SqSelect: Automatic assessment of Failed Error Propagation in state-based systems*

Alfredo Ibias[a], Manuel Núñez[a,b]

[a]Design and Testing of Reliable Systems research group
Universidad Complutense de Madrid, Madrid, Spain
[b]Institute of Knowledge Technology
Universidad Complutense de Madrid, Madrid, Spain

ARTICLE INFO

Keywords:
Software Testing
Failed Error Propagation
Expert Systems
Information Theory

Abstract

Current software systems are inherently complex and this fact strongly complicates, and makes more expensive, to validate them. Therefore, it is a must to provide methodologies, supported by tools, that can direct validation activities so that they focus on specific aspects of the system (e.g. its critical parts, common errors produced by developers, components that are expensive to fix after deployment, etc). Among the different validation techniques, testing is the most widely used. In this paper we focus on one of the main problems when testing systems with many components: the likelihood of Failed Error Propagation (FEP). FEP appears when we have faulty components such that their wrong behaviour is not revealed when isolatedly testing them but that might produce an error when they are combined with other components.

Given a component, it is not possible to automatically assess the likelihood of FEP. However, previous work has shown that there is a strong correlation between the likelihood of FEP and an Information Theory notion called Squeeziness. Recent work has shown that it is possible to compute different values of Squeeziness (essentially, Squeeziness depends on a positive real value) and some of them are more suitable to estimate FEP.

In this paper we present our tool SqSelect. SqSelect receives either a specific system or its more important characteristics (number of states, maximum and minimum number of outgoing transitions from a state, size of the input and output alphabets) and returns interesting data that can help the tester to estimate the presence of FEP. In particular, our tool provides the most promising value(s) of the parameter associated with Squeeziness so that the likelihood of FEP can be more accurately estimated. In order to compute these values, our tool relies on an artificial neural network that has been extensively trained (compressing the information from around 250,000 systems and around 1,500,000 executions).

1. Introduction

The main goal of an expert system is to replace a human expert in the process of providing a decision or verdict. Usually, expert systems have to deal with big amounts of information so that they are indeed more effective than the replaced human experts because they will not be able to process all the needed information. Expert systems currently deal with very heterogeneous systems: they can assess the risks of cost overruns in real power plants (Islam et al., 2019), rank hockey players according to difference performance metrics (Gu et al., 2019), deal with different environments in the Health domain (Díaz et al., 2020; García de Prado et al., 2017), or detect security attacks (Roldán et al., 2020), among many others. The goal of this paper is to introduce an expert system such that given a state-based system, and using an artificial neural network (ANN) previously trained, provides relevant information, in a structured and comprehensive way, that will help testers by guiding some of the testing activities.

Software Testing (Ammann and Offutt, 2017; Myers et al., 2011) is the main validation technique to detect faults in software systems. Although Software Testing was traditionally considered to be mainly *informal*, this situation has changed. Actually, from the 1990s it is well-known and understood that testing can be formalised (Gaudel, 1995). Thus, formal approaches to testing is a flourishing field of study (Cavalli et al., 2015; Hierons et al., 2009) and there are many tools supporting the theoretical frameworks (Marinescu et al., 2015; Shafique and Labiche, 2015). One of the main scenarios where formal methods for testing are fundamental is black-box testing: the tester provides inputs to the system under test (SUT), without having access to its internal structure, observe its reaction (outputs) and analyse whether these reactions are expected or not.

Testing a complex system consists of many strands. For example, depending on the amount of resources available for testing, we can perform activities such as analyse the isolated behaviour of different components, assess the robustness of a system or check whether a system is using more resources than expected. However, one facet is unavoidable: current software systems are composed of many compon-

* This work has been supported by the Spanish MCIU-FEDER (grant number FAME, RTI2018-093608-B-C31), the Region of Madrid (grant number FORTE-CM, S2018/TCS-4314), the Region of Madrid – Complutense University of Madrid (grant number PR65/19-22452) and the Santander – Complutense University of Madrid (grant number CT63/19-CT64/19).

aibias@ucm.es (A. Ibias); mn@sip.ucm.es (M. Núñez)
https://alfredoibias.com/ (A. Ibias); http://antares.sip.ucm.es/manolo/ (M. Núñez)
ORCID(s): 0000-0002-3122-4272 (A. Ibias); 0000-0001-9808-6401 (M. Núñez)

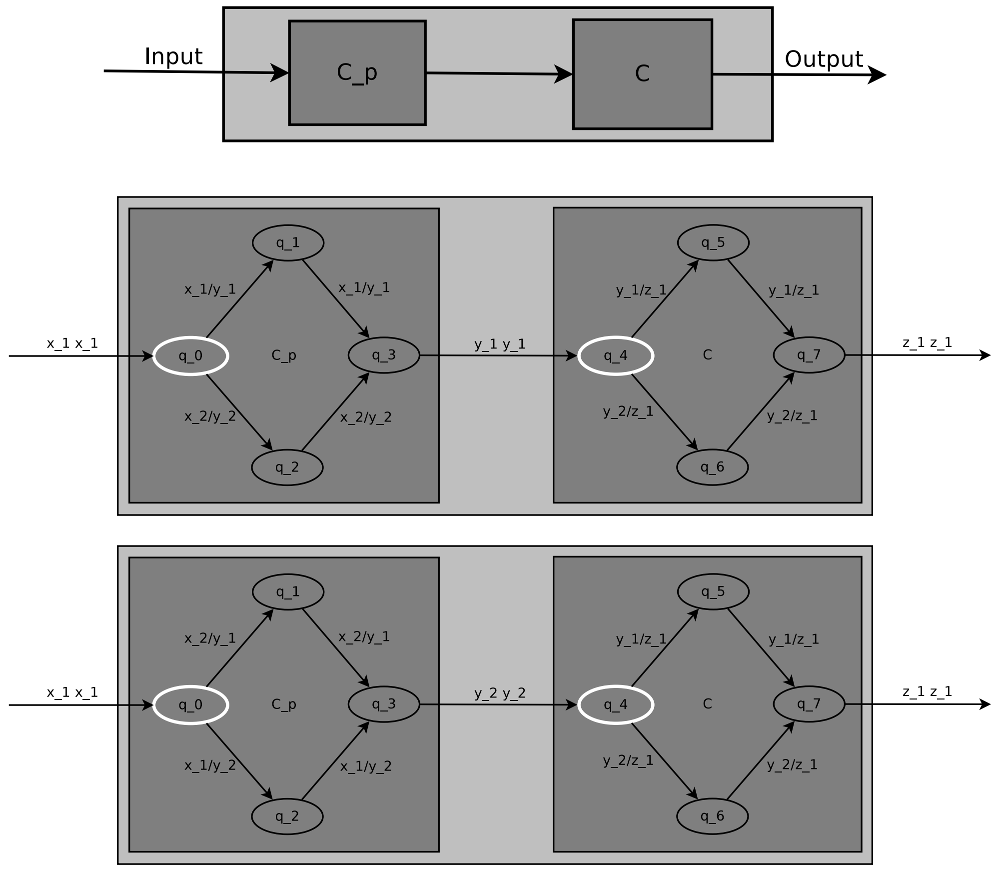

Figure 1: Representation of our testing scenario.

ents and an appropriate testing plan must assess how different units/components work together. One of the main problems that can appear when testing components that are working together is Failed Error Propagation (Laski et al., 1995; Woodward and Al-Khanjari, 2000) (FEP). Suppose that we execute a faulty statement of a component, so that the resulting internal state becomes faulty, but the differences between the faulty and correct state are not reflected in the output provided by the component. If this component works alone, then FEP does not represent a problem but if we *transfer* the (faulty) state to a different component, then an unforeseen error might appear.

In Figure 1 we graphically show the scenario where FEP can appear. We use Finite State Machines (FSMs) to represent components but any other state-based formalism could be used and the adaption of our framework would be straightforward. We have two components, C_P and C. The former receives a sequence of inputs. In particular, this sequence can be provided by the tester. Then, C_P performs some computations and sends a sequence of inputs to C. In order to not violate the assumption concerning a black-box framework, we assume that the actions sent from C_P to C cannot be observed by the tester. C will produce a sequence (they will represent outputs of the whole system) that can be observed by the tester. In this context, the component C_P could produce an unexpected sequence, this is the one received by C, but C might produce the same result from the expected and unexpected sequences: C would introduce a form of FEP that makes it more difficult to find faults in C_P. These faults could be revealed if C_P is combined with another component C'.

Assume that we want to implement the component C_p given in the middle part of Figure 1 and that this component will be paired with component C. In this setting, it will be difficult to unmask a faulty implementation of C_p, such as the one shown in the lower part of Figure 1, because C returns the same response, the sequence $z_1 z_1$, to the sequences $y_1 y_1$ (produced by a correct implementation of C_p receiving $x_1 x_1$) and $y_2 y_2$ (produced by a faulty implementation of C_p also receiving $x_1 x_1$). Note, as we already said, that a tester will not be able to observe whether the sequence provided to C is $y_1 y_1$ or $y_2 y_2$.

Several empirical studies have shown that FEP hampers testing (Masri et al., 2009; Santelices and Harrold, 2011; Wang et al., 2009): in 13% of the examined programs, a total of 60% or more of the tests suffered from FEP (Masri et al., 2009). Although it is not clear how Software Testing could be better designed to ameliorate the problems caused by FEP, it is important to reduce the probability that test cases will suffer from FEP and, therefore, it is useful to have different metrics that can help us to identify parts of a system that are more likely to lead to FEP (Androutsopoulos et al., 2014). There is a line of research looking for measures to indirectly estimate FEP and it seems that an Information Theory measure called Squeeziness (Clark and Hierons, 2012; Clark et al., 2019) is a good candidate. Actually, an empirical study (Androutsopoulos et al., 2014) of 30 programs and more than 7×10^6 tests showed that the Spearman rank correlation of Squeeziness with FEP is close to 0.95. Squeeziness was adapted to a black-box testing framework (Ibias et al., 2019) and the correlation with FEP was also very high (around 0.80 for both Pearson and Spearman correlations).

The original notion of Squeeziness relies on Shannon's entropy (Shannon, 1948). Although this is a standard notion, it is not the only possibility. Actually, it is possible to define an infinite number of notions of entropy, one for each $\alpha \in \mathbb{R}_+$: Rényi's entropy (Rényi, 1961). Shannon's entropy corresponds to Rényi's entropy when $\alpha = 1$. Recent work (Ibias and Núñez, 2020) has studied the different notions of Squeeziness induced by each value of α and, interestingly enough, has shown that for each system there is a different best value of α. By best value we mean the value such that the corresponding notion of Squeeziness has the highest rank correlation with FEP. In particular, even though $\alpha = 1$ always provides good correlations, all the experiments revealed that the best value was never reached with this value. Therefore, given a system, it would be interesting to know the optimum value of α so that the FEP of the system can be estimated with a higher precision. In order to compute this value, there are two important facts that must be taken into account. First, the computation of this value for a specific system needs a non-negligible amount of computing power. Second, it has been observed (Ibias and Núñez, 2020) that similar systems obtain similar values. Thus, it would be enough to have an approximate method to compute a good enough approximation of the desired value. We have developed a tool, called SqSelect, that receives either a specific system or its defining characteristics (among others, number of states, maximum and minimum number of outgoing transitions, size of the input and output alphabets) and provides several interesting data that can help the tester to decide the amount of testing that should be used to evaluate FEP. Specifically, SqSelect provides the value of Squeeziness that more accurately estimates the likelihood of having cases of FEP in the system. In order to compute our solution, integrated in SqSelect, we rely on an ANN that we have extensively trained using $249{,}000$ different systems and a total of $1{,}494{,}000$ executions, obtaining a mean loss of only 1.280935

Concerning related work, there is plenty of work on Information Theory and testing (Androutsopoulos et al., 2014; Clark et al., 2015; Clark and Hierons, 2012; Feldt et al., 2016, 2008; Ibias et al., 2019, 2021; Miranskyy et al., 2012; Pattipati and Alexandridis, 1990; Pattipati et al., 1990; Sagarna et al., 2007; Yoo et al., 2013) where theoretical frameworks, sometimes supported by empirical evidence, are presented and discussed. However, to the best of our knowledge, there do not exist frameworks where relevant Information Theory measures are automatically (and using a small computing power) estimated. There is a recent proposal to use expert systems in testing (Cañizares et al., 2019) but their focus is on memory systems and the underlying testing approach is Metamorphic Testing. It is natural that expert and recommender systems use artificial neural networks, and we may mention some recent work (de Mesquita Sá Junior et al., 2019; Ayala et al., 2020), but their goal is not related to testing and their field of application are very far from ours. Finally, it is worth mentioning that artificial neural networks are a good tool to confront testing activities (Serna M. et al., 2019) but they have never been used to estimate Information Theory measures. There is some work in the field of profiling tools for error propagation but our work diverges in several aspects and, therefore, it is difficult to compare it with them. PROPANE (Hiller et al., 2002) needs the source code to work, that is, it can be classified as a white-box approach. EPIC (Hiller et al., 2004) considers a running system. Therefore, it can be considered to be a black-box approach. A similar consideration applies to later work dealing with the analysis of operating systems (Coppik et al., 2017; Johansson and Suri, 2005), multi-threaded programs (Chan et al., 2017), automotive systems (Piper et al., 2015) and software architecture (Abdelmoez et al., 2004). Although SqSelect also works with a black-box, the abstract representation of the systems that we consider, represented as FSMs, is very far from the systems studied in these profiling approaches. However, the most important difference between the previously mentioned approaches and ours concerns how error propagation is studied. The former ones analyse error propagation of the system with the goal of detecting dependencies between components and determine which components can have the most harmful errors for the whole system (because their errors propagate more than others). In our case, our goal is to determine which components can hide errors from being detected, that is, which components do not propagate errors. In Figure 2 we sketch the main features of our tool and of the tools and studies more related to ours. Specifically, we mention whether the study uses Information Theory, is tool-supported, works with a white/black box or considers error propagation or failed error propagation.

The rest of the paper is structured as follows. In Section 2 we introduce the notations and concepts that we will use. Section 3 provides a high level description of the behaviour of SqSelect and explains how our ANN was designed, trained and tested. In Section 4 we present the different modes that are available in our tool. In Section 5 we discuss some design decisions and justify the chosen options. In Section 6 we give some representative case studies to show the behaviour of our tool. Finally, in Section 7 we present our conclusions and some lines for future work.

2. Background

In this section we present the basic concepts and notions that are required to understand the work presented in this paper.

Tool or framework	Information Theory	Tool supp.	Black box	White box	FEP	Error Prop.	Field of application
SqSelect	General Squeeziness	Yes	Yes	No	Yes	No	FSMs
(Ibias et al., 2019)	Squeeziness	Yes	Yes	No	Yes	No	FSMs
TrEKer (Coppik et al., 2017)	No	Yes	Yes	No	No	Yes	Operating Systems Kernels
IPA (Chan et al., 2017)	No	Yes	No	Yes	No	Yes	Analysis of Multithreads
PBM (Piper et al., 2015)	No	Yes	Yes	No	No	Yes	Automotive Systems
(Androutsopoulos et al., 2014)	Squeeziness	No	No	Yes	Yes	No	Source Code
(Clark and Hierons, 2012)	Squeeziness	No	No	Yes	Yes	No	Source Code
(Johansson and Suri, 2005)	No	No	Yes	No	No	Yes	Operating Systems
EPIC (Hiller et al., 2004)	No	Yes	Yes	No	No	Yes	Data Errors in Software
(Abdelmoez et al., 2004)	No	No	No	Yes	No	Yes	Software Architecture
PROPANE (Hiller et al., 2002)	No	Yes	No	Yes	No	Yes	Source Code

Figure 2: Comparison of SqSelect with other (failed) error propagation tools and frameworks.

2.1. Basic concepts

Given a set A, we let A^* denote the set of finite sequences composed from elements of A; as usual, we consider that $\epsilon \in A^*$ denotes the empty sequence. We let A^+ denote the set of non-empty sequences of elements of A. A^k denotes the set of sequences with length $k \geq 1$. We let $|A|$ denote the cardinal of set A. Given a sequence $\sigma \in A^*$, we have that $|\sigma|$ denotes its length. Given a sequence $\sigma \in A^*$ and $a \in A$, we have that σa denotes the sequence σ followed by a and $a\sigma$ denotes the sequence σ preceded by a.

Throughout this paper we let I be the set of input actions and O be the set of output actions. In our context, it is important to distinguish between input actions and *inputs* of the system. Specifically, an input of a system is a non-empty sequence of input actions, that is, an element of I^+. We have a similar situation concerning output actions and *outputs* of the system. A *Finite State Machine* is a (finite) labelled transition system in which transitions are labelled by an input/output pair.

Definition 1. A tuple $M = (Q, q_{in}, I, O, T)$ is a *Finite State Machine* (FSM), where Q is a finite set of states, $q_{in} \in Q$ is the initial state, I is a finite set of input actions, O is a finite set of output actions, and $T \subseteq Q \times I \times O \times Q$ is the transition relation. A transition $(q, i, o, q') \in T$, also denoted by $(q, i/o, q')$, means that from state q after receiving the input action i it is possible to move to state q' and produce the output action o.

We say that M is *deterministic* if for all $q \in Q$ and $i \in I$ there exists at most one pair $(q', o) \in Q \times O$ such that $(q, i/o, q') \in T$. In this paper we consider deterministic FSMs.

In this paper we use this simple formalism to define processes but our methodology can be easily adapted to deal with any other state-based formalism as long as it provides notions of inputs and outputs. Note that FSMs, despite their simplicity, are frequently used to represent a wide variety of systems: state-based software systems, logic circuits, communication protocols, etc. An FSM can be seen as a diagram where nodes denote states of the FSM and transitions are represented by arcs between the nodes. We use a double circle to denote the initial state.

As stated in the previous definition, we consider that FSMs are deterministic. This restriction is taken to mimic the white-box scenario where Squeeziness was originally introduced and considered, as usual, that programs are deterministic.

Definition 2. Let $M = (Q, q_{in}, I, O, T)$ be an FSM. We say that $(i_1, o_1) \ldots (i_k, o_k) \in (I \times O)^*$ is a *trace* of M if there exist states $q_1 \ldots q_k \in Q$ such that for all $1 \leq j \leq k$ we have $(q_{j-1}, i_j/o_j, q_j) \in T$, where $q_0 = q_{in}$. We denote by $traces(M)$ the set of traces of M. Note that $\epsilon \in traces(M)$.

Let $s = i_1 \ldots i_k \in I^*$ be a sequence of input actions. We define $out_M(s)$ as the set

$$\{o_1 \ldots o_k \in O^* \mid (i_1/o_1) \ldots (i_k, o_k) \text{ trace of } M\}$$

Note that if M is deterministic, then this set is either empty or a singleton. In the last case we will simply write $out_M(s) = o_1, \ldots, o_k$.

We define dom_M as the set $\{s \in I^* \mid out_M(s) \neq \emptyset\}$. Similarly, we define $image_M$ as the set

$$\{o_1 \ldots o_k \in O^* \mid \exists s \in I^* : o_1 \ldots o_k \in out_M(s)\}$$

We denote by $dom_{M,k}$ the set $dom_M \cap I^k$. Similarly, we denote by $image_{M,k}$ the set $image_M \cap O^k$.

Next we present a simple example to illustrate the previous concepts.

Figure 3: FSM example.

Example 1. In Figure 3 we have an FSM with 10 states, S0 is its initial state (the double circle), a set of inputs $I = \{a, b, c\}$ and set of outputs $O = \{x, y, z\}$. For example, $(a, z)(a, x)$ is a trace that takes the system from state S0 to state S4, $out_M(aa) = zx$ while $out_M(ba) = \emptyset$. We have

$$dom_M = \{a, b, c, aa, ab, bb, ca, cb, cc\}$$
$$image_M = \{z, y, zx, zz, yz\}$$

Moreover, $dom_{M,1} = \{a, b, c\}$ and $image_{M,2} = \{zx, zz, yz\}$.

Finally, we review the concept of Failed Error Propagation. This kind of error appears when we have a fault in the program but the error associated with this fault does not have an effect in the output of the program. Using the RIP model (Ammann and Offutt, 2017) (stating that three conditions must be present for a failure to be observed: Reachability, Infection and Propagation), we might reach a fault such that the infection is not propagated to the final (observable) state. The lack of access to the internal structure of the SUT negates the tester the possibility to detect these faults in a black-box scenario[1]. As we have already mentioned in the introduction of this paper, empirical studies have shown that many systems suffer from FEP (Masri et al., 2009): in 13% of the analysed programs, a total of 60% or more of the tests suffered from FEP.

It might be thought that if the previous faults do not alter the outputs, then we should not worry about them. However, complex forms of FEP include faults whose errors do not propagate to the outputs in some cases, but they generate wrong outputs in other cases. These forms of FEP are especially dangerous because their detection depends on executing the right test, the one that propagates the error to the output. Due to resources, time and budget restrictions this could be a hard work, because the testing process has to be restricted to the application of some selected cases: the ones that are more promising at detecting faults. However, if the right test is not in the selected test suite (maybe because it will only detect this fault and the other tests could detect more than one fault at the same time), then the error will remain undetected.

[1] Note that these faults can be observed in a white-box scenario because the tester has access to the code and, therefore, it is possible to follow the produced error.

2.2. Shannon-based Squeeziness in a black-box setting

Before we introduce the notion of Squeeziness, we define some auxiliary concepts. First, note that FSMs can be seen as functions that transform inputs into outputs. Projections of these functions restrict the function to inputs of a certain length. Finally, we review the notion of *collision*, which happens when two different inputs produce the same output.

Definition 3. Let $M = (Q, q_{in}, I, O, T)$ be an FSM. We define $f_M : dom_M \rightarrow image_M$ as the function such that for all $s \in dom_M$ we have $f_M(s) = out_M(s)$. Given $k > 0$, we define $f_{M,k}$ as the function $f_M \cap (I^k \times O^k)$, where f_M denotes the associated set of pairs. Let $t \in image_M$. We define $f_M^{-1}(t)$ to be the set $\{s \in I^{<\omega} \mid f_M(s) = t\}$.

Let $s_1, s_2 \in I^{<\omega}$. We say that s_1 and s_2 collide for M if $s_1 \neq s_2$ and $f_M(s_1) = f_M(s_2)$.

Example 2. Consider again the FSM depicted in Figure 3. For example, f_M maps $aa \mapsto zx$, $ab \mapsto zz$ and $bb \mapsto yz$, among others. The inputs a, ca and cb collide; the inputs a and c also collide.

Squeeziness has been successfully used to estimate the existence of FEP in white-box (Clark and Hierons, 2012; Clark et al., 2019) and black-box (Ibias et al., 2019) scenarios. It represents the amount of information lost by a function. Thus, Squeeziness for an FSM can be defined as the Squeeziness of the function that represents the FSM. In order to properly compute it, it was necessary to define how inputs are chosen and outputs are returned. A probabilistic view, where a random variable is associated with each set of relevant inputs/outputs, can be considered. Specifically, a random variable can be associated with the set of inputs/outputs of a certain length (that is, there are different random variables associated with $I^1, I^2, \ldots, O^1, O^2, \ldots$). Since $dom_{M,k}$ includes the inputs of length equal to k that M can perform and $image_{M,k}$ includes the outputs of length equal to k that M can produce after receiving an element of $dom_{M,k}$, random variables ranging over each set are defined as $\zeta_{dom_{M,k}}$ and $\zeta_{image_{M,k}}$, respectively.

Definition 4. Let S be a set and ζ_S be a random variable over S. We denote by π_S the probability distribution induced by ζ_S.

Let $M = (Q, q_{in}, I, O, T)$ be an FSM and $k > 0$. Let us consider two random variables $\zeta_{dom_{M,k}}$ and $\zeta_{image_{M,k}}$ ranging, respectively, over the domain and image of $f_{M,k}$. The Squeeziness of M at length k is defined as

$$Sq_k(M) = H(\zeta_{dom_{M,k}}) - H(\zeta_{image_{M,k}})$$

where $H(\zeta_S)$ denotes the (Shannon's) entropy of the random variable ζ_S that ranges over the set S, which is defined as

$$H(\zeta_S) = -\sum_{s \in S} \pi_S(s) \cdot \log_2(\pi_S(s))$$

There is an important remark concerning random variables associated with inputs and outputs: given an FSM M, $k > 0$ and a random variable $\mathcal{X}_{dom_{M,k}}$, we have that the probability distribution of the random variable $\mathcal{X}_{image_{M,k}}$ is completely determined. This is because for each element $t \in image_{M,k}$ we have that

$$P_{\mathcal{X}_{image_{M,k}}}(t) = \sum_{s \in f_M^{-1}(t)} P_{\mathcal{X}_{dom_{M,k}}}(s)$$

Therefore, the formulation of Squeeziness is

$$Sq_k(M) = -\sum_{t \in image_{M,k}} \left(\sum_{s \in f_M^{-1}(t)} P_{\mathcal{X}_{dom_{M,k}}}(s) \right) R_M(t)$$

where the term $R_M(t)$ is equal to

$$\sum_{s \in f_M^{-1}(t)} \frac{P_{\mathcal{X}_{dom_{M,k}}}(s)}{P_{\mathcal{X}_{dom_{M,k}}}(f_M^{-1}(t))} \log_2 \left(\frac{P_{\mathcal{X}_{dom_{M,k}}}(s)}{P_{\mathcal{X}_{dom_{M,k}}}(f_M^{-1}(t))} \right)$$

Example 3. Consider again the FSM given in Figure 3. If we assume that inputs are uniformly distributed (later we will explain why this is a reasonable assumption), then we have that

$$\begin{aligned} Sq_1(M) = & \\ & = -3 \cdot \frac{1}{3} \log_2(\tfrac{1}{3}) - \left(-\frac{1}{3} \log_2(\tfrac{1}{3}) - \frac{2}{3} \log_2(\tfrac{2}{3}) \right) \\ & \approx 1.585 - 0.918 = 0.667 \end{aligned}$$

Using similar computations we have that

$$Sq_2(M) \approx 2.585 - 1.459 = 1.126$$

2.3. Rényi's-based Squeeziness in a black-box setting

The original work on Squeeziness used Shannon's entropy, but there exists a general definition of entropy, depending on a parameter, called *Rényi's entropy* (Rényi, 1961).

Definition 5. Let S be a set and $\mathcal{X}_S$ be a random variable over S. Let $\alpha \in \mathbb{R}_+ \setminus \{1\}$. The *Rényi's entropy* of the random variable $\mathcal{X}_S$ with respect to α, denoted by $H_\alpha(\mathcal{X}_S)$, is defined as:

$$H_\alpha(\mathcal{X}_S) = \frac{1}{1-\alpha} \log_2 \left[\sum_{s \in S} P_{\mathcal{X}_S}(s)^\alpha \right]$$

We have that when α tends to 1, Rényi's entropy becomes Shannon's entropy (Rényi, 1961), that is,

$$\lim_{\alpha \to 1} H_\alpha(\mathcal{X}_S) = H(\mathcal{X}_S) = -\sum_{s \in S} P_{\mathcal{X}_S}(s) \log_2(P_{\mathcal{X}_S}(s))$$

Squeeziness of an FSM using Rényi's entropy has been recently defined (Ibias and Núñez, 2020) in the same way as (Shannon's) Squeeziness was defined in Definition 4.

Definition 6. Let $M = (Q, q_{in}, I, O, T)$ be an FSM and $k > 0$. Let us consider two random variables $\mathcal{X}_{dom_{M,k}}$ and $\mathcal{X}_{image_{M,k}}$ ranging, respectively, over the domain and image of $f_{M,k}$. Let $\alpha \in \mathbb{R}_+ \setminus \{1\}$. Rényi's Squeeziness of M at length k with respect to α is defined as

$$Sq_{\alpha,k}(M) = H_\alpha(\mathcal{X}_{dom_{M,k}}) - H_\alpha(\mathcal{X}_{image_{M,k}})$$

In the Appendix I of the paper we provide an equivalent, but simpler to compute, formulation of Rényi's Squeeziness (see Lemma 1).

The previously defined notion of Squeeziness is parameterised by the distribution over the inputs of the function (that is, over the input sequences that the FSM can perform). If we know the actual distribution, then we can use this. If we do not know the distribution, then there is a need to choose one and we now discuss two approaches to do this.

2.3.1. Maximum entropy principle

We can select the distribution that maximises entropy. If there are no further restrictions, maximum entropy is obtained with a uniform distribution (Acharya et al., 2017; Cover and Thomas, 1991). Then, under this distribution, the weight of a single element of $dom_{M,k}$ is $\frac{1}{|dom_{M,k}|}$ and the weight of the inverse image of an output $t \in image_{M,k}$ is equal to $\frac{|f_M^{-1}(t)|}{|dom_{M,k}|}$.

Under these assumptions, and after some algebraic manipulations, the formula for Rényi's Squeeziness becomes:

$$Sq_{\alpha,k}(M) = \frac{1}{1-\alpha} \log_2 \left(\frac{|dom_{M,k}|}{\sum_{t \in image_{M,k}} |f_M^{-1}(t)|^\alpha} \right)$$

As usual, we have two special cases: α tending to 1 and α tending to ∞. If α tends to 1, then we are using Shannon's entropy and we have the following simplified formulation (Ibias et al., 2019):

$$Sq_{1,k}(M) = \frac{1}{|dom_{M,k}|} \sum_{t \in image_{M,k}} |f_M^{-1}(t)| \log_2(|f_M^{-1}(t)|)$$

If α tends to ∞, then we are using min-entropy and, after some algebraic manipulations, we obtain the following formulation:

$$Sq_{\infty,k}(M) = \log_2 \left(\max_{t \in image_{M,k}} |f_M^{-1}(t)| \right)$$

2.3.2. Maximum loss of information

Another option is to consider the worst case scenario, that is, the scenario where the probability distribution induces the maximum loss of information. In order to maximise the loss of information, we need to maximise Squeeziness. In this case, the probability distribution is the one that is uniformly distributed in the largest inverse image of an element of the outputs and zero elsewhere (Clark and Hierons, 2012). Formally, consider $t' \in image_{M,k}$ such that for all $t \in image_{M,k}$ we have that $|f_M^{-1}(t')| \geq |f_M^{-1}(t)|$. Then,

$$\mu_{dom_{M,k}}(s) = \begin{cases} \frac{1}{|f_M^{-1}(t')|} & \text{if } s \in f_M^{-1}(t') \\ 0 & \text{otherwise} \end{cases}$$

Using this probability distribution, Rényi's Squeeziness becomes:

$$Sq_{\infty,k}(M) = \log_2 |f_M^{-1}(t')|$$

In this case, unlike the previous ones, Squeeziness does not depend on the value of α. In particular, the two special cases (α tending to 1 and α tending to ∞) have the same formulation.

2.4. Probability of Collisions

FEP happens when the expected and faulty inputs, received from another component, produce the same output. Therefore, a collision (see Definition 3) is an indication of FEP and it is useful to compute the probability of having collisions in the FSM under study. This probability is given by PColl (Clark and Hierons, 2012).

Definition 7. Let M be an FSM and $k > 0$. Let $image_{M,k} = \{t_1, \dots, t_n\}$ and for all $1 \leq i \leq n$ let $I_i = f_{M,k}^{-1}(t_i)$ and $m_i = |f_{M,k}^{-1}(t_i)|$. We have that $d = \sum_{i=1}^{n} m_i$ is the size of the input space.

Given a uniform distribution over the inputs, the probability of s and s' both being in the same set I_i is equal to $p_i = \frac{m_i(m_i-1)}{d(d-1)}$. We have that the probability of having a collision in M for sequences of length k, denoted by $PColl_k(M)$, is given by

$$PColl_k(M) = \sum_{i=1}^{n} \frac{m_i(m_i-1)}{d(d-1)}$$

With regard to this definition, a topic that has been already addressed is the potential to use $PColl_k(M)$ instead of Squeeziness. The problem with using $PColl_k(M)$ is that it is hard to compute. While this also applies to Squeeziness, the latter has the advantage of being an information theoretic measure. As a result, we can use Information Theory to either estimate or bound measures (Boreale and Paolini, 2014; Chothia et al., 2014), what will suffice for our final goal.

2.5. Artificial Neural Networks

We review the main concepts around Artificial Neural Networks (ANN). They try to mimic the brain behaviour. We use them to infer the value of α that more appropriately asses the likelihood of the presence of cases of FEP. There are two types of ANNs: *classification* and *regression*. The former ones classify the input in one of many predefined classes while the latter ones return a real number that represents the value associated to the input in a one dimensional space.

An ANN is composed of different layers, where each layer has an associated activation function and a set of neurons. The layers of an ANN are commonly grouped in three types:

- Input layer: the layer that receives the input. It has as many neurons as the dimension of the input.
- Hidden layers: the layers inside the ANN that are not accessible from outside. They have various sizes and activation functions.
- Output layer: the last layer of the ANN. It has as many neurons as classes for classification ANNs, or one neuron in the case of regression ANNs.

A neuron is a simple agent that performs the following steps:

- Compute the weighted sum of the outputs of the previous layer. For each neuron of the previous layer, the agent has an associated weight that multiplies the output value of that neuron. Then, the agent sums all the weighted output values.
- Apply the activation function. The agent applies the activation function associated to its layer to the result of the previous step. These activation functions are non-linear.

Activation functions are non-linear because if they were linear, then the ANN would be equivalent to a single neuron with the proper weights. There are many activation functions (the interested reader is referred to classical material on ANNs (Goodfellow et al., 2016; Jain et al., 1996)). In this paper we consider the linear activation function for the output layer (because we use a regression ANN) and the leaky ReLU activation function, that we will explain later, for the hidden layers. Note that using a linear activation function is equivalent to not using any activation function at all.

Finally, ANNs use the *back-propagation algorithm* (Rumelhart et al., 1986) to learn the values of the weighs of each neuron. This learning method needs a set of examples given as input/output pairs (do not confuse with our inputs and outputs), which we call the *training set*. This algorithm performs, for each element of the training set, the following steps:

- Compute the result of the ANN for the given input.
- Compute the loss of the ANN result with respect to the expected result. The loss function usually is the mean squared error (squared L2 norm) between both values.
- Compute the gradient of the loss function with respect to each weight by the chain rule, trying to minimise the loss. This computation is done from the last layer to the previous ones, in an incremental way.

Using this method the ANN updates its weights, e ciently learning the hidden function that associates the inputs to the outputs of each element of the training set.

After the learning, a common practice is to test how well the ANN performs on previously unseen examples. In order to perform this check, it is necessary to have another set of examples which we call the test set . With this test set, the accuracyof the ANN is computed. For classi cation ANNs the accuracy is de ned as the ratio between the number of elements of the test set that have been well classi ed and the cardinal of the tests set. For regression ANNs there is no such concept of accuracy; the performance of the ANN is given by the mean loss of the examples of the test set. This concept of accuracy/mean loss can be extended to the performance of the ANN on the training set, although in this case it only measures how well the ANN learned the elements of the training set.

3. Methodology

In this section we review the main concepts behind the construction of our tool SqSelect.[2] In particular, we will explain how the ANN driving the behaviour of SqSelect was designed, trained and tested.

Squeeziness is useful to asses the likelihood of FEP in a system. Besides, PColl (see De nition 7) is a reference measure of FEP in a system. Therefore, for a given system, we would like to know the best value of to compute Rényi's Squeeziness, that is, the value of that better assess the likelihood of having cases of FEP. In other words, the value whose associated Rényi's Squeeziness has a higher correlation with PColl. In order to compute these correlations (one for each) we consider families of systems, that is, we grouped the FSMs according to their characteristics. Then, we compute the di erent Rényi's Squeeziness values (for di erent values of) for all the systems of a family and measure the correlation, for each, between those values of Rényi's Squeeziness and the values of PColl.

There are two standard options to compute correlations: Pearson correlation, which focuses on the proportionality between the variables, and Spearman correlation, which focuses on the monotony between the variables. We will focus on the rst option and during the rest of the paper when we simply say correlation we are referring to Pearson correlation (we will sometimes use Spearman correlation and we will clearly identify it). A discussion about this decision and its implications is given in Section 5.

If we are able to compute the best value of for a family of systems, then each time that our tool receives a system belonging to the family we only need to compute its Rényi's Squeeziness using this However, the number of di erent system families is in nite, even if we use a few parameters to de ne each family. Therefore, it is impossible to compute this best notion for each potential family. In order to circumvent this restriction, SqSelect implements an approximation method to compute a good enough value of for any system provided to the tool, even if its family has not been analysed before. SqSelect uses an ANN as approximation method. We decided to rely on this machine learning technique because our preliminary experiments showed that the best values of , for di erent families, have low correlation to the parameters that codify those families. Therefore, we need an approximation technique that could deal with this lack of correlation. In Figure 4 we give this distribution for the entries of the dataset that we used to train our ANN. Each point corresponds to a di erent family of systems and depth: the points to the right of each value in the x-axis represent di erent search depths. We can observe how, for the same parameter, the distribution of the points along the possible values of is almost uniform for each of its values, showing no correlation between and the considered parameter.

After training the ANN using a huge number of cases, we can use it to approximate the value of that will be better for a given system. In order to do so, we need this system to be modelled as an FSM, from where we can obtain their characteristic parameters. These parameters will be the inputs of the ANN underlying the behaviour of SqSelect.

Finally, knowing the value of whose Rényi's Squeeziness will approximate better the likelihood of having a case of FEP in a system, the only remaining step is to compute the actual Squeeziness of the system. In order to do so we implement the methodology de ned in our previous work (Ibias and Núñez, 2020). First of all, we assume that the probability distribution of the inputs follows a uniform distribution. With this choice we know that we are maximising entropy. In addition, we can use the simpli ed formulation given in Lemma 1. Second, we need to x a search depth k. This value will indicate the maximum length of the input sequences that we will be used for testing (and therefore, the expected output sequences). Once we have set the search depth, we must obtain two values in order to compute Rényi's Squeeziness:

- Number of inputs: the number of input sequences of length k of the FSM. This number is stored in a variable inputs.
- Inverse image of the outputs: the number of input sequences that lead to the same output sequence. These values are stored in a dictionary mapOtoI that keeps, for each output sequence of length k, the number of inputs that lead to it.

In order to compute these values, we have to collect the traces of the FSM until the desired depth is reached. If a trace reaches the desired depth (or a deadlock), then the inputs count is increased by 1 and the output obtained is searched in the mapOtoI dictionary to increase by 1 the count of inputs that lead to that output. Once we processed all these values, we only need to apply the formula given in Lemma 1.

3.1. The ANN underlying SqSelect

In order to infer the value that will return the better approximation of the likelihood of the presence of cases of

[2] Our tool is freely available at https://github.com/Colosu/SqSelect/ .

a) b)

c) d)

e)

Figure 4: Correspondence between parameters and of elements of the dataset: states (a), maximum transitions (b), minimum transitions (c), input alphabet size (d) and output alphabet size (e).

FEP in the system, we have developed an ANN using PyTorch (Paszke et al., 2019). We may consider that this ANN is the core of SqSelect. During the rest of this section we review the di erent phases involved in the creation of the ANN.

3.1.1. The dataset

First of all, we need a dataset to train the ANN. In order to obtain this dataset, we developed a program such that given a family of FSMs and a search depth, it computes the value of such that Rényi's Squeeziness values have a higher correlation with PColl values. We execute this program for di erent families and di erent depths. For each family and depth we added a row to the dataset with the family parameters, the depth and the best value of .

In order to get examples of each family, we developed a java script that generates, using the parameters of the family, a given number of FSMs of each of them. This script uses the automatalib (Isberner et al., 2015) library to represent and manipulate FSMs and saves the newly randomly generated FSMs in .dot les. With this script, we generated 100 examples of each family. The di erent families were de ned by combining the following parameter values:

- Number of states: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Figure 5: ReLU vs leaky ReLU.

- Maximum number of transitions: 5, 10, 15, 20.
- Minimum number of transitions: 1, 5, 10.
- Input alphabet size: 10, 20, 30, 40, 50.
- Output alphabet size: 10, 20, 30, 40, 50.

Let us remind that, as we said before, our experiments showed that there is no correlation between each single parameter and the computed best This is the reason why we combine them into an ANN.

3.1.2. The structure of the ANN

Our ANN implemented a regression ANN because we want to obtain an approximation of the best without restraining the results to some prede ned classes.

Since we are dealing with a problem of dimensionality versus complexity our ANN is quite simple while being very e ective. It has 4 layers:

- An input layer of size 6.
- A hidden layer of size 12, with a leaky ReLU activation function.
- A hidden layer of size 3, with a leaky ReLU activation function.
- An output layer of size 1, with a linear activation function.

Our input layer has six neurons because it receives the ve parameters that determine the family of FSM and a sixth parameter: the search depth. We consider this additional parameter because our rst experiments showed that the best strongly depends on the established search depth.

We use leaky ReLU activation functions because they improve the actual ReLU activation function by avoiding the so called dying ReLU problem and speeding up the learning (He et al., 2015). We can compare both activation functions in Figure 5.

The dying ReLU problem happens because the ReLU activation function returns zero for all negative inputs. This evolves into some neurons dying because they always return a zero value and, therefore, they do not help to discriminate the input. Also, it is unlikely that the neuron will ever output again non-zero values. The leaky ReLU activation function solves this problem by returning a small negative value, instead of zero, what allows the neuron to add something to the discrimination of the input. In addition, it will be able in the future to output non-zero values.

3.1.3. The learning process

Having the ANN and the dataset, we normalised the data in the dataset, we shu ed the entries of the dataset and divided it into two sets: a set containing 70~ of the entries, for training, and a set with 30~ of the entries, for testing. Following this procedure we ensure that in the test set there is a representative of each FSM class that conforms the dataset, and of each search deep. This will ensure that we can assess the generalisation capability of our ANN with the test set. We set the loss function to the mean square error and the optimisation algorithm to the standard gradient descend with learning rate 0:01. With this setting, we trained our ANN to approximate the values with the training set, obtaining a mean loss of 1:280935 (1:333986 for Spearman). This loss is small enough to claim that our ANN learned really well the training dataset.

We used the test set to assess how good the training has been. In this case, we obtained a mean loss 1:326940 (1:368981 for Spearman).

4. The SqSelect tool

In this section we present our tool to facilitate the assessment of the FEP of a system by computing relevant values associated with the Squeeziness of the system. SqSelect has four di erent modes, depending on what input parameters are available and what results are requested. These modes are:

- Alpha mode. The tool receives a set of parameters (see below) pro ling the system of interest, and the search depth, and returns the value providing a Squeeziness that is closer to the true likelihood of the presence of cases of FEP.
- Alpha le mode. The tool receives a search depth and a system and computes the corresponding value.
- Squeeziness le mode. SqSelect receives a search depth and a system and computes the best value and reports the likelihood of the presence of cases of FEP in the form of a Squeeziness value.
- Squeeziness range mode. The tool receives a search depth and a system and computes the Squeeziness of the system for di erent values of. Then, it gives back a graph with the likelihood of the presence of cases of FEP in the system associated to each value of.

The tool also provides the best value of for smaller search depths in alpha and alpha le modes. In Squeeziness le mode, SqSelect provides Rényi's Squeeziness for smaller search depths. These values might be useful if the tester, in view of additional information, decides to reduce the search depth associated with testing.

Note that in the rst two modes we only compute the value of . If the tester wants to obtain the value of Rényi's Squeeziness for that, then it is enough to use the third mode. The rationale is that computing Squeeziness, even for medium-size search depths, needs an important amount of computing time. If a tester either only needs the value of or desires to use an alternative measure (for this), then he does not need to consume those computing resources.

The set of parameters used in the Alpha mode is given by:

- Number of states of the FSM
- Maximum number of transitions outgoing from a state of the FSM
- Minimum number of transitions outgoing from a state of the FSM
- Input alphabet size of the FSM
- Output alphabet size of the FSM

In the second and third modes, SqSelect computes these values from the actual system.

The system received in the Alpha le mode, the Squeeziness le mode and the Squeeziness range mode should be a .dot le (the standard for representing graphs in plain text), where the rst node is marked with the red colour (Isberner et al., 2015).

In each mode (except in the Squeeziness range mode) there is an option called Use Spearman correlation that tells the tool to use the best value of according to the Spearman correlation instead of the default Pearson correlation.

Finally, all modes have an option to save the generated plot to a le, in case it is needed.

Next, we analyse the main characteristics of each mode.

4.1. Alpha mode

SqSelect receives a set of parameters corresponding to the system and the search depth. Then, SqSelect calls the ANN that computes the best for assessing the likelihood of the presence of cases of FEP in the system. The tool shows this value in its results panel. An example of execution is shown in Figure 6.

This mode is intended for users that do not have the description of their systems in the format used by our tool, so that they cannot use other modes. Therefore, with this mode they can still get a generic best for their system. Afterwards, they can compute Rényi's Squeeziness, for that , of their system using their own algorithm.

4.2. Alpha le mode

SqSelect receives an actual FSM and the search depth. Then, SqSelect computes the same parameters as the ones that are needed for the Alpha mode and works as in that mode. An example of execution is shown in Figure 7.

Figure 6: Alpha mode example: Co ee machine case study.

Figure 7: Alpha le mode example: TCP case study.

This mode is intended for users that, although having the system in a valid format, do not want our tool to compute Rényi's Squeeziness. This is the case when they have a faster, cheaper or in general better implementation of the formula from Lemma 1. SqSelect will obtain the best for their system and then the users will compute its Rényi's Squeeziness using either another tool or their own algorithm.

4.3. Squeeziness le mode

SqSelect receives a system and the search depth. Then, SqSelect computes the same parameters as the ones that are needed for the Alpha mode. After the best is obtained, the tool proceeds to compute Rényi's Squeeziness for the selected search depth and this . As an additional information, SqSelect computes values of for smaller depths and the corresponding Rényi's Squeeziness. All the values are shown as a graph, while the value corresponding to the selected search depth appears in the results panel. An example of execution is shown in Figure 8.

This is the main mode of SqSelect. We expect that most users will have their system in the valid format, they will give

Figure 8: Squeeziness le mode example: Logic circuit case study.

Figure 9: Squeeziness range mode example: Banckard case study.

it to the tool and they will get the value of Rényi's Squeeziness that better assess the likelihood of having cases of FEP for that system. If the users develop di erent versions of their system, then they can compare the Rényi's Squeeziness values obtained for each version and select the one having a lower Rényi's Squeeziness, because it is the one with smaller likelihood to su er cases of FEP. If the users only have one version of the system, and similar to the previous mode, they still can use the results displayed in the graph to decide the search depth that they want to use for testing.

4.4. Squeeziness range mode

SqSelect receives a system and the search depth. In contrast to the previous modes, SqSelect does not compute the best . Instead, it computes all the values of Rényi's Squeeziness for a selected number of values $\in [0;5]$. This interval is prede ned and was chosen because in our experiments we never obtained a best value of greater than 5 and the selected values are representative of the best values for the families of systems appearing in the dataset. The tool shows these values in its graph panel. An example of execution is shown in Figure 9.

This mode is intended for users that do not want to know only Rényi's Squeeziness for the best , but that want to know it for a huge range of values of

5. Discussion

In this section we discuss some decisions we took along the development of SqSelect.

The rst decision was which correlation to use as a reference for the selection of . Pearson correlation focuses on the linearity of the points that are being correlated. Therefore, higher (in absolute value) correlations imply that the points maintain a proportionality while correlations near 0 imply that the points are distributed more in a point cloud way. Meanwhile, Spearman correlation focuses on the monotony of the points that are being correlated. Therefore, higher (in absolute value) correlations imply that the points maintain monotony while correlations near 0 imply that many points break the monotony. The main goal of our research is to nd values of whose Rényi's Squeeziness has a higher similarity to the likelihood of the appearance of FEP. In order to have higher similarity, it is not enough to keep monotony; it is also necessary to have some kind of proportionality. Therefore, we think that it is better to stick to Pearson but if users prefer to use Spearman, then we allow them to choose it. In order to implement this duality, we obtained two datasets, one for the best values of according to Pearson correlation and another one according to Spearman correlation. We used the same ANN structure with both datasets, obtaining one ANN that learned the Pearson dataset and another one that learned the Spearman dataset. In any case, our experiments showed that the di erence between the Pearson and Spearman correlations is minimal.

The second decision was which machine learning technique to use. Currently, there are a myriad of options, from a classical decision tree algorithm to complex deep learning algorithms or even reinforcement learning algorithms. However, not all these techniques would perform well in our scenario. For example, reinforcement learning techniques require that the problem can be expressed as a Markov Decision Process (MDP), what in fact is a $1\frac{1}{2}$ game (Chatterjee et al., 2004). However, our problem cannot be rede ned (at least, in an easy and intuitive way) as an MDP. Decision trees and the linear regression techniques cannot be applied to our problem due to the small correlation between the elements of the dataset, that we explained before. Therefore, we were left with methods that were complex enough to manage the apparent non-correlation between the elements of the dataset. From this set of methods, we chose a simple one, because we wanted to avoid using approaches that were overpowered for the task at hand. That is why we decided to use arti cial neural networks. However, we would like to justify with empirical evidence that this was the right choice.

During the development of SqSelect we considered the

Regression Model	Pearson Training Loss	Pearson Test Loss	Spearman Training Loss	Spearman Test Loss
Linear Regression	1:523924	1:575762	1:547626	1:527375
Ridge Regression (alpha = 5)	2:153214	2:182028	2:110532	2:103499
Ridge Regression (alpha = 1)	1:734578	1:775401	1:737016	1:728423
Ridge Regression (alpha = 0.01)	1:524000	1:575669	1:547695	1:527758
Ridge Regression (alpha = 0.00001)	1:523924	1:575762	1:547626	1:527375
Lasso Regression (alpha = 0.01)	2:469327	2:489932	2:392343	2:383564
Lasso Regression (alpha = 0.001)	1:556424	1:609071	1:580521	1:564440
Lasso Regression (alpha = 0.00001)	1:523928	1:575770	1:547630	1:527369
Lasso Regression (alpha = 1e-10)	1:523924	1:575762	1:547626	1:527375
Shallow ANN (2 hidden layers)	1.280935	1.326940	1.333986	1.368981
Deep ANN (6 hidden layers)	1:236317	1:372902	2:396616	2:387157
Deep ANN (8 hidden layers)	1:284181	1:381884	2:400117	2:383044

Figure 10: Comparison of regression models (sorted by model complexity).

use of a more complex (deep) neural network. However, our preliminary experiments did not show any improvement over the shallow ANN that we are using. Actually, we usually obtained cases of over tting while experimenting with deep neural networks. Next we give the details of one of the (representative) settings that we used. We considered a deep neural network with six hidden layers and with the following neurons per layer: 6 neurons for the input layer, 100 for the rst hidden layer, 75, 50, 25, 12 and 3 for the subsequent hidden layers, and 1 neuron for the output layer (remind we are training a regression ANN). With this deep ANN we obtained the following results for the Pearson dataset: training loss equal to 1:236317 and test loss equal to 1:372902. The results for the Spearman dataset were considerably worse: 2:396616 and 2:387157, respectively. Another example is a neural network with eight hidden layers and with the following neurons per layer: 6 neurons for the input layer, 12 for the rst hidden layer, 48, 128, 75, 50, 25, 12 and 3 for the subsequent hidden layers, and 1 neuron for the output layer. With this deep ANN we obtained the following results for the Pearson dataset: training loss equal to 1:284181 and test loss equal to 1:381884. The results for the Spearman dataset were also considerably worse: 2:400117 and 2:383044, respectively. Therefore, the results are slightly worse than the ones corresponding to our shallow ANN (in both Pearson and Spearman datasets and in both training and test sets). In addition, this kind of networks needs extra computation resources and this extra complexity does not pay back with an increase on performance. After this comparison we considered that our shallow neural network was a better option.

A second experiment allowed us to compare the results provided by our ANN and a multi-variable linear regression model (Kutner et al., 2003). We trained an sklearn (Pedregosa et al., 2011) linear regression model, which automatically adapts to the multivariable case, with our datasets. Then, we computed model loss using the same method as we used for our ANN network, the mean square error, and obtained the following results for the Pearson dataset: training loss equal to 1:523924 and test loss equal to 1:575762. The results for the Spearman dataset were very similar: 1:547626 and 1:527375, respectively. Therefore, the results show that our ANN outperforms the LNR that we implemented. We also trained 4 di erent Ridge regression models and 4 different Lasso regression models. In all the cases we computed the model loss as the mean square error between the predictions and the real values. The results corresponding to all these experiments are displayed in Figure 10.

As a further exploration of these alternatives, we computed the con dence boundaries for each one, using Fixed-Width bands (Macskassy and Provost, 2004) (see Figures 11 and 12). We decided to compute these con dence boundaries for the rst 100 elements of the test dataset because, although including more elements would a ect the shape of the con dence bands, we want to address the di erences between di erent models instead of generate good bands for a given model (Macskassy and Provost, 2004). The red line presents the real value of, the blue line presents the value of obtained using the regression model; and the light blue zone presents the con dence interval in which we can assure, from the answer of our regression model, that the true value of will be with a 95~ of con dence.

These con dence boundaries show that our shallow ANN gets tter con dence boundaries and that the obtained values of t better the real values. Speci cally, there are some alternatives like Ridge Regression (alpha = 5) and Lasso Regression (alpha = 0.01) that are considerably worse, but the other alternatives are not better than ours.

6. Case studies

In this section we present some case studies where we use our tool in order to assess the likelihood of FEP in some FSM. In order to ensure that these FSM are representative, we used a recently collected benchmark (Neider et al., 2019) of FSM modelling real-world systems. We explore how to apply our tool to 4 FSM: the classical co ee machine, the TCP protocol, a logic circuit, and a bankcard system. We hope that this section will be a useful reference for anyone that wants to use SqSelect.

Linear Regression

Ridge Regression (alpha = 5)

Ridge Regression (alpha = 1)

Ridge Regression (alpha = 0.01)

Ridge Regression (alpha = 0.00001)

Lasso Regression (alpha = 0.01)

Figure 11: Confidence boundaries plots (part 1).

6.1. The coffee machine

This system represents the classical coffee machine that has been extensively used as an example of FSM. We use this system to show how to use the alpha mode. In this case, we need some assumptions: we only want to get the best and we do not have the FSM modelled in an appropriate format for the tool. Therefore, we have to use the system to obtain the different parameters that SqSelect uses for computing the best . We obtained the following parameters:

- Number of states: 6.
- Maximum number of transitions: 4.
- Minimum number of transitions: 4.
- Input alphabet size: 4.
- Output alphabet size: 3.

Finally, setting that we want to explore the FSM up to a depth of 5, we have to introduce these values in the tool. We obtained a value of equal to 0.10638013, as it is displayed in Figure 6.

6.2. The TCP protocol

This system represents the widely known TCP protocol, massively used in the communications through internet. Specifically, it represents the TCP protocol from the perspective of the server, what is an FSM with 55 states. We use this system to show how to use the Alpha file mode. In this case, we

Lasso Regression (alpha = 0.001) | Lasso Regression (alpha = 0.00001)

Lasso Regression (alpha = 1e * 10) | Shallow ANN (2 hidden layers)

Deep ANN (6 hidden layers) | Deep ANN (8 hidden layers)

Figure 12: Con dence boundaries plots (part 2).

assume that we only want to get the best and that we have the protocol modelled as an FSM in dot format. We also set that we want to explore the FSM up to a depth of 50.

Therefore, we load the le in SqSelect and set the search depth to 50. The tool computes a value of equal to 0.346357, as it can be seen in Figure 7.

6.3. A logic circuit

This system represents a dk27 logic circuit (Yang, 1991). We use this system to show how to use the Squeeziness le mode. We assume that we have the system in the appropriate format and that we set a search depth equal to 10. This system has 7 states and 14 transitions and SqSelect returns the results displayed in Figure 8.

From these results, we get that the best for a search depth of 10 was = 0.045744844 and it gives a Rényi's Squeeziness value equal to 2.4884. However, if we carefully analyse the plot, then we observe that using inputs of length 10 is not the optimal option concerning a trade-o between amount of testing and likelihood of su ering from FEP. As seen in the plot, inputs of length greater than 6 will su er more FEP than inputs of the said length. Inputs of lengths 4 and 5 will also su er more FEP than inputs of length 6. In contrast, inputs of length smaller than 4 will be less prone to have FEP than inputs of length 6. However, these last inputs will hardly explore more than half of the system (at least, in this case). Therefore, if resources are scarce and the tester might have to reconsider whether to explore the SUT up to a smaller depth, then a better option is to test using inputs of length 6.

6.4. The bankcard

This system represents the behaviour of an ATM when authenticating a debit or credit card. We use this system to show how to use the Squeeziness range mode. In order to do so, we assume that we want to know how Squeeziness evolves for different alphas, so that we can have a global vision of how likely is that our system is affected by cases of FEP without having to stick to only one. Since this system has only 7 states, we set the search depth to 5. We obtained the result displayed in Figure 9.

7. Conclusions

We have developed the tool SqSelect that can be used to assess the likelihood of the presence of FEP in a system. It relies in the concept of Squeeziness, which has been previously proven to be useful for this task. Moreover, SqSelect implements an extended version of Squeeziness based on Rényi's notion of entropy, instead of the classical Shannon's entropy notion. This general version relies on a parameter, named α, to perform the computation. In previous work we empirically showed that the election of the α parameter is not an easy task. This is due to the fact that the reliability of the assessment of the likelihood of the presence of FEP strongly varies among different values of α. Therefore, we have to develop a method to obtain the best value of α to assess the likelihood of the presence of FEP in the system. In SqSelect we used this method to get data for training an artificial neural network that infers the best α for a given system. Then, the tool uses this α to compute Rényi's Squeeziness, so that the user can have an idea of how prone the system is to have cases of FEP. Moreover, SqSelect implements another three modes, so it can be used by a wider kind of users.

As future work, there are some open research lines that can improve the efficiency and usefulness of SqSelect. One line of future work is the improvement of the computation of Rényi's Squeeziness, so that SqSelect can be more efficient and quick. Second, we would like to integrate SqSelect with other tools that automatise testing, so that it can be used as a previous step to assess how much testing should be performed. In particular, we would like to incorporate the efficient and systematic generation and processing of mutants (Cañizares et al., 2018; Delgado-Pérez et al., 2019; Gómez-Abajo et al., 2018, 2021; Gutiérrez-Madroñal et al., 2019) so that we can offer mutation testing features. Finally, we would like to extend our tool to deal with other FSM-based formalism. We are particularly interested in distributed systems where communications can be asynchronous (Hierons et al., 2017, 2018; Merayo et al., 2018a,b).

Appendix I: Alternative definition of Rényi's Squeeziness

Lemma 1. Let $M = (Q, q_{in}, I, O, T)$ be an FSM, $k > 0$ and $\alpha \in \mathbb{R}_+ \setminus \{1\}$. Let us consider a random variable $\zeta_{dom_{M,k}}$ ranging over the domain of $f_{M,k}$. We have that

$$Sq_{\alpha,k}(M) = \frac{1}{1-\alpha}\log_2\left(\frac{\sum_{s\in dom_{M,k}} P_{dom_{M,k}}(s)^{\alpha}}{\sum_{t\in image_{M,k}}\left(\sum_{s\in f_M^{-1}(t)} P_{dom_{M,k}}(s)\right)^{\alpha}}\right)$$

If α tends to 1 then we obtain Shannon's entropy (Rényi, 1961) and we have

$$Sq_{1,k}(M) = -\sum_{t\in image_{M,k}}\left(\sum_{s\in f_M^{-1}(t)} P_{dom_{M,k}}(s)\right) R_M(t)$$

where the term $R_M(t)$ is equal to

$$\sum_{s\in f_M^{-1}(t)} \frac{P_{dom_{M,k}}(s)}{P_{dom_{M,k}}(f_M^{-1}(t))}\log_2\left(\frac{P_{dom_{M,k}}(s)}{P_{dom_{M,k}}(f_M^{-1}(t))}\right)$$

If α tends to ∞ then we obtain min-entropy (Rényi, 1961) (that is, $H_\infty(X) = -\log_2(\max_i p_i)$) and we have

$$Sq_{\infty,k}(M) = \log_2\left(\frac{\max_{t\in image_{M,k}} \sum_{s\in f_M^{-1}(t)} P_{dom_{M,k}}(s)}{\max_{s\in dom_{M,k}} P_{dom_{M,k}}(s)}\right)$$

Proof

$$Sq_{\alpha,k}(M) = H_\alpha(\zeta_{dom_{M,k}}) - H_\alpha(\zeta_{image_{M,k}}) =$$

$$= \frac{1}{1-\alpha}\log_2\left(\sum_{s\in dom_{M,k}} P_{dom_{M,k}}(s)^{\alpha}\right) - \frac{1}{1-\alpha}\log_2\left(\sum_{t\in image_{M,k}} P_{image_{M,k}}(t)^{\alpha}\right) =$$

$$= \frac{1}{1-\alpha}\log_2\left(\frac{\sum_{s\in dom_{M,k}} P_{dom_{M,k}}(s)^{\alpha}}{\sum_{t\in image_{M,k}} P_{image_{M,k}}(t)^{\alpha}}\right) =$$

$$= \frac{1}{1-\alpha}\log_2\left(\frac{\sum_{s\in dom_{M,k}} P_{dom_{M,k}}(s)^{\alpha}}{\sum_{t\in image_{M,k}}\left(\sum_{s\in f_M^{-1}(t)} P_{dom_{M,k}}(s)\right)^{\alpha}}\right)$$

When $\alpha = 1$, the result has been proven in previous work (Ibias et al., 2019). Finally, if $\alpha = \infty$, then the proof is the following:

$$Sq_{\infty;k}(M) = H_{\infty}(dom_{M;k}) - H_{\infty}(image_{M;k})$$

$$= -\log_2\left(\max_{s\in dom_{M;k}} P_{dom_{M;k}}(s)\right) + \log_2\left(\max_{t\in image_{M;k}} \sum_{s\in f_M^{-1}(t)} P_{dom_{M;k}}(s)\right)$$

$$= \log_2\left(\frac{\max_{t\in image_{M;k}} \sum_{s\in f_M^{-1}(t)} P_{dom_{M;k}}(s)}{\max_{s\in dom_{M;k}} P_{dom_{M;k}}(s)}\right)$$

Appendix II: Deployment of the tool

In this appendix we briefly analyse some issues related to the deployment of SqSelect. Specifically, we will discuss the inclusion of our ANN, developed using PyTorch (therefore, written in python), in SqSelect, a tool that has been developed using java. Although we found some libraries that claim to (easily) perform this integration, we were unable to make them work. We contemplated several alternatives: deploy the ANN in a controlled server, execute the python code from java, integrate the python code in java and use another library for the ANN. The second and third options, in addition to being more complex than the first one, have the problem of dealing with the communication between the java virtual machine and the python libraries, what is a really complex task. The last option has the problem of having to refactor the entire ANN, without being sure that the new library will be able to being properly integrated into java. Finally, the first option, despite of being not so complex, has the security risks associated to having a server running in the background. Luckily, those risks could be easily overcome using some tricks: we execute the server only in localhost, avoiding the possibility of accessing it from outside the computer; we use an uncommon port in order to both avoid collision with other server utilities and make it difficult to find the correct port; and we use a specific kind of message so that the server does not read any other type of message. Therefore, we decided to perform a small workaround: we set up a Flask server in python that receives the ANN input parameters by using a JSON form and returns the ANN output (the best value) in a JSON response. SqSelect has to set up this server in localhost, avoiding any call from outside the computer.

This option arises a potential security concern: whether deploying a server each time that SqSelect is initialised is secure. In other words, we have to evaluate whether this action will be a backdoor to execute python code in the host computer. We are aware of this concern but discard it because the server is deployed in localhost, without access to internet. So, it will not be an open port for potential external attackers. Also, it does not use a common port, so it will be difficult to find the open port from inside the own computer. Finally, the Flask application only accepts one type of messages, rejecting any other input.

References

Abdelmoez, W., Nassar, D.E.M., Shereshevsky, M., Gradetsky, N., Gunnalan, R., Ammar, H.H., Yu, B., Mili, A., 2004. Error propagation in software architectures, in: 10th IEEE Int. Software Metrics Symposium, METRICS'04, IEEE Computer Society. pp. 384–393.

Acharya, J., Orlitsky, A., