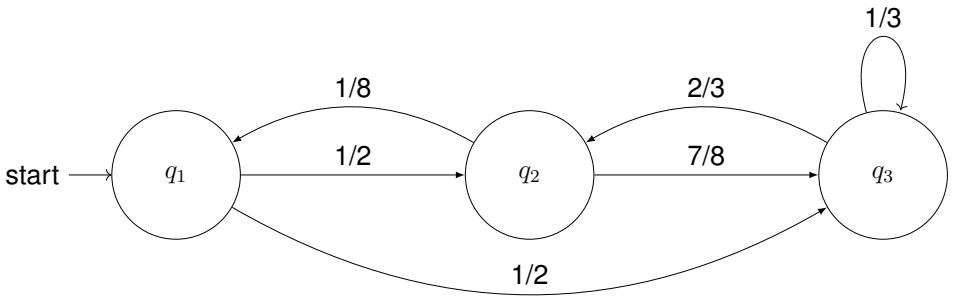


Figure 1.3 – An example of an MC with 3 states.

Expressive power. The expressive power of such models resides in their ability to display that every step along the way only depends on the step that came right before it. In other words, knowing the present state, one does not need the past to predict an outcome. We can easily compute the probability of having a certain result (or being in a certain state) after i steps [BLN04]. Furthermore, there exist higher order Markov Chains [CFN04], aiming to add more memory to the process by extending the dependencies to previous steps also (not only the step that came right before). Markov

Chains are essential to the understanding of random processes, because one can explicitly compute many quantities of interest [Nor98]. Mathematically, if X is the random variable governing the random process represented in the MC, then the probability of X_i (the ith outcome of X) with regards to the Markovian property (order 1 in this case) can be written as follows:

$$P(X_i = x_i) = P(X_i = x_i | X_{i-1} = x_{i-1}, ..., X_0 = x_0) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

In addition, Markov Chains were used in simulation [Bré13], medicine [LD01], economics [CR01] and many other fields that handle random processes, due to their power of explicitly showing the different probabilities and paths of reaching or avoiding an objective (state) starting from a certain point. However, this modeling formalism is confronted with a big limitation: the size of a Markov model can quickly increase in order to model large systems (or random processes) making it hard to compute and handle. This is what researchers call the state-space explosion problem.

In this thesis, the focus is on behaviors and security assessments and as mentioned earlier, this type of model can be used for describing a series of transitions before reaching a malicious state. As a consequence one can maybe predict the probability of reaching a malicious state in future steps. However, from an expressiveness point of view, it cannot describe the event (the action) leading to a malicious state or the events describing safe behavior in the model.
Therefore, in order to have a clearer representation and a better use of the expressiveness of such formalism, many derivations or subfamilies were defined such as Labeled Markov Chains (LMC) [Des+04] and Hidden Markov Models (HMM) [Edd96].

Labeled Markov Chains (LMC) are the first type of subfamilies, where a set of labeled actions is defined and added to each transition in order to enhance the descriptive power of an MC. In the following, we recall the definition of an LMC and we show an example.

Definition 1.3.2 *A Labeled Markov Chain (LMC) is a tuple* (Q, q_0, M, A, F)*, where*

- Q *is a set of states;*
- $q_0 \in Q$ *is an initial state;*
- $M : (Q \times Q) \longrightarrow [0, 1]$ *is a transition matrix, with:* $\forall q_1 \in Q, \sum_{q_2 \in Q} M(q_1, q_2) = 1$*;*

- A *is a set of actions;*
- $F : Q \times Q \longrightarrow A \cup \{\perp\}$ *is a function that assigns an action to each transition, such that* $F(q, q') \neq \perp$ *iff* $M(q, q') > 0$*, with* $\perp$ *a symbol corresponding to the absence of transitions.*

This formalism grants more information than regular Markov chains, and can be used for describing a series of actions (finite or infinite) and their consequences. This extension could be used for behavior analytics in the service of security for instance. In Figure 1.4, an example of an LMC with three states, six transitions and three actions is given.

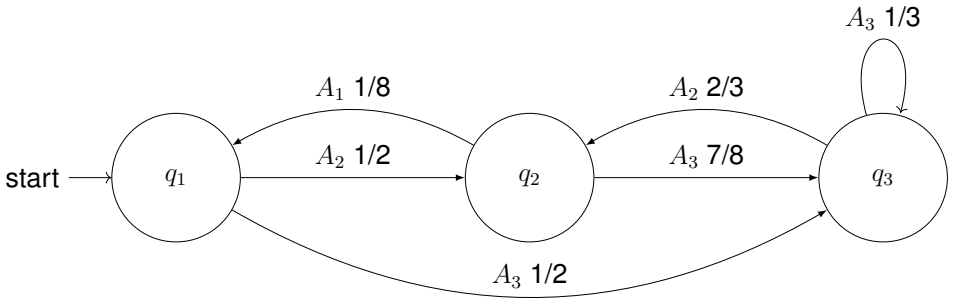


Figure 1.4 – An example of an LMC with 3 states and 3 actions.

Another type of subfamilies is Hidden Markov Models (HMM), that is represented as a sequence of "hidden" states that are interconnected by state transition probabilities. The main advantage about this formalism is that the same execution (over states) can correspond to several observation sequences (traces). In other words, each state has a symbol-emission probability distribution and can emit a number of observable symbols (or observations). An HMM is defined below, and an example follows.

Definition 1.3.3 *A Hidden Markov Model (HMM) is a tuple* (Q, π, M, O, B)*, where*

- Q *is a set of states;*
- $\pi : Q \longrightarrow [0, 1]$ *is an initial probability distribution,* $\sum_{q \in Q} \pi(q) = 1$*;*
- $M : (Q \times Q) \longrightarrow [0, 1]$ *is a transition matrix, with :* $\forall q_1 \in Q, \sum_{q_2 \in Q} M(q_1, q_2) = 1$*;*
- O *is a set of observations;*
- $B : (Q \times O) \longrightarrow [0, 1]$ *is a function associating the observations probability distribution to each state, such that :* $\forall q \in Q, \sum_{o \in O} B(q, o) = 1$*.*

In Figure 1.5, an example of an HMM with three states and two possible observations is shown. Each state has to be associated with one observation (y_1 or y_2 in this case) according to the probability distribution.

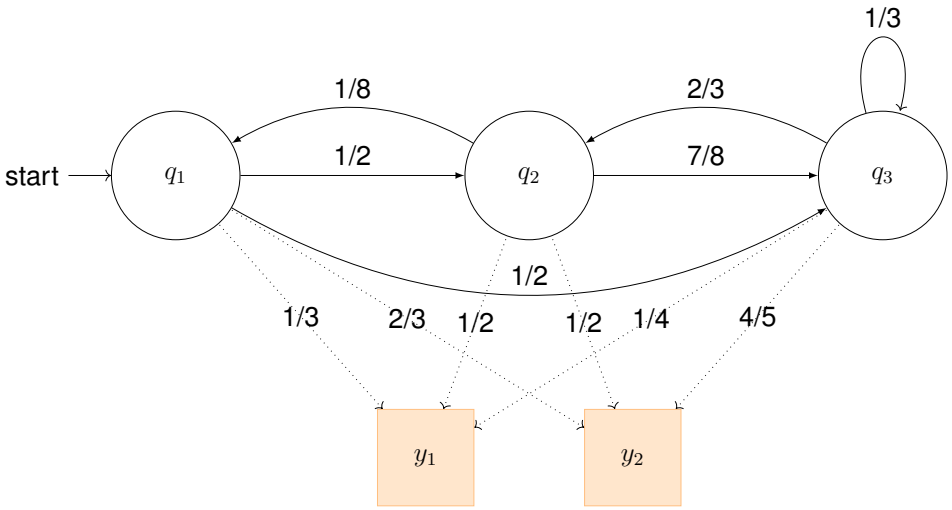


Figure 1.5 – An example of a Hidden Markov Model with 3 states and 2 observations.

A description of HMM theory has been written by Rabiner [Rab89]. HMMs have been widely used in the domains of speech recognition and more generically as a general statistical modeling technique for linear problems in computational sequence analysis [SWS93]. For instance it was applied in [WSS94] to the field of evolution in protein structural modeling.

Example 1.3.1 *In Figure 1.6, we show a possible execution of the HMM illustrated in Figure 1.5, showing the sequence of hidden states and their corresponding observations.*

Remark that, for the same execution q_1, q_2, q_3, q_2, q_1 *we could have had a different observation sequence (trace), for instance* y_2, y_2, y_2, y_1, y_1 *with probability* $2/45$ *or* y_1, y_1, y_1, y_1, y_2 *with probability* $1/72$ *instead of* y_2, y_1, y_2, y_1, y_2 *whose probability is* $4/45$ *.*

For now, the characteristics of some Markov models have been discussed. However, in this framework, the evaluation of a model in regards to certain criteria (that will be defined later on) is also very important, as well as the formal verification of a model as already seen in the first section. The evaluation of a model is addressed in this context as the measuring of the quality of a learned model compared to other models or to the real world process that it represents. In contrast, formal verification

is the process of checking whether the model satisfies a set of properties that are not necessarily related to its quality. In the following, we overview the evaluation and the formal verification of the presented models.

Evaluation. The evaluation of the Discrete Time Markov Models presented above is about measuring the quality of a Markov Chain (respectively an LMC) with respect to the real random process. In other words, the evaluation of such models consists in comparing the model and the reality in order to have an insight about the representational quality of the model. Therefore, to measure the conformity of the MC with the actual process, one can simulate data from the MC (which can be performed in linear time using a naive algorithm) and compare it with the real world data. Otherwise, one can calculate the likelihood of the real data knowing the MC, i.e. how much is it likely that this data came from this MC. In consequence, the crucial need about evaluating the quality of an MC (or LMC) is prior knowledge about the represented process (data, expertise etc.). The same techniques apply for the evaluation of HMMs.

Formal verification. The formal verification of a Discrete Time MC (respectively a Discrete Time LMC) has been studied extensively. For instance, a tool was developed in [KNP11] for automatic verification and another tool was also developed in the context of reliability modeling of various systems [Her+03]. In practice, any kind of adequate formal verification technique like classical Model Checking, Probabilistic Model Checking or Statistical Model Checking can be used with these models without major constraints. The same also applies for HMMs but with a slight constraint: one should be able to discern (if needed) states and transitions from observations while verifying

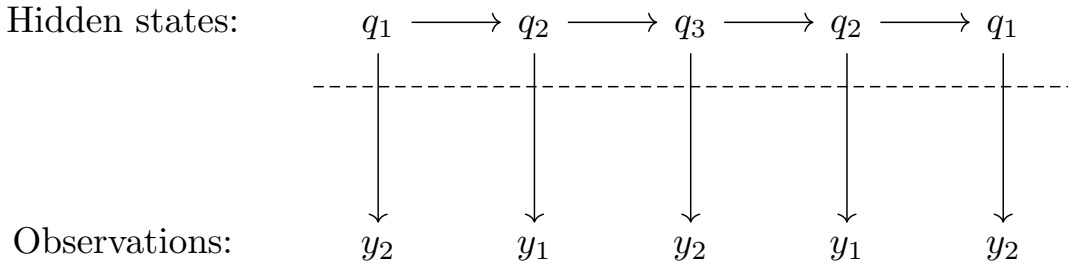


Figure 1.6 – An example of an execution of the Hidden Markov Model of Figure 1.5.

the model.

In the following, we present another type of discrete time formalisms called Dynamic Bayesian Networks (DBN), an extension of Bayesian Networks (BN) that are useful for a wide range of problems.

1.3.2 Bayesian Networks and Dynamic Bayesian Networks

Bayesian networks [Cha91; Nea+04] are a family of probabilistic models that are capable or concisely representing relationships between variables. Contrary to the previously introduced formalisms, Bayesian networks are static, thus the edges in their graphs do not represent transitions but rather dependencies. In the following, we recall the formal definition of a Bayesian Network and we show an illustration in Figure 1.7.

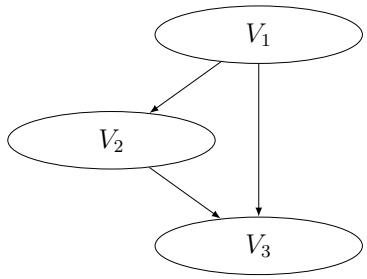
Definition 1.3.4 *[Cha91] A Bayesian Network (BN) is defined as a pair* (G, θ) *such that*

- $G = (V, E)$ *is a DAG (Directed Acyclic Graph) where* V *is a set of random variables* $(V_1, V_2, ..., V_n)$ *constituting the nodes of* G *and* E *is a set of edges.*
- $\theta = \{P(V_i \mid Pa(V_i))\}$ *is the set of conditional probability distributions of each node knowing its parents, with* $Pa(V_i)$ *the parents of* V_i *in the graph* G*.*

One can notice that a variable depends on its *Markov Blanket* (parents, children and other parents of the children) in the DAG. In other words, the *Markov blanket* of a node is the only information one needs to predict the behavior of that node and its children. For instance, knowing the values of the parents random variables is all the information one needs to compute the value of the child's random variable. In addition, the "root" nodes without any parent, are independent random variables with already known distributions.

In Figure 1.7, we can see a BN with three random variables and their corresponding conditional probability distributions. In this example each of the random variables is Boolean and can only take two values, True or False, but in practice there are no restrictions on the number of values that a discrete random variable can take.

Any complete probabilistic model must, either explicitly or implicitly, represent the *joint distribution*, i.e. the probability of every possible event as defined by the values of all



	V_2	
V_1	T	F
F	0.4	0.6
T	0.01	0.99

V_1	
T	F
0.2	0.8

		V_3	
V_2	V_1	T	F
F	F	0.4	0.6
F	T	0.01	0.99
T	F	0.2	0.80
T	T	0.35	0.65

Figure 1.7 – An example of a Bayesian Network BN with 3 random variables.

the variables [Pea01]. Bayesian Networks achieve compactness by factoring the joint distribution into local conditional distributions for each variable given its parents. Furthermore, any joint distribution over a set of variables can be represented by a fully connected Bayesian Network [Cha91]. The general interpretation of Bayesian Networks states that the full joint distribution is written as:

$$P(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} P(V_i \mid Pa(V_i))$$

On a side note, data generated from a Bayesian network, or any other probabilistic graphical model with dependencies, is said to be *faithful* if it correctly replicates the independence properties of the model. Bayesian networks offer an appropriate way to address many artificial intelligence problems, in which one would like to come to conclusions probabilistically rather than only as logic properties. The *expressive power* of such models is that they concisely represent a lot of information, making it less prone to state-space explosion problems. For instance in the work of [Phi06] Bayesian Networks are used in the learning and modeling of complex systems. This matter will be discussed in the following. However, BNs do not provide direct means for representing temporal dependencies, this is why alternatives have been proposed.

With this in mind, we introduce, define and discuss Dynamic Bayesian Networks (DBNs), an extension of Bayesian Networks for explicitly representing random processes.

Definition 1.3.5 *The generic definition of a k time slice Dynamic Bayesian Network (DBN) with Markov order $k-1$ can be found in [MR02]. We are interested in the two time slice Dynamic Bayesian Network (Markov order 1) that is defined as $(B_1, B_{\longrightarrow})$, with B_1 a BN representing the initial network and $B_{\longrightarrow}$ also a BN with two time slices $t-1$ and t representing $P(V_t \mid V_{t-1})$ by a DAG, and such that:*

$$P(V_t \mid V_{t-1}) = \prod_{i=1}^{n} P(V_t^i \mid Pa(V_t^i))$$

with n the number of random variables, V_t^i the node V_i at time t and $Pa(V_t^i)$ the parents of V_i in the graph at times t and $t-1$ (while considering an order 1 Markovian model). Remark that this formalism describes discrete time stochastic processes and that the term "dynamic" only means that we are modeling a dynamic system, not that the network itself is evolving with time. Remark also that the Markovian property holds for Dynamic Bayesian Networks.

The joint probability distribution in this case is obtained on a sequence of length T, and is written as :

$$P(V_{1:T}) = \prod_{t=1}^{T} \prod_{i=1}^{n} P(V_t^i \mid Pa(V_t^i))$$

Expressive power. The expressive power of BNs is transferred to DBNs, therefore DBNs are capable of displaying a lot of information without being very complex. Besides, DBNs can be compared to HMMs. For instance, it is shown in [MR02] that HMMs can be represented using DBNs. More precisely, a DBN can represent an exponential number of HMM states concisely. Furthermore, the nodes in a DBN represent random variables, and each random variable can be observed or not. On the other hand, in an HMM the hidden states are always the same. The latter shows that from an observability point of view, DBNs offer more flexibility. Moreover, we show in Example 1.3.2 that a DBN can be converted into a Discrete Time Markov Chain. A state in a DBN is the combined observations (values) of all the random variables at a certain time t. Note that for the sake of simplicity in what follows, we write V_{i_t} for variable V_i at time t.

Example 1.3.2 *Consider two random variables V_1 and V_2 with a Boolean outcome 0 (False) or 1 (True) with different probabilities. In order to represent the evaluation of*

these variables with a DTMC, we need to create a state for each combination of pair of values (see Figure 1.8).

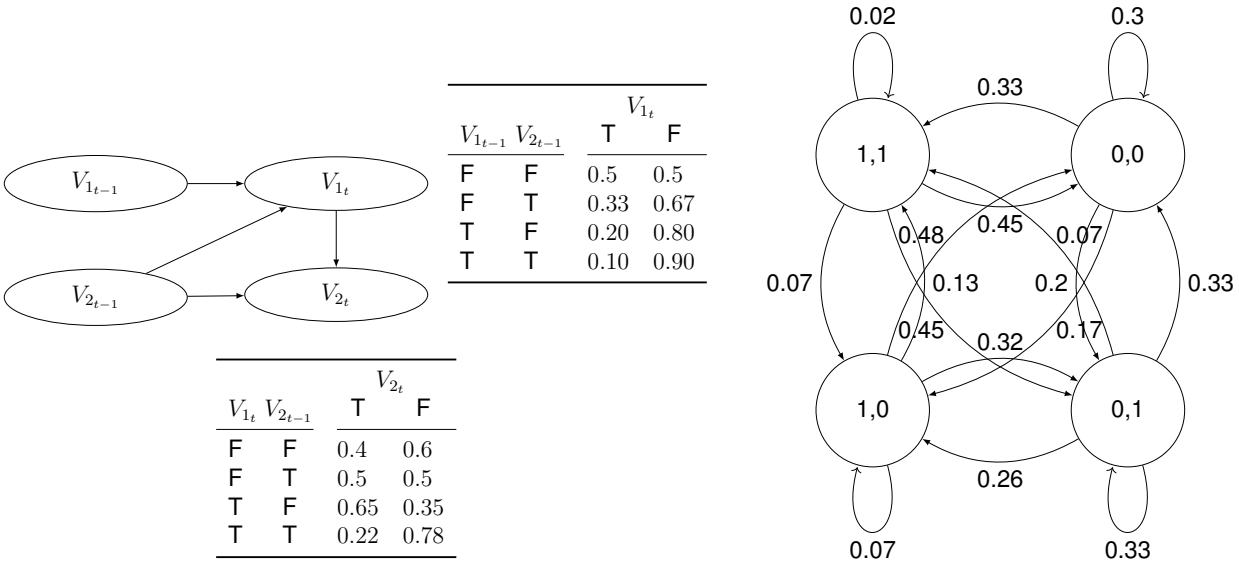


Figure 1.8 – An example showing the $B_{\longrightarrow}$ of a DBN (to the left) transformed into a DTMC (to the right). Note that in the states of the DTMC the value on the left is for V_1 and the one on the right for V_2.

The outcome of the random variable V_1 at time t depends on its own outcome at time $t-1$ as well as the outcome of V_2 at time $t-1$ (see Figure 1.8). However, the outcome of the random variable V_2 at time t depends on its own outcome at time $t-1$ as well as the outcome of V_1 at time t. The probabilities on the transitions are computed using the conditional probability distribution for each variable. For example, to compute the probability to go from state $(1,1)$ to state $(0,0)$ we multiply $Pr(V_{1_t} = F|V_{1_{t-1}} = T, V_{2_{t-1}} = T) = 0.9$ and $Pr(V_{2_t} = F|V_{1_t} = F, V_{2_{t-1}} = T) = 0.5$. We can notice that the size of the DTMC (number of states and transitions) grows exponentially with the number of variables in the DBN.

We add that the Bayesian Network B_1 (representing the initial state) of the DBN on the left can be used for computing the probabilities of each state in the DTMC to be an initial state. We can deduce that DBNs represents advantages over DTMCs in this case, where they can represent more concisely the same quantity of information. The objective from this example is to show that each formalism is created for certain purposes, but that does not mean that one is superior to another.

In practice, DBNs are traditionally used for answering probabilistic questions about

the state of a system at a certain step in a time sequence. Let Y be the set of observed variables ($Y_{1:t}$ are the observed variables ranging from 1 to t) and X the set of non observed ones. Given $l > 0$ and $h > 0$, the questions in practice are classified into different categories:

- Filtering: compute $P(X_t \mid Y_{1:t})$, i.e. the unobserved variables at time t using the observed variables at times ranging from 1 to t (all the observed information).
- Smoothing: compute $P(X_{t-l} \mid Y_{1:t})$, i.e. the unobserved variables at the previous time step $t-1$ using the observed variables at times ranging from 1 to t (all the observed information).
- Prediction: compute $P(X_{t+h} \mid Y_{1:t})$, i.e. the unobserved variables at an upcoming time step $t+h$ using the observed variables at times ranging from 1 to t (all the observed information).
- Decoding: compute $argmax P(X_{1:t} \mid Y_{1:t})$, i.e. the most likely values for all of the unobserved variables using all of the observed variables.
- Classification: compute $P(Y_{1:t})$, i.e. all the observed variables at times ranging from 1 to t.

The learning of BNs and DBNs can be divided into two parts, structure (or dependencies) learning and parameters learning [Gha97]. All learning techniques for these models are inspired by the expectation-maximization (EM) algorithm [DLR77]. In particular, EM is applied when there is missing data. Otherwise classical statistical estimation methods, e.g. Maximum Likelihood Estimate (MLE) or Maximum A Posteriori (MAP), are used for parameter learning.

Evaluation. The evaluation of such models, in practice, is commonly made while learning them. We identify three types of methods for structure learning: score-based, constraint-based and hybrid methods [DSA11]. Using a score-based method like a greedy search algorithm [Chi02] for instance, consists in searching for models and consistently picking either the "best", or a "better", model using an evaluation or a selection criterion like the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC) [Sch+78]. This procedure only stops when a model that cannot be enhanced is found (thus the name greedy). Furthermore, we can evaluate such models by comparing them between each others. To compare the structure there exist distance measures like the Structural Hamming Distance (SHD) [Ham50; Tra13; TBA06], and to compare the parameters there exist metrics like the Kullback-Leiber Divergence

(KLD) [VH14].

Formal verification. The verification of this type of models is very uncommon in the Model Checking community, in a way that classical and probabilistic Model Checking techniques are not adapted to them and almost never used, to the best of our knowledge. However, Statistical Model Checking (SMC) was applied for formal verification of certain properties on DBNs in [Lan], which was made possible by the fact that DBNs are models that we can sample and that are purely stochastic, two sufficient conditions for applying SMC.

To conclude this presentation of state of the art formalisms, two last families of models will be defined next, as well as their expressive power, evaluation and formal verification.

1.3.3 Other Formalisms

In this subsection, Probabilistic Automata and Petri Nets will be briefly introduced, henceforth completing this survey on discrete time graphical probabilistic formalisms.

Probabilistic Automata (PA). Probabilistic Automata [SL94; Rab63] are a family of transition systems that can allow non-determinism in addition to probabilities. In the following, we define a Probabilistic Automaton (PA) and illustrate it using an example. We write $Dist(Q)$ for the set of probability distributions on the set Q, i.e. the set of functions $\pi : Q \longrightarrow [0, 1]$ such that $\sum_{q \in Q} \pi(q) = 1$.

Definition 1.3.6 *A Probabilistic Automaton (PA) [SL94] is a transition system* $\mathcal{A} = (Q, \pi_0, A, P)$*, where*

- Q *is a set of states;*
- $\pi_0 \in Dist(Q)$ *is the initial probability distribution over the states;*
- A *is an alphabet of actions;*
- $P : Q \times A \longrightarrow Dist(Q)$ *is a probabilistic transition function.*

Note that, depending on the application, a probabilistic automaton can have accepting states, i.e. states in which the sequence of actions is allowed to end. A probabilistic automaton can also be equipped with a non deterministic probabilistic transition relation.

The example shown in Figure 1.9 illustrates a finite PA with two states and five transitions. We can notice that there exists non-deterministic behavior in the PA of Figure 1.9, due to the non-probabilistic choice that has to be made between transitions a and b, each leading to a different distribution. The non-deterministic decision is usually made externally, by a certain technique (like schedulers [BK08]), and it leads to the probability distributions for the following transitions. Subsequently, simulation-based techniques like SMC are not adequate for such models.

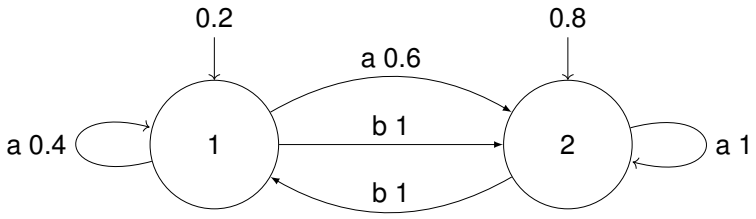


Figure 1.9 – An example of a Probabilistic Automaton with two states.

This formalism is commonly used in the domain of grammatical inference, where it is used for representing distributions over strings and is learned using different algorithms like ALERGIA [De 10]. However, this type of formalisms is not adequate for modeling behavior in our framework, because of additional techniques or parameters that should be used for preventing non-determinism or to transform non-deterministic uncertainties into pure probabilities.

Petri Nets (PN). Petri nets [Pet77; Pet81] are mainly used for describing concurrent systems, which means that by definition this formalism allow non-determinism. As a result, simulation based verification is out of the picture. Furthermore, in some cases it is hard to adapt classical Model Checking techniques to this type of models. In what follows, Petri Nets are formally defined and an example illustrating a Petri Net is shown in Figure 1.10.

Definition 1.3.7 *A Petri Net (PN) is a tuple* $R = (P, T, Pre, Post, M_0)$ *[Bra83], such that*

- P *is a finite (non empty) set of places;*
- T *is a finite (non empty) set of transitions such that* $T \cap P = \emptyset$*;*
- $Pre : T \times P \longrightarrow \mathbb{N}$ *is the pre-incidence (or input) function;*
- $Post : P \times T \longrightarrow \mathbb{N}$ *is the post-incidence (or output) function;*

- $M_0 \in \mathbb{N}^P$ *is the initial marking of the net.*

A state of the PN is defined by the application M, attributing a *marking* to each place, i.e. a number of tokens that are present in each place. $M(p)$ is equal to the number of tokens in the place $p \in P$. The initial marking M_0 is the marking with which we start the execution of the PN.

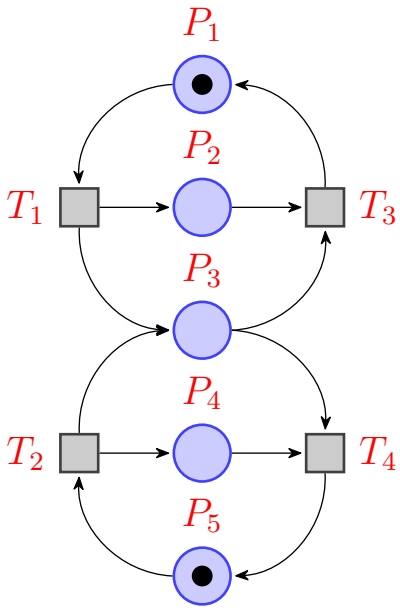


Figure 1.10 – An example of a Petri Net with 5 places and 4 transitions.

In Figure 1.10, a Petri Net with five places, four transitions and two tokens (one in P_1 and one in P_5) is illustrated. An execution of the PN consists in the successive firing of *allowed* transitions, i.e. transitions having enough tokens in the Pre corresponding places. When an allowed transition is fired, tokens are removed from the Pre corresponding places and others are placed in the $Post$ corresponding places. As a consequence, a **marking graph** can be generated from the PN and can represent the latter as a (potentially infinite) transition system. The evolution of this graph corresponds to the execution of the PN.

Many types of Petri Nets exist in the literature, like high-level Petri Nets, colored Petri Nets [JR12] and probabilistic extensions of Petri Nets (probabilistic Petri Nets [Alb+08] and stochastic Petri Nets [BK98]). This formalism and its derivatives are commonly used in Process Mining, learned by the *alpha* algorithm (and its extensions) [Van11]

in order to learn processes using a certain type of data; do diagnostics and detect problems; identify the sources of the problems; and finally recommend solutions for the problems. It could seem like a good formalism to use in behavior analytics and malicious event detection, however these models typically detect bottlenecks in a concurrent system and potential deadlocks in a process. Thus, the application of Petri Nets is mainly industrial and is more relevant to the Business Intelligence domain [Van11].

1.4 Discussion

After this concise review, one can clearly induce, by taking a glimpse on the state of the art presented in this chapter, that the first step in doing model based machine learning is choosing the appropriate modeling formalism. The same applies for formal verification. It is useless to learn a model that cannot be formally checked if this was the initial objective. Several verification techniques and learning formalisms have been discussed in this chapter and, before moving to the next one, we propose a brief synthesis.

So far, based on what has been presented, one should have an insight about the functionality and utility of some of the state of the art formalisms. Recall that one of the main objectives of this thesis is to bring closer the learning domain and the formal verification domain. Therefore, before introducing the preliminaries and the models that were directly used in this framework, we discuss similarities and inspirational works that we already have explored in the literature.

Links have been established between several formalisms for different applications, e.g. direct passages from HMMs to PAs were created in [DDE05], allowing to apply learnability results and induction algorithms developed in one formalism to the other. In addition, regardless of complexity issues, Petri Nets can be represented as (potentially infinite) transition systems, HMMs can be represented as DBNs and DBNs can be represented as Markov Chains (with the risk of state-space explosion). Besides, continuous time extensions exist to the previously cited formalisms, like continuous time Markov Chains for instance, that can be formally verified [Azi+96]; and continuous time Bayesian Networks [NSK02] (that are not commonly used due to their complexity) on which inferences could be done after transforming them into probabilistic timed automata [NSK02]. Additionally, Dynamic Bayesian Networks have been used for formal

verification using SMC in [Lan] which has been a major inspiration for the work in this thesis.

Other works on model optimization heuristics with regards to a certain objective and neighborhood search techniques were also notably inspiring for this work. For instance in the works of [Mis+15], a heuristic was used for minimizing the outcome values of observed variables in a hierarchical Bayesian model. Similarly, in [Lim+06; Lim+09] substructural neighborhoods for local searches in the Bayesian optimization algorithm were proposed. By exploring the literature, we had a better knowledge about the different domains. Furthermore, this exploration gives a better intuition on how to dissect the problematic into several smaller problems in order to target them and resolve them one by one.

To sum up, all of the discussed models in this chapter have multiple functionalities and are widely used in the domain of data science, but as previously mentioned they all have one common insufficiency in regards to the objective of this thesis: the discretization of time. Furthermore, the practices that have been listed in this chapter do not have the same objective and approach as the one adopted in this work. In chapter 3, a formalized problem statement and a detailed description of the adopted strategy will be presented. In the next chapter some continuous time formalisms will be presented as a background with the required preliminaries for this work.

CHAPTER 2

BACKGROUND

Each of the formalisms that have been introduced in the previous chapter has its own advantages and limitations. Nonetheless, all of them have a common insufficiency, the discretisation of time, which can be described as a representational bias in the learning of these formalisms. Between every two consecutive events there is a discrete time step. Therefore, there is no real measure of time that shows the actual elapsed time between events, the delay before any event occurs in the beginning of a process, nor the delay at the end of a process. A continuous time formalism ensures no loss of information concerning time. This is important in our framework, especially when it comes to learning behavioral models and verifying properties on them. In particular, from a security point of view, it is better to use continuous time modeling formalisms that allow knowing exactly when to act and not only what action to take; for example when predicting a system failure or forecasting future user tendencies.

To explore the dynamics of a wide variety of systems behavior based on collected event streams, there exist many advanced continuous time modeling formalisms: for instance, continuous time Bayesian networks [NSK02], Markov jump processes [RT13], Poisson networks [RGH05] and graphical event models (GEMs) based on Conditional Intensity Models [GMX11]. In this thesis we are interested in *Recursive Timescale Graphical Event Models* (or RTGEMs) [GM16] a sub-family of GEMs, that present many advantages compared to the other formalisms. In particular, they are designed to universally approximate a reasonable class of marked point processes (see [DV07] for details).

In the following, marked point processes (m.p.p.) are briefly introduced as a prerequisite for the framework. Graphical Event Models and their different subfamilies are then defined and discussed.

2.1 Marked Point Processes

Let S be a totally ordered set. A point process is a set of points that are positioned in S. A marked point process is a point process that contains additional features at each point. In other words, a marked point process (m.p.p.) is composed of a point process and marks associated witch each point. If we consider a set of marks M, an m.p.p. can be expressed as the pair:

$$\{(s_i, m_i) : i = 1, ..., n\}$$

where $s_i \in S$, $m_i \in M$ and $s_i \leq s_{i+1}$ for all $1 \leq i \leq n$.
Henceforth, we assume that the space S that is used is the one-dimensional space of time, and the marks are labels describing each point. Marked point processes are used for expressing event streams in this framework, i.e. random labeled (marked) events (points) that arrive at different times $\{t_i\}$ (unidimensional space). The word point is used for describing events as being instantaneous in the time dimension and for the sake of simplicity, the notation l for labels is going to be used instead of m for marks.

In practice, we tend to use a timeline to represent data that arrive at irregular intervals. A timeline can be defined as a sequence of pairs $\{time, event\}$ capturing the relative frequency and ordering of events [WP13]. In figure 2.1, an example is shown of how a timeline can be seen as a multivariate marked point process.

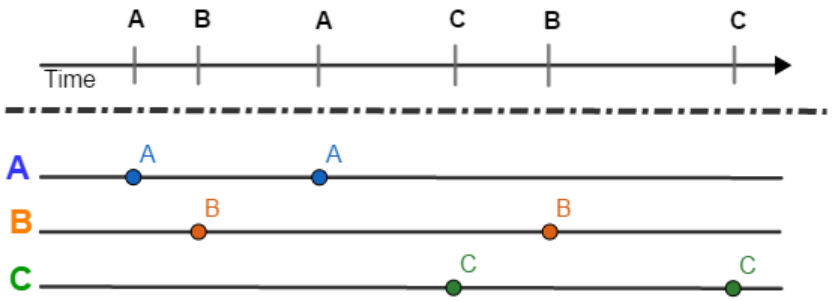


Figure 2.1 – An example showing the decomposition of a timeline into a marked point process.

Time is often seen with an evolutionary character, i.e. what happens now could depend on the past but surely not on the future. In the following, event streams are defined as well as stochastic models that are capable of representing such an evolutionary process.

If we take a step back, a point process can be described by unrolling a stochastic model that defines inter arrival times between different events, i.e. defining the time of the next event based on all the times of previous events. In the following, we will explore a family of stochastic models based on conditional intensity functions, called Conditional Intensity Models (CIM). We start by defining the data type that is generally used for learning these models.

Event streams. An event stream consists in a timed sequence of events with strictly increasing timestamps. An event stream can be written as: $(t_1, l_1), ..., (t_n, l_n)$, with $0 < t_i < t_{i+1} < t^*$ ($t^* = t_{n+1}$) for all $1 \leq i \leq n-1$ and where l_i are labels chosen from a finite set of labels $\mathcal{L}$. Hence, a single event is defined by the pair (t_i, l_i) but for the sake of simplicity we sometimes refer to an event only by its label l when the context is clear enough. We note that in the following, $t_0 = 0$ and $t^* = t_{n+1}$ are used as conventions, as well as the fact that two events cannot occur at the exact same time. In the following, we use t^* to denote the length (or size) of a given sample. An event stream can be considered as an m.p.p. and the data that can be derived from it can be written as x_{t^*}. We write $|x_{t^*}|$ for the size of our data x_{t^*} (the number of events in the sequence, here it is n). The history at time t is the set of all the events that occurred before t: h_i denotes the ith history $h_i = (t_1, l_1), ..., (t_{i-1}, l_{i-1})$. With this in mind, we define a conditional intensity function. In the following, and for the sake of simplicity, data will sometimes be written as x, history as h and time as t, without indexing them explicitly at every instance, but providing enough context to understand their use.

Definition 2.1.1 *[DV07] A conditional intensity function, defines the risk of having a given event at a certain time, depending on the observed history of events. We recall that $E[X]$ is the expected value of a random variable X. Mathematically a conditional intensity function λ is written as:*

$$\lambda(t) = \lim_{\Delta_t \to 0} \frac{E[N(t, t+\Delta_t)|h_t]}{\Delta_t}$$

where $N(T)$ denotes the number of points (events) occurring in a time interval T.

In other words, the conditional intensity function can globally specify the mean inter arrival times between two events in a process, because it defines the mean number of

events that are occurring in a fixed time interval. Furthermore, we recall an important assumption for the following about this approach: there cannot be more than one event in an infinitesimal interval of time.

Before moving forward to the definition of a CIM, we start with an introductory example about a common probability distribution based on an intensity function. Consider for instance the Poisson distribution [Hai73], that is a probability distribution representing a number of events occurring in a fixed interval of time under the assumption that any event is independent with regards to the duration from the previous event. A Poisson distribution is used for modeling processes such as the number of cars arriving to a garage between two given times. The probability function of a random variable X that follows a Poisson distribution of parameter λ (where also the expectation and the variance are equal to λ) is given by: $P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$. We note that in order to sample from such distribution, the inter arrival times between every two events should be computed. The inter arrival time follows the exponential distribution in the case of a Poisson distribution, with a mean of $1/\lambda$, i.e. the mean time between every two events in a Poisson distribution is $1/\lambda$.

A CIM that models a marked point process must treat each point (or event) individually, thus each label $l \in \mathcal{L}$ must be associated with its own conditional intensity function $\lambda_l(t \mid h)$. The conditional intensity function, describes the risk of having the event l at time t given a certain history h of events (the totality of the events or a part of them depending on the CIM). For instance, in a *Markovian* CIM where an event can be particularly dependent from a set of events (called parents), the conditional intensity functions satisfy the following property:

$$\lambda_l(t \mid h) = \lambda_l(t \mid [h]_{Pa(l)})$$

where $Pa(l)$ is the set of parents of l and $[h]_{Pa(l)}$ is the history of the parents only.

Definition 2.1.2 *[DV07] Formally, a CIM θ is a set of indexed conditional intensity functions $\{\lambda_l(t \mid x)\}_{l \in \mathcal{L}}$. The data likelihood function is the joint density function of all the points in the m.p.p. that can be factorized into a product of all conditional intensity*

functions [DV07] and can be written as:

$$p(x \mid \theta) = \prod_{l \in \mathcal{L}} \prod_{i=1}^{n} \lambda_l(t_i \mid h_i; \theta)^{1_l(l_i)} e^{-\Lambda_l(t_i|h_i;\theta)}$$

where $\Lambda_l(t \mid h; \theta) = \int_0^t \lambda_l(\tau \mid x; \theta) d\tau$ *represents the compensator [DV07] for the data* x *and the indicator function* $1_l(l')$ *is one if* $l' = l$ *and zero otherwise.*

In the next subsection, Piecewise-Constant Conditional Intensity Models (PCIMs) are briefly presented in order to give an insight about Graphical Event Models.

2.1.1 Piecewise-Constant Conditional Intensity Models

Piecewise-Constant Conditional Intensity Models (PCIMs) are based on the assumption that intensity functions are constant in "pieces" or in time intervals. In other words, there is a mapping associated with these models that assigns a parameter λ for every label in $\mathcal{L}$ according to an *active* state. The active state is determined in function of the time t and the data x.
These models are important because they form the basis of graphical event models and they ensure fast and efficient learning and inference. These models are related to Poisson networks [Tru+05], since Poisson networks also contain piecewise constant parameters in their mapping (for more details see the following and [GMX11]). However, it was experimentally shown in [GMX11] that PCIMs are in practice, by orders of magnitude, faster to train than Poisson networks.

Definition 2.1.3 *Let* $\mathcal{L}$ *be a set of labels. Each label* $l \in \mathcal{L}$ *is associated to a set of discrete states* Σ_l*. For each label* l *and each state* $s \in \Sigma_l$ *we have a parameter* λ_{ls}*. An active state* s *for each label (correspondingly the active parameter* λ_{ls}*) has to be determined by a mapping* $\sigma_l : T \times X \longrightarrow \Sigma_l$ *(with* T *the set of all possible times and* X *the set of all possible data). Therefore, a PCIM is defined by local structures* $S_l = (\Sigma_l, \sigma_l(t, x))$ *and local piecewise constant parameters* λ_{ls}*. Let* $S = \{S_l\}_{l \in \mathcal{L}}$ *and* $\theta = \{\lambda_{ls}\}_{l \in \mathcal{L}, s \in \Sigma_l}$*, then the PCIM data likelihood knowing the structure and the parameters in this case can be written as:*

$$p(x \mid S, \theta) = \prod_{l \in \mathcal{L}} \prod_{s \in \Sigma_l} \lambda_{ls}^{M_{ls}(x)} e^{-\lambda_{ls} T_{ls}(x)}$$

where $M_{ls}(x)$ is the number of occurrence of events of type l while s is active in the event sequence x, and $T_{ls}(x)$ is the total duration while s was active for event type l.

In practice, the learning of a PCIM is divided into two processes: the *local* structure learning process that could be done using a decision tree to alternate between active states and define adequate mapping from the data to the states set similarly to [CHM97]; and the *parameters* learning process that uses statistical estimates depending on the type of the model and knowing the structure. Commonly, a greedy search is used (similarly to the learning of Dynamic Bayesian Networks) where for a fixed structure (for instance a given decision tree and a given mapping for states) the parameters are calculated, and the structure is continuously modified until we cannot find a better model based on a selection criterion or a certain "gain".

The theory of CIMs is a root to many continuous time formalisms. It is an expressive tool that can help build graphical models such as Poisson networks [RGH05] and GEMs, that will be detailed later on. Moreover, PCIMs favor the discretization of states instead of the discretization of time, creating a set of constant parameters associated with each "scenario" in time and history of past events in which an event can occur. In the following, we define a family of graphical models that is directly used in this framework, based on the concepts that have been presented so far.

2.2 Graphical Event Models

Definition 2.2.1 *A Graphical Event Model (GEM) is defined as $\mathcal{G} = ((\mathcal{L}, E), \theta)$ where $(\mathcal{L}, E)$ is a directed graph and θ a set of parameters. A GEM can represent event streams of the type x as defined in section 2.1, as well as the dependencies between the different labels (or events) in time. In this case, the likelihood of the data knowing the graph and its parameters is written as:*

$$p(x_{t^*} \mid t^*) = \prod_{i=1}^{|x_{t^*}|} \lambda_{l_i}(t_i \mid h_i) \prod_{i=1}^{|x_{t^*}|+1} e^{-\sum_{l \in \mathcal{L}} \int_{t_{i-1}}^{t_i} \lambda_l(\tau|h_i) d\tau} \tag{2.1}$$

with $h_{|x_{t^*}|+1}$ representing the entire history of the event stream including x_{t^*}. In figure 2.2, an example of a GEM is shown with four labels (nodes) A, B, C and D, each with its corresponding parameter (conditional intensity function). The structure of a GEM

(directed graph) has no constraints over cycles or loops, giving it a superior degree of expressive power compared to other graphical models based on DAGs, like Dynamic Bayesian Networks for instance, that cannot directly allow cycles or loops in a single time-slice.

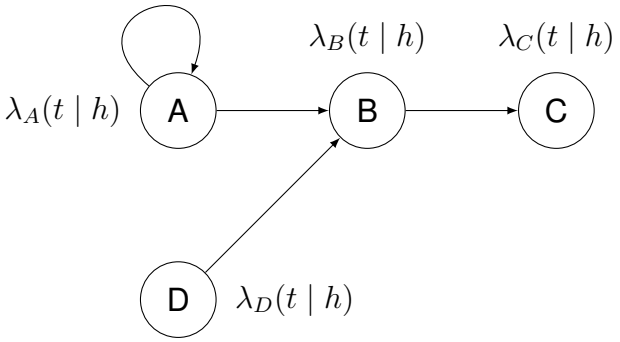


Figure 2.2 – An example of a 4 labels GEM.

In this formalism, one should note that the conditional intensity functions $\lambda_l(t \mid h)$ are not *piecewise-constant*, which means that they do not take constant values for a certain period of time so the models are a family of CIMs and not PCIMs.
In this work we are only interested in *Markov* m.p.p.s with respect to a certain GEM. The conditional intensity functions $\lambda_l(t \mid h)$ in this case satisfy the following property for all t and h:

$$\lambda_l(t \mid h) = \lambda_l(t \mid [h]_{Pa(l)})$$

where $Pa(l)$ are the parents of l in $\mathcal{G}$. This means that the conditional intensity of a certain label l at time t only depends on the history of the parents of l and not the entire history of the process.

The dependencies between the events can be easily depicted on the graph of the GEM. In the graph of figure 2.2 for instance, one can identify that label B has two parents A and D, hence $\lambda_B(t \mid h)$ depends on the past history of A and D. However, label (or node) D does not have any parents and subsequently has a constant rate $\lambda_D(t \mid h) = \lambda_D$ of occurrence.
It has been noted in Chapter 1 that any joint distribution over a set of variables can be represented by a fully connected Bayesian Network. Likewise, a fully connected GEM can represent any m.p.p. with labels in $\mathcal{L}$ [GM16].

In the following, we introduce a more interesting subfamily of GEMs that is more granular and can be more easily used in practice.

2.2.1 Timescale Graphical Event Models

A Timescale GEM (TGEM) is a subfamily of GEM where each dependency between two events is defined for a given finite *timescale* which specifies the temporal horizon and the granularity of the dependency represented by that edge. Formally, a timescale is a set T of half-open intervals $(a, b]$ (with $a \geq 0$ and $b > a$) that form a partition of some interval $(c, t_h]$ (with $c \geq 0$), where t_h is the highest value of T and is called the *horizon* of an edge e.

Definition 2.2.2 *A TGEM $M = (\mathcal{G}, \mathcal{T})$ consists of a GEM $\mathcal{G} = ((\mathcal{L}, E), \theta)$ and a set of timescales $\mathcal{T} = T_{e_{(e \in E)}}$ corresponding to the edges E of the graph of $\mathcal{G}$.*

We write $c_l(h, t)$ as the *parent count vector* of bounded counts (of occurrences) over the intervals in the timescales of the parents of l (in order to have bounded counts, a maximum threshold should be fixed). We provide an example to explicitly show parent count vectors and parameters (see Example 2.2.1).

The conditional intensity of a node (or a variable) labeled l in the GEM only depends on the history of the number of occurrences of its parents within the corresponding timescale.
The conditional intensity functions now have parameters, which are piecewise-constant:

$$\lambda_l(t \mid h) = \{\lambda_{l,c_l(h,t)}\}_{c_l(h,t) \in C_l}.$$

We use C_l to denote the set of all possible parent count vectors of label l. We note that each edge has a timescale. For the following example (Figure 2.3), we consider that all TGEMs are bounded by 1 (making the parent count vectors binary), thus only the fact that a parent has occurred (or not) within the corresponding timescale is important and not the number of times a parent occurs.

Example 2.2.1 *Consider the TGEM illustrated in Figure 2.3. We have $\mathcal{L} = \{A, B, C, D\}$, so we can list the different parent count vectors in C_l associated with each label:*

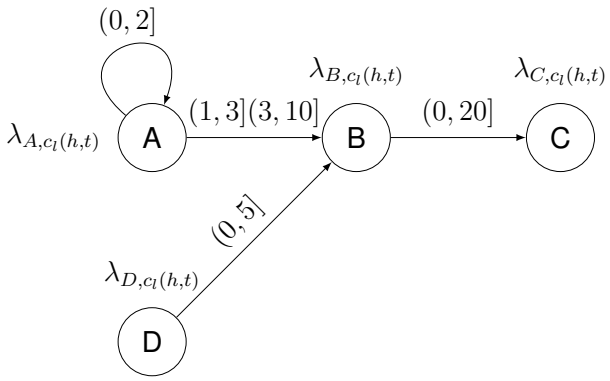


Figure 2.3 – An example of a 4 labels TGEM.

$C_A = \{0, 1\}$, $C_B = \{0, 1\} \times \{0, 1\} \times \{0, 1\}$, $C_C = \{0, 1\}$ *and* $C_D = \emptyset$ *. Thus, for the event* B *for example,* $c_B(h, t) = [0, 1, 1]$ *means that there was no* A *in* $[t - 3, t - 1)$*, there was an* A *in* $[t - 10, t - 3)$ *and there was a* D *in* $[t - 5, t)$*. Hence, the conditional intensity functions for the variable* B *are of the form:* $\lambda_{B,000}$*,* $\lambda_{B,001}$*,* $\lambda_{B,010}$*,* $\lambda_{B,011}$*,* $\lambda_{B,100}$ $\lambda_{B,101}$*,* $\lambda_{B,110}$ *and* $\lambda_{B,111}$*. The same applies to the rest of the variables, except the variable* D *that is independent of all other variables, so its conditional intensity function is written as* λ_D*. All conditional intensity functions are equal to constants making them piecewise-constant depending on the corresponding combination of parents.*

In the case of TGEMs, the likelihood of the data in equation 2.1 is simplified and written as follows:

$$p(x_{t^*} \mid t^*) = \prod_{l \in \mathcal{L}} \prod_{j \in C_l} \lambda_{l,j}^{n_{t^*,l,j}(x_{t^*})} e^{-\lambda_{l,j} d_{t^* l,j}(x_{t^*})} \tag{2.2}$$

where the sufficient statistics $n_{t^*,l,j}(x_{t^*})$ and $d_{t^* l,j}(x_{t^*})$ are the count of l-events and the durations, respectively, when the parent count vector was equal to j (a certain combination of parents). In [GM16], the sufficient statistics are formally written as:

$$n_{t^*,l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|} \mathbf{1}(l_i = l)\mathbf{1}(c_l(h_i, t_i) = j)$$

$$d_{t^* l,j}(x_{t^*}) = \sum_{i=1}^{|x_{t^*}|+1} \int_{t_{i-1}}^{t_i} \mathbf{1}(c_l(h_i, \tau) = j) d\tau$$

with $t_{|x_{t^*}|+1}$ used to represent the duration d between the last event occurrence and the final time t^* in the data.

2.2.2 Recursive Timescale Graphical Event Models

In practice, the learning process of GEMs is not an easy task, especially because they are not piecewise-constant intensity models, thus the learning of the conditional intensity functions can become hard (since they are not constants). The difficulty of learning GEMs, along with the necessity of having a "universal" Graphical Event Model that is easy to learn and to handle, led to the creation of a subclass of TGEMs called *Recursive Timescale Graphical Event Models* (RTGEMs).

Definition 2.2.3 *The family of RTGEMs is defined recursively to be the finite closure of the empty model* $\mathcal{M}_0 = ((\mathcal{L}, \{\}), \{\})$ *under a set of allowed operators* $\mathcal{O}_F = \{add, split, extend\}$ *(discussed in the following).*

RTGEMs as described in [GM16] can universally approximate any smooth multivariate temporal point process. RTGEMs are learned recursively using the set of operators introduced above. More details will be given in the following.
Furthermore, it is proven in the works of [GM16] that using a greedy search algorithm to learn this class of models always shows structural and parametric consistency. In other words, the learned model converges in probability to the optimal model when the learning data size is increased, unlike the more general cases of TGEMs and GEMs for which, to the best of our knowledge, consistency has not been proven.

A finite consistent RTGEM learning procedure [GM16] is to do a forward greedy search (for model construction) followed by a backward greedy search (for model refinement), both based on model selection, and using data that is faithful (see Chapter 1). The Bayesian Information Criterion (BIC) [Sch+78] is a very general model criterion that has been adapted to select the "best" (or a "better") RTGEM when doing a greedy search, and is written as follows for a model M:

$$S_{t^*}(M) = \log(p(x_{t^*} \mid t^*; M, \widehat{\lambda}_{t^*,l,j}(x_{t^*}))) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot \log(t^*),$$

where the left term of the difference is the likelihood of the data knowing the history, the model and the calculated maximum likelihood estimates (m.l.e.) of the λ functions (written $\widehat{\lambda}$). The right term represents the complexity of the model, with $\mathrm{Dim}(M) = \sum_{l \in \mathcal{L}} \mid C_l \mid$ as the dimension of an RTGEM, multiplied by the logarithm of the length of the sample t^* (t^* is also referred to as sample size in the following) used for learning to

define the overall complexity.
Given a model M, the maximum likelihood estimate $\widehat{\lambda}$ of the λ parameters for M (knowing its structure) is as follows [GM16]:

$$\widehat{\lambda}_{t^*,l,j}(x_{t^*}) = \frac{n_{t^*,l,j}(x_{t^*})}{d_{t^*l,j}(x_{t^*})}$$

Learning. The set of elementary operators allowed in the learning of RTGEMs in a forward search algorithm is the following: $\mathcal{O}_F = \{add, split, extend\}$. The forward search algorithm usually starts from an empty model (only containing nodes that are not connected) . The "add" operator adds a non-existing edge to a model and its corresponding timescale $T = (0, c]$, with c a constant (also called horizon). The "split" operator splits one interval $(a, b]$ in the timescale of a chosen edge into two intervals $(a, \frac{a+b}{2}], (\frac{a+b}{2}, b]$. The extend operator extends the horizon of a chosen edge by adding the interval $(t_h, 2t_h]$, with t_h being the previous horizon.

In the learning algorithm proposed in [GM16], a backward search follows a forward search. This means that, implicitly, symmetric backward operators should be used. For the sake of convenience, and since there are no further information in the literature about these operators, one can write $\mathcal{O}_F^{-1} = \{reverse\_add, reverse\_split, reverse\_extend\}$ for these symmetric backward operators.

The "reverse_add" operator removes a chosen edge with only one interval in its timescale (to make it the exact inverse of the "add" operator). The "reverse_split" operator is used for merging two consecutive intervals in a timescale on a chosen edge that have been initially split. The "reverse_extend" operator removes the highest (the last) interval in a timescale on a chosen edge only if the upper bound of this interval was initially created by an extend.

If we explore more closely the dependencies in an RTGEM we can see that the number of dependencies that an edge represents is equal to the number of intervals in its timescale. Therefore, removing an edge that contains more than one interval is not an "elementary" operation: it is like removing multiple edges. This is why the operation remove_edge (and analogously the add operator) can only be used for removing edges (adding edges) with one interval in their timescale.

Sampling. The sampling of such models is similar to the sampling of Poisson Networks [RGH05] and it was also adapted on Continous Time Bayesian Networks (CTBNs) [FS08]. The needed "elementary" sampling techniques are demonstrated using the example in Figure 2.4 where different simple RTGEMs are shown. In Figure 4.16a the sampling is straightforward since there is only a single constant rate λ_A. In Figure 4.16b, the sampling of A is straightforward but the sampling of B will depend on the sampling of A. As a consequence, A must be sampled before B, then the active λ_B should be chosen respectively in regards to the occurrences of A along the timeline, and finally B can be sampled using the corresponding λ_B value in each section of the timeline.

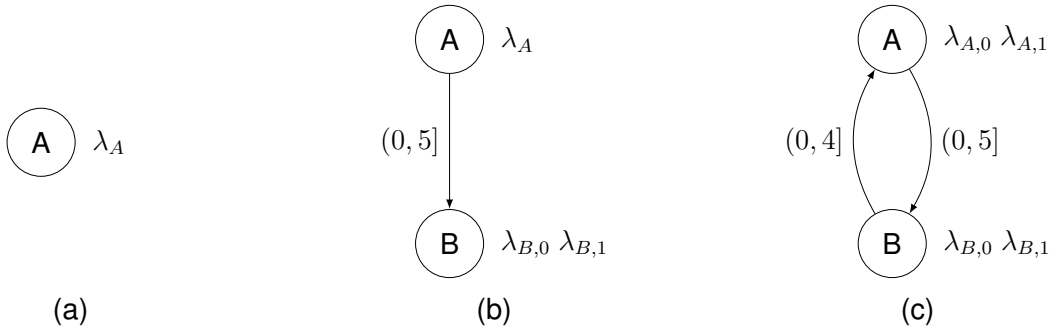


Figure 2.4 – An example showing different RTGEMs to illustrate elementary sampling techniques.

Finally, in Figure 4.16c, things become more complicated: none of the variables can be sampled straightforwardly because of the cycle between A and B. Therefore, some sort of "competition" (based on the rejection sampling concept [Flu90]) is created between the variables and two time values τ_A and τ_B are sampled using the corresponding rates λ_{A,c_l} and λ_{B,c_l} (c_l is the corresponding parent configuration, $c_l = 0$ initially). The highest value (between τ_A and τ_B) is rejected because the one that comes before is supposed to change the corresponding conditional intensity function of the other (making the latter one wrongly sampled). Note that the corresponding rates are the conditional intensity functions that are active (with regards to the parents configuration) at the moment of the sampling. Afterwards, the lowest value (between τ_A and τ_B) is accepted if it is within the firing period. The firing period is the lowest interval on the timescale of the edge entering the corresponding node (for A it is between 0 and 5), because usually after that interval there's a switch of parents configuration and subsequently a switch in the conditional intensity function.

Using these elementary techniques, sampling can be generalized to a larger number of nodes, but two operations (illustrated in Figure 2.5) need to be performed beforehand.

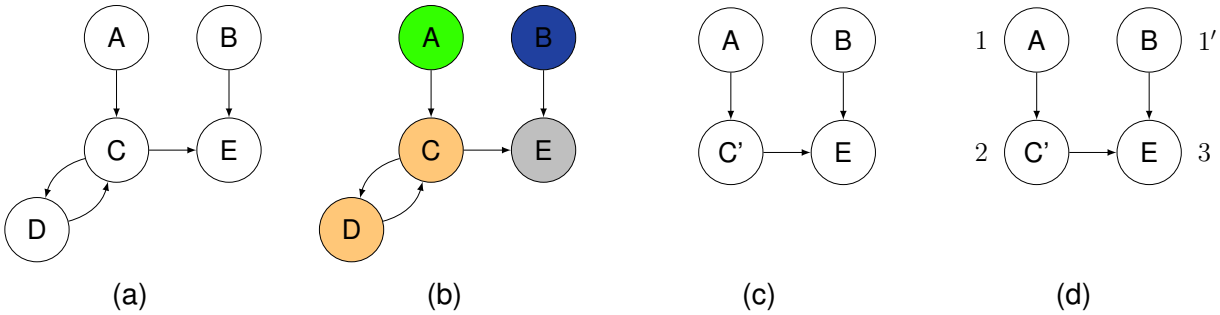


Figure 2.5 – An example from (a) to (d) showing how an RTGEM (we omitted the timescales for a better representation) can be transformed into ordered SCCs.

The first operation is based on the concept of Strongly Connected Components (SCCs). A Strongly Connected Component (SCC) in graph theory [Cor+01] is a directed subgraph where there exists a path between every single pair of nodes. The first operation is illustrated in Figure 2.5b where strongly connected components (nodes of the same color) are brought together forming a new "node" locally and the directed graph is transformed to a DAG at the end of the operation. The second operation (illustrated in Figure 2.5d) is topological ordering [Cor+01], and it defines an order that can be used for sampling the new nodes of the DAG using the three elementary operations. Note that the topological order is not unique, for instance 1 and $1'$ could be sampled in random order (1 before $1'$ or vice versa).

The sampling procedure described above is called *Forward Sampling* due to the progressive sampling of the SCCs (one by one) following a topological order. Furthermore, the sampling of each SCC is based on rejection sampling [Flu90] as previously described. The sampling algorithm and the complexity of generating one sample from an RTGEM will be defined in what follows (Algorithm 1). Algorithm 1 takes an RTGEM, its ordered SCCs (SCC_set) and a sample length t^* as inputs and generates an event stream of the form x_{t^*}. In our case, the computation of the SCCs is performed using Kosaraju's algorithm [Sha81], that has linear complexity with regards to the total number of nodes (n) and the total number of edges (e) ($\mathcal{O}(n+e)$). However, any other algorithm that allows computing SCCs could be used instead.

Once SCCs are computed, the essence of sampling is to go through each SCC and apply the rejection sampling technique within this SCC (it is straightforward in the case

Algorithm 1 Forward Sampling RTGEMs

```
   input: RTGEM; SCC_set; t*
   output: x_{t*}
 1: for(i = 0; i < n_s; i++)
 2:    t = 0
 3:    while(t < t*)
 4:       Rejection_Sampling(nodes[i])
 5:       Update_time(t)
 6:       if(t < t*)
 7:          Update_Stream(x_{t*})
 8:          Update_Lambdas
 9:       else
10:          break
11:       endif
12:    endwhile
13: endfor
14: return   x_{t*}
```

of SCCs with one node and no loops). The variable t defines the time of the next event. The function $Rejection\_Sampling$ takes as a parameter the nodes of the concerned SCC and samples them as previously described. The function $Update\_time(t)$ updates the value of t as previously described in this section. In line 6 of the algorithm, we ensure the respect of the t^* bound (desired sample length) while adding the one sampled event in x_{t^*} ($Update\_Stream$ on line 9) and to update the active parameters (λ) due to the potential change of parents configurations ($Update\_Lambdas$ on line 10).

