Graphical event model learning and verification for security assessment

Dimitri Antakly, Benoit Delahaye, Philippe Leray

Graphical Event Model Learning and Verification for Security Assessment

Dimitri Antakly[1,2], Benoît Delahaye[2], and Philippe Leray[2]

[1] GFI informatique
[2] Université de Nantes/LS2N UMR CNRS 6004 , Nantes, France
dimitri.antakly@gfi.fr
{benoit.delahaye,philippe.leray}@univ-nantes.fr

Abstract. The main objective of our work is to assess the *security* of a given real world system by verifying whether this system satisfies given properties and, if not, how far it is from satisfying them. We are interested in performing formal verification of this system based on event sequences collected from its execution. In this paper, we propose a preliminary model-based approach where a Graphical Event Model (GEM), learned from the event streams, is considered to be representative of the underlying system. This model is then used to check a certain *security property*. If the property is not verified, we also propose a search methodology to find another *close* model that satisfies it. Our approach is generic with respect to the verification procedure and the notion of distance between models. For the sake of completeness, we propose a distance measure between GEMs that allows to give an insight on how far our real system is from verifying the given property. The interest of this approach is illustrated with a toy example.

Keywords: Model-based learning · Formal verification · Graphical Event Models (GEMs) · Event streams.

1 Introduction

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 behaviors and act at the right moment to intercept them. Some of the existing modeling formalisms are better tailored for the verification of given properties or hypothesis, others for learning behaviors and dependencies. Probabilistic finite automaton, for instance, were used in modeling and verification of known or desired behaviors [5]. Petri Nets were used in modeling and verification of several parallel tasks as well as in Process Mining [8]. Probabilistic graphical models were used in Machine Learning for the representation of dependencies between the different variables of a system.

Each of these cited formalisms has its own advantages and disadvantages. Nonetheless, all of the formalisms cited above and the ones that are in the same

family have a common flaw, the discretisation of time, which can be described as a representational bias in the learning of these formalisms. Thus, 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, Markov jump processes [6], Poisson networks and graphical event models (GEMs) [2]. In this work we are interested in *Recursive Timescale Graphical Models* (or RTGEMs) [2] a sub-family of GEMs, that present advantages compared to the other formalisms.

Appropriate learning and verification techniques should be adapted for the type of formalism that we wish to use. Standard model checking, for example, is used as an verification method [1]. 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) [4], which is an efficient technique based on simulations and statistical results. SMC has been successfully applied to probabilistic graphical models such as dynamic Bayesian networks (DBNs) in [3]. In the same way, SMC could be easily adapted to RTGEMs.

The main objective of this work is to learn a model (if one exists) that is at the same time representative of the real world system and secure. 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. Hence, we propose a strategy where we choose to learn the "optimal" RTGEM (the one that most fits the data). If this model does not satisfy a specific *security* property we seek to find another RTGEM, in its *close* neighborhood, that does. To do so, an appropriate model-based strategy is proposed and a distance measure is introduced in order to compare two RTGEMs. The strategy we propose contains three main steps, the learning of the model, the space exploration and model verification phase, and finally the distance calculation.

This paper is divided into four sections, section 2 consists in definitions and some background context that will be useful further on. Section 3 contains the proposed strategy, that is illustrated by a toy example in section 4. Section 5 is reserved for the conclusion and perspectives.

2 Background

The data we are using consists in timed sequences with strictly increasing timestamps (we use $t_0 = 0$ and $t^* = t_{n+1}$ as conventions). Thus, our data is written x_{t^*} for the sequence of events $(t_1, l_1), ..., (t_n, l_n)$, with $0 < t_i < t_{i+1} < t^*$ for all $1 \leq i \leq n-1$ and where l_i are labels chosen from a finite label vocabulary $\mathcal{L}$. We write $|x_{t^*}|$ for the size of our data x_{t^*} (the number of events in the sequence). The history at time t is the set of all the events that occurred before t, h_i denotes the ith history $h_i = (t_1, l_1), ..., (t_{i-1}, l_{i-1})$.

2.1 Graphical Event Models

A *Graphical Event Model* (GEM) is defined as a directed graph $\mathcal{G} = (\mathcal{L}, E)$ that can represent data of the type x_{t^*} as given above, as well as the dependencies between the different labels (or events) in time. In this work we are only interested in *Markov* GEMs, where the *conditional intensity functions* $\lambda_l(t \mid h)$ satisfy the following property:

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

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. We also note that conditional intensity functions are *piecewise-constant*, which means that they take constant values for a certain period of time based on the observed history. More details about GEMs can be found in [2].

2.2 Recursive Timescale Graphical Event Models

Recursive Timescale Graphical Event Models as described in [2] are a class of GEMs 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 $(0, t_h]$, where t_h is the highest value of T and is called the *horizon* of an edge e. An RTGEM $M = (\mathcal{G}, \mathcal{T})$ consists of a GEM $\mathcal{G} = (\mathcal{L}, E)$ and a set of timescales $\mathcal{T} = T_{e(e \in E)}$ corresponding to the edges E of the graph $\mathcal{G}$. The "recursive" form of this formalism comes from the fact that it is constructed using a forward search algorithm, usually starting from an empty model (only containing nodes that are not connected). 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 "add" operator adds a non-existing edge to the model and its corresponding timescale $T = (0, c]$, with c a constant. 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.

The conditional intensity functions now have parameters (they are also piecewise-constant), i.e. $\lambda_l(t \mid h) = \lambda_{l,c_l(h,t)}$ where the index $c_l(h, t)$ is the *parent count vector* of bounded counts over the intervals in the timescales of the parents of l. For the following example, we consider that all RTGEMs are bounded by 1, thus only the fact that a parent has occurred (or not) within the corresponding timescale is important.

Example 1. Consider the TGEM illustrated in figure 1. We have $\mathcal{L} = \{A, B, C, D\}$; for the event B for example, $c_B(h, t) = [0, 1, 1]$ means that there was no A in $[t-3, t)$, there was an A in $[t-6, t-3)$ and there was a D in $[t-5, t)$. Hence, the conditional intensity functions for the variable B are of the form: $\lambda_{B,000}$,

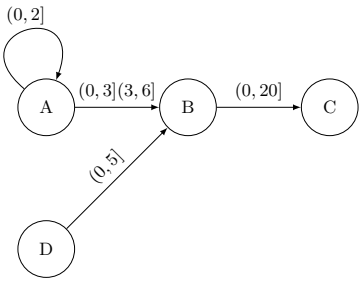

Fig. 1. Example of a four variables RTGEM

$\lambda_{B,001}$, etc. All conditional intensity functions are equal to constants making them piecewise-constant depending on the corresponding combination of parents.

In the learning algorithm proposed in [2], a backward search follows a forward search. This means that, implicitly, the authors use symmetric operators in their backward search. For the sake of convenience we write: $\mathcal{O}_F^{-1} = \{remove\_edge, fusion, remove\_interval\}$ for these symmetric operators, that do the inverse of the ones cited beforehand. In order to learn the fittest RTGEM, a Greedy Forward-Backward search is applied, based on an adapted Bayesian Information Criterion (BIC) score to select the RTGEM that most represents the data.

3 Proposed Approach

3.1 Problem Statement

In the learning phase of a model, where we want to learn the "fittest" model that best represents reality, we tend to adapt scores and metrics that evaluate the complexity and resemblance of the different learned models compared to the real data ($\mathcal{D}$), in order to chose the optimal one. From a *security* point of view, it is also important to verify if our model (that represents reality to a certain degree) satisfies certain security rules or properties, and if not, how far the current model is from verifying them. The probability that a model M verifies a security property ϕ is written $P(\phi \mid M)$. The problem we want to solve can be written as follows:

$$\exists M^*, M^* = argmax P(D \mid M) \; with \; P(\phi \mid M) > c \tag{1}$$

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 graph. We denote $l_\phi \in \mathcal{L}_\phi$ the labels of the events concerned by the security property ϕ . The problem as stated in equation 1 cannot be solved using classical multi-objective optimization heuristics, because of the fact that a qualitative property cannot be optimized, it is either true or false (it cannot become "truer").

3.2 Proposed Strategy

The strategy we propose to solve (1) is described in the following generic algorithm consisting in three main steps. 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 calculation between two models.

Algorithm 1 Proposed Strategy

input: $\mathcal{D}, \phi$
output: M^*, Δ
1: $M^o = \underset{M \in RTGEM}{\text{argmax}} \; P(\mathcal{D} \mid M)$
2: $\mathcal{N} = \mathcal{N}_c(\mathcal{N}_{l_\phi}(M^o))$
3: $M^* = \text{find}\{M \in \mathcal{N}, P(\phi \mid M) > c\}$
4: $\Delta = \text{SHD}(M^o, M^*)$

The first line of Algorithm 1 corresponds to the learning phase of the fittest RTGEM M^o (section 2.2). Lines 2 and 3 correspond to the model space exploration phase and model verification, where we try to find a model M^*, in the "close" neighborhood of M^o, that verifies the security property. This step will be explained in details in sections 3.3 and 3.4. The last line of the algorithm consists in calculating the distance between the optimal and the selected model (if one exists). The *distance* measure we propose will be defined and explained in section 3.5.

3.3 Model Space Exploration

The security properties we would like to verify address a number of particular variables l_ϕ in our model. The notation $\mathcal{N}_{l_\phi}(M^o)$, on line 2 of Algorithm 1, defines the neighborhood of M^o limited by the labels l_ϕ that are concerned by the security property ϕ. The neighborhood $\mathcal{N}$ that we consider, is the transitive closure ($\mathcal{N}_c$) of the previous neighborhood, limited by the number of allowed operators that is fixed beforehand. 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 l_ϕ of the model M^o, while staying in $\mathcal{N}$, in order to find a model that verifies the property. We check after each operation, if the obtained model satisfies the property or not. The search stops immediately when we find a model that verifies the property. The operations that are allowed are the ones in the sets $\mathcal{O}_F$ and $\mathcal{O}_F^{-1}$.

The find function can be defined using any search technique: 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 $P(\phi \mid M)$ in order to make it higher than c.

3.4 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. The second type of queries addresses time or the timing of given events. By adapting these queries (or a conjunction of them) to the system's security standards we obtain our *security properties* that allow us to classify a model as normal (or dangerous) from a security point of view. Certain types of properties can be formalized using an extended version of LTL (Linear Time Logic) [1], with the addition of past time intervals over the variables. For example we can write $\square^{100}(C \Rightarrow A_{(0,5]} \wedge B_{(0,10]})$, 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.

These types of properties could be verified using exact verification methods like standard model checking techniques [1], but standard model checking is subject to state space explosion and to the best of our knowledge was never adapted to Graphical Event Models. In practice we could also use an approximation method, like Statistical Model Checking (SMC) [4] that consists in simulating the model and verifying on each sampled data sequence if the given property is verified.

3.5 Distance Between Models

To the best of our knowledge, there is no existing metric of distance between RTGEMs. In the literature, the popular Hamming distance has been adapted for some probabilistic graphical models such as Bayesian networks [7]. 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. Consider two RTGEMs with the same set of labels $\mathcal{L}$, $G_1 = (\mathcal{L}, E_1)$ and $G_2 = (\mathcal{L}, E_2)$, we define:

$$\mathrm{SHD}(G_1, G_2) = \sum_{e \in E_{sd}} 1 + \sum_{e \in E_{inter}} d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) \tag{2}$$

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. A timescale in an RTGEM can be represented by a vector $v = [0, a, b, c, ...]$ where the values are the successive timestamp 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. Thus, we define the elementary distance as follows:

$$d(\mathcal{T}(e, G_1), \mathcal{T}(e, G_2)) = \frac{v_{nid}}{v_{nid} + v_{id}} \tag{3}$$

Equation (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.

4 Toy example

The purpose of the following example is to illustrate the interest of the proposed strategy on a real life application. We consider a prepaid card online service, where the possible actions are *Recharge*, *Check account*, *Transfer money* (to transfer money to his or to another bank account) and *Log out*. We suppose that the optimal RTGEM (M^o) that best fits the real behavior of users is as shown in figure 2. A security query ϕ that we can verify on this example can 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 property $P(\phi \mid M^o) > 0.8$. In other words we would like to ensure a behavior for users that we consider safe: 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.

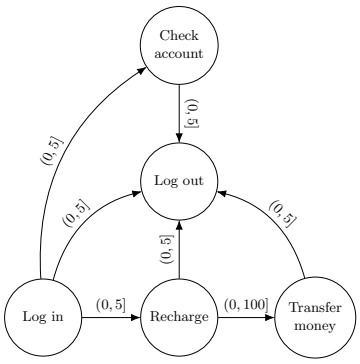

Fig. 2. The learned model

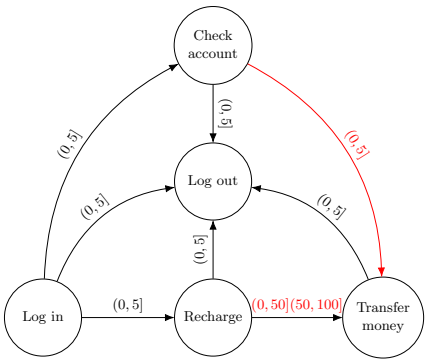

Fig. 3. A modified more secure model

We notice, only from looking at the model, that $P(\phi \mid M^o)$ is low and that the learned behavior does not verify the security property mainly because of the missing dependency between "*Check account*" and "*Transfer money*". Hence, 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 ($l_\phi = \{\text{Transfer Money}, \text{Recharge}, \text{Check account}\}$), we can obtain the RTGEM (M^*) of figure 3, where the modifications are in red. Clearly, this obtained model is more *secure* considering the security property ϕ, because we now have a smaller interval between "recharge" and "transfer money" that is taken into consideration and we have added the edge between "check account" and "transfer money".

The distance is $\text{SHD}(M^o, M^*) = 1.333$ in this case, because of the added edge and the split on the interval.

5 Conclusion and Perspectives

In conclusion, we have proposed a preliminary 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 the real data, 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.
In the future we plan to apply our algorithm and start the experimentation on a real world case study to evaluate its complexity and advantages, especially in systems verification, in order to compare it with other verification approaches. Finally, we are also planning on studying the correlation between the relative distance measure we compute and the change in the satisfaction probability of a certain property.

References

1. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
2. Gunawardana, A., Meek, C.: Universal models of multivariate temporal point processes. In: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics. pp. 556–563 (2016)
3. Langmead, C.J.: 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. pp. 201–212. Life Sciences Society (2009)
4. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: International Conference on Runtime Verification. pp. 122–135. Springer (2010)
5. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning probabilistic automata for model checking. In: Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on. pp. 111–120. IEEE (2011)
6. Rao, V., Teh, Y.W.: Fast MCMC sampling for Markov jump processes and extensions. The Journal of Machine Learning Research **14**(1), 3295–3320 (2013)
7. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing Bayesian network structure learning algorithm. Machine learning **65**(1), 31–78 (2006)
8. Van Der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin Heidelberg (2014)