Let n_s be the number of SCCs, n_m the maximum number of nodes in a single SCC and λ_{max} the highest λ value of the RTGEM. In the rejection sampling phase, the more events we sample the more operations (*Update* functions) we trigger for each node inside the SCC (thus we use n_m for the upper bound in the complexity evaluation). To sample the maximum amount of events we need to compute the smallest arrival times (τ) for each node. The mean inter arrival times are defined as $\frac{1}{\lambda}$ (as previously mentioned) for a Poisson distribution (which is the case for RTGEMs). Therefore, the upper bound for the maximum number of triggered operations is defined as $\frac{t^*}{1/\lambda_{max}} = t^* \cdot \lambda_{max}$. Therefore, the complexity of Algorithm 1 can be evaluated as $\mathcal{O}(n_s \times n_m \times t^* \cdot \lambda_{max})$. Remark that, after a closer observation we can write the complexity as $\mathcal{O}(n \times t^* \cdot \lambda_{max})$ with n the total number of nodes. Indeed, the sampling process goes through all nodes

a single time, and each node triggers a maximum of $t^* \cdot \lambda_{max}$ constant time operations.

To the best of our knowledge, there is no published tool to learn and handle this type of models. Besides, as mentioned before, it is experimentally proven that these models (type of PCIMs) are an order of magnitude faster to learn than other continuous time modeling formalisms (in particular Poisson networks) [GMX11], and that they can universally approximate any m.p.p. [GM16], making them very attractive to inquire into. Moreover, we will emphasize, in chapters 3 and 4 on how we benefit from the different advantages of RTGEMs that were presented in this section.

2.3 Conclusion

As shown in this chapter, one cannot deny the importance of continuous time modeling formalisms from an applicative point of view. Graphical Event Models are families of models that can represent any event streams. In particular, for security assessments, RTGEM can be built to explicitly represent and process a lot of information. Furthermore, RTGEMs can be sampled, so they can be verified using Statistical Model Checking (SMC).

Although the sampling of RTGEMs could become expensive (as previously shown), we use it in our experiments to sample with a fixed target. In other words, we adapt a simulation technique that is described in Chapter 3 that allows us to reduce the sampling time in practice. We have found that RTGEMs are adequate to do behavior analytics, in order to distinguish malicious from normal behavior, because they can describe event based and temporal dependencies between a sequence of actions.

In the following chapter, we describe the problem in details and properly formalize it. In addition, we will introduce in details the strategy that we proposed in order to address the problem.

CHAPTER 3

GENERIC STRUCTURE OF A MODEL BASED LEARNING AND FORMAL PROPERTIES VERIFICATION STRATEGY

In this chapter, we give a brief contextualization of the problem we wish to solve in order to reach the objectives of this thesis. In the following, we introduce the problem, formalize it and propose a strategy that can solve it. We also propose a generic algorithm describing the different steps of the strategy in details. Afterwards, we justify and describe the methods that we chose in order to address each section of the algorithm. Some of the results of this chapter have been presented in [ADL19].

Consider a real life system, like an E-store, a car rental service, an online bank, a manufacturing system or a social network. These types of systems like many others are observable, which means that by observing them we can collect data logs that describe the real behavior and evolution of the system (maybe in different executions). We consider, for the following, that the data is as described in 2.1, so that our data set $\mathcal{D}$ consists of timed events $E \in \mathcal{L}$, $\mathcal{L}$ being the set of labels. Using this data set we would like to learn the "fittest" model, that best represents the data. We recall that learning a model is important from a security point of view because it allows easier comparison with *reference behavior models*, enables broader applications of formal verification (also broader types of properties to verify) and allows the sampling of more data.

In the learning phase of a model, we want to learn the fittest model that best represents reality. Therefore, in order to select this fittest model, we tend to adapt scores and metrics that evaluate the complexity and resemblance of different learned models compared to the real data. From a *security* point of view, it is also important to verify if

our model (that represents reality up to a certain degree) satisfies given security rules or properties. If security properties do not hold on the model, we are also interested, from a security point of view, in computing how far the current model is from verifying them in order to have an insight on the "danger level" of the model. A possible approach is to adapt a measure inspired by methods of model comparison that allows us to evaluate the dangerousness accordingly. By proceeding as such we can find (or estimate) a model that is safe, i.e. that can generate data with "safe behavior".The probability that a model M verifies a security query φ is written $P(\varphi \mid M)$. In order to stay coherent with the notations of the previous chapter, we write $P(D \mid M)$ for the likelihood of the data knowing the model (structure and parameters). Remark that the previously mentioned likelihood (used to compute the maximum of likelihood) is different from the *marginal likelihood* of the data knowing only the structure of the model. It was shown in [CHM97] that AIC and BIC scores are approximations of the *marginal likelihood* (this will be addressed in the following). The problem we are stating can be formalized as follows:

$$\exists M^*, M^* = argmax(P(D \mid M)) \; with \; P(\varphi \mid M) > c, \tag{3.1}$$

with $c \in [0, 1]$ a given constant.
The security properties we are looking to verify are *qualitative* and generally address a limited number of events in our model. We denote $\mathcal{L}_\varphi$ the set of labels of the events concerned by the security query φ. We write l_φ for a label in $\mathcal{L}_\varphi$. Security properties are explained and defined in details later on in the current chapter.

Intuitively we tend to address this kind of problems as optimization problems. However, the problem as stated in equation 3.1 cannot be solved using a multi-objective optimization heuristics such as [Mir+16], because of the fact that a qualitative property cannot be optimized, it is either true or false. In other words, we only have models that are not secure ($P(\varphi \mid M) < c$) and models that are secure ($P(\varphi \mid M) > c$). For instance, consider two models M and M' that satisfy $P(\varphi \mid M) > c$ and $P(\varphi \mid M') > c$ so that both are "secure"; having $P(\varphi \mid M) > P(\varphi \mid M')$ does not mean that M is "more secure" than M'. As a consequence Equation 3.1 cannot be optimized using a multi-objective function, thus we decompose it and proceed otherwise.

We will be using a sufficiently generic algorithm to explicitly represent and detail the adopted procedure step by step along this chapter.

3.1 Proposed strategy

The strategy we propose can be represented using a generic algorithm consisting in three main steps, where the first step is the learning phase, the second step is the model space exploration phase and model verification, and the last step is the distance computation between two models. Every step of the following Algorithm 2 is detailed later on.

Algorithm 2 Proposed Strategy

input: $\mathcal{D}, \varphi$
output: M^*, Δ
1: $M^o = \underset{M \in Chosen\_Formalism}{\operatorname{argmax}} P(\mathcal{D} \mid M)$
2: $\mathcal{N} = \mathcal{N}_c(\mathcal{N}_{\mathcal{L}_\varphi}(M^o))$
3: $M^* = \text{find}\{M \in \mathcal{N}, P(\varphi \mid M) > c\}$
4: $\Delta = \text{Distance}(M^o, M^*)$

The first line of Algorithm 2 corresponds to the learning phase of a model. It consists in choosing an adequate modeling language and learning the fittest model M^o. The choice of the formalism and its learning will be discussed in section 3.1.2. Lines 2 and 3 of Algorithm 2 correspond to the model space exploration phase and model verification, where we seek a model M^*, in the "close" neighborhood of M^o written $\mathcal{N}_c(\mathcal{N}_{\mathcal{L}_\varphi}(M^o))$, that verifies the security property. This step will be explained in details in sections 3.1.2 and 3.1.3. The last line of the algorithm consists in computing the distance between the fittest model and the model that we select after the neighborhood search (if one exists). The notion of *distance* we propose is later defined and explained in section 3.1.4.

3.1.1 Formalism choice and learning

The formalism we chose for this framework are Recursive Timescale Graphical Event Models (RTGEMs), due to the many advantages they offer. To begin with, they are continuous in time, which guarantees no loss of time related information and increases the reliability of verifying time related properties. On the other hand, as mentioned before, they are universal in the sense that they can represent any data set of the form x (as seen in Chapter 2) consisting of events streams in time. Furthermore, RTGEMs are simple to handle and easy to learn, which is mainly due to their "recursive" type and straightforward structure. Finally, they were experimentally proven to

be of orders of magnitude faster to train and more efficient than other formalisms, in particular Poisson networks, as shown in [GMX11; GM16].

Learning. RTGEMs are learned using a two steps Forward-Backward Greedy search technique. A Greedy search has by definition a "best" mode. This mode consists in choosing the best model in every step.
The learning of RTGEMs consists in starting from an initially empty RTGEM (only the nodes (labels) are present in the graph without any edge), and iteratively applying modifications on the initial model, this phase is called the Forward search.

An iteration consists in separately testing all possible operators if we are in "best" mode and some of the possible operators (and maybe all of them) if we are in "better" mode, and henceforth picking only one operator to apply. Remark that not every operator is possible in every iteration, for instance from the empty graph we can only apply the "ADD" operator in the first iteration, in order to add an edge whose timescale can be used in next iterations to split or to extend. Note that for the sake of simplicity, the expression "empty graph" is used in this framework to denote a graph that only contains the nodes without any edge.

After each selected operator (either randomly or deterministically) a test is made based on the Bayesian Information Criterion (BIC) [Sch+78] score before permanently applying the operator. Since we are in Greedy "best" mode, we test all possible operators separately one by one on the empty graph and only apply the operator that gives the best BIC score for the resulting graph. We stop the learning procedure when we cannot improve the BIC score anymore.

In the Forward search phase, the allowed operators are the $\mathcal{O}_F = \{add, split, extend\}$ forward operators that allow to construct the graph and to increase its size by detecting dependencies. In the Backward search phase the allowed operators are the reverse $\mathcal{O}_F^{-1} = \{reverse\_add, reverse\_split, reverse\_extend\}$ backward operators that allow to refine the model and reduce its complexity. Note that in a "best" mode Greedy Search technique we choose the best graph at every step (highest BIC score), thus in practice we rarely find backward operators in the Backward phase that improve the score.

Let $M = (\mathcal{G}, \mathcal{T})$ be an RTGEM such that $\mathcal{T}$ is a set of timescales and $\mathcal{G} = ((\mathcal{L}, E), \theta)$ where $(\mathcal{L}, E)$ is a directed graph and θ the set of parameters. We recall the formula of the BIC score for RTGEMs from Chapter 2:

$$S_{t^*}(M) = \log(p(x_{t^*} \mid t^*; M, \widehat{\lambda}_{t^*,l,j}(x_{t^*}))) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot \log(t^*).$$

Remark that the BIC score can be decomposed into a sum of local scores on each label as follows:

$$\begin{aligned} S_{t^*}(M) &= \log\left(\prod_{l \in \mathcal{L}} \prod_{j \in C_l} \lambda_{l,j}^{n_{t^*,l,j}(x_{t^*})} e^{-\lambda_{l,j} d_{t^* l,j}(x_{t^*})}\right) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot \log(t^*) \\ &= \sum_{l \in \mathcal{L}} \sum_{j \in C_l} (n_{t^*,l,j}(x_{t^*}) \log(\lambda_{l,j}) - \lambda_{l,j} d_{t^* l,j}(x_{t^*})) - \sum_{l \in \mathcal{L}} \mid C_l \mid \cdot \log(t^*) \end{aligned}$$

Hence, there is no need to compute the entire BIC score after each modification concerning a node (or its parents timescales) but only the local evolution of the score. For instance, the local BIC score for node A having one parent with one interval in the timescale can be written as $n_{t^*,A,0}(x_{t^*}) \log(\lambda_{A,0}) - \lambda_{A,0} d_{t^*,A,0}(x_{t^*}) + n_{t^*,A,1}(x_{t^*}) \log(\lambda_{A,1}) - \lambda_{A,1} d_{t^*,A,1}(x_{t^*}))$. The only real obstacle in the learning of an RTGEM consists in choosing a default horizon for the "add" operator. As already mentioned in Chapter 2, the "add" operator adds an edge and its corresponding timescale $T = (0, c]$, with c the horizon. In the works of [GM16], where RTGEM and their learning was first introduced, the choice of the default horizon was never addressed. In practice, while conducting our first experiments, we noticed that the choice of the horizon is critical for the learning. Indeed, a small change in its value causes drastic changes in the BIC score and in the detected dependencies, while a bad starting horizon leads to no learned edges. In order to learn consistent models we must therefore add pertinent default horizon(s).

In our earliest experiments on learning RTGEMs we adopted a "universal" default horizon that we used for all edges regardless of the destination and source nodes. As we proceeded, we found better learning results when we started defining a specific horizon for each edge to add between every pair of nodes. Since the "add" operator adds a dependency within a certain time between two nodes (events), it seemed logical to specify a horizon by referring to the inter arrival times between the corresponding events in the data set. We applied different data based heuristics (mean value, median

value, mode value etc.) and tested them as specific horizons for every pair of nodes. We found, during preliminary tests, that using the medians of inter arrival times as a horizon for the "add" operator between the corresponding nodes gave the best overall quality for learning. However, the time and space complexity of the algorithm for the median heuristic are both quadratic in function of the number of events n in the data ($\mathcal{O}(n^2)$). The number of events n is usually big, as a consequence this computation can become costly.

However, in his Master thesis [Phi20] in collaboration with LS2N's DuKe team, Philipp Behrendt provided evidence that we can adapt a more powerful heuristic for calculating specific horizons based on the work of [BS518]. In the works of [BS518], a new family of Graphical Event Models was introduced called Proximal Graphical Event Models (PGEMs) destined to be used for representing real life systems whose events are considered "proximal". In other words, PGEMs are specific to applications where an event is most likely to depend on the event that is the closest to him in the past. Furthermore, a method that is also data driven, was proposed in [BS518] for calculating a specific and unique "proximal horizon" for each pair of nodes in the graph. They consider that this horizon is optimal for the construction algorithm of their PGEMs (that only contain one timescale on each edge).

The idea of this approach is to find a default horizon where the distribution of event counts differs maximally from the corresponding duration across the parent configurations (recall C_l from Chapter 2). It is the value that maximizes the likelihood for their learning (for a formal proof one should refer to [BS518]). In this work we use this "proximal heuristic" only to compute the starting horizons for the "add" operator, and then we switch to the Greedy Forward-Backward algorithm to continue the learning procedure.

Now that we have justified the use of RTGEMs and detailed their learning procedure we proceed to the model space exploration and formal verification procedures.

3.1.2 Model space exploration

The security properties we would like to verify address a number of particular labels $\mathcal{L}_\varphi$ in our model. Hence, in practice, the number of targeted labels (events, nodes) is

smaller than the overall number of labels. The notation $\mathcal{N}_{\mathcal{L}_\varphi}(M^o)$, on line 2 of Algorithm 2, defines the neighborhood of M^o specified by the labels $l_\varphi \in \mathcal{L}_\varphi$ that are concerned by the security query φ. The objective of the space exploration is to find a model in the neighborhood of M^o that satisfies the query φ. Therefore, we seek to modify the parameters λ_{l_φ} of l_φ by altering the model, and henceforth disrupting the distribution of events l_φ. The overall operations (modifications) that are allowed in the space exploration are the operators in the sets $\mathcal{O}_F$ and $\mathcal{O}_F^{-1}$, in order to stay in the same family of models (RTGEMs). We propose a space exploration technique where the explored neighborhood is not constant and may vary after several iterations if we struggle to find a model that satisfies φ.

We suggest that the initial neighborhood $\mathcal{N}_{\mathcal{L}_\varphi}(M^o)$ consists of the nodes of the graph corresponding to the concerned labels, as well as the nodes in the same Strongly Connected Components as the concerned nodes (recall definition in section 2.2.2 of Chapter 2). In Figure 3.1 we show an example of a Directed Acyclic Graph (DAG) showing the seven SCCs of a given RTGEM, in order to give an illustration for a better understanding of what follows. Note that in Figure 3.1, the blue SCCs are the ones containing the target nodes in $\mathcal{L}_\varphi$.

We allow modifications that targets the initial neighborhood $\mathcal{N}_{\mathcal{L}_\varphi}(M^o)$ (for the space exploration). We name these modifications *level 0 operators* assuming that these operators will not change the SCCs of the model. In other words, *level 0 operators* are modifications that we do internally on the edges (dependencies) inside the SCCs of the concerned labels l_φ. Furthermore, in *level 0 operators* we disallow the adding of edges between two concerned SCCs, even if it does not disrupt the SCCs of the model. In practice, we choose to start with these *level 0 operators*, because in most scenarios they have a direct effect on the parameters λ_{l_φ}. Henceforth, they can contribute the most in making the model satisfy the property. In some cases *level 0 operators* are not enough. As a result we define another class of operators that we name *level 1 operators*.

Level 1 operators allow modifications on an updated version of $\mathcal{N}_{\mathcal{L}_\varphi}(M^o)$, that now also include the parents of nodes l_φ and their corresponding SCCs. In other words, *level 1 operators* allow the addition and/or the editing of dependencies between the concerned SCCs containing l_φ and $P_a(l_\varphi)$. In some extreme cases when we do not find

a satisfying model using the first two levels, one can use *level 2 operators* that allow every possible modification on the RTGEM. *Level 2 operators* are not recommended in this framework, since they do not necessarily target the right nodes and are more likely to disrupt other distributions that do not present direct advantages in the service of the property φ.

The neighborhood $\mathcal{N}$ that we consider on the same line 2, is the transitive closure ($\mathcal{N}_c$) of the previous neighborhood, limited to the number of allowed operators that is fixed beforehand and should not be very high (in order to stay close to the fittest model). The objective of the find function is to determine which operator to apply between the allowed ones (level 0,1 or 2). We check if the initial model verifies the security property in the first step of our find function before doing any space exploration. The idea of the model space exploration (in line 3) consists in doing a finite number of operations on the concerned labels $\mathcal{L}_\varphi$ of the model M^o, while staying in $\mathcal{N}$, in order to find a

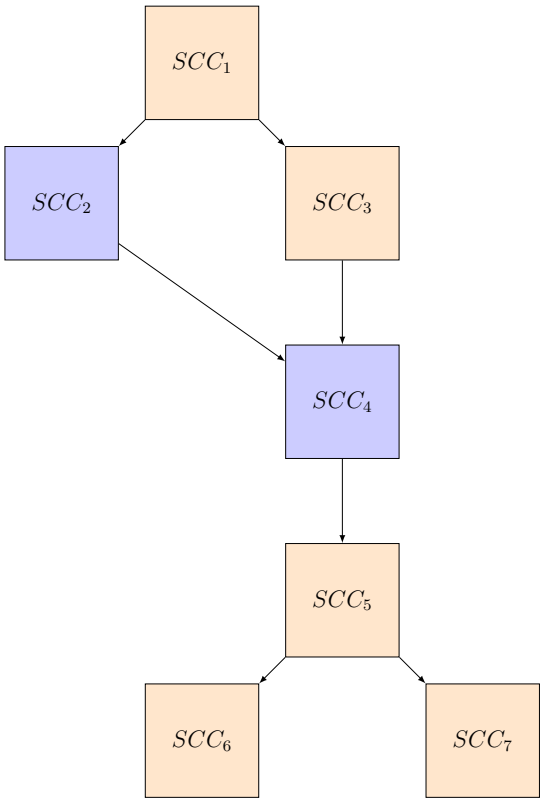


Figure 3.1 – An example of a DAG showing the strongly connected components of an RTGEM. The SCCs in blue are the concerned SCCs.

model that verifies the property. We check after each operation whether the obtained model satisfies the property. The search stops immediately when we find such a model.

The find function can be defined using any search technique: it can be an exhaustive technique like DFS (Depth First Search) or BFS (Breadth First Search) for example. It can also be random, like the random walk technique or a greedy search with an objective to improve $P(\varphi \mid M)$ in order to make it higher than c. We provide more discussion on the quality of the space exploration function in the perspectives.

In the following we define the security properties as well as the verification procedure.

3.1.3 Model verification

In practice, we are interested in two main types of queries that can be verified on continuous-time graphical models. The first type of queries targets the order or number of occurrences of given events. As an example "the event A occurs more than 10 times in the last 20 time units", or "the event B follows the event A within an interval of (5,10] time units more than 5 times in the last 1000 time units", or any conjunction of such queries (even considering multiple variables). The second type of queries addresses time or the timing of given events. As an example "the delay before the first occurrence of A is more (or less) than 20 time units", or "the inter-arrival time between A and B, in the last 1000 time units, is more (or less) than 20 time units", or any conjunction of such queries. In addition, conjunctions of queries of the two types cited above can be considered as a query. By adapting these queries to the system's security standards we obtain our *security queries* that allow us, depending on the answers we get (true or false) on the traces we check, to classify a model as safe (or dangerous) from a security point of view.

Certain types of queries, like the ones that do not include counting, can be formalized using an extended version of LTL (Linear Time Logic) [BK08], with the addition of past time intervals over the variables. For example we can write $\Diamond^{100} C \wedge A_{(0,5]} \wedge B_{(0,10]}$, meaning that eventually, before 100 times units, we must have a C preceded by an A and a B within their respective intervals. Another example of an extended LTL query, using the *globally* operator, can be written as $\Box^{100}(C \Rightarrow A_{(0,5]} \wedge B_{(0,10]})$, meaning that

all the occurrences of C within the next 100 time units (if it ever occurs) must imply the occurrence of A and B in the past within their respective timescales.

Our contribution is independent from the method that is chosen for verifying these queries. These types of queries could be verified using exact verification methods like standard Model Checking techniques [BK08], but classical Model Checking is subject to state space explosion and to the best of our knowledge was never adapted to Graphical Event Models.

In our experiments we use Statistical Model Checking (SMC) [LDB10], hence we compute for each model M_i that we choose in the neighborhood $\mathcal{N}$ its optimal parameters $\widehat{\lambda} = \mathrm{argmax}_{\lambda} P(\mathcal{D} \mid M_i, \lambda)$. We then simulate M_i in order to compute an estimate $\widehat{P}(\varphi \mid M_i, \widehat{\lambda})$ of $P(\varphi \mid M_i, \widehat{\lambda})$ using the collected samples.

The only drawback about using SMC in practice is that the sampling of RTGEMs is costly when there are cycles in the model. However, in our experiments, we gained much time by using "targeted simulations" and only sampling the target events (more details on this in Chapter 4). We have already introduced the sampling of RTGEMs in chapter 2. The presented technique is called Forward sampling and is also common to Poisson networks [RGH05]. By definition, Forward sampling consists in ordering topologically the different SCCs, before sampling them one by one. In other words, in order to sample the SCC that has topological order 3 for instance, we first need to sample its parents SCCs that have the orders 1 and 2. However, the sampling of the SCC that has topological order 4 is not mandatory for the sampling of the SCC that has topological order 3. In figure 3.2, we show an example of how we cut the graph into two parts, an upper and a lower part with regards to the last concerned SCC's topological order.

In our framework, when we are confronted with the example of Figure 3.2, we only simulate the upper part of the graph (we call that sampling with early stopping) because the lower part does not give further information that are useful in our verification technique. In other words, we are only interested in sampling the parents of the concerned SCCs in order to sample the concerned SCCs afterwards and stop the sampling. Therefore we are not obliged to sample SCC_0 in this example. Remark that the SCCs are recomputed after each modification done by *level 1 and 2 operators*, because they can cause the disruption of the SCCs by changing the topological order or

by adding/removing nodes from SCCs.

We formally define the above described procedure of the early stopping sampling in Algorithm 3 below. The inputs needed are the targeted nodes $\mathcal{L}_\varphi$ and $\mathcal{G}$ (graph and parameters) of the model (see Chapter 2). We then proceed to compute the set of Strongly Connected Components (SCC) of the graph. The next step (line 2) consists in computing a topological order of the SCCs. In the following (line 3) the function $\mathrm{Select}_{\mathrm{SCC}}$ is applied on each of the labels $l_\varphi \in \mathcal{L}_\varphi$ in order to select the targeted SCCs (containing the labels l_φ). Afterwards (line 4), the maximum of the topological order of the targeted SCCs ($imax$) is computed using the function $\mathrm{max\_order}$. Hence, $imax$ consists in the new stopping criterion for the Forward Sampling. The notation $SCC\_set[Topological\_order[i]]$ denotes the SCC with topological order i. And the function $\mathrm{Sample}(SCC\_set[Topological\_order[i]])$ consists in sampling the nodes inside of the corresponding SCC as seen in Chapter 2.

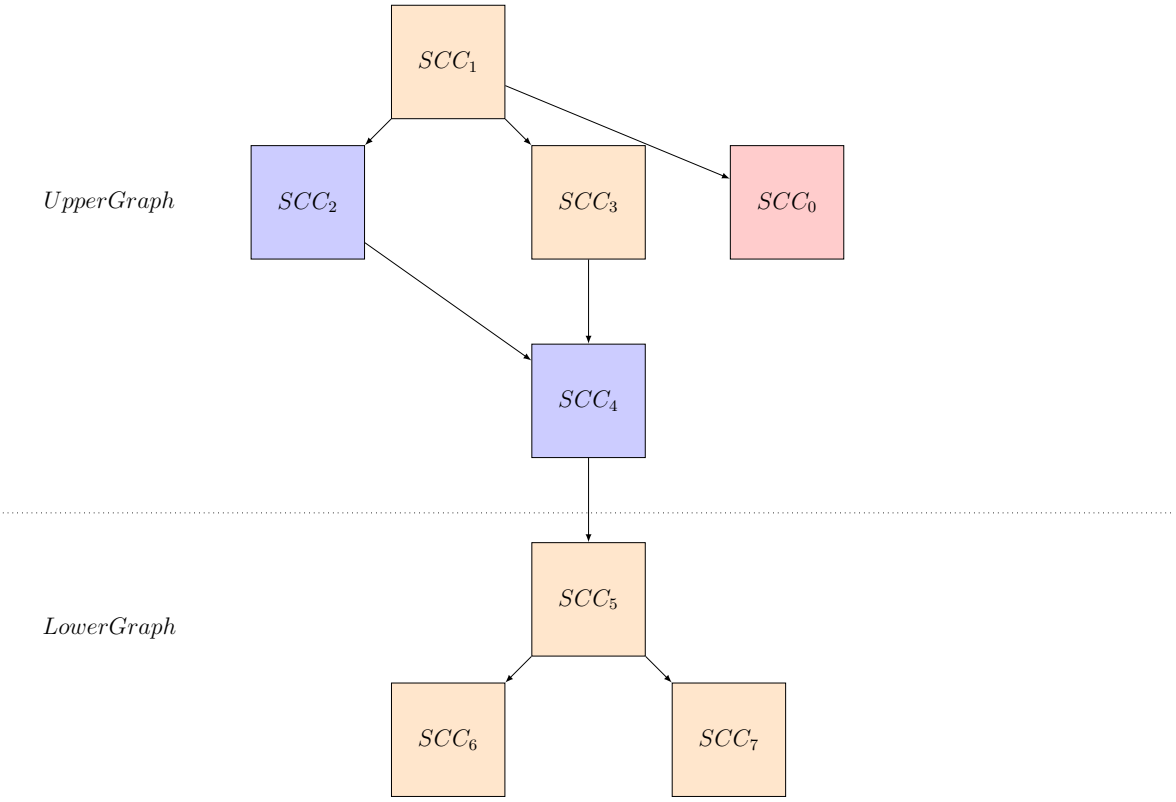


Figure 3.2 – An example of a DAG that is cut into an upper and a lower part based on the last ordered concerned SCC. For a faster sampling with the same outcome we can stop sampling at the last SCC in the upper part.

Algorithm 3 Targeted Sampling with Early Stopping

input: $\mathcal{G}, \mathcal{L}_\varphi$
1: $SCC\_set = \mathrm{Compute}_{\mathrm{SCC}}\left(\mathcal{G}\right)$
2: $Topological\_order = \underset{\forall\, S \in SCC\_set}{\mathrm{Compute}_{\mathrm{Topological}}}\left(S\right)$
3: $SCC\_set_\varphi = \underset{\forall\, l_\varphi \in \mathcal{L}_\varphi}{\mathrm{Select}_{\mathrm{SCC}}}\left(l_\varphi\right)$
4: $imax = \max\_\mathrm{order}\{order / SCC\_set[order] \in SCC\_set_\varphi\}$
5: $i = 1$
6: **while** $i \leq imax$
7: $\quad \mathrm{Sample}(SCC\_set[Topological\_order[i]])$
8: $\quad i++$
9: **end while**

In the following we define a distance measure that we adapted to RTGEMs in order to give an insight about the differences between two RTGEMs.

3.1.4 Distance between models

To the best of our knowledge, there is no existing metric between RTGEMs. The most intuitive distance measure that comes to mind is the minimal number of operations needed to move from an RTGEM to the other since it is a recursive procedure. However, such a distance is not accurate in our context because not all operators add (or remove) the same information every time.

In the literature, the popular Hamming distance [Ham50] has been adapted for some probabilistic graphical models such as Bayesian networks [TBA06]. In the following, we propose an extension of the Structural Hamming Distance (SHD), adapted to RTGEMs, where we evaluate the amount of differing information on two different edges.

A timescale in an RTGEM can be represented by a vector $v = [0, a, b, c, ...]$ where the values are the successive endpoints values. We write v_1 and v_2 for the values of a timescale (on a given edge that is present in both graphs) of G_1 and G_2 respectively. We write $v_{id} = |v_1 \cap v_2|$, for the identical endpoints in the two vectors; and $v_{nid} = |v_1 \setminus v_2| + |v_2 \setminus v_1|$, for the endpoints that are not identical in the two vectors.

Consider two RTGEMs with the same set of labels $\mathcal{L}$, $G_1 = ((\mathcal{L}, E_1), \theta_1)$ and $G_2 = ((\mathcal{L}, E_2), \theta_2)$, we define :

$$\mathrm{SHD}(G_1, G_2) = \sum_{e \in E_{sd}} 1 + \sum_{e \in E_{inter}} d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)), \tag{3.2}$$

where $E_{sd} = \{E_1 \setminus E_2\} \cup \{E_2 \setminus E_1\}$ are the edges of each model that are not present in the other one and $E_{inter} = E_1 \cap E_2$ is the set of edges that are present in both models. $\mathcal{T}(e, G_1)$ and $\mathcal{T}(e, G_2)$ are the lists of endpoints of the intervals on the timescales of the corresponding edge e in graph G_1 and G_2 respectively. Thus, we define the elementary distance as follows :

$$d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = \frac{v_{nid}}{v_{nid} + v_{id}} \tag{3.3}$$

Equation 3.2 corresponds to adding 1 to the global distance when the edge (or the dependency between two nodes) exists in a graph but not the other, and adding a value d in $[0, 1)$ corresponding to the difference between the timescales when an edge exists in both graphs. One can show that SHD is indeed a distance metric (i.e. satisfies distance axioms). A proof is given in Appendix B.

However, this distance measure suffers from a "scaling" disadvantage when it comes to comparing quantitative information. For instance, consider three vectors $v_1 = [0, 1, 2]$, $v_2 = [0, 1.01, 1.98]$ and $v_3 = [0, 5, 10]$, representing the endpoints of three different timescales to compare using the distance measure. By computing the elementary distances between (v_1, v_2) and (v_1, v_3) with respect to equation 3.3, we notice that they are equal although v_1 and v_2 cover almost the same timescale compared to v_3 that covers a different timescale. Therefore, Philipp Behrendt in his Master thesis [Phi20], proposed an extension to this SHD where he takes into account the relative quantitative differences inside the timescales to give a fairer distance measure, that is also more adapted when using *proximal horizons* for the learning.

The idea is to find matches (if existing) between the endpoints of the two timescales based on the mutual minimal absolute difference. In other words, consider two vectors v_1 with size l and v_2 with size k. A match is a pair (v_{1_i}, v_{2_j}) such that the closest element from $v_{1_i} \in v_1$ (with $i = 1, ..., l$) is $v_{2_j} \in v_2$ (with $j = 1, ..., k$) and that the closest element from v_{2_j} is also the same v_{1_i}. This *proximal distance measure* extension proposes a refinement on the elementary distance. The set of matches is written V_{id}^* for the sake

of coherence. Formally for vectors v_1 and v_2 considered above, we can write:

$$V_{id}^{*} = \{(v_{1_i}, v_{2_j}) \in v_1 \times v_2 : \mathbf{cl}(v_{1_i}, v_2) = v_{2_j} \wedge \mathbf{cl}(v_{2_j}, v_1) = v_{1_i}\},$$

with $\mathbf{cl}$ a function that finds the closest element to v_{1_i} in v_2: $\mathbf{cl}(v_{1_i}, v_2) = argmin_{v_{2_p}}(|v_{1_i} - v_{2_p}|)$. The number of matched endpoints (considered identical) is written $v_{id} = |V_{id}^{*}|$ and the number of unmatched endpoints v_{nid}. The elementary distance can now be written:

$$d^{*}(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = \frac{1}{v_{nid} + v_{id}} \left(\sum_{(v_{1_i}, v_{2_j}) \in V_{id}^{*} \setminus (0,0)} \frac{\mid v_{1_i} - v_{2_j} \mid}{min(v_{1_i}, v_{2_j})} \right) + \frac{v_{nid}}{v_{nid} + v_{id}}$$

For each pair of matched endpoints, the sum of the relative differences (scaled by its minimum) is taken into account in order to penalize the fact that it is not a one hundred percent match (not perfectly identical). Furthermore, remark that if there are no detected matches (except $(0, 0)$) we will have $d^{*}(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2))$, in other words we will have the same elementary distance and hence the same SHD as presented before. Unfortunately, we show in Appendix B using a counter example that this function does not satisfy the triangle inequality. It is therefore only a semimetric and the term "distance" is an abuse of language.

In the rest of this work, we use this *proximal distance measure* written SHD^{*} for the sake of coherence, since we calculate the default horizons based on *proximal horizons* for all models. The fact that SHD^{*} is only a semimetric is not problematic in our context. Before getting to the experimental chapter of this work, we show a toy example to illustrate the application of the entire strategy proposed in this chapter.

3.2 Toy example

The purpose of the following example is to illustrate the interest of the proposed strategy on a real life application and to show how we compute a distance measure between two RTGEMs.
We consider a prepaid card online service, where a card owner should sign in to his account in order to do a number of actions before signing out. The possible actions are:

— *Recharge*: add more money to his card account,
— *Check account*: check his card account's balance,
— *Transfer money*: transfer money to his or to another bank account,
— *Log out*.

We suppose that the fittest RTGEM (M^o) that best fits the real behavior of users is as shown in Figure 3.3. This corresponds to the first step of our procedure. A security query φ that we can verify on this model could be of the form $\Box^{1000}(\text{Transfer Money} \Rightarrow \text{Recharge}_{(0,20]} \vee \text{Check account}_{(0,5]})$, and for instance we want the model to satisfy the security property $Pr(\varphi \mid M^o) > 0.8$. In other words we would like to ensure a behavior that we find normal for users on this service and safe from a security point of view: every time a user wants to transfer money, an action where he checks his account must have occurred right beforehand or a recharging of his account must have occurred not long ago (because he may have made some purchases very recently after the recharge and is aware of his balance). If our system does not verify this property we would say that the average user should be more "careful" while using the service and that the global behavior of the service is not secure. We note that the parameters of *Log out* are omitted for the sake of simplicity because it forms an SCC that is not concerned and that is of high topological order (it is not sampled in the targeted sampling technique that we use).

One can see, only from looking at the model (parameters and structure), that $P(\varphi \mid M^o)$ is low and that the learned behavior has low chances of verifying the security property mainly because of the missing dependency (edge) between "*Check account*" and "*Transfer money*". Indeed, we conducted a small experiment where we use quantitative model checking (that is described in details in the next chapter) to calculate an estimate to $Pr(\varphi \mid M^o)$ written $\widehat{p}(\varphi \mid M^o)$. We compute $\widehat{p}(\varphi \mid M^o) = 0.61$. Hence, the average users are transferring money without checking their accounts first. Furthermore, after recharging and using their card they never directly check their accounts but they sometimes directly transfer money.

By doing a limited number of (allowed) operations on the labels that are addressed by φ ($l_\varphi = \{\text{Transfer Money}, \text{Recharge}, \text{Check account}\}$), we can obtain the RTGEM (M^*) of Figure 3.4 (step 2 of the proposed Algorithm 2), where the modifications are in red. Intuitively, this obtained model structure has more chances of satisfying the security property (step 2 of the proposed Algorithm 2), because we now have a smaller interval

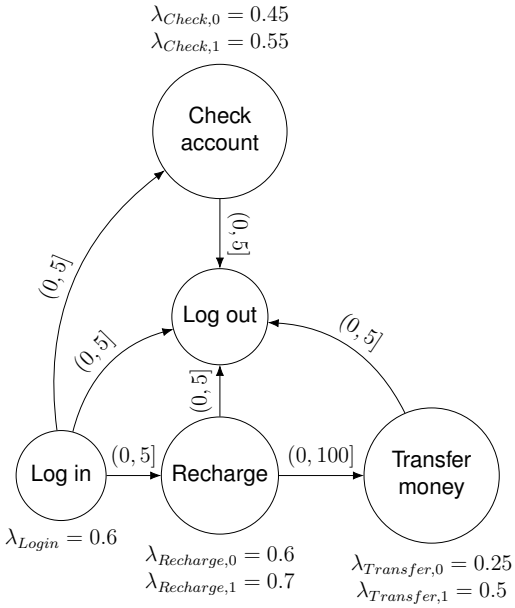


Figure 3.3 – The learned model

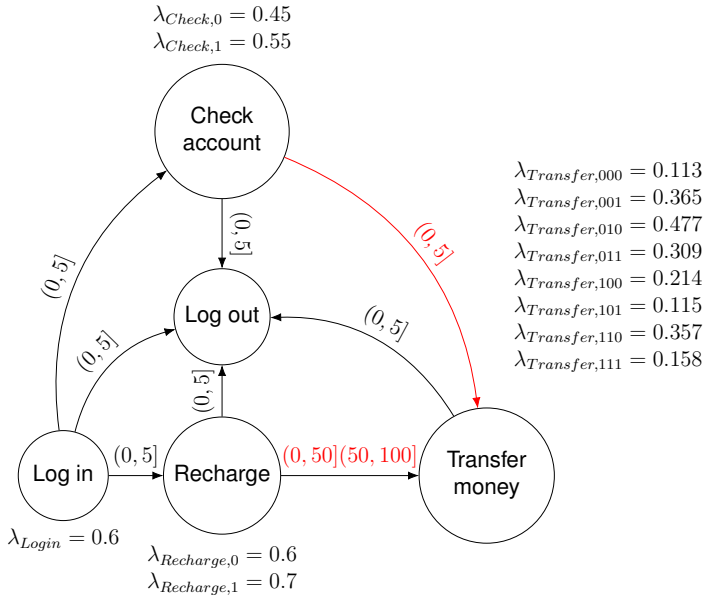


Figure 3.4 – A modified more secure model

between "recharge" and "transfer money" that is taken into consideration and we have added the edge between "check account" and "transfer money". In other words, using the second (found) model structure we are more likely to satisfy the security property. Indeed, we conducted the same quantitative SMC technique used on the learned model, in order to compute an estimate to $Pr(\varphi \mid M^*)$ written $\widehat{p}(\varphi \mid M^*)$. We computed $\widehat{p}(\varphi \mid M^*) = 0.85 (> 0.8)$.

The distance is $\mathrm{SHD}^*(M^o, M^*) = 1.333$ in this case, because of the added edge and the split on the interval (step 3 of the procedure). By following the steps of our proposed algorithm in this example, we have found a model M^o from data logs and a model M^*, which is intuitively more likely to verify the security property (and does verify it in practice) and that is at a distance of 1.333 from M^o, after doing two allowed operations on the concerned variables l_φ.

This toy example is tailored to give an intuition on how the structure and the added/ refined dependencies can visibly play a role in the verification of a security property on a model. In the following chapter, we conduct similar experiments and discuss the calibration and the SMC techniques used in order to give proper results.

3.3 Conclusion

In this chapter, we have proposed a detailed model-based strategy in learning and verification for security assessments, as well as a distance measure between graphical models. Our strategy consists in learning the model that best represents given real data (based on appropriate scores and metrics), in checking if a *close* model exists that verifies a certain security property and in computing a distance, that we defined, between the two models to see how far the *fittest* model is from verifying the property. We recall that the strategy we are proposing is also generic with respect to the model's choice, the verification procedure and the notion of distance between models.

We justified in this chapter that what we are proposing is a very good combination of techniques to apply in each step of our generic algorithm. We have worked on improving some of them, in particular the sampling phase in SMC, in order to get a faster version of the algorithm. The early stopping sampling allowed us to gain notable computation time (this will be discussed in the next chapter). The next chapter contains the experimental results on the tests we conducted to evaluate the advantages and the complexity of the proposed strategy in systems verification.

CHAPTER 4

GRAPHICAL EVENT MODEL LEARNING AND VERIFICATION FOR SECURITY ASSESSMENTS

In this chapter, we present a series of experiments validating and evaluating the performance of each step of the previously defined strategy, as well as a complete pipeline that validates the entire strategy. The only tool that is used in the following experiments is a library/branch of PILGRIM called "evential" (see Appendix A) that was created to handle Graphical Event Models. The library was implemented by a group of the DuKe team in Polytech Nantes and the biggest part of it was tailored to fulfill the main objectives of this thesis. I partially contributed in the earliest stages of implementation/architecture of the library and actively participated in the agile testing phase for validation.

In the following sections, we give results and evaluations for the learning phase, the model exploration phase and the verification phase.

4.1 Learning of RTGEMs

In the beginning of this section we present the results of an experiment that can be described as follows: Many random graphs are generated (called references) with a prefixed complexity. From these graphs, data samples of different sizes are simulated. From each of the sampled data a graph is relearned using the Forward-Backward Greedy search technique. The mean learning times are computed as well as the distances between each learned graph and the corresponding reference. The benchmark of the collected data is shown in what follows.

The purpose of the test above is to show the variations of the learning duration of RTGEMs in function of their complexities and the sample size. Similarly, we evaluate the accuracy of the learning in function of the sample size and the different RTGEM complexities.

In the following, we show some results about the performance of the learning phase. The sample size for the this phase is defined by the "end time" t^* of the sampling. Note that all the sampling operations start from $t_0 = 0$ in this framework. There is no precise time unit used in the data of this framework. As a consequence we use the default "t.u." notation for the end time of the sample and for the endpoints of the timescales. All the tests presented in this chapter have been executed on a Windows 10 Enterprise i5 laptop, with a 2.3 GHz CPU and 8GB of RAM.

In a first experiment, a batch of fifty random RTGEMs with a complexity (dimension) of $|C_l| = 8$ were generated using the random graph generator of the PILGRIM "evential" library. The random graph generator function is based on the Erdos-Renyi model [AGW90]. The function takes 8 parameters: the number of desired nodes; the density of having an edge for each possible (non existing) edge; a seed (in order to reproduce the same model again); the maximum number of allowed parents; the probability of applying an operator to increase the number of intervals on a timescale; the maximum number of allowed time intervals on a timescale; a restricted list of default horizons from which one horizon is randomly assigned to timescales and a restricted list of default parameters (λ) to randomly assign to nodes. The random graph generator function uses its parameters and the set of Forward operators to construct a random RTGEM. We execute it several times (more than fifty) with pretested parameters in order to generate graphs and only use the ones with a dimension $|C_l| = 8$ (we eventually need fifty graphs of dimension $|C_l| = 8$). The idea of using pretested parameters is to minimize the number of needed executions of the random generator before gathering the target number of needed graphs with the same desired dimension. Event logs are sampled from each generated model several times with different end times t^*. Finally a model is then relearned from the sampled data and the distance between the relearned model and the reference model is evaluated (see Figure 4.1, that will be detailed in what follows).

The mean values of learning times are computed and some of them are shown in Table 4.1, where we can see the variations in learning duration depending on the size of the sample and the complexity of the RTGEM. The same protocol was used for the rest of the experiments for graphs with higher complexity, using the same sampling and learning functions. Note that we fix the same number of nodes (in the random graph generator) for every batch of graphs for the sake of convenience. In addition, we choose the values of the restricted list of default parameters (λ) in a way not to bias the tests (values between 0.1 and 1.5).

t^* \ $\mathrm{Dim}(M)$	$\lvert C_l\rvert = 8$	$\lvert C_l\rvert = 12$	$\lvert C_l\rvert = 32$	$\lvert C_l\rvert = 44$	$\lvert C_l\rvert = 80$
500 t.u.	0.3 sec	0.54 sec	1.72 sec	2.24 sec	4.21 sec
1000 t.u.	0.62 sec	0.94 sec	4.27 sec	4.74 sec	8.57 sec
10 000 t.u.	1.72 sec	6 sec	42 sec	1 min 6 sec	1 min 58 sec
20 000 t.u.	5 sec	14.35 sec	1 min 25 sec	2 min 32 sec	4 min 40 sec
25 000 t.u.	7.23 sec	18.97 sec	2 min 41 sec	5 min 38 sec	11 min 20 sec
50 000 t.u.	41.67 sec	58.51 sec	6 min 38 sec	13 min 37 sec	24 min 11 sec

Table 4.1 – Variations in learning time depending on the size of the sample and the complexity of the model.

From these results we can conclude that the bigger the dimension and the sample size, the more time consuming the learning is, which is not surprising. The time complexity of the learning algorithm can be evaluated using the worst case iteration time (upper bound). An upper bound for an iteration in a Greedy Search algorithm for RTGEMs can be defined in terms of the model's specifications (structure and parameters) and the input sample size. The Greedy Search process for RTGEMs goes through all the possible operators and tests each operator. The testing of an operator consists in: virtually applying it (we do not keep it, in order to test the rest of the operators one by one and choose the best), updating the corresponding parameters (λ) and updating the BIC score (change the local score for the label concerned by the operator). In order to compute the complexity we must define a maximum number of possible operators for a given RTGEM. Let e be the number of existing edges in the model and n the number of existing nodes. Thus, the total number of possible edges is n^2 and the number of non-existing edges is $n^2 - e$. Let max_{TS} be the maximum number of time intervals on a timescale in the given RTGEM. Let the maximum number of possible operations

be N. N can be expressed in terms of e, n and max_{TS}. For each non-existing edge corresponds one possible operator (an add operator). Consider the worst case where each existing edge contains max_{TS} time intervals and that all time intervals are originally coming from the split operator (so we can reverse split them). In this case there are max_{TS} possible split operators plus $max_{TS} - 1$ possible reverse split operators and one possible extend operator. As a consequence the maximum number of possible operators can be written as follows:

$$N = (n^2 - e) + (2 \cdot e \cdot max_{TS})$$

The parameters update is relative to the computing of the sufficient statistics from the input data (see Chapter 2). Following the same logic as in the sampling complexity (see Chapter 2) for the upper bound, the most times we call the function to update the λ is defined by $t^* \cdot \lambda_{max}$. The update of the BIC score is done locally and in constant time. Therefore, the complexity can be expressed as $\mathcal{O}(N \times t^* \cdot \lambda_{max})$.
In Figure 4.1, we show plots representing the variation of the mean of the normalized distances in function of the size of the sample for different complexities. The standard deviation is indicated by the shadowed curve around the plot for each complexity. The normalized distance is the $SHD^*(reference, learned)$ divided by the maximum SHD^* for each batch of graphs that is equal to the square of the total number of nodes $|\mathcal{L}^2|$. We write $D(reference, learned) = \frac{SHD^*(reference, learned)}{|\mathcal{L}^2|}$.

The main observations that can be made based on this plot is the common trend between all graphs, from every complexity, to get closer to the reference model as the sample size increases. Moreover, we can see that the more complex the reference model is the harder it is to learn a model that is close to it (more data is needed). However, we can observe (mainly between $t^* = 10000$ and $t^* = 25000$) some fluctuations that are due to the learning of inaccurate models. We conjecture that this is happening when there is enough data to select an RTGEM (best BIC score with regards to data, see Chapter 3) that models inaccurate behaviors, but still not enough data to learn the accurate dependencies. We did not further explore explanations for this observation but we discuss it later in the perspectives.

An observation we have made while conducting the experiment, that could help conjecture the previous statement, is that for frequent events, the learning algorithm has

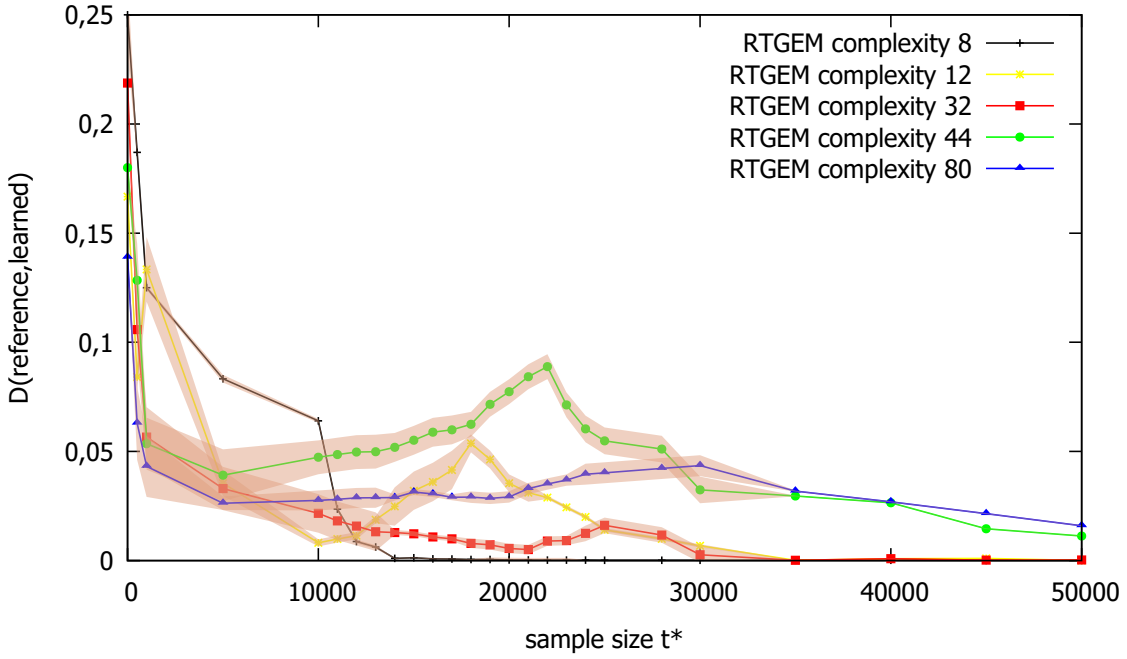


Figure 4.1 – Mean normalized distances between learned RTGEMs with different complexities and their corresponding references, in function of the sample size t^*.

a tendency to learn "self loops" (a loop on the concerned node itself) even if they do not exist in the reference model. As a consequence, knowing that the edges have the most effect on the distance function we use, we observe a significant increase in distance between the learned and the reference model for this scenario. However, this bias caused by the insufficiency of data is later on eliminated when we have more data (as shown in the plots).

Many other "bad" learnings, that we did not observe while conducting the tests, may have also occurred. In addition, many possible scenarios that we cannot intuitively imagine could have happened with other batches of randomly generated graphs. Furthermore, we have concluded from separate tests that in many cases correct edges are harder to learn than correct timescales. In other words, when we learn the correct dependency we often easily learn the correct timescales that go with it, but the correct dependency is not always easy to learn.

As previously mentioned, the objective of this section is not to discuss the flaws of

the learning algorithm but to evaluate its performance. The observations we discussed in the previous paragraph are not mathematically proven and correspond to a limited amount of tested graphs. Furthermore, there are many hyper parameters that were not taken into account when generating random models, such as the number of nodes (that is prefixed for each complexity) and the parameters λ that were carefully chosen to avoid detailed balance (see [GM16]). Detailed balance is the analogy of faithfulness for Dynamic Bayesian Networks (see [MR02]), it is when parameters are selected in a biased way for the reference model that they may obscure certain dependencies. In other words, when we sample from a reference model that contains detailed balance in order to to relearn the model, the learning procedure is not consistent, meaning that if the data size increases we do not guarantee a perfect learning. The last statement is due to the obscured dependencies (by the choice of parameters) that will not be learned. Thus, in order to make a complete study about the learning procedure and draw a valid conclusion, one should study every possible variation of hyper parameters.

Now that we have presented the results we obtained when evaluating the learning phase, we show results about the model exploration and formal verification phases.

4.2 Formal Verification and neighborhood exploration

In this section we will justify the choice and study the performances of the chosen exploration and verification techniques. We recall that after learning a model, if it does not satisfy a given property, we search in its delimited neighborhood for a model that does satisfy the property. The chosen verification technique for this work is Statistical Model Checking (SMC), which is a simulation based technique. The chosen search technique is the Random Walk, where a random operator is chosen and applied. Note that the operator is chosen from the set of allowed operators (see Chapter 3) that targets the concerned nodes.

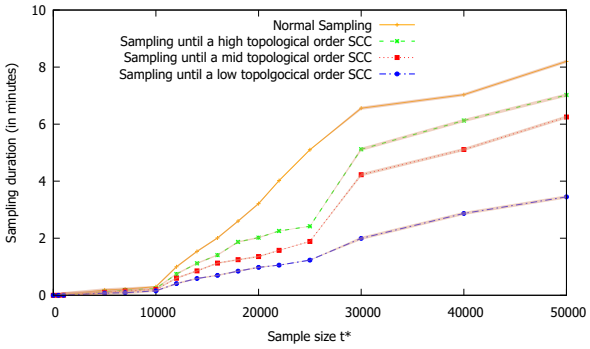
In the following, we show a benchmark for the different sampling performances by comparing the normal sampling with the "early stopping" sampling we proposed in Chapter 3, in order to only simulate the events of interest. Similarly to the previous test, batches of fifty RTGEMs for each fixed complexity were randomly generated and sampled using this technique. In this test we compare the mean duration of normal sampling of a set

of graphs and the mean duration of early stopping sampling of these graphs. Different stopping points are considered for the sampling with early stopping. The standard deviation is shown by the shaded curve.

Three early stop cases are considered for each set of models, one where the last concerned SCC has a low topological order meaning that the upper graph has a notably smaller size (in number of SCCs) than the lower graph (see Chapter 3). Another case is considered where the last concerned SCC has a mid topological order, i.e. the upper and the lower graph have approximately the same size. A final case is considered where the last concerned SCC to sample has a high topological order, i.e. the upper graph has a notably bigger size than the lower graph. Recall that cycles are the most time consuming to sample (from the definition in Chapter 2). In this experiment, we therefore made sure that the presence of cycles is balanced in the graph so that it does not bias the tests. In order to do so, we added a condition before accepting a randomly generated graph into the batch for the tests. In the added condition, we detect cycles in all SCCs going from the first topological order until the $k/2$ (or $k/2+1$) topological order, with k the highest topological order. We then compare the number of previously detected cycles with the number of detected cycles in all SCCs going from the $k/2$ (or $k/2+1$) topological order until the last topological order k. We reject each random graph that does not have the same number of cycles in each half. It was efficient in practice, in terms of rejection rate, since we also used pretested parameters in the random generator (e.g. a maximum number of allowed parents fixed to 1). The results are shown in the plots of Figure 4.2.

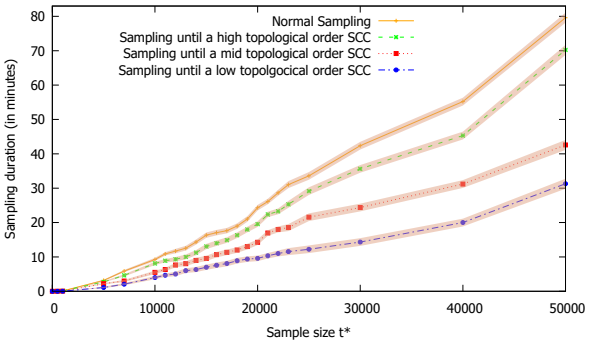
Note that the test for $t^* = 50000$ for the most complex graphs was omitted because of the high sampling duration.
We can conclude from the presented results that in most scenarios the sampling with early stopping offers a big advantage in terms of computation speed, in comparison to the normal sampling. The time saving gap is more visible when the targeted nodes are situated in the upper part of the DAG representing the SCCs of the model. Furthermore, there are no disadvantages of using an early stopping technique while sampling since we are only interested in the target nodes in this framework. We therefore use the early stopping sampling in the rest of this work.

Now that we have an insight about the sampling performances, we can test the simula-



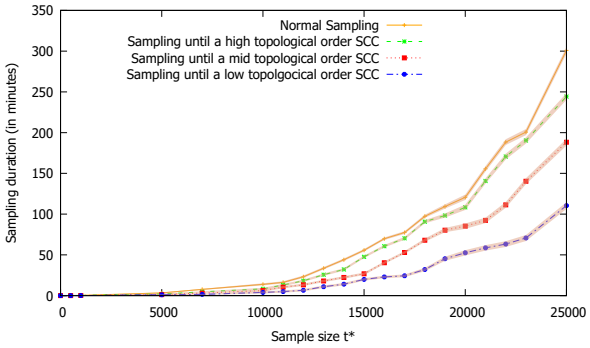
(a)

The mean sampling times between the normal stopping and the early stopping sampling for model complexity $\mathrm{Dim}(M) = 12$, in function of the sample size t^*.



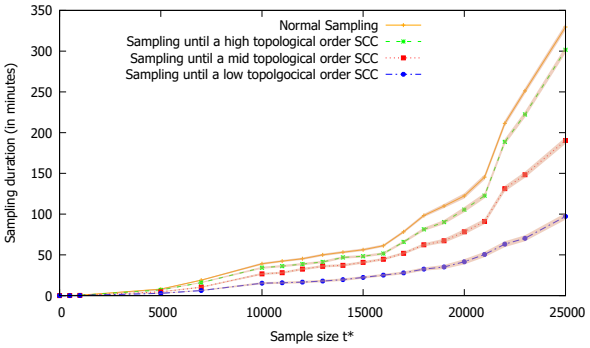
(b)

The mean sampling times between the normal stopping and the early stopping sampling for model complexity $\mathrm{Dim}(M) = 32$, in function of the sample size t^*.



(c)

The mean sampling times between the normal stopping and the early stopping sampling for model complexity $\mathrm{Dim}(M) = 44$, in function of the sample size t^*.



(d)

The mean sampling times between the normal stopping and the early stopping sampling for model complexity $\mathrm{Dim}(M) = 80$, in function of the sample size t^*.

Figure 4.2 – Plots comparing the mean sampling times between the normal stopping and the early stopping sampling forthe different model complexities, in function of the sample size t^*.

tion based Statistical Model Checking (SMC) performance on these models. To the best of our knowledge, there are no published works that adapt Statistical Model Checking to this type of formalism. As a consequence, we start by testing a simple property in Example 4.2.1 by using a quantitative approach (with Monte Carlo simulations) and a qualitative approach using the Sequential Probability Ratio Test (SPRT) defined in the following (also see Chapter 1).

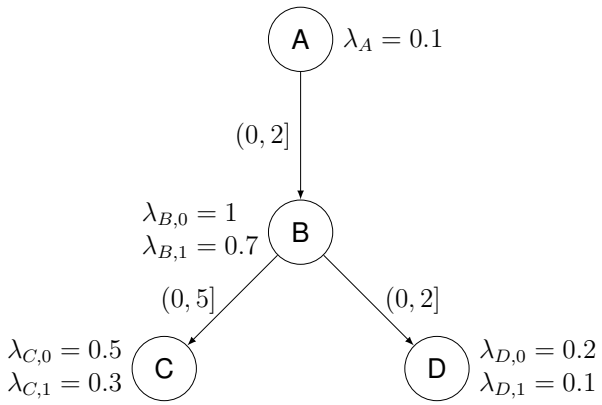


Figure 4.3 – The RTGEM M_1 on which we apply Statistical Model Checking to verify the query φ_1.

Example 4.2.1 *Consider the graph of the RTGEM M_1 in Figure 4.3 and the query $\varphi_1 = \Box^{100}(B[0, 5])$. We are looking to verify that for an event stream of 100 time units we have a B that occurs at least every 5 time units. We want the model to satisfy the security property $Pr(\varphi_1|M_1) > 0.80$. For the sake of coherence we always use the name security query for φ that is checked on the samples (traces) of the model, and the name security property for the qualitative inequality $Pr(\varphi|M) > c$ with M the model and c a threshold ($c \in [0, 1]$).*

The first test is for the *quantitative* approach based on Monte Carlo simulations. Consider a stochastic system $\mathcal{S}$, a formal property φ to check on the system and a probability γ. We write $Pr(\varphi \mid \mathcal{S}) = \gamma$ when a random execution of $\mathcal{S}$ has the probability γ to satisfy φ. When we apply SMC in practice, the probability γ is unknown and our aim is to compute it. However, we cannot compute the exact value of γ but an estimate $\widehat{\gamma}$ due to the nature of SMC that is an approximation method. We recall from Chapter 1, that the estimate is given with a precision δ, such that $|\widehat{\gamma} - \gamma| \leq \delta$ and $Pr(|\widehat{\gamma} - \gamma| > \delta) < \varepsilon$ with ε the probability of making an error. As a consequence, the result $\widehat{\gamma} \geq \theta + \delta$ means that the system $\mathcal{S}$ satisfies φ with a probability higher than θ and with a confidence $1-\varepsilon$.

The technique used for the approximation of γ in this framework is the Monte Carlo estimation. The Monte Carlo method consists in drawing n independent samples ω_i with $i \in \{1, ..., n\}$ from the stochastic system $\mathcal{S}$. Note that $\mathcal{S}$ must be purely stochastic and governed by a unique probability distribution. Let the Bernoulli variable defining the outcome of the samples with regards to φ be $\mathcal{B}$. Hence, the possible values for a

given sample (or trace) w_i are $b(w_i) = 0$ or $b(w_i) = 1$. We can write:

$$\widehat{\gamma} = \frac{1}{n}\sum_{i=1}^{n} b(w_i) \approx E[B]$$

with $\sum_{i=1}^{n} b(w_i)$ the number of times that $\mathcal{S}$ has satisfied φ (the number of successes).

We can see that this technique is straightforward and easy to apply. However, it requires a large number of simulations to become precise and reliable. The method we use in order to determine the required number of simulations is based on the work of Chernoff [Che+52] and Okamoto [Oka59]. Based on the previously cited works, we can define an Okamoto bound by:

$$(Pr(|\widehat{\gamma} - \gamma| > \delta) \leq 2e^{(-2n\delta^2)}$$

for any δ with $0 < \delta < 1$. Therefore, by fixing ε and δ, the Okamoto bound can determine a minimum number N of simulations required to accomplish the above described Monte Carlo estimation. From the Okamoto bound we have that $\varepsilon \leq 2e^{(-2n\delta^2)}$ and hence the formula we have been using in this work:

$$N \geq \frac{ln(2) - ln(\varepsilon)}{2\delta^2}$$

The quantitative Monte Carlo approach presented above computes an estimate $\widehat{p}(\varphi_1|M_1)$ to $Pr(\varphi_1|M_1)$ with a certain precision. Unfortunately, to the best of our knowledge there are no exact techniques that are adapted to this kind of formalism in order to compute the real probability $Pr(\varphi_1|M_1)$. Table 4.2 shows the results for this quantitative test.

	δ	ε	N	Duration	$\widehat{p}(\varphi_1\|M_1)$
Test 1	0.02	0.01	6622	5 min 54 sec	0.9177 (> 0.80 property satisfied)
Test 2	0.008	0.005	46808	52 min 18 sec	0.92179 (> 0.80 property satisfied)
Test 3	0.005	0.003	130044	2 hr 8 min	0.92182 (> 0.80 property satisfied)

Table 4.2 – Different results for quantitative Statistical Model Checking of query φ_1 on M_1 with Monte Carlo simulations.

Remark that although the property we are verifying is qualitative, it is also possible to apply a quantitative approach, have a numerical estimate (with a certain precision)

and afterwards compare the estimate with the desired threshold.
As expected, the results for this approach show that for a high precision (low δ) and a low relative error margin (low ε) the number of required simulations becomes high.

We perform a second experiment in the following to test the *qualitative* approach. The idea of taking observations into account when conducting a simulation based technique was first introduced in [DR29]. The Sequential Probability Ratio Test (SPRT) that we introduced in the literature review is based on the same logic, it minimizes the number of required simulations when it comes to hypothesis testing (see Chapter 1). The objective of this section is to give a detailed description of the algorithm we use to apply SPRT in our experiments.

Algorithm 4 describes the SPRT procedures for parameters α and β, and probabilities p_0 and p_1 the endpoints of the indifference region (see Chapter 1). We recall that the *strength* of the test is defined by the pair (α, β) of bounding errors. The notations m and f_m respectively define the number of simulations and the SPRT score. The score f_m is updated in function of the outcome ($\mathcal{B}_i$) of the Bernoulli variable $\mathcal{B}$ which describes the SMC procedure (an outcome of 1 for success and 0 for failure). Remark that in practice we use the $\log$ of the values in the tests to avoid handling large floats.

Algorithm 4 SPRT

input: $(p_0, p_1, \alpha, \beta)$
$m \longleftarrow 0, f_m \longleftarrow 0$
while $\log(\frac{\beta}{1-\alpha}) < f_m < \log(\frac{1-\beta}{\alpha})$
    $m \longleftarrow m + 1$
    $f_{m+1} \longleftarrow f_m + \mathcal{B}_i \log(\frac{p_0}{p_1}) + (1 - \mathcal{B}_i) \log(\frac{1-p_0}{1-p_1})$
if $f_m \leq \log(\frac{\beta}{1-\alpha})$
    return H_1
else
    return H_0

We assume that SPRT gives the optimal number of needed simulations [You05b] when it comes to qualitative SMC. More details about expected sample sizes and the optimality of sequential tests are found in [You05b].

For this example, we use a centered indifference region of size δ, and centered in

0.8 (the target value). The number of simulations is not fixed beforehand, and thus depends on the query and the outcome of the Statistical Model Checking on each simulation. In other words, the number of simulations tends to increase if the stopping criteria are hard to satisfy, and that is when the true probability of $Pr(\varphi_1|M_1)$ (that we are not explicitly computing in this approach) is inside the indifference region that we have set.

We apply the SPRT on the previous example knowing that it computes a decision about the security property $Pr(\varphi_1|M_1) > 0.8$. Table 4.3 shows the results for this qualitative test.

	δ	α	β	m	Duration	$Pr(\varphi_1\|M_1) > 80\%$
Test 1	0.02	0.01	0.01	167	8.18 sec	Property satisfied
Test 2	0.008	0.005	0.005	275	16.21 sec	Property satisfied
Test 3	0.005	0.003	0.003	344	41.29 sec	Property satisfied

Table 4.3 – Different results for qualitative Statistical Model Checking of query φ_1 on model M_1 with Sequential Probability Ratio Test.

We test the same approaches with a more complicated property in Example 4.2.2.

Example 4.2.2 *Consider the graph of RTGEM M_2 in Figure 4.4 and the query $\varphi_2 = \square^{100}(A \longrightarrow C[0,5] \cup D[0,8])$. We are looking to verify that over an event stream of 100 time units each time an A occurs, we should verify that we also have a C and a D respectively within 5 and 8 units in the past. We want the model to satisfy the security property: $Pr(\varphi_2|M_2) > 0.50$.*

The results of the application of the quantitative and qualitative approaches are shown respectively in Table 4.4 and Table 4.5.

	δ	ε	N	Duration	$\widehat{p}(\varphi_2\|M_2)$
Test 1	0.02	0.01	6622	5 min 31 sec	0.3978 (< 0.5 property NOT satisfied)
Test 2	0.008	0.005	46808	49 min 27 sec	0.417879 (< 0.5 property NOT satisfied)
Test 3	0.005	0.003	130044	2 hr 2 min	0.417789 (< 0.5 property NOT satisfied)

Table 4.4 – Different results for quantitative Statistical Model Checking of query φ_2 on M_2 with Monte Carlo simulations.

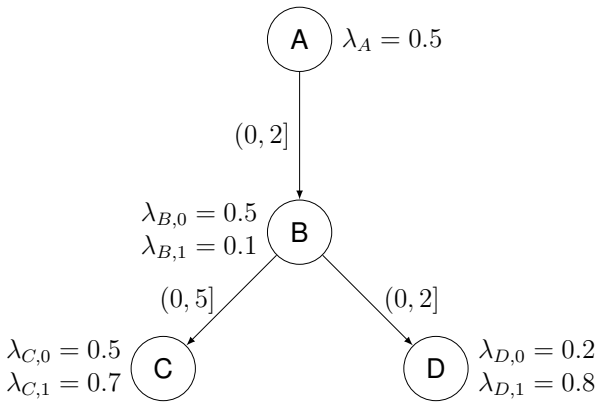


Figure 4.4 – The RTGEM M_2 on which we apply Statistical Model Checking to verify the query φ_2.

	δ	α	β	m	Duration	$Pr(\varphi_1 \| M_2) > 50\%$
Test 1	0.05	0.01	0.01	87	4.63 sec	Property NOT satisfied
Test 2	0.02	0.01	0.01	124	6.17 sec	Property NOT satisfied
Test 3	0.005	0.01	0.01	165	11.32 sec	Property NOT satisfied
Test 4	0.008	0.005	0.005	181	15.88 sec	Property NOT satisfied
Test 5	0.005	0.003	0.003	239	20.09 sec	Property NOT satisfied

Table 4.5 – Different results for qualitative Statistical Model Checking of query φ_2 on M_2 with Sequential Probability Ratio Test.

The number of simulations is independent from the property in the quantitative approach, and dependent on the property in the qualitative approach. Moreover, notice that in these tests we did not have any conflicting results between the two approaches.

Remark that the two approaches cannot be empirically compared. They are situational and each one is used for its own purpose. We previously mentioned that the properties we are interested in verifying are qualitative. In a qualitative approach, the number of simulations does not increase as fast as in a quantitative approach when we require higher "reliability" for the test.

However, we will use both techniques in parallel in the following experiments to avoid the excessive number of simulations in the qualitative approach when the true probability of the property is inside the indifference region. Accordingly, this allows to avoid a number of simulations that could be "useless" in the quantitative approach. As a consequence we make sure to combine the advantages of both approaches and that we

eventually converge to an estimate if the qualitative approach fails to give an "early decision" about the property. The algorithm is presented in the following.

We recall that SMC is the verification technique we use for formal verification of models in our strategy. Furthermore, as a space exploration technique the find function that we use in this framework is the Random Walk technique. The reason behind this choice is that it is very practical and instantaneous, we simply choose an operator between the allowed ones (from the allowed *level* and that are restricted by the targeted nodes) and apply it to the model. Since no score calculation is done nor any testing, the modifications are practically instantaneous. We do not further explore, in the scope of this work, if this has a negative effect on the quality of the search compared to other techniques. However, we know that it has the disadvantage of not being exhaustive, it does not explore the entire space.

4.3 Testing the proposed strategy

In this section we test the proposed algorithm on synthetic data. We build a pipeline to experimentally show that the proposed strategy attains the objectives discussed in Chapter 3. We first begin by describing the experimental protocol and recalling the objectives of this study. This section is divided into three parts, the first one is dedicated to the description of the pipeline, the second one is reserved for the results and the last one is for the interpretation and the discussion of the results.

4.3.1 Experimental protocol

The design of this experiment is introduced in what follows, and illustrated in Figure 4.5. This experiment is built in order to answer the following questions:

- Consider a set of data that is *statistically* secure (or not secure), with regards to a security property, i.e. the data represents a safe behavior (or a dangerous one). Does the learned model, from this data, satisfy (or not) the security property accordingly?
- Consider a model that does not verify the security property. Can we find in its neighborhood a model that does verify the property?

- Consider a model that is far, in probability, from satisfying a given security property. How hard is it to find a model in its neighborhood (if one exists) that satisfies the property?

For the sake of simplicity, we refer to the first test as the OK case, the second test as the KO_1 case and the third test as the KO_2 case. As shown in Figure 4.5 we begin our pipeline by applying the proposed algorithm on a set of synthetic data that was previously generated from a model that satisfies a security property with respect to a security query φ. The first step is the learning of the model that best represents the data for the OK case. We proceed by applying our chosen formal verification technique (SMC), as described in this chapter, on the learned model. We then run the space exploration technique on the learned model if it does not verify the property. If a model that verifies the security property is found in the neighborhood of the initial model, the SHD^* distance is then computed between the found model and the initial one (that best represents the data). For what follows, we use the name *proximal secure model* for a model that we find in the neighborhood of the learned model and that does verify the security property.

We execute the same procedure again on two separate sets of synthetic data that were previously generated from models that do not satisfy the security property (cases KO_1 and KO_2). The data that is used for learning the model in case KO_2 is supposed to be, in probability, further than the data that is used for learning the model in case KO_1 from verifying the security property. We note that for the sake of coherence, we sometimes use the notation M^* to denote the *proximal secure model*, via space exploration (if one exists).

The description of the models, the security property, the calibration of the Statistical Model Checking techniques and the results are shown in details in the next subsection.

4.3.2 Experimental results for the proposed strategy

The synthetic data sets used for learning the models contain event streams with labels "Login", "Logout", "Check account", "Recharge" and "Transfer money". These sets were generated from RTGEMs having different structures and parameters describing different behaviors on a prepaid card online service. The security query that we use in this experiment is: $\Box^{1000}(\text{Transfer Money} \Rightarrow \text{Recharge}_{(0,20]} \vee \text{Check account}_{(0,5]})$, in order

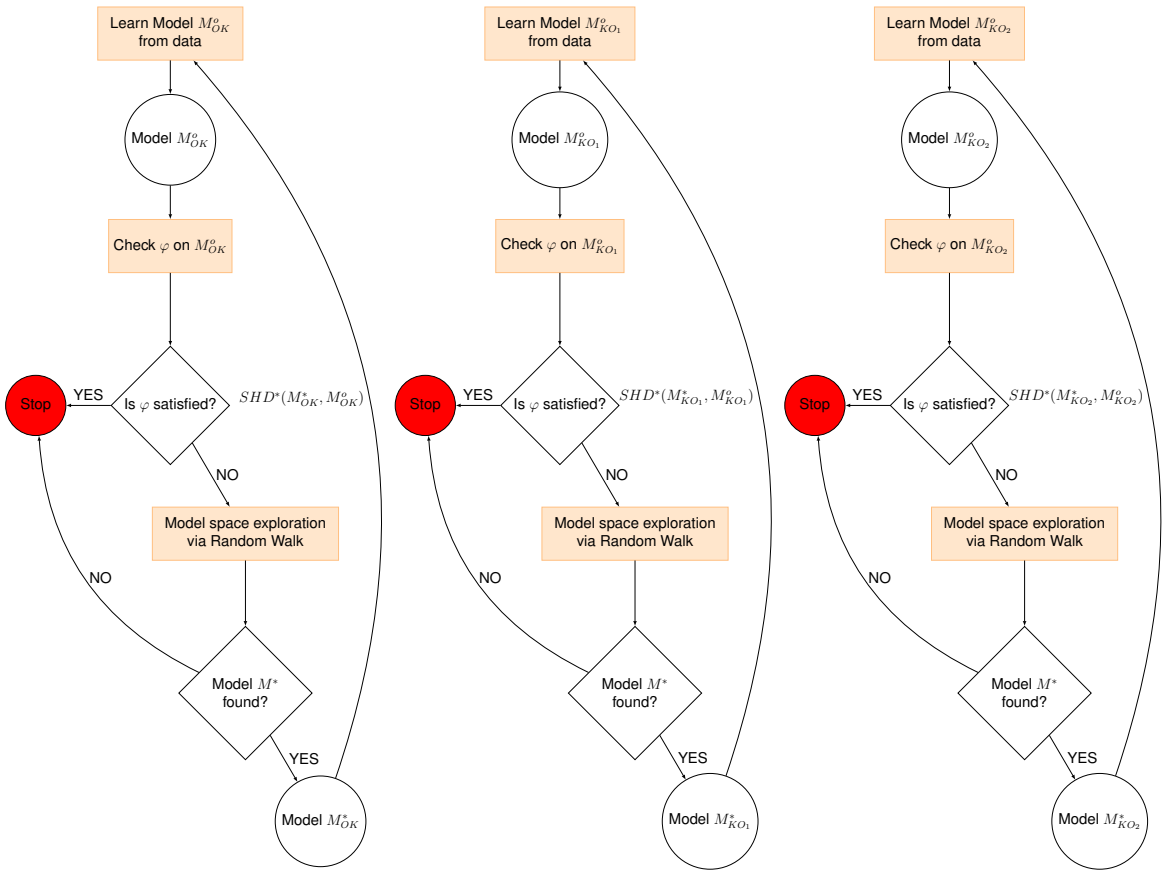


Figure 4.5 – An illustration of the experimental protocol that is conducted in this section.

to stay consistent with the toy example presented in Chapter 3. We also want the model to satisfy the security property: $Pr(\varphi|M) > 0.8$.

For the experiments, the Statistical Model Checking techniques that we use are: the qualitative Sequential Probability Ratio test and the quantitative estimation based on Monte Carlo simulations in parallel (as previously mentioned). We present the procedure in Algorithm 5. The idea is to have an additional break when we reach the number of simulations (n) computed by the calibration of the quantitative Monte Carlo estimation. By proceeding as such we make sure to converge to an estimated value ($\frac{s_m}{m}$) of the probability if the SPRT fails to gives an answer regarding the security property.

The SMC algorithms are calibrated as shown in Table 4.6. Remark that the maximum number of simulations that we do is determined by the number of simulations corresponding to the quantitative technique, in the case where SPRT does not converge faster to a result.

In the first test, we run our algorithm on the corresponding set of data to learn the

Algorithm 5 SPRT + Monte Carlo estimation

input: $(p_0, p_1, \alpha, \beta, \delta, \varepsilon)$
$m \longleftarrow 0, f_m \longleftarrow 0, n \longleftarrow \frac{ln(2)-ln(\varepsilon)}{2\delta^2}, s_m \longleftarrow 0$
while $(log(\frac{\beta}{1-\alpha}) < f_m < log(\frac{1-\beta}{\alpha}))$ && $(m < n)$
  $m \longleftarrow m + 1$
  $f_{m+1} \longleftarrow f_m + \mathcal{B}_i log(\frac{p_0}{p_1}) + (1 - \mathcal{B}_i) log(\frac{1-p_0}{1-p_1})$
  $s_{m+1} \longleftarrow s_m + \mathcal{B}_i$
if $f_m \leq log(\frac{\beta}{1-\alpha})$
  return H_1
else if $m >= n$
  return $\frac{s_m}{m}$
else
  return H_0

	α	β	δ	ε	Number of simulations
SPRT	0.05	0.05	0.01	-	Unknown beforehand
Monte Carlo simulations	-	-	0.01	0.05	18444

Table 4.6 – Calibration of the SMC techniques for the following experiments.

model M^o_{OK}. The obtained model is shown in Figure 4.6. For the sake of simplicity we do not represent the parameters of the label "Logout" for all of the remaining figures, since it is not a target node with regards to φ.

In Table 4.7, we show the obtained results after the completion of the first test. The formal verification with SMC is repeated twenty times in order to establish mean values for the required duration and number of simulations before acquiring a result. We write $\widehat{t}$ for the mean duration and $\widehat{m}$ for the mean number of simulations over twenty executions for all remaining tests.

	$Pr(\varphi\|M) > 0.8$	$\widehat{t}$	$\widehat{m}$
M^o_{OK}	Property satisfied	5 sec (± 0.43 sec)	277 simulations (± 49 simulations)
M^*_{OK}	-	-	-

Table 4.7 – Results of the first test on M^o_{OK}.

The learned model M^o_{OK}, that best represents the data, verifies the security property in this case. Hence, no further exploration is made and the test stops. We recall that all the experiments are performed on a Windows 10 Enterprise i5 laptop, with a

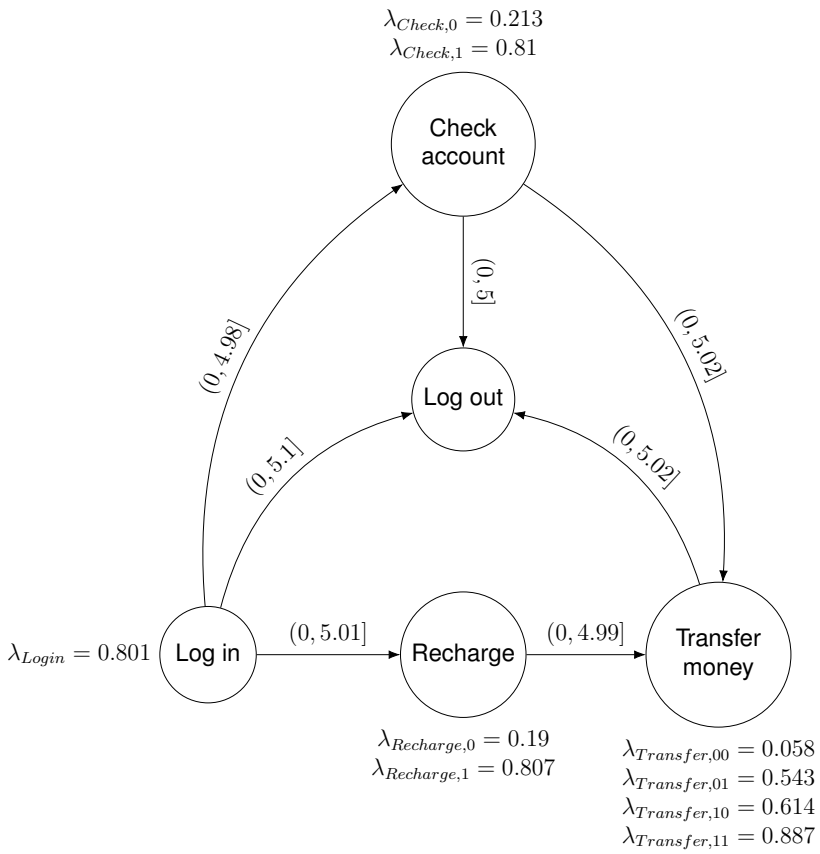


Figure 4.6 – A representation of the learned model M^o_{OK} from the set of "secure behavior" data.

2.3 GHz CPU and 8GB of RAM. The RTGEM learning and sampling are coded in C++ under the main PILGRIM private library, consequently the code is not accessible. The verification algorithms and the proposed strategy in this framework are coded on a separate branch also using C++ and inheriting from the PILGRIM library. The complexity of the used SMC algorithm (previously defined in this chapter) is the same as the complexity of the sampling (see Chapter 2). The data that is generated during the SMC process (samples for instance) is non-persistent.

For the second test, we run our algorithm on the corresponding set of data to learn the model $M^o_{KO_1}$. The obtained model is shown in Figure 4.7. The model $M^o_{KO_1}$ does not satisfy the security property. Hence, the space exploration technique is repeated twenty separate times in this test, in order to check whether every time we find a model that satisfies the property. Similarly, we check whether it is the same proximal secure model that is found every time or if there are different possible ones.

The space exploration is executed on the learned model $M^o_{KO_1}$ and only one proximal secure model ($M^*_{KO_1}$) is found in its neighborhood (see Figure 4.8). As previously mentioned, we use the Random Walk technique to explore the neighborhood of the learned model. Based on preliminary experiments, a compromise number of twenty five allowed modifications (chosen operators) is set for this test. The allowed operators are *level 0* and *level 1* operators (see Chapter 3). Remark that the proximal secure model changes reality to a certain degree by modifying the learned model (that degree is defined by the SHD^* distance with the learned model). We do not seek to explain the structure or the dependencies of the proximal secure model, the essential idea to withhold is that it is a combination of parameters (a mathematical object) that represents (and can simulate) safe behavior. So the idea of the space exploration is to find a mathematical object, a combination of parameters, that provide samples of behaviors which are "secure" with regards to the property, but that may not make sense with regards to reality.

In Table 4.8, we show some results of the algorithm for this test. We write $\widehat{o}$ for the average number of applied modifications before finding a proximal secure model. We write $\widehat{\tau}$ the average duration of the space exploration process (plus the formal verification at each step), and n for the number of distinct proximal secure models found. The success rate is the number of times we found a proximal secure model over the total number of executed Random Walks. We use the same notations for the remaining test.

	$\widehat{o}$	$\widehat{\tau}$	n	Success Rate	$SHD^*(M^*_{KO_1}, M^o_{KO_1})$
$M^o_{KO_1}$	12 operators (± 2 operators)	13 min 52 sec (± 30 sec)	1	$20/20$	1

Table 4.8 – Some performances of the algorithm applied on the second set of data (see Figure 4.5).

The last results for this test are shown in Table 4.9 after the completion of the experiment. The formal verification with SMC is repeated twenty times for both $M^o_{KO_1}$ and $M^*_{KO_1}$ in order to establish mean values for the required duration and number of simulations before converging on a result. Note that the SMC was executed only one time for all the intermediate models obtained at each step of the exploration process

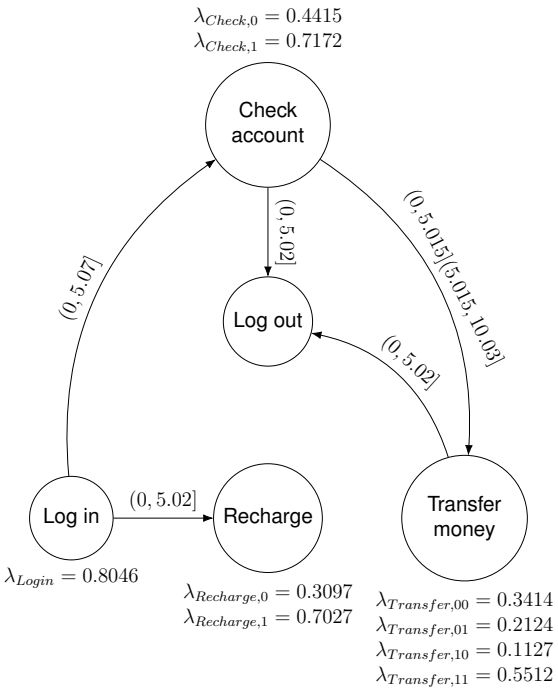


Figure 4.7 – A representation of the learned model $M^o_{KO_1}$ from the corresponding set of "not secure" data.

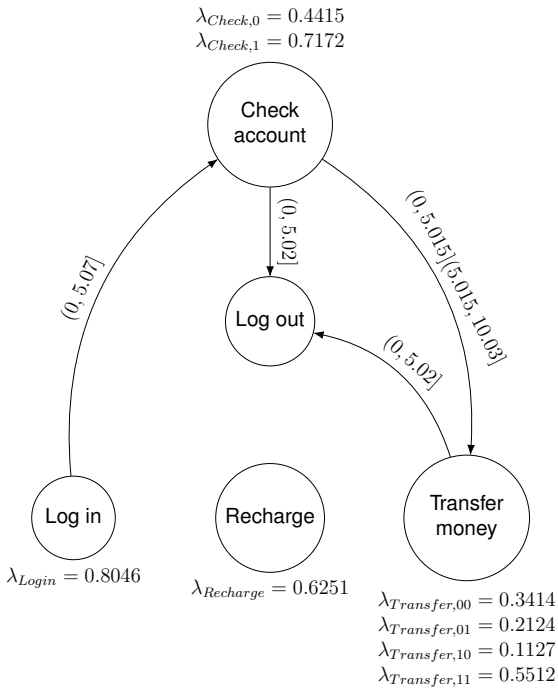


Figure 4.8 – A representation of the found model $M^*_{KO_1}$ which satisfies the query φ in the neighborhood of $M^o_{KO_1}$.

while constructing $M^*_{KO_1}$.

	$Pr(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_1}$	Property NOT satisfied	1 min 2 sec (± 11 sec)	4018 simulations (± 101 simulations)
$M^*_{KO_1}$	Property satisfied	2 min 17 sec (± 25 sec)	5512 simulations (± 122 simulations)

Table 4.9 – Results of the second test on $M^o_{KO_1}$.

For the third and final test, we run our algorithm on the corresponding set of data to learn the model $M^o_{KO_2}$. The obtained model is shown in Figure 4.9. The model $M^o_{KO_2}$ does not satisfy the security property. Hence, the space exploration technique is repeated twenty separate times in order to check whether every time we find a model that satisfies the property. Similarly, we check whether it is the same proximal secure model that is found every time or if there are different possible ones.

The space exploration is executed on the learned model $M^o_{KO_2}$ and two proximal secure models ($M^*_{KO_{2_1}}$ and $M^*_{KO_{2_2}}$) are found in its neighborhood (see Figure 4.10 and Figure 4.11). Based on preliminary experiments, a compromise number of one hundred allowed modifications in the Random Walk (chosen operators) is set for this test.The

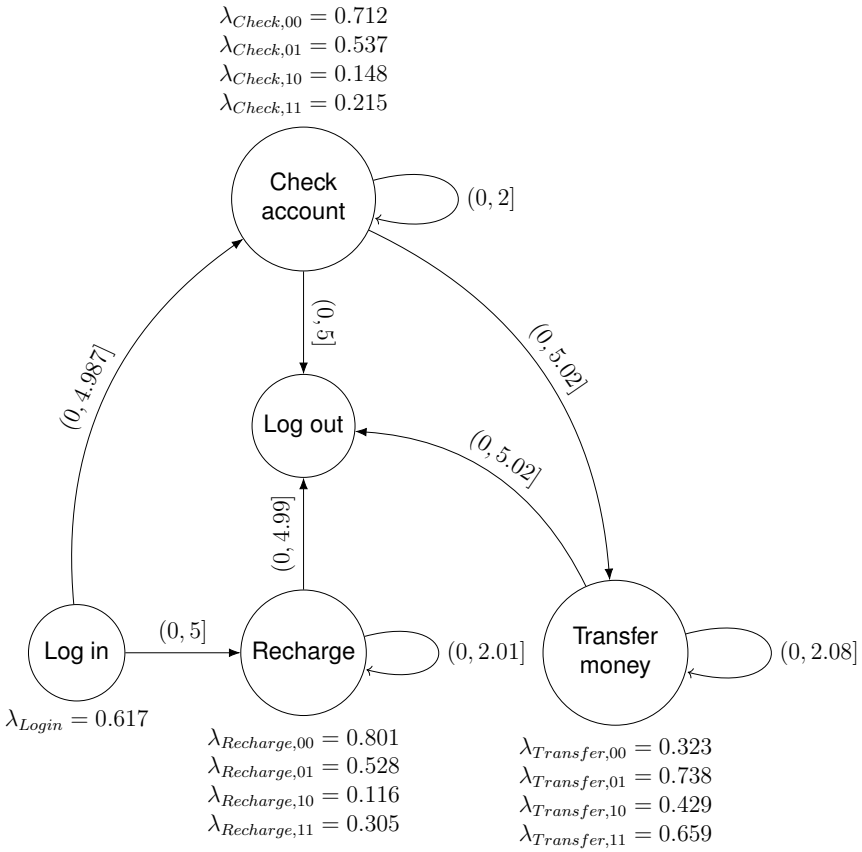


Figure 4.9 – A representation of the learned model $M^o_{KO_2}$ from the corresponding set of "not secure" data.

allowed operators are *level 0*, *level 1* and *level 2* operators (see Chapter 3).

In Table 4.10, we show some results of the algorithm for this test. We use the same notations as before, however for the distance we show the two distance measures with respectively the first and the second found proximal secure model. Notice that the success rate is not perfect in this test, three times out of twenty we did not find a proximal secure model (failure rate of $3/20$). The proximal secure model $M^*_{KO_{2_1}}$ (Figure 4.10) is found twelve times and the proximal secure model $M^*_{KO_{2_2}}$ (Figure 4.11) is found five times.

	$\widehat{o}$	$\widehat{\tau}$	n	Success Rate	$SHD^*(M^*_{KO_2}, M^o_{KO_2})$
$M^o_{KO_2}$	88 operators (± 8 operators)	1 hr 15 min (± 9 min)	2	$17/20$	3 - 3.333

Table 4.10 – Some performances of the algorithm applied on the third set of data (see Figure 4.5).

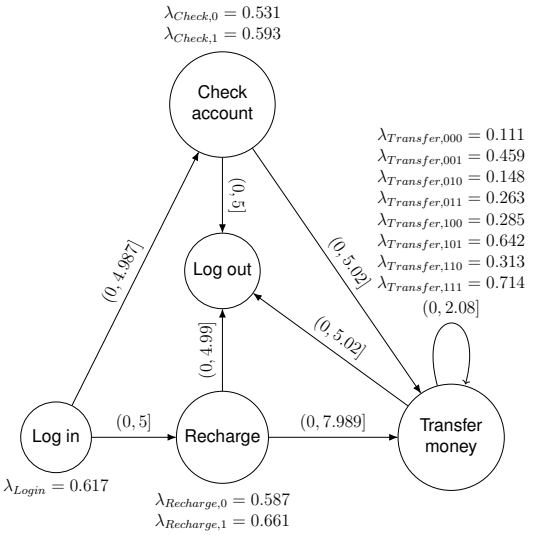


Figure 4.10 – A representation of $M^*_{KO_{2_1}}$ which satisfies the security property in the neighborhood of $M^o_{KO_2}$.

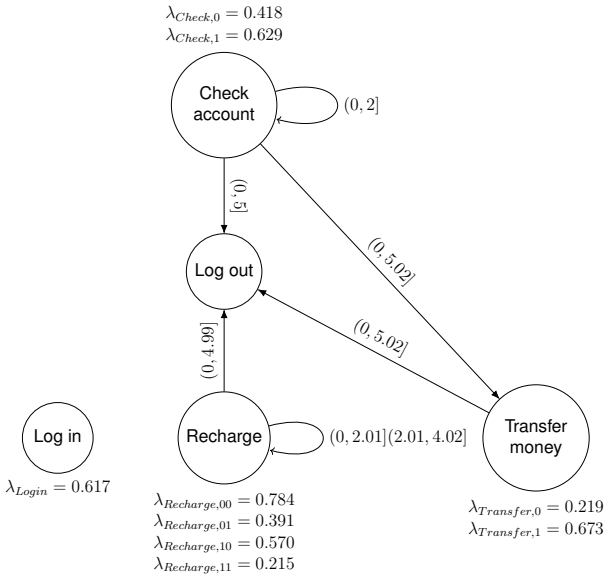


Figure 4.11 – A representation $M^*_{KO_{2_2}}$ which satisfies the security property in the neighborhood of $M^o_{KO_2}$.

The last results for this test are shown in Table 4.11. The formal verification with SMC is repeated twenty times for $M^o_{KO_2}$ and both $M^*_{KO_{2_1}}$ and $M^*_{KO_{2_2}}$ in order to establish mean values for the required duration and number of simulations before acquiring a result. Note that also in this test, the SMC was executed only one time for all the intermediate models obtained at each step of the exploration process.

	$Pr(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_2}$	Property NOT satisfied	3 min 22 sec (± 11 sec)	2155 simulations (± 69 simulations)
$M^*_{KO_{2_1}}$	Property satisfied	11 min 31 sec (± 25 sec)	7512 simulations (± 136 simulations)
$M^*_{KO_{2_2}}$	Property satisfied	7 min 19 sec (± 25 sec)	4286 simulations (± 114 simulations)

Table 4.11 – Results of the third test on $M^o_{KO_2}$.

Now that we have shown the different metrics and results we obtained from running the pipeline we described in Figure 4.5, we proceed to the discussion and interpretation of the results.

4.3.3 Discussion and interpretation of the results

In the OK case, we notice that the SPRT technique quickly converged to an answer, which means that the model satisfies the security property with high probability

(significantly higher than the threshhold of 0.8). When we verify that the model satisfies our security property we stop the procedure since we do not need to explore the neighborhood of the learned model.

In the KO_1 case, the SPRT took longer to give a result because the model was not very far in probability from satisfying the property. In other words, SPRT requires a low number of simulations and converges quickly when the probability that the model satisfies the query is far from the given threshold. This explains why we assumed for KO_1 case that the model is not very far from verifying the property in contrast to the KO_2 case that is far from verifying the property. Therefore, in order to be sure that the latter statement is true, we applied the quantitative Monte Carlo simulations technique (calibrated as previously shown in this chapter) to calculate approximates of the exact probabilities. We obtained the following values $\widehat{p}(\varphi \mid M^o_{KO_1}) = 0.728$ and $\widehat{p}(\varphi \mid M^o_{KO_2}) = 0.585$.

A space exploration was then initiated twenty separate times in the objective of finding a close model that does verify the property. The same model was found twenty times, and is at a distance of 1 of the learned model. We cannot conjecture on the level of dangerosity of the model by only having one reference distance. We need to have more tests in order to define whether a distance of 1 is a big or a small distance. As a consequence we performed a last test where we learn a model that is in probability further away from satisfying the property. The experimental hint that the KO_2 case is further in probability than the KO_1 case is that the SPRT technique converged quicker to a negative result (as previously mentioned).

We recall that we do not seek to intuitively (based on the structure of the model itself) find an explanation to the reason why the models we find satisfy the property. What matters most is that they are mathematical objects that offer samples of "safe" behaviors, and from which we can compute a distance measure from the fittest model to asses the dangerousness level. In the last case, we were lucky to find two models in the neighborhood of the learned model that satisfy the property. We found one model more frequently than the other, it was the one with the closer distance. We cannot be sure that we always find the closest model more often (or first), since we use a Random Walk technique. In the next chapter, we discuss in the perspectives other techniques that can guarantee the finding of the closest model if one exists. We recall that all the

found proximal secure models M^* changes reality to a certain level and that we seek to find mathematical objects that provide samples of behaviors which are "secure" with regards to the property.

We now have all the observations needed to answer the questions we asked in the beginning of this chapter. Based on the conducted experiments, we can see that if the data comes from a secure (or not) behavior, the learned model is also checked as secure (or not) accordingly. We can also see that we have found for both the KO_1 and KO_2 cases, models in their respective neighborhood that verify the security property. This last statement allows us to assume that it is possible to find "secure" models by exploring nearby models. In addition, we have shown that more than one secure model can be found by the space exploration method for the same learned model. And finally, by comparing the last two cases, we can see that the process of finding a secure model becomes harder and more time consuming (88 modifications versus 12 modifications on average) when the model is further in probability from satisfying the property.

We can also deduce, by comparing the distances $SHD^*(M^*, M^o)$ in the last two cases, that the third case corresponds to a more dangerous behavior than the second case. This comparison allows us to give more meaning to the distance measure. On a side note, we can see that the models we handled in case KO_2 are not very complex yet the procedure can become costly. This is mainly due to the Forward sampling technique for RTGEMs that can quickly become time consuming. We also discuss in the next chapter perspectives on how to reduce this cost.

Finally, note that it is irrelevant to cross-compare the SHD^* distances between the models obtained from different tests (compute the distance $SHD^*(M^*_{KO_2}, M^o_{KO_1})$ for instance) in order to try to find a certain correlation. A very simple counterexample can be found by imagining two very distant models that both satisfy the same security property. The distance measure is not in any way related to the security property. The distance measure is only relevant in the scope of the same test when comparing the reality (fittest model) to the model that satisfies our "standards". This allows us have an insight about the danger level of the real behavior by showing how far it is from verifying the security rules.

4.4 Conclusion

We recall that the main objective of this thesis is to bring closer together the model based learning and the formal verification fields of study, by using the advantages of both to elaborate a useful application. To the best of our knowledge, what we have proposed and presented in the last two chapters is a novel strategy to *measure* levels of dangerous behaviors in event streams. This strategy is based on the learning and manipulation of Graphical Event Models and the adaptation of a formal verification technique. Furthermore, we conducted experiments in order to test the quality of the proposed algorithm and answered questions related to its performance.

We previously justified in Chapter 3 the choices of formalisms and techniques we adopt in the different steps of the proposed algorithm. In this chapter, we discussed the performance of the adopted techniques and formalisms. We separately tested and evaluated each step of the proposed algorithm on RTGEMs. In addition, we have built an experiment to validate the proposed strategy and to study its performance.

To end this chapter, we can draw two main conclusions about the RTGEMs and the strategy we proposed. The first one concerns the flexibility and genericity, which are positive traits of our strategy, that are introduced by the use of RTGEMs. The RTGEM formalism is easy to learn and to manipulate. As we have seen in the experiments, their only downside is the high cost of the sampling. We use a simulation-based technique for formal verification so we are confronted with a high number of simulations, hence a lot of computing time. However, we have shown a way of notably reducing this effect, when we are applying Statistical Model Checking, by using an early stopping criteria for the sampling.

The second conclusion we can draw is about the strategy. When we apply the algorithm on the different proposed cases, aside from the main measure of distance that we aim to compute, we have shown that many metrics and measures can potentially be evaluated. As an example one can measure the number of different proximal secure models M^* that can be found in the defined neighborhood, each at a different distance. In addition, we note that the difficulty of finding a *proximal secure model* can significantly increase when the model is "dangerously" further from satisfying the security

property.

In the following we present perspectives for future works, and a global conclusion to this work.

CONCLUSION

In this work we have established a solid basis for working with Graphical Event Models for security assessments. And most importantly we have accomplished our goal of combining the application of the probabilistic graphical learning and the formal verification fields. The main problematic of this thesis is to use data generated from a certain application in order to learn a model of behavior that is at the same time best representative of the data, and safe from a security point of view. We have managed to reach this goal in practice, by providing experimental evidence that if the "fittest" learned model does not satisfy a given security property, we are able to find in its neighborhood (if one exists) a *proximal secure model* that does satisfy the security property. A distance measure is adapted in order to compare the *proximal secure model* (if found) and the fittest model. This measure gives an insight about how far the learned behavior is from a secure behavior. The previous metric can then be used to rate the level of dangerousness of the real behavior. In what follows, we sum up the literature review and the contributions of this thesis, and we present perspectives for future works.

In the first two chapters of this thesis we gave an introductory state of the art review, showing the advantages and disadvantages of the discrete time probabilistic graphical modeling formalisms. The presented formalisms inspired this work and due to their lack of time-related-information we have been pushed to explore continuous time probabilistic graphical modeling formalisms. We have also discussed that continuous time formalisms offer more benefits from a security point of view with regards to the data type (event streams). In addition, we have presented Model Checking and Statistical Model Checking (SMC) as formal verification techniques. Based on our desired objectives, we have justified the use of Recursive Timescale Graphical Event Models (RTGEMs) as a learning/modeling formalism and Statistical Model Checking (SMC) as a formal verification technique.

In the last two chapters, we formalized the problem and proposed a solution in the form of a generic algorithm. We then justified the choice of the methods and techniques

used to fulfill the instructions provided by the algorithm. We chose the following combination of techniques: RTGEMs as a modeling formalism, due to their easy learning (compared to other continuous time formalisms) and manipulability; Statistical Model Checking as a formal verification technique; the Random Walk as a space exploration technique and the *proximal distance measure* as a measuring method. We have also built a pipeline of experiments in order to provide evidence that the work is complete. We provided answers to many questions that can resume the results we obtained by performing the proposed experimental protocol. We recall that no further theoretical proof has been established in order to formally demonstrate the following results and that they are purely based on the tests we have performed. The answers to our questions that are also capable of summing up our results are listed below:

- With exemption of detailed balance, the bigger the data size the more accurate the learning (see Chapter 4 and [GM16]).
- Excessive sampling of an RTGEM is costly especially if it contains cycles, but we use targeted sampling with early stopping (as proposed in Chapter 3) when possible in order to reduce the computing time.
- If the input data we use represents a safe behavior (or not) with regards to the security property, then the learned model satisfies the security property when formally checked (or not) accordingly.
- If the learned "fittest" model does not satisfy the given security property, a model can be found in its neighborhood (sometimes after many instances of the search technique) that does satisfy the property (named proximal secure model).
- The further the fittest model is, in probability, from satisfying the security property the harder it is to find a proximal secure model in its neighborhood.
- The distance measure, used for rating the dangerousness of the fittest model by comparing it to the proximal secure model, increases when the fittest model is, in probability, far from satisfying the property.
- Sometimes while executing a Random Walk instance we do not find a proximal secure model, in particular for more dangerous models.

In the last section of this thesis, we discuss perspectives in order to improve the current work.

Perspectives

To bring this work to a final point, we present several perspectives that we envisaged for future works. To begin with, the learning process of an RTGEM could be explored in more details. In other words, we noticed in our tests some odd fluctuations while in the learning process (see Chapter 4). In order to explain this impression we tried to build some small tests and realized that bigger tests must be done. These tests should include the variation of the different possible hyper parameters when randomly constructing a graph (number of nodes, the parameters, the horizons, number of edges, number of timescales, density of timescales, etc.) in order to have a complete diagnostic for a better understanding. The first basic confronted difficulty in such a procedure is the big cost of these operations. Another difficulty is the ability of finding explanations/correlations while avoiding representational biases. In other words, the choice of parameters and timescales on reference models must be done in a way not to obscure certain dependencies (see Chapter 1 and detailed balance in [GM16]).

A perspective that is more directly related to the proposed strategy targets the Forward Sampling technique. As we have seen in Chapter 4, the sampling of RTGEM could become costly even with the early stopping rule we proposed. Consequently, an approach where we can abstract and approximate the totality of the effects that the parents have on the targeted nodes can be considered. In other words, to have only one parameter $\lambda_{l_\varphi, effects}$ for each concerned label $l_\varphi \in L_\varphi$, that is computed with a certain heuristic with regards to all the parents $Pa_\varphi(l_\varphi)$. A method to accomplish that is to disrupt dependencies (or remove them) in a way that can give us the desired structure that we discussed above. By proceeding as such, we transform a given model M to M' on which we can relearn the desired "abstracted" parameters using the same input data that was used for learning the model M. The difficulty is to be able to find a formal demonstration that the model M' is a similar mathematical object to the model M to a certain extent (or a precision measure to evaluate how much they are the same), from a sampling point of view.

Another perspective concerning the space exploration technique can also be envisioned. In this work we chose the Random Walk technique as an exploration method due to its simplicity and relative efficiency. A more complete method could be used,

like an exhaustive Breadth First Search (or Depth First Search) which guarantees the exploration of all the possibilities. However, this technique could become costly if we do not consider a smart approach where we create a caching memory to store the already explored RTGEMs in the neighborhood. The reason behind that is the usage of operators and inverse operators for modifications. For instance, we can undo a forward operator by doing a backward operator to fall back on a previously tested model, and vice versa. A second space exploration technique can also be tested: A Greedy Search method that would consider improving the security property in probability. In other words, a quantitative SMC technique should be associated with the greedy search in order to compute the approximate probability of satisfying the security query for a model after each possible modification. When a model that improves the probability of satisfying the security property is found, it is selected as a new reference and the search continues. Although we are not sure about whether it is better than the exhaustive search or the Random Walk, we think it is an interesting approach to explore.

Finally, one last goal that we consider achieving in the near future is an experimentation with real world data. Due to accessibility issues to real world faithful data, we were not able to perform this kind of experiment yet. It is something we are planning to do with GFI informatique on different use cases, particularly on data generated from a road safety application. An approach for multi-task transfer learning for Timescale Graphical Event Models that can be combined in practice with our proposed strategy for this kind of application can be found in [ML19]. In other words, since the different models of behavior are somehow supposed to be *close* in practice, transfer learning by using previous information in order to do Model Checking might be accurate. Although we think that there is a lot of data treatment to do in order to adapt it to the required format, we think that our methods are mature enough to be now used in this context.

Bibliography

[Abo+15] Mohamed Abomhara et al., « Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks », *in*: *Journal of Cyber Security and Mobility* 4.*1* (2015), pp. 65–88.

[ADL19] Dimitri Antakly, Benoit Delahaye, and Philippe Leray, « Graphical event model learning and verification for security assessment », *in*: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 2019, pp. 245–252.

[AGW90] Richard Arratia, Louis Gordon, and Michael S Waterman, « The Erdos-Rényi law in distribution, for coin tossing and sequence matching », *in*: *The Annals of Statistics* (1990), pp. 539–570.

[AK86] Krzysztof R. Apt and Dexter Kozen, « Limits for automatic verification of finite-state concurrent systems », *in*: *Inf. Process. Lett.* 22.*6* (1986), pp. 307–309.

[Alb+08] Massimiliano Albanese et al., « A constrained probabilistic petri net framework for human activity detection in video », *in*: *IEEE Transactions on Multimedia* 10.*8* (2008), pp. 1429–1443.

[Azi+96] Adnan Aziz et al., « Verifying continuous time Markov chains », *in*: *International Conference on Computer Aided Verification*, Springer, 1996, pp. 269–276.

[Bai+04] Christel Baier et al., « Model checking action-and state-labelled Markov chains », *in*: *International Conference on Dependable Systems and Networks, 2004*, IEEE, 2004, pp. 701–710.

[Bas+10] Ananda Basu et al., « Statistical abstraction and model-checking of large heterogeneous systems », *in*: *Formal Techniques for Distributed Systems*, Springer, 2010, pp. 32–46.

[BG14] Grant Blank and Darja Groselj, « Dimensions of Internet use: amount, variety, and types », *in*: *Information, Communication & Society* 17.*4* (2014), pp. 417–435.

[Bie+03] Armin Biere et al., « Bounded model checking. », *in*: *Advances in computers* 58.*11* (2003), pp. 117–148.

[BK08] Christel Baier and Joost-Pieter Katoen, *Principles of model checking*, MIT press, 2008.

[BK98] Falko Bause and Pieter S. Kritzinger, « Stochastic Petri nets: An introduction to the theory », *in*: *ACM SIGMETRICS Performance Evaluation Review* 26.*2* (1998), pp. 2–3.

[BLN04] Gely P. Basharin, Amy N. Langville, and Valeriy A. Naumov, « The life and work of AA Markov », *in*: *Linear algebra and its applications* 386 (2004), pp. 3–26.

[Bra83] G.W. Brams, *Réseaux de Petri: théorie et pratique*, vol. 2, Masson, 1983.

[Bré13] Pierre Brémaud, *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, vol. 31, Springer Science & Business Media, 2013.

[BS518] Debarun Bhattacharjya, Dharmashankar Subramanian, and Tian Gao, « Proximal graphical event models », *in*: *Advances in Neural Information Processing Systems*, 2018, pp. 8136–8145.

[CDL08] Edmund Clarke, Alexandre Donzé, and Axel Legay, « Statistical model checking of mixed-analog circuits with an application to a third order Δ-Σ modulator », *in*: *Haifa Verification Conference*, Springer, 2008, pp. 149–163.

[CE81] Edmund M. Clarke and E. Allen Emerson, « Design and synthesis of synchronization skeletons using branching time temporal logic », *in*: *Workshop on Logic of Programs*, Springer, 1981, pp. 52–71.

[CFN04] Wai Ki Ching, Eric S. Fung, and Michael K. Ng, « Higher-order Markov chain models for categorical data sequences », *in*: *Naval Research Logistics (NRL)* 51.*4* (2004), pp. 557–574.

[Cha91] Eugene Charniak, « Bayesian networks without tears. », *in*: *AI magazine* 12.*4* (1991), pp. 50–50.

[Che+52] Herman Chernoff et al., « A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations », *in*: *The Annals of Mathematical Statistics* 23.*4* (1952), pp. 493–507.

[Che80] Brian F. Chellas, *Modal logic: an introduction*, Cambridge university press, 1980.

[Chi02] David Maxwell Chickering, « Optimal structure identification with greedy search », *in*: *Journal of machine learning research* 3.*Nov* (2002), pp. 507–554.

[CHM97] David Maxwell Chickering, David Heckerman, and Christopher Meek, « A Bayesian approach to learning Bayesian networks with local structure », *in*: *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1997, pp. 80–89.

[Cor+01] Thomas H. Cormen et al., « Introduction to algorithms second edition », *in*: *The Knuth-Morris-Pratt Algorithm, year* (2001).

[CR01] Antonio F.B. Costa and M.A. Rahim, « Economic design of X charts with variable parameters: the Markov chain approach », *in*: *Journal of Applied Statistics* 28.*7* (2001), pp. 875–885.

[DDE05] Pierre Dupont, François Denis, and Yann Esposito, « Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms », *in*: *Pattern recognition* 38.*9* (2005), pp. 1349–1371.

[De 10] Colin De la Higuera, *Grammatical inference: learning automata and grammars*, Cambridge University Press, 2010.

[Des+04] Josée Desharnais et al., « Metrics for labelled Markov processes », *in*: *Theoretical computer science* 318.*3* (2004), pp. 323–354.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y Vardi, « Improved automata generation for linear temporal logic », *in*: *International Conference on Computer Aided Verification*, Springer, 1999, pp. 249–260.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin, « Maximum likelihood from incomplete data via the EM algorithm », *in*: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.*1* (1977), pp. 1–22.

[DR29] Harold French Dodge and H.G. Romig, « A method of sampling inspection », *in*: *The Bell System Technical Journal* 8.*4* (1929), pp. 613–631.

[DSA11] Rónán Daly, Qiang Shen, and Stuart Aitken, « Learning Bayesian networks: approaches and issues », *in*: *The knowledge engineering review* 26.*2* (2011), pp. 99–157.

[DV07] Daryl J. Daley and David Vere-Jones, *An introduction to the theory of point processes: volume II: general theory and structure*, Springer Science & Business Media, 2007.

[Edd96] Sean R. Eddy, « Hidden markov models », *in*: *Current opinion in structural biology* 6.*3* (1996), pp. 361–365.

[Flu90] Bernard D Flury, « Acceptance–rejection sampling made easy », *in*: *SIAM Review* 32.*3* (1990), pp. 474–476.

[FS08] Yu Fan and Christian R. Shelton, « Sampling for Approximate Inference in Continuous Time Bayesian Networks. », *in*: *ISAIM*, 2008.

[Gab+80] D.M. Gabbay et al., *On the Temporal Basis of Fairness, POPL: 163-173*, 1980.

[GCP99] Orna Grumberg, E.M. Clarke, and Doron Peled, *Model checking*, 1999.

[Ger+95] Rob Gerth et al., « Simple on-the-fly automatic verification of linear temporal logic », *in*: *International Conference on Protocol Specification, Testing and Verification*, Springer, 1995, pp. 3–18.

[Gha97] Zoubin Ghahramani, « Learning dynamic Bayesian networks », *in*: *International School on Neural Networks, Initiated by IIASS and EMFCSC*, Springer, 1997, pp. 168–197.

[GM16] Asela Gunawardana and Christopher Meek, « Universal Models of Multivariate Temporal Point Processes », *in*: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016, pp. 556–563.

[GMX11] Asela Gunawardana, Christopher Meek, and Puyang Xu, « A model for temporal dependencies in event streams », *in*: *Advances in Neural Information Processing Systems*, 2011, pp. 1962–1970.

[Hai73] Frank A. Haight, *Handbook of the Poisson Distribution*, 1973.

[Ham50] Richard W Hamming, « Error detecting and error correcting codes », *in*: *Bell System technical journal* 29.*2* (1950), pp. 147–160.

[Her+03] Holger Hermanns et al., « A tool for model-checking Markov chains », *in*: *International Journal on Software Tools for Technology Transfer* 4.*2* (2003), pp. 153–172.

[Hér+04] Thomas Hérault et al., « Approximate probabilistic model checking », *in*: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, 2004, pp. 73–84.

[HJ94] Hans Hansson and Bengt Jonsson, « A logic for reasoning about time and reliability », *in*: *Formal aspects of computing* 6.*5* (1994), pp. 512–535.

[Hoe94] Wassily Hoeffding, « Probability inequalities for sums of bounded random variables », *in*: *The Collected Works of Wassily Hoeffding*, Springer, 1994, pp. 409–426.

[JR12] Kurt Jensen and Grzegorz Rozenberg, *High-level Petri nets: theory and application*, Springer Science & Business Media, 2012.

[JSD17] Cyrille Jegourel, Jun Sun, and Jin Song Dong, « Sequential schemes for frequentist estimation of properties in statistical model checking », *in*: *International Conference on Quantitative Evaluation of Systems*, Springer, 2017, pp. 333–350.

[KA98] Nagendra Kumar and Andreas G. Andreou, « Heteroscedastic discriminant analysis and reduced rank HMMs for improved speech recognition », *in*: *Speech communication* 26.*4* (1998), pp. 283–297.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker, « PRISM 4.0: Verification of Probabilistic Real-time Systems », *in*: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ed. by G. Gopalakrishnan and S. Qadeer, vol. 6806, LNCS, Springer, 2011, pp. 585–591.

[KS76] John G. Kemeny and J. Laurie Snell, *Markov Chains*, Springer-Verlag, New York, 1976.

[Lan] Christopher J Langmead, « Generalized Queries and Bayesian Statistical Model Checking in Dynamic Bayesian Networks: Application to Personalized Medicine », *in*: *Proceedings of The 8th Annual International Conference on Computational Systems Bioinformatics (CSB)*, vol. 201, p. 212.

[Lap+07] Sophie Laplante et al., « Probabilistic abstraction for model checking: An approach based on property testing », *in*: *ACM Transactions on Computational Logic (TOCL)* 8.*4* (2007), p. 20.

[LD01] Javier Llorca and Miguel Delgado-Rodrıéguez, « Competing risks analysis using Markov chains: impact of cerebrovascular and ischaemic heart disease in cancer mortality », *in*: *International journal of epidemiology* 30.*1* (2001), pp. 99–101.

[LDB10] Axel Legay, Benoit Delahaye, and Saddek Bensalem, « Statistical model checking: An overview », *in*: *International conference on runtime verification*, Springer, 2010, pp. 122–135.

[Lim+06] Claudio F. Lima et al., « Substructural neighborhoods for local search in the Bayesian optimization algorithm », *in*: *Parallel Problem Solving from Nature-PPSN IX*, Springer, 2006, pp. 232–241.

[Lim+09] Claudio F. Lima et al., « Loopy substructural local search for the Bayesian optimization algorithm », *in*: *International Workshop on Engineering Stochastic Local Search Algorithms*, Springer, 2009, pp. 61–75.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck, « The glory of the past », *in*: *Workshop on Logic of Programs*, Springer, 1985, pp. 196–218.

[Mao+11] Hua Mao et al., « Learning probabilistic automata for model checking », *in*: *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, IEEE, 2011, pp. 111–120.

[Mir+16] Seyedali Mirjalili et al., « Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization », *in*: *Expert Systems with Applications* 47 (2016), pp. 106–119.

[Mis+15] Nikita Mishra et al., « A probabilistic graphical model-based approach for minimizing energy under performance constraints », *in*: *ACM SIGPLAN Notices*, vol. 50, 4, ACM, 2015, pp. 267–281.

[ML19] Mathilde Monvoisin and Philippe Leray, « Multi-task Transfer Learning for Timescale Graphical Event Models », *in*: *European Conference on Symbolic and Quantitative Approaches with Uncertainty*, Springer, 2019, 313–323.

[MR02] Kevin P. Murphy and Stuart Russell, « Dynamic Bayesian networks: representation, inference and learning. 2002 », *in*: *University of California, Berkeley* (2002).

[Nea+04] Richard E. Neapolitan et al., *Learning bayesian networks*, vol. 38, Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[Nor98] James Robert Norris, *Markov chains*, 2, Cambridge university press, 1998.

[NSK02] Uri Nodelman, Christian R. Shelton, and Daphne Koller, « Continuous time Bayesian networks », *in*: *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 2002, pp. 378–387.

[Oka59] Masashi Okamoto, « Some inequalities relating to the partial sum of binomial probabilities », *in*: *Annals of the institute of Statistical Mathematics* 10.*1* (1959), pp. 29–35.

[Pea01] Judea Pearl, « Bayesian networks, causal inference and knowledge discovery », *in*: *UCLA Cognitive Systems Laboratory, Technical Report* (2001).

[Pet77] James L Peterson, « Petri nets », *in*: *ACM Computing Surveys (CSUR)* 9.*3* (1977), pp. 223–252.

[Pet81] James L Peterson, *Petri net theory and the modeling of systems*, Prentice Hall PTR, 1981.

[Phi06] Leray Philippe, *Bayesian Networks: Learning and modeling of complex systems*, 2006.

[Phi20] Behrendt Philipp, *Learnability of Timescale Graphical Event Models*, tech. rep., Polytech Nantes, 2020.

[Pnu77] Amir Pnueli, « The temporal logic of programs », *in*: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[Pri03] Arthur N. Prior, *Time and modality*, OUP Oxford, 2003.

[Rab63] Michael O. Rabin, « Probabilistic automata », *in*: *Information and control* 6.*3* (1963), pp. 230–245.

[Rab89] Lawrence R. Rabiner, « A tutorial on hidden Markov models and selected applications in speech recognition », *in*: *Proceedings of the IEEE* 77.*2* (1989), pp. 257–286.

[RGH05] Shyamsundar Rajaram, Thore Graepel, and Ralf Herbrich, « Poisson-networks: A model for structured point processes », *in*: *Proceedings of the 10th international workshop on artificial intelligence and statistics*, 2005, pp. 277–284.

[RT13] Vinayak Rao and Yee Whye Teh, « Fast MCMC sampling for Markov jump processes and extensions », *in*: *The Journal of Machine Learning Research* 14.*1* (2013), pp. 3295–3320.

[RVV08] Anne Rozinat, Manuela Veloso, and Wil M.P. Van Der Aalst, « Using hidden markov models to evaluate the quality of discovered process models », *in*: *Extended Version. BPM Center Report BPM-08-10, BPMcenter. org* 161 (2008), pp. 178–182.

[Sch+78] Gideon Schwarz et al., « Estimating the dimension of a model », *in*: *The annals of statistics* 6.*2* (1978), pp. 461–464.

[Sel+09] Maarten H.W. Selfhout et al., « Different types of Internet use, depression, and social anxiety: The role of perceived friendship quality », *in*: *Journal of adolescence* 32.*4* (2009), pp. 819–833.

[Sha81] Micha Sharir, « A strong-connectivity algorithm and its applications in data flow analysis », *in*: *Computers & Mathematics with Applications* 7.*1* (1981), pp. 67–72.

[SL94] Roberto Segala and Nancy Lynch, « Probabilistic simulations for probabilistic processes », *in*: *International Conference on Concurrency Theory*, Springer, 1994, pp. 481–496.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha, « Statistical model checking of black-box probabilistic systems », *in*: *International Conference on Computer Aided Verification*, Springer, 2004, pp. 202–215.

[SWS93] Collin M. Stultz, James V. White, and Temple F. Smith, « Structural analysis based on state-space modeling », *in*: *Protein Science* 2.*3* (1993), pp. 305–314.

[TBA06] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis, « The max-min hill-climbing Bayesian network structure learning algorithm », *in*: *Machine learning* 65.*1* (2006), pp. 31–78.

[Tra13] Ghada Trabelsi, « New structure learning algorithms and evaluation methods for large dynamic Bayesian networks », PhD thesis, Université de Nantes; Ecole Nationale d'Ingénieurs de Sfax, 2013.

[Tru+05] Wilson Truccolo et al., « A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects », *in*: *Journal of neurophysiology* 93.*2* (2005), pp. 1074–1089.

[Van11] Wil Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*, vol. 2, Springer, 2011.

[Van14] W. Van Der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Springer Berlin Heidelberg, 2014, ISBN: 9783642434952.

[VH14] Tim Van Erven and Peter Harremos, « Rényi divergence and Kullback-Leibler divergence », *in*: *IEEE Transactions on Information Theory* 60.*7* (2014), pp. 3797–3820.

[VM98] Christian Vogler and Dimitris Metaxas, « ASL recognition based on a coupling between HMMs and 3D motion analysis », *in*: *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, IEEE, 1998, pp. 363–369.

[Wal45] Abraham Wald, « Sequential tests of statistical hypotheses », *in*: *The annals of mathematical statistics* 16.*2* (1945), pp. 117–186.

[WP13] Jeremy C Weiss and David Page, « Forest-based point process for event prediction from electronic health records », *in*: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 547–562.

[WSS94] James V White, Collin M Stultz, and Temple F Smith, « Protein classification by stochastic modeling and optimal filtering of amino-acid sequences », *in*: *Mathematical biosciences* 119.*1* (1994), pp. 35–75.

[Wur+16] Jacob Wurm et al., « Security analysis on consumer and industrial IoT devices », *in*: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2016, pp. 519–524.

[You05a] Hakan L. Younes, « Probabilistic verification for “black-box” systems », *in*: *International Conference on Computer Aided Verification*, Springer, 2005, pp. 253–265.

[You05b] Hakan L. Younes, *Verification and planning for stochastic processes with asynchronous events*, tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 2005.

[ZH09] Nicole Zillien and Eszter Hargittai, « Digital distinction: Status-specific types of internet usage », *in*: *Social Science Quarterly* 90.*2* (2009), pp. 274–291.

A. DESCRIPTION OF THE "EVENTIAL" LIBRARY

In this appendix section we briefly present the RTGEM library that is part of the PILGRIM project. The main components of the library that were used in this work are presented in this section. We give a description of the different parts of the library and their respective functionality. This library was created in order to learn, sample and handle RTGEMs. We recall that although there are a few published works on Graphical Event Models (in particular RTGEMs), there are no available libraries to handle these formalisms. We use the doxygen documentation figures to explicitly show the different components of the library. The code of PILGRIM and of the evential library is written in C++. Note that not all the functionalities of the library are introduced in this section. However, this library is intended to be used only for Recursive Timescale Graphical Event Models and not any type of Graphical Event Models. The library is constructed to only allow the construction of RTGEM, either with the learning procedure or manually or randomly.

We begin by exploring and illustrating the elementary classes of the library. The structure of the class RTGEM is shown in Figure 4.12. This class stores the structure (graph) and the parameters (λ, also called the effects) of an RTGEM object. The three main components of this class are the classes "SCCs", "Nodes" and "Edges". The class "SCCs" is used for computing and storing the different strongly connected components of an RTGEM object. The class "Nodes" contains the different nodes of the graph that are either set manually or computed automatically from the "Events" class (see Figure 4.13). An adjacency matrix is computed from the nodes and the different existing edges. The class "Edges" is used for setting manually or automatically (after the learning procedure) the list of edges and their corresponding status (existing, absent, used to exist). The status of the edge is implemented because it is adequate for the Greedy learning technique, where we need to test each operation separately (apply it then remove it) and see if it improves the BIC score.

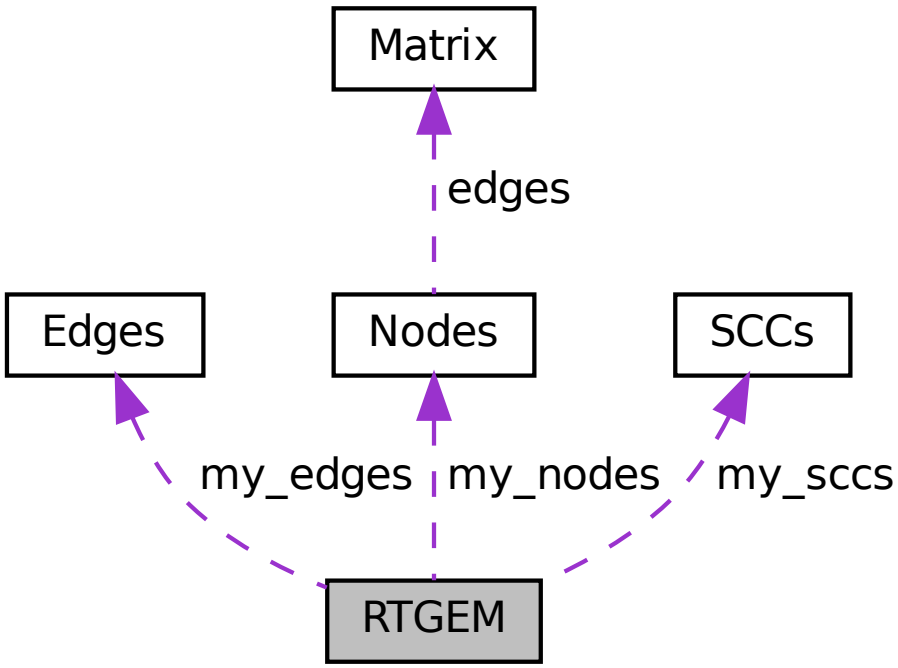


Figure 4.12 – An illustration of the RTGEM class components in the "evential" library.

The timescales are represented as trees in the Edge class. The tree structure allows to better identify split and extend operators, in order to establish conditions for reverse operators. The effects are computed depending on the structure and the initial data that is either manually set or externally transmitted to the program. The effects can also be set manually. In the sampling function, the object RTGEM is used for computing the corresponding effect (λ parameter) of a certain node in function of the date in the timeline of the sample.

The core of the library is the Learning class illustrated in Figure 4.13. The name "Learning" should not be misleading, because this class is not only used for learning RTGEMs. The Learning class consists of the "Neighbours" class whose methods are used for computing the table of possible neighbors (possible modifications) of a certain RTGEM. It also consists of the "Date" class that is mainly used for computing the sample size t^* of the input data (data used for learning the RTGEM and/or data that is sampled from an RTGEM).

In addition to the RTGEM class and the Events class that allows to import/export events and/or use their labels to name the nodes, the Learning class also consists of the "hrz_mat" class. The "hrz_mat" allows to compute the proximal horizons (see Chapter 2) for every pair of nodes and stores them into a matrix. Afterwards, the horizons are passed accordingly only to the existing edges in the adjacency matrix.

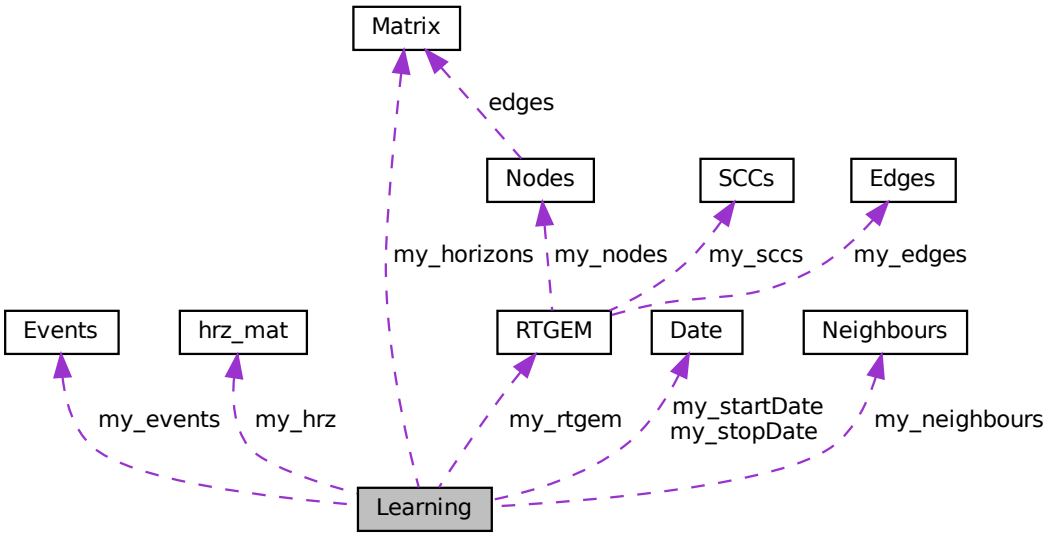


Figure 4.13 – An illustration of the Learning class components in the "evential" library.

The Random Walk function we use in our algorithm is implemented using mainly the Neighbors class to construct a more strict table of Neighbors from which we can randomly pick a possible modification. The early stopping function is implemented using the SCCs class and the normal sampling function. The SMC functions and the formal properties are implemented separately from the library but also in C++.

B. PROOF FOR THE SHD DISTANCE METRIC

In this Appendix we prove that the SHD ($\mathrm{SHD}(G_1, G_2) = \sum_{e \in E_{sd}} 1 + \sum_{e \in E_{inter}} d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2))$) we propose in Chapter 3 is a distance metric. We recall that $E_{sd} = \{E_1 \setminus E_2\} \cup \{E_2 \setminus E_1\}$ are the edges of each model that are not present in the other one and $E_{inter} = E_1 \cap E_2$ are the edges that are present in both models. $\mathcal{T}(e, G_1)$ and $\mathcal{T}(e, G_2)$ are the lists of endpoints of the intervals on the timescales of the corresponding edge e in graph G_1 and G_2 respectively.

Consider three RTGEM graphs G_1, G_2 and G_3, we illustrate in Figure 4.14 that E_1 can be partitioned into $\{A_1\} \cup \{A_{12}\} \cup \{A_{13}\} \cup \{A_{123}\}$, E_2 can be partitioned into $\{A_2\} \cup \{A_{12}\} \cup \{A_{23}\} \cup \{A_{123}\}$ and E_3 into $\{A_3\} \cup \{A_{13}\} \cup \{A_{23}\} \cup \{A_{123}\}$. We recall that the elementary distance on a corresponding edge $e \in A_{ij}$ in any two RTGEM graphs G_i and G_j (i and j being indexes) can be written as: $d(\mathcal{T}(e, G_i), \mathcal{T}(e, G_j)) = \frac{v^{ij}_{nid}}{v^{ij}_{nid} + v^{ij}_{id}}$, where $v^{ij}_{nid} = |v_i \setminus v_j| + |v_j \setminus v_i|$ and $v^{ij}_{id} = |v_i \cap v_j|$ with v_i and v_j the vectors where the values of the successive endpoints of the timescale are stored.

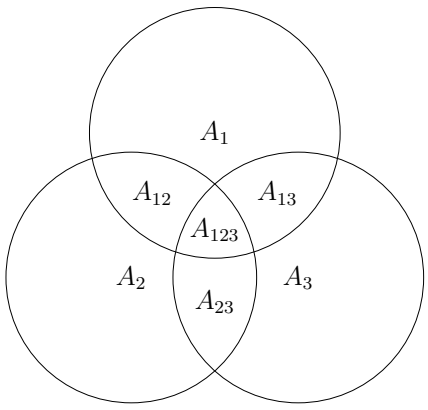


Figure 4.14 – Intersections between the sets of edges of three RTGEM graphs.

We start by showing that the elementary distance measure is always positive and satisfies the three following properties:

1. $\forall G_1$, $\forall G_2$ and $\forall e \in A_{12}$: $d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = 0 \Leftrightarrow \mathcal{T}(e, G_1) = \mathcal{T}(e, G_2))$
2. $\forall G_1$, $\forall G_2$ and $\forall e \in A_{12}$: $d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = d(\mathcal{T}(e, G_2), \mathcal{T}(e, G_1))$
3. $\forall G_1$, $\forall G_2$, $\forall G_3$ and $\forall e \in A_{123}$: $d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_3)) \leq d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) + d(\mathcal{T}(e, G_2), \mathcal{T}(e, G_3))$

To begin with, our elementary distance is always positive since v_{nid}^{ij} and v_{id}^{ij} are always positive. For the first property, if $d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = 0$ then by definition $v_{nid}^{12} = 0$ implying that all values on the timescales are identical, thus meaning $\mathcal{T}(e, G_1) = \mathcal{T}(e, G_2)$; and it is analogical for the other way around ($\mathcal{T}(e, G_1) = \mathcal{T}(e, G_2)) \Rightarrow d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = 0$). The second property is trivially satisfied since the computation of v_{nid}^{12} and v_{id}^{12} is done by comparing, value by value, two vectors of endpoints which is a symmetric operation.

The third property is also satisfied on the elementary distance, let A_{123} be the set of edges that are in common between three RTGEM graphs G_1, G_2 and G_3 (see figure 4.14). Consider an edge $e \in A_{123}$, we write for this edge: $\mathcal{T}(e, G_1) = \{T_1\} \cup \{T_{12}\} \cup \{T_{13}\} \cup \{T_{123}\}$, $\mathcal{T}(e, G_2) = \{T_2\} \cup \{T_{12}\} \cup \{T_{23}\} \cup \{T_{123}\}$ and $\mathcal{T}(e, G_3) = \{T_3\} \cup \{T_{23}\} \cup \{T_{13}\} \cup \{T_{123}\}$ for its corresponding timescales in G_1, G_2 and G_3 respectively. The possible intersections between these timescales are represented in Figure 4.15. For the sake of simplicity we write T_i (with i an index) for the cardinals of the corresponding above sets.

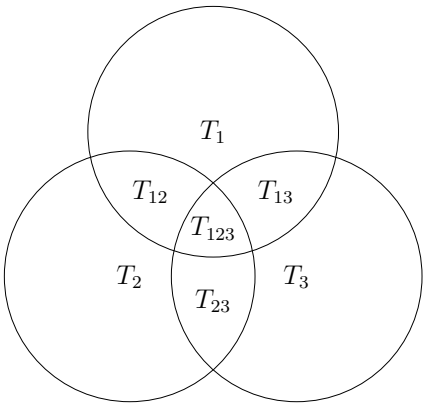


Figure 4.15 – Intersections between the timescales of the same edge in three RTGEM graphs.

The elementary distance on a given edge $e \in A_{123}$ (for the three RTGEM graphs),

between G_1 and G_3 can be written as:

$$d(\mathcal{T}(e,G_1),\mathcal{T}(e,G_3)) = \frac{v_{nid}^{13}}{v_{nid}^{13}+v_{id}^{13}} = \frac{T_1+T_{12}+T_{23}+T_3}{T_1+T_{12}+T_{23}+T_3+T_{13}+T_{123}} \tag{4.1}$$

For the third property we need to show that the above distance in equation 4.1 is always smaller than $d(\mathcal{T}(e,G_1),\mathcal{T}(e,G_2)) + d(\mathcal{T}(e,G_2),\mathcal{T}(e,G_3))$. In other words, we need to prove the inequality:

$$d(\mathcal{T}(e,G_1),\mathcal{T}(e,G_3)) - d(\mathcal{T}(e,G_1),\mathcal{T}(e,G_2)) - d(\mathcal{T}(e,G_2),\mathcal{T}(e,G_3)) \leq 0$$

We have:

$$d(\mathcal{T}(e,G_1),\mathcal{T}(e,G_2)) = \frac{T_1+T_{13}+T_{23}+T_2}{T_1+T_{13}+T_{23}+T_2+T_{12}+T_{123}}$$

and

$$d(\mathcal{T}(e,G_2),\mathcal{T}(e,G_3)) = \frac{T_2+T_{12}+T_{13}+T_3}{T_2+T_{12}+T_{13}+T_3+T_{23}+T_{123}}$$

Let $F_1 = T_1+T_{12}+T_{23}+T_3+T_{13}+T_{123}$, $F_2 = T_1+T_{13}+T_{23}+T_2+T_{12}+T_{123}$ and $F_3 = T_2+T_{12}+T_{13}+T_3+T_{23}+T_{123}$. After finding the common denominator, the numerator can be written as:

$$(T_1+T_{12}+T_{23}+T_3)\times F_2\times F_3 - (T_1+T_{13}+T_{23}+T_2)\times F_2\times F_3 - (T_2+T_{12}+T_{13}+T_3)\times F_2\times F_3 \tag{4.2}$$

By carefully developing equation 4.2 we can show that it is always negative (it only contains negative terms with positive valued variables). Therefore, the elementary distance satisfies the desired properties.

In order to prove the triangle inequality on the SHD we need to prove that $\mathrm{SHD}(G_1,G_3) \leq \mathrm{SHD}(G_1,G_2) + \mathrm{SHD}(G_2,G_3)$. By referring to Figure 4.14, we can write:

$$\mathrm{SHD}(G_1,G_3) = \sum_{\{A_1\}\cup\{A_3\}\cup\{A_{23}\}\cup\{A_{12}\}} 1 \quad + \quad \sum_{\{A_{13}\}\cup\{A_{123}\}} \left(\frac{v_{nid}^{13}}{v_{nid}^{13}+v_{id}^{13}}\right) \tag{4.3}$$

Similarly, let $\mathcal{P}_1 = \{A_1\}\cup\{A_2\}\cup\{A_{23}\}\cup\{A_{13}\}$, $\mathcal{P}_2 = \{A_2\}\cup\{A_3\}\cup\{A_{13}\}\cup\{A_{12}\}$,

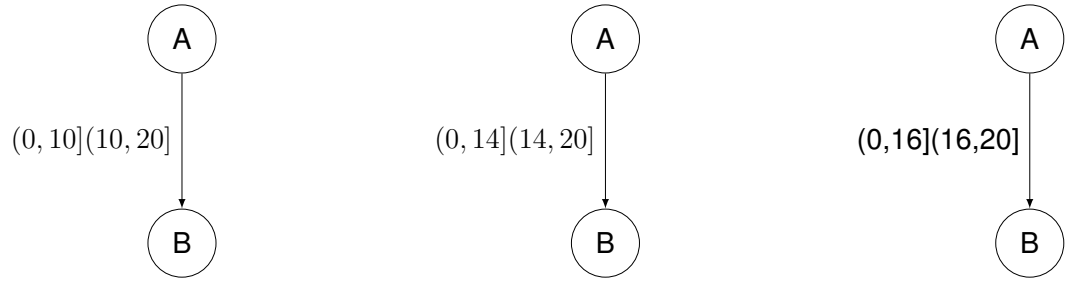
$\mathcal{Q}_1 = \{A_{12}\} \cup \{A_{123}\}$ and $\mathcal{Q}_2 = \{A_{23}\} \cup \{A_{123}\}$ we can write:

$$\mathrm{SHD}(G_1, G_2) + \mathrm{SHD}(G_2, G_3) = \sum_{\mathcal{P}_1} 1 \quad + \quad \sum_{\mathcal{P}_2} 1 \quad + \quad \sum_{\mathcal{Q}_1} \left(\frac{v_{nid}^{12}}{v_{nid}^{12} + v_{id}^{12}}\right) \quad + \quad \sum_{\mathcal{Q}_2} \left(\frac{v_{nid}^{23}}{v_{nid}^{23} + v_{id}^{23}}\right) \tag{4.4}$$

We can see that (even without developing the equations) the terms that appear in equation 4.3 also appear in equation 4.4. The sets $\{A_1\}$, $\{A_3\}$, $\{A_{23}\}$ and $\{A_{12}\}$ (the entire left term of equation 4.3) appear in $\mathcal{P}_1$ and $\mathcal{P}_2$ along with other sets (on which we do the sum of 1). In the right term of equation 4.3 we go through the set $\{A_{13}\}$ to do the sum of the $\frac{v_{nid}^{13}}{v_{nid}^{13}+v_{id}^{13}}$ (which is lesser than 1), contrarily to equation 4.4 where we go through the set $\{A_{13}\}$ to do the sum of 1. Finally, for the set $\{A_{123}\}$ that appears in both equations we previously proved that $\frac{v_{nid}^{13}}{v_{nid}^{13}+v_{id}^{13}} \leq \frac{v_{nid}^{12}}{v_{nid}^{12}+v_{id}^{12}} + \frac{v_{nid}^{23}}{v_{nid}^{23}+v_{id}^{23}}$. After this identification we can see that equation 4.3 is always lesser or equal to equation 4.4 and that the SHD satisfies the triangle inequality.

The remaining distance axioms (Identity of indiscernibles and Symmetry) can also be proven on the SHD. As a consequence, SHD is proved to be a distance metric.

However the SHD^* measure is only a semimetric (not a metric), because it is positive and only satisfies the identity of indiscernibles and the symmetry property analogically to the SHD but does not satisfy the triangle inequality. We propose a counter example in Figure 4.16.



(a) RTGEM Graph G_x (b) RTGEM Graph G_y (c) RTGEM Graph G_z

Figure 4.16 – Three RTGEMs graphs G_x, G_y and G_z with one edge e used as counter example for the triangle inequality in SHD^*

For the triangle inequality: consider the three vectors $v_x = [0, 10, 20]$, $v_y = [0, 14, 20]$ and $v_z = [0, 16, 20]$, corresponding to the edge e that is the same in the three graphs G_x, G_y and G_z of Figure 4.16 ($e \in A_{xyz}$). We recall that $V_{id}^{*12} = \{(v_{1_i}, v_{2_j}) \in v_1 \times v_2 :$

$\mathbf{cl}(v_{1_i}, v_2) = v_{2_j} \wedge \mathbf{cl}(v_{2_j}, v_1) = v_{1_i}\}$, with $\mathbf{cl}$ a function that finds the closest element to v_{1_i} in v_2: $\mathbf{cl}(v_{1_i}, v_2) = argmin_{v_{2_p}}(|v_{1_i} - v_{2_p}|)$. The elementary distance in this case is written:

$$d^*(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = \frac{1}{v^{12}_{nid} + v^{12}_{id}} \left(\sum_{(v_{1_i}, v_{2_j}) \in V^{*12}_{id} \setminus (0,0)} \frac{\mid v_{1_i} - v_{2_j} \mid}{min(v_{1_i}, v_{2_j})} \right) + \frac{v^{12}_{nid}}{v^{12}_{nid} + v^{12}_{id}}$$

Remark that in d^* we count the found matches in v^{ij}_{id} and not in v^{ij}_{nid}. We have $d^*(\mathcal{T}(e, G_x), \mathcal{T}(e, G_z)) = 0 + \frac{v^{xz}_{nid}}{v^{xz}_{nid} + v^{xz}_{id}} = 1/2$ (no matches because 10 is closest to 16 but 16 is closest to 20), $d^*(\mathcal{T}(e, G_x), \mathcal{T}(e, G_y)) = \frac{1}{3} \cdot \frac{|10-14|}{min(10,14)} + 0 = 2/15$ (matching 10 and 14) and
$d^*(\mathcal{T}(e, G_y), \mathcal{T}(e, G_z)) = 1/21$ (matching 14 and 16). Therefore, for this example we have:

$$\mathrm{SHD}^*(G_x, G_z) = 0 + d^*(\mathcal{T}(e, G_x), \mathcal{T}(e, G_z)) = 1/2$$

$$\mathrm{SHD}^*(G_x, G_y) = 0 + d^*(\mathcal{T}(e, G_x), \mathcal{T}(e, G_y)) = 2/15$$

$$\mathrm{SHD}^*(G_y, G_z) = 0 + d^*(\mathcal{T}(e, G_y), \mathcal{T}(e, G_z)) = 1/21$$

hence the triangular inequality does not hold because: $\mathrm{SHD}^*(G_x, G_z) > \mathrm{SHD}^*(G_x, G_y) + \mathrm{SHD}^*(G_y, G_z)$.

RÉSUMÉ

Depuis sa création au début des années 1990, la perception d'internet par les entreprises et le grand public n'a cessé d'évoluer. Le monde dans lequel nous vivons aujourd'hui est totalement dépendant d'Internet qui est partie prenante de la vie de chaque individu. Par conséquent l'utilisation de machines, d'ordinateurs, de smartphones et d'appareils connectés connaît une croissance exponentielle d'année en année. Les chercheurs et les sociologues ont même identifié quatre types d'Internet. Il s'agit de l'Internet des contenus (Google, Wikipedia, etc.), de l'Internet des personnes (réseaux sociaux), de l'Internet des objets (cloud, objets connectés, etc.) et de l'Internet des lieux (mobilité, Google maps, etc.) [ZH09; Sel+09; BG14]. En conséquence, les machines connectées sont devenues le point d'entrée le plus ciblé pour les agents malveillants. Toutes les machines connectées communiquent via d'importants réseaux et serveurs régis par des systèmes complexes. Plus les systèmes se compliquent, plus les techniques d'exploitation frauduleuses deviennent sophistiquées.

Dans la plupart des emplois ou des applications de la vie courante, les employés, les travailleurs, les consommateurs et les utilisateurs utilisent des appareils connectés. Ceci les rend vulnérables non seulement à des exploits externes, mais également à des abus internes qui peuvent devenir dangereux. C'est pourquoi il est important de surveiller le comportement des utilisateurs et des travailleurs dans leur environnement pour garantir un flux de travail « sécurisé ». Par exemple, la surveillance du comportement des chauffeurs de camion peut permettre au superviseur de vérifier s'ils ont eu suffisamment de temps de repos pendant le trajet. Les informaticiens ont déployé beaucoup d'efforts dans la recherche de construction de périmètres sûrs dans lesquels les appareils connectés peuvent être utilisés [Wur+16; Abo+15]. Dans ce travail, nous utilisons l'analyse comportementale afin d'identifier un comportement normal d'un comportement dangereux tout en évaluant son niveau de dangerosité. La clé pour établir une analyse de comportement adéquate est un ensemble de données fidèles enregistrées à partir du système concerné. En 2020, nous générons en dix minutes plus de données que l'ensemble des données enregistrées par l'humanité

jusqu'en 2003. Ainsi, il apparaît que le problème ne réside pas dans la génération des données ou dans leur stockage mais dans notre capacité à extraire des informations utiles de cette énorme quantité de données. C'est dans ce contexte que les techniques d'exploration de données [Van11; Van14] révèlent toute leur utilité. Dans cette thèse, nous ne traitons pas directement ce problème, mais nous insistons sur l'importance de la fidélité des données car les techniques que nous utilisons sont basées sur ces données.

Cette thèse a été réalisée en collaboration avec le groupe GFI informatique [1]. GFI est un groupe international qui propose principalement des solutions et expertises informatiques, touchant une grande variété de domaines (banque, sécurité routière, supervision, cloud, solutions IA et bien d'autres). Le Centre d'Innovation et d'Expertise (CIE) de GFI dirige de nombreux projets de recherche et développement (R&D). Les principales attentes de GFI à propos de cette thèse sont d'explorer l'analyse comportementale, y compris comment elle peut être appliquée en sécurité ou en supervision, et d'utiliser des techniques innovantes afin de se distinguer des solutions disponibles sur le marché.

Afin de créer un accès sécurisé aux données dans un système réel et d'assurer leur sécurité contre toute menace potentielle à venir, il faut connaître les dépendances et le comportement des différents composants du système. Ainsi on peut identifier les comportements malveillants afin d'agir au bon moment pour les intercepter. Nous pouvons ici commencer à voir l'importance du Machine Learning qui permet de construire un modèle du système concerné sur la base de données ou de certaines connaissances antérieures sur le système. De nombreux types de formalismes de modélisation existent dans la littérature, chacun de ces types ayant été développé avec un but précis. Certains sont mieux adaptés à la vérification de propriétés ou d'hypothèses données, d'autres à l'apprentissage des comportements et des dépendances ; dans ce travail, nous nous intéressons à tous ces aspects. Les automates finis probabilistes, par exemple, ont été utilisés dans la modélisation et la vérification des comportements connus ou souhaités [Mao+11; SL94; Rab63]. Les réseaux de Petri [Pet77; Pet81] ont été utilisés dans la modélisation et la vérification de plusieurs tâches parallèles ainsi que dans le Process Mining [Van14]. Les modèles de Markov cachés ont été largement utilisés

1. Site Web : gfi.world

dans la reconnaissance de la parole et des images, ainsi que dans l'évaluation de la qualité des processus découverts [RVV08; KA98; VM98]. Des modèles graphiques probabilistes ont été utilisés pour l'apprentissage automatique et la représentation des dépendances entre les différentes variables d'un système [MR02].

Chacun de ces formalismes a ses propres avantages et inconvénients. Néanmoins, tous les formalismes cités ci-dessus et ceux qui sont dans la même famille ont un défaut commun, la discrétisation du temps. Ce dernier peut être décrit comme un biais de représentation dans l'apprentissage de ces formalismes. Entre chaque deux événements consécutifs, il y a un pas de temps mais il n'y a pas de mesure quantitative du temps qui montre le temps réel écoulé entre les événements, le délai avant qu'un événement ne se produise au début d'un processus, ou le retard à la fin du processus. Ainsi, du point de vue de la sécurité, il est préférable d'utiliser des formalismes de modélisation temporelle continue qui permettent de savoir plus précisément quand agir et pas seulement quelle action entreprendre; par exemple lors de la prédiction d'une défaillance du système ou de la prédiction des tendances futures des utilisateurs.

Pour explorer la dynamique d'une grande variété de comportements de systèmes basés sur des flux d'événements collectés, il existe de nombreux formalismes avancés de modélisation en temps continu : par exemple, les réseaux bayésiens à temps continu [NSK02], les Markov jump processes [RT13], les réseaux de Poisson [RGH05] et les modèles d'événements graphiques (Graphical Event Models GEMs) [GMX11]. Dans ce travail, nous sommes particulièrement intéressés par les *Recursive Timescale Graphical Models* (RTGEMs) [GM16] une sous-famille de GEM, qui présentent des avantages par rapport aux autres formalismes. En particulier, ils sont conçus pour approximer universellement tout processus ponctuel marqué (marked point process m.p.p.) lisse, non explosif, stationnaire et multivarié [DV07].

Les techniques d'apprentissage et de vérification formelle appropriées doivent être adaptées au type de formalisme que nous souhaitons construire. Le Model Checking, par exemple, est utilisé comme méthode formelle de vérification [BK08]. Cette méthode a été appliquée à de nombreux formalismes, mais d'après nos connaissances, jamais adaptée aux RTGEM. Une autre solution valable pour la vérification sont les méthodes d'approximation, telles que la vérification du modèle statistique

(SMC) [LDB10], qui est une technique efficace basée sur des simulations et des résultats statistiques. La SMC a été appliquée avec succès à des modèles graphiques probabilistes tels que les réseaux bayésiens dynamiques (DBN) dans [Lan]. Par conséquent, la SMC pourrait être adaptée aux RTGEM. Lorsque nous considérons des techniques basées sur de la simulation (telles que la SMC), on pourrait penser à les utiliser directement sur les données originales et non sur les données échantillonnées à partir d'un modèle appris. Néanmoins, les données réelles que nous collectons peuvent être bruyantes, rares ou incomplètes et peuvent ainsi masquer différents scénarios rendant les techniques basées sur la simulation moins efficaces. C'est pourquoi, il est important d'apprendre un modèle probabiliste et d'en échantillonner les données pour éviter d'ignorer (totalement ou partiellement) certains scénarios. De plus, des techniques comme l'*échantillonnage d'importance* (importance sampling) peuvent être utilisées sur le modèle pour augmenter la fréquence d'apparition d'événements rares, ce qui ne peut pas être réalisé directement sur les données d'origine.

Les principaux objectifs poursuivis au cours de ce travail sont d'abord la création de liens entre le domaine d'apprentissage basé sur des modèles graphiques et le domaine de vérification formelle, afin de bénéficier des avantages de chaque domaine pour les évaluations de sécurité. D'autre part, la recherche d'un modèle à la fois représentatif des données d'entrée et sûr d'un point de vue sécurité fût entreprise. Nous ne sommes pas seulement intéressés par l'évaluation de l'aptitude d'un modèle à l'aide de techniques de notation standard mais également à son adéquation du point de vue de la sécurité. En particulier, il est probable que le modèle appris ne satisfait pas les propriétés de sécurité données. Par conséquent, nous proposons une stratégie où nous choisissons d'apprendre le RTGEM "le plus adapté" (celui qui est le plus représentatif des données). Si nos normes de sécurité ne sont pas vérifiées sur le modèle appris, nous proposons également une méthodologie de recherche pour trouver un autre modèle *proche* qui les satisfasse. Pour ce faire, une stratégie appropriée est proposée et une mesure de distance est introduite afin de comparer deux RTGEM. En comparant le modèle "le plus adapté" et celui trouvé dans son voisinage (s'il en existe un) qui répond à la norme de sécurité, nous donnons un aperçu de la dangerosité du comportement détecté (appris). Notre approche est générique par rapport à la procédure de vérification et à la notion de distance entre modèles. Dans un souci d'exhaustivité, la stratégie que nous proposons est ensuite testée sur des données synthétiques. Le

plan de la thèse est donné ci-dessous.

Cette thèse est composée de quatre chapitres qui sont résumés dans ce qui suit. Le premier chapitre contient les formalismes et les techniques de pointe qui ont inspiré ce travail. Le deuxième chapitre contient les préliminaires nécessaires à cette thèse et les deux derniers chapitres constituent la principale contribution de ce travail.

Dans le premier chapitre, une revue de l'état de l'art sur les formalismes temporels discrets couramment utilisés est établie. En particulier, les systèmes de transition, les modèles de Markov, les réseaux bayésiens dynamiques, les réseaux de Petri et les automates probabilistes sont introduits. Nous rappelons formellement la définition de chaque formalisme présenté et discutons de ses avantages et inconvénients. Nous présentons également une synthèse sur le pouvoir expressif, l'évaluation et la vérification de chaque formalisme présenté. En procédant comme tel, nous constatons que la discrétisation du temps est en réalité un problème d'un point de vue sécuritaire, et que nous devons adopter un formalisme temporel continu afin d'atteindre nos objectifs. De plus, nous définissons formellement des propriétés linéaires qui seront ensuite utilisées pour définir les propriétés de sécurité mentionnées ci-dessus. Enfin, nous discutons de deux techniques de vérification formelles : le Model Checking et le Model Checking Statistique (SMC), ainsi que de leurs avantages et de leurs inconvénients. Nous justifions également le choix du SMC comme technique de vérification utilisée dans le reste de ce travail.

Dans le deuxième chapitre, nous introduisons les prérequis, définitions et notations de base qui sont utilisés dans le reste du travail. Nous présentons et définissons formellement les marked point processes (m.p.p.) et les modèles d'intensité conditionnelle (Conditional Intensity Models CIMs). Plus important encore, nous décrivons le type de données considérées pour cette étude. De plus, nous introduisons et définissons formellement les différentes familles de modèles d'événements graphiques, leurs procédures d'apprentissage et leurs limites. L'objectif principal de ce chapitre est de

justifier le choix des Recursive Timescale Graphical Event Models (RTGEMs) comme technique de modélisation formelle, en équilibrant leurs avantages et leurs inconvénients, et en démontrant leur facilité de manipulation.

Dans ce chapitre, nous définissons formellement un RTGEM comme étant une sous famille de Timescale Graphical Event Models (TGEM) construite avec des opérateurs récursifs prédéfinis. Un TGEM est un couple $M = (\mathcal{G}, \mathcal{T})$ où $\mathcal{G}$ est un Graphical Event Model (GEM) $\mathcal{G} = ((\mathcal{L}, E), \theta)$ constitué d'un graph $(\mathcal{L}, E)$ (L ensemble des noeuds et E ensemble des arcs) et d'un ensemble de paramètres θ; et $\mathcal{T} = T_{e_{(e \in E)}}$ un ensemble de timescales correspondant aux arcs E du graphe de $\mathcal{G}$. On note que ce type de formalisme est Markovien et que les noeuds dépendent des occurrences (historique) de leurs parents dans le passé (vis-à-vis des timescales). On utilise la notation $c_l(h, t)$ pour dénoter les combinaisons possibles d'occurrences des parents (s'ils existent). Par conséquent, pour un noeud il y a autant de paramètres que de combinaisons c_l possible. Un noeud n'ayant pas de parent n'a qu'un seul paramètre. Dans la figure 4.17 on montre un exemple de RTGEM à quatre noeuds et quatre arcs. Les opérateurs avec lesquels on construit un RTGEM sont divisés en deux groupes, les opérateurs Forward (pour aggrandir le graphe) et les opérateurs Backward (pour raffiner le graphe).

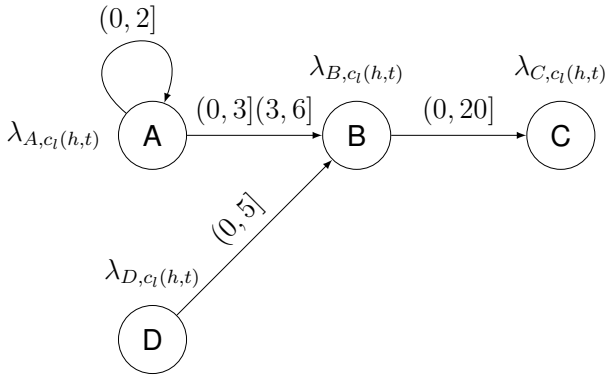


Figure 4.17 – Un exemple d'un RTGEM à 4 noeuds et 4 arcs.

Les opérateurs Forward sont les suivants : $\mathcal{O}_F = \{add, split, extend\}$, l'opérateur "add" ajoute un arc inexistant à un modèle et son timescale correspondant $T = (0, c]$, avec c une constante (également appelée horizon). L'opérateur "split" fractionne un intervalle $(a, b]$ dans le timescale d'un arc choisi en deux intervalles $(a, \frac{a+b}{2}], (\frac{a+b}{2}, b]$. L'opérateur "extend" étend l'horizon d'un arc choisie en ajoutant l'intervalle $(t_h, 2t_h]$, t_h étant l'horizon précédent. Les opérateurs Backward sont les suivants :

$\mathcal{O}_F^{-1} = \{reverse\_add, reverse\_split, reverse\_extend\}$, l'opérateur "reverse_add" supprime un arc choisi avec un seul intervalle dans son timescale (pour en faire l'inverse exact de l'opérateur "add"). L'opérateur "reverse_split" est utilisé pour fusionner deux intervalles consécutifs dans un timescale sur un arc choisi qui ont été initialement séparés. L'opérateur "reverse_extend" supprime l'intervalle le plus élevé (le dernier) d'une échelle de temps sur un bord choisi uniquement si la limite supérieure de cet intervalle a été initialement créée par une extension.

Dans le troisième chapitre, nous présentons les bases de notre contribution. Nous énonçons explicitement le problème et le formalisons. Nous proposons une solution au problème via une stratégie générique basée sur un modèle présenté sous la forme d'un algorithme. Nous détaillons la stratégie proposée pour l'apprentissage et la vérification des évaluations de sécurité. Nous définissons également une mesure de distance entre les modèles graphiques. Notre stratégie consiste à apprendre le modèle le plus représentatif du système sous-jacent. Nous vérifions ensuite si le modèle appris satisfait une propriété de sécurité donnée. Si ce n'est pas le cas, une recherche d'un modèle *proche* qui vérifie la propriété de sécurité donnée est effectuée. Enfin, nous adaptons une mesure de distance qui est calculée entre les deux modèles pour voir dans quelle mesure le modèle *fittest* est capable de vérifier la propriété. À notre avis, la combinaison de techniques que nous choisissons d'appliquer à chaque étape de notre algorithme générique proposé est la meilleure, bien que certaines d'entre elles puissent être améliorées (ce qui est discuté dans les perspectives).

Dans la phase d'apprentissage d'un modèle, nous voulons apprendre le modèle le plus adapté qui représente le mieux la réalité. Par conséquent, afin de sélectionner ce modèle le plus adapté, nous avons tendance à adapter les scores et les métriques qui évaluent la complexité et la ressemblance des différents modèles appris par rapport aux données réelles. D'un point de vue *securité*, il est également important de vérifier si notre modèle (qui représente la réalité jusqu'à un certain degré) satisfait les règles ou propriétés de sécurité données. Si les propriétés de sécurité ne sont pas vérifiées par le modèle, nous sommes également intéressés, d'un point de vue sécurité, à calculer dans quelle mesure le modèle actuel est loin de les vérifier afin d'avoir un aperçu du "niveau de danger" du modèle. La probabilité qu'un modèle M vérifie une requête

de sécurité φ s'écrit $P(\varphi \mid M)$. Afin de rester cohérent avec les notations du chapitre précédent, nous écrivons $P(D \mid M)$ pour la vraisemblance des données connaissant le modèle (structure et paramètres). Le problème que nous posons peut être formalisé comme suit:

$$\exists M^*, M^* = argmax(P(D \mid M)) \; avec \; P(\varphi \mid M) > c, \tag{4.5}$$

avec $c \in [0, 1]$ une constante donnée.
Les propriétés de sécurité que nous cherchons à vérifier sont *qualitatives* et traitent généralement un nombre limité d'événements dans notre modèle. On note $\mathcal{L}_\varphi$ l'ensemble des étiquettes des événements concernés par la requête de sécurité φ.

Intuitivement, nous avons tendance à traiter ce type de problèmes comme des problèmes d'optimisation. Cependant, le problème énoncé dans l'équation 4.5 ne peut pas être résolu en utilisant une heuristique d'optimisation multi-objectif telle que [Mir+16], car une propriété qualitative ne peut pas être optimisée, elle est soit vraie soit fausse. En d'autres termes, nous n'avons que des modèles qui ne sont pas "sûrs" ($P(\varphi \mid M) < c$) et des modèles qui sont "sûrs" ($P(\varphi \mid M) > c$). Par exemple, considérons deux modèles M et M' qui satisfont $P(\varphi \mid M) > c$ et $P(\varphi \mid M') > c$ afin que les deux soient "sécurisés"; avoir $P(\varphi \mid M) > P(\varphi midM')$ ne signifie pas que M est "plus sûr" que M'. Par conséquent, l'équation 4.5 ne peut pas être optimisée à l'aide d'une fonction multi-objectif, alors nous la décomposons et procédons autrement.

La stratégie proposée peut être représentée à l'aide d'un algorithme générique composé de trois étapes principales, la première étape étant la phase d'apprentissage, la deuxième étape étant la phase d'exploration de l'espace modèle et la vérification du modèle, et la dernière étape étant le calcul de la distance entre deux modèles.

Algorithm 6 Stratégie Proposée

input: $\mathcal{D}, \varphi$
output: M^*, Δ
1: $M^o = \underset{M \in Chosen\_Formalism}{\operatorname{argmax}} P(\mathcal{D} \mid M)$
2: $\mathcal{N} = \mathcal{N}_c(\mathcal{N}_{\mathcal{L}_\varphi}(M^o))$
3: $M^* = \text{find}\{M \in \mathcal{N}, P(\varphi \mid M) > c\}$
4: $\Delta = \text{Distance}(M^o, M^*)$

La première ligne de l'algorithme 6 correspond à la phase d'apprentissage d'un

modèle. Elle consiste à choisir un langage de modélisation adéquat et à apprendre le modèle le plus adapté M^o ("fittest model"). Les lignes 2 et 3 de l'algorithme 6 correspondent à la phase d'exploration de l'espace modèle et à la vérification du modèle, où nous recherchons un modèle M^*, dans le voisinage "proche" de M^o écrit $\mathcal{N}_c(\mathcal{N}_{\mathcal{L}_\varphi}(M^o))$, qui vérifie la propriété de sécurité. Cette étape est expliquée en détail dans ce chapitre. La dernière ligne de l'algorithme consiste à calculer la distance entre le modèle le plus adapté et le modèle que nous sélectionnons après la recherche de voisinage (s'il en existe un). La notion de *distance* que nous proposons est définie et expliquée dans ce chapitre.

Dans le dernier chapitre, nous construisons des tests pour évaluer les performances de chaque étape de l'algorithme proposé. Nous montrons les performances de l'apprentissage et de l'échantillonnage des RTGEM, ainsi que l'application de SMC sur ce formalisme. De plus, nous construisons un protocole d'expériences afin de montrer que ce que nous proposons remplit réellement les objectifs de départ que nous nous sommes fixés. Nous prouvons que si nous avons des données provenant d'un système sécurisé, le RTGEM appris sera également sécurisé en ce qui concerne la propriété de sécurité que nous proposons, et vice versa. Nous montrons expérimentalement que la technique d'exploration de voisinage afin de trouver un modèle *proche* qui satisfasse la propriété (au cas où le modèle appris ne la satisfait pas) fonctionne réellement. Enfin, nous montrons comment, en pratique, nous pouvons évaluer la dangerosité d'un modèle appris par rapport à un comportement nominal.

La conception de cette expérience est présentée dans ce qui suit et illustrée dans la figure 4.18. Cette expérience est construite afin de répondre aux questions suivantes:

- Considérons un ensemble de données qui est *statistiquement* sécurisé (ou non sécurisé), en ce qui concerne une propriété de sécurité, c'est-à-dire que les données représentent un comportement sûr (ou dangereux). Le modèle appris, à partir de ces données, satisfait-il (ou non) la propriété de sécurité en conséquence?
- Considérons un modèle qui ne vérifie pas la propriété de sécurité. Peut-on trouver dans son voisinage un modèle qui vérifie la propriété?
- Considérons un modèle qui est loin, probablement, de satisfaire une propriété de sécurité donnée. Est-il difficile de trouver un modèle dans son voisinage (s'il

en existe un) qui satisfait la propriété?

Par souci de simplicité, nous appelons le premier test le cas OK, le deuxième test le cas KO_1 et le troisième test le cas KO_2. Comme le montre la figure 4.18, nous commençons notre pipeline en appliquant l'algorithme proposé sur un ensemble de données synthétiques qui a été précédemment généré à partir d'un modèle qui satisfait une propriété de sécurité par rapport à une requête de sécurité φ. La première étape est l'apprentissage du modèle qui représente le mieux les données pour le cas OK. Nous procédons en appliquant notre technique de vérification formelle (SMC) choisie, comme décrit dans ce chapitre, sur le modèle appris. Nous exécutons ensuite la technique d'exploration spatiale sur le modèle appris s'il ne vérifie pas la propriété. Si un modèle qui vérifie la propriété de sécurité se trouve dans le voisinage du modèle initial, la distance qu'on a proposé dans le chapitre précédent SHD^* est alors calculée entre le modèle trouvé et le modèle initial (qui représente le mieux les données).

Nous exécutons à nouveau la même procédure sur deux ensembles distincts de données synthétiques qui ont été précédemment générées à partir de modèles qui ne satisfont pas la propriété de sécurité (cas KO_1 et KO_2). Les données utilisées pour l'apprentissage du modèle dans le cas KO_2 sont censées être, en probabilité, plus éloignées que les données qui sont utilisées pour l'apprentissage du modèle dans le cas KO_1 ont vérifié la propriété de sécurité. Nous notons que pour des raisons de cohérence, nous utilisons parfois la notation M^* pour désigner le *modèle sécurisé*, via l'exploration spatiale (si elle existe).

Nous montrons brièvement les résultats obtenus dans les tableaux suivants.
Dans le tableau 4.12, nous montrons les résultats obtenus après la fin du premier test. La vérification formelle avec SMC est répétée vingt fois afin d'établir des valeurs moyennes pour la durée requise et le nombre de simulations avant d'acquérir un résultat. Nous écrivons $\widehat{t}$ pour la durée moyenne et $\widehat{m}$ pour le nombre moyen de simulations sur vingt exécutions pour tous les tests restants.

	$Pr(\varphi\|M) > 0.8$	$\widehat{t}$	$\widehat{m}$
M^o_{OK}	Propriété satisfaite	5 sec (± 0.43 sec)	277 simulations (± 49 simulations)
M^*_{OK}	-	-	-

Table 4.12 – Résultats du premier test sur M^o_{OK}.

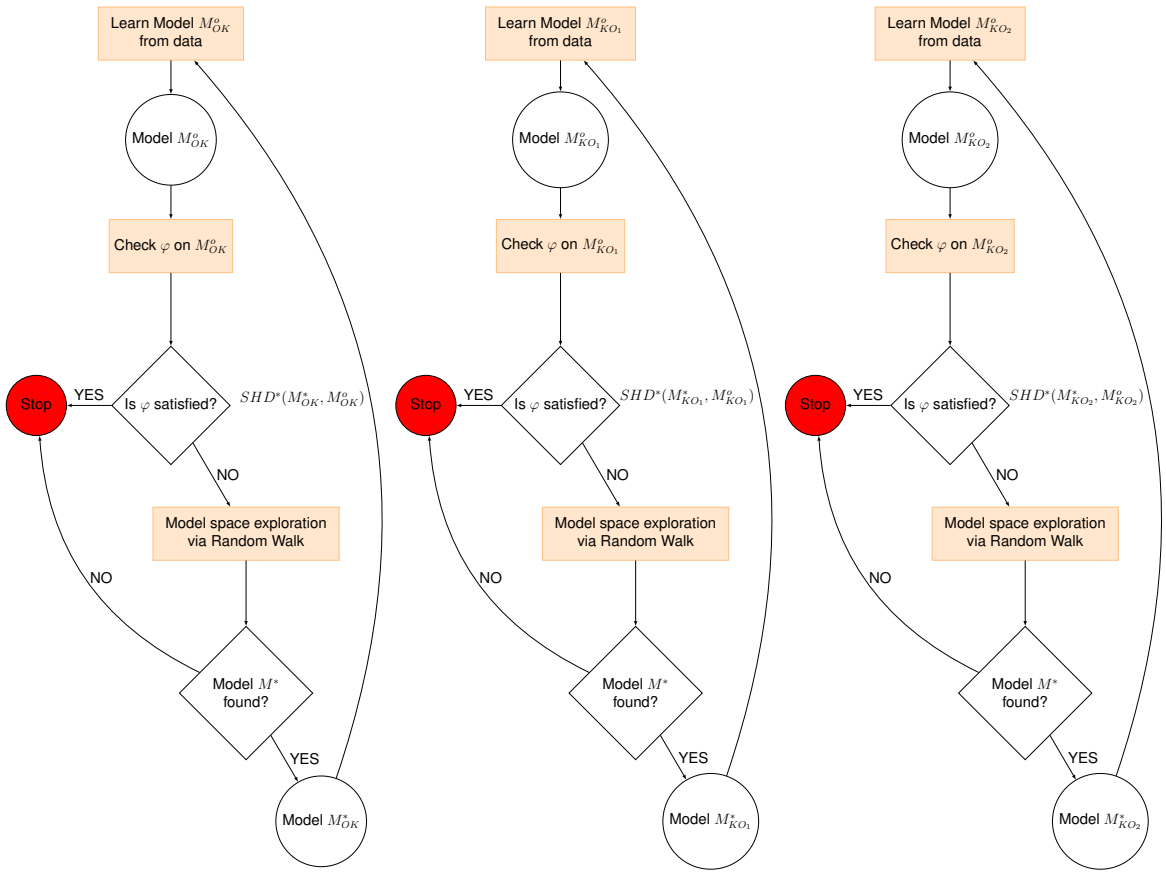


Figure 4.18 – Une schématisation du protocole expérimental proposé.

Quelques résultats du deuxième test sont présentés dans le tableau 4.13. La vérification formelle avec SMC est répétée vingt fois pour $M^o_{KO_1}$ et $M^*_{KO_1}$ afin d'établir des valeurs moyennes pour la durée requise et le nombre de simulations avant de converger vers un résultat.

	$Pr(\varphi \mid M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_1}$	Propriété NON satisfaite	1 min 2 sec (± 11 sec)	4018 simulations (± 101 simulations)
$M^*_{KO_1}$	Propriété satisfaite	2 min 17 sec (± 25 sec)	5512 simulations (± 122 simulations)

Table 4.13 – Résultats du second test sur $M^o_{KO_1}$.

Quelques résultats du dernier test sont présentés dans le tableau 4.14. La vérification formelle avec SMC est répétée vingt fois pour $M^o_{KO_2}$ et $M^*_{KO_{2_1}}$ et $M^*_{KO_{2_2}}$ (deux modèles sécurisés trouvés dans ce cas) afin d'établir la moyenne pour la durée requise et le nombre de simulations avant d'acquérir un résultat.

	$Pr(\varphi\|M) > 0.8$	$\widehat{t}$	$\widehat{m}$
$M^o_{KO_2}$	Propriété NON satisfied	3 min 22 sec (± 11 sec)	2155 simulations (± 69 simulations)
$M^*_{KO_{2_1}}$	Propriété satisfaite	11 min 31 sec (± 25 sec)	7512 simulations (± 136 simulations)
$M^*_{KO_{2_2}}$	Propriété satisfaite	7 min 19 sec (± 25 sec)	4286 simulations (± 114 simulations)

Table 4.14 – Résultats du dernier test sur $M^o_{KO_2}$.

Pour conclure, nous rappelons qu'aucune preuve théorique n'a été établie afin de démontrer formellement les résultats présentés et qu'ils sont uniquement basés sur les tests que nous avons effectués. Les réponses aux questions posées précédemment et qui sont également capables de résumer nos résultats sont énumérées ci-dessous:

- À l'exception de "detailed balance", plus la taille des données est grande, plus l'apprentissage est précis (voir Chapitre 4 et [GM16]).
- Un échantillonnage excessif d'un RTGEM est coûteux, surtout s'il contient des cycles, mais nous utilisons un échantillonnage ciblé avec un arrêt précoce lorsque cela est possible afin de réduire le temps de calcul.
- Si les données d'entrée que nous utilisons représentent (ou non) un comportement sûr en ce qui concerne la propriété de sécurité, alors le modèle appris satisfait la propriété de sécurité lorsqu'il est formellement vérifié (ou non) en conséquence.
- Si le modèle "le plus adapté" appris ne satisfait pas la propriété de sécurité donnée, un modèle sécurisé peut être trouvé dans son voisinage (parfois après de nombreuses instances de la technique de recherche) qui satisfait la propriété.
- Plus le modèle le plus adapté est, en probabilité, loin de satisfaire la propriété de sécurité, plus il est difficile de trouver un modèle sécurisé proximal dans son voisinage.
- La mesure de distance, utilisée pour évaluer la dangerosité du modèle le plus adapté en le comparant au modèle sécurisé, augmente lorsque le modèle le plus adapté est, en probabilité, loin de satisfaire la propriété.
- Parfois, lors de l'exécution d'une instance de la technique d'exploration, nous ne trouvons pas de modèle sécurisé, en particulier pour les modèles plus dangereux.

Finalement, nous présentons quelques perspectives que nous avons envisagées pour de futurs travaux. Pour commencer, le processus d'apprentissage d'un RTGEM pourrait être exploré plus en détail. En d'autres termes, nous avons remarqué dans nos tests quelques fluctuations étranges lors du processus d'apprentissage (voir chapitre

4). Afin d'expliquer cette impression, nous avons essayé de construire de petits tests et avons réalisé que des tests plus importants devaient être effectués. Ces tests doivent inclure la variation des différents hyper paramètres possibles lors de la construction aléatoire d'un graphe (nombre de nœuds, paramètres, horizons, nombre d'arcs, nombre de timescale, densité d'échelles de temps, etc.) afin d'avoir un diagnostic complet pour une meilleure compréhension. La première difficulté fondamentale rencontrée dans une telle procédure est le coût élevé de ces opérations. Une autre difficulté est la capacité de trouver des explications/corrélations tout en évitant les biais de représentation. En d'autres termes, le choix des paramètres et des échelles de temps sur les modèles de référence doit se faire de manière à ne pas masquer certaines dépendances.

Une perspective qui est plus directement liée à la stratégie proposée cible la technique d'échantillonnage direct. Comme nous l'avons vu au chapitre 4, l'échantillonnage de RTGEM pourrait devenir coûteux même avec la règle d'arrêt précoce que nous utilisons. Par conséquent, une approche où nous pouvons résumer et approximer la totalité des effets que les parents ont sur les nœuds ciblés peut être envisagée. En d'autres termes, pour n'avoir qu'un seul paramètre $\lambda_{l_\varphi, effets}$ pour chaque étiquette concernée $l_\varphi \in L_\varphi$, qui est calculé avec une certaine heuristique par rapport à tous les parents $Pa_\varphi(l_\varphi)$. Une méthode pour y parvenir est de perturber les dépendances (ou de les supprimer) d'une manière qui peut nous donner la structure souhaitée dont nous avons discuté ci-dessus. En procédant comme tel, nous transformons un modèle donné M en M' sur lequel nous pouvons réapprendre les paramètres "abstraits" souhaités en utilisant les mêmes données d'entrée qui ont été utilisées pour apprendre le modèle M. La difficulté est de pouvoir trouver une démonstration formelle que le modèle M' est un objet mathématique similaire au modèle M dans une certaine mesure (ou une mesure de précision pour évaluer combien ils sont les mêmes), à partir d'un point de vue d'échantillonnage.

Une autre perspective concernant la technique d'exploration spatiale peut également être envisagée. Dans ce travail, nous avons choisi la technique Random Walk comme méthode d'exploration en raison de sa simplicité et de son efficacité relative. Une méthode plus complète pourrait être utilisée, comme une recherche exhaustive en largeur (ou profondeur en premier) qui garantit l'exploration de toutes les possibilités.

Cependant, cette technique pourrait devenir coûteuse si nous ne considérons pas une approche intelligente où nous créons une mémoire cache pour stocker les RTGEM déjà explorés dans le quartier. La raison derrière cela est l'utilisation d'opérateurs et d'opérateurs inverses pour les modifications. Par exemple, nous pouvons annuler un opérateur avant en faisant un opérateur arrière pour se replier sur un modèle précédemment testé, et vice versa. Une deuxième technique d'exploration spatiale peut également être testée: une méthode de recherche Greedy qui envisagerait d'améliorer la propriété de sécurité en probabilité. En d'autres termes, une technique SMC quantitative devrait être associée à la recherche Greedy afin de calculer la probabilité approximative de satisfaire la requête de sécurité pour un modèle après chaque modification possible. Lorsqu'un modèle qui améliore la probabilité de satisfaire la propriété de sécurité est trouvé, il est sélectionné comme nouvelle référence et la recherche se poursuit. Bien que nous ne sachions pas si c'est mieux que la recherche exhaustive ou la Random Walk, nous pensons que c'est une approche intéressante à explorer.

Enfin, un dernier objectif que nous envisageons d'atteindre dans un avenir proche est l'expérimentation de données réelles. En raison de problèmes d'accessibilité aux données fidèles du monde réel, nous n'avons pas encore pu effectuer ce type d'expérience. C'est quelque chose que nous prévoyons de faire avec GFI informatique sur différents cas d'utilisation, notamment sur les données générées par une application de sécurité routière. Bien que nous pensons qu'il y a beaucoup de traitement de données à faire pour l'adapter au format requis, nous pensons que nos méthodes sont suffisamment mûres pour être maintenant utilisées dans ce contexte.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE NANTES

Titre : Apprentissage et Vérification Statistique pour la Sécurité

Mot clés : Apprentissage de modèles, Vérification formelle, Statistical Model Checking, Recursive Timescale Graphical Event Models, Flux d'évènements, Evaluation de sécurité.

Résumé : Les principaux objectifs poursuivis au cours de cette thèse sont en premier lieu de pouvoir combiner les avantages de l'apprentissage graphique probabiliste de modèles et de la vérification formelle afin de pouvoir construire une nouvelle stratégie pour les évaluations de sécurité. D'autre part, il s'agit d'évaluer la sécurité d'un système réel donné. Par conséquent, nous proposons une approche où un "Recursive Timescale Graphical Event Model (RTGEM)" appris d'après un flux d'évènements est considéré comme représentatif du système sous-jacent. Ce modèle est ensuite utilisé pour vérifier une propriété de sécurité. Si la propriété n'est pas vérifiée, nous proposons une méthodologie de recherche afin de trouver un autre modèle qui la vérifiera. Nous analysons et justifions les différentes techniques utilisées dans notre approche et nous adaptons une mesure de distance entre Graphical Event Models. La mesure de distance entre le modèle appris et le *proximal secure model* trouvé nous donne un aperçu d'à quel point notre système réel est loin de vérifier la propriété donnée. Dans un soucis d'exhaustivité, nous proposons des séries d'expériences sur des données de synthèse nous permettant de fournir des preuves expérimentales que nous pouvons atteindre les objectifs visés.

Title: Machine Learning and Statistical Verification for Security

Keywords: Model-based learning, Formal verification, Statistical Model Checking, Recursive Timescale Graphical Event Models (RTGEMs), Event streams, Security assessments.

Abstract: The main objective of this thesis is to combine the advantages of probabilistic graphical model learning and formal verification in order to build a novel strategy for security assessments. The second objective is to assess the *security* of a given system by verifying whether it satisfies given properties and, if not, how far is it from satisfying them. We are interested in performing formal verification of this system based on event sequences collected from its execution. Consequently, we propose a model-based approach where a Recursive Timescale Graphical Event Model (RTGEM), learned from the event streams, is considered to be representative of the underlying system. This model is then used to check a *security property*. If the property is not verified, we propose a search methodology to find another *close* model that satisfies it. We discuss and justify the different techniques we use in our approach and we adapt a distance measure between Graphical Event Models. The distance measure between the learned "fittest" model and the found *proximal secure model* gives an insight on how far our real system is from verifying the given property. For the sake of completeness, we propose series of experiments on synthetic data allowing to provide experimental evidence that we can attain the desired goals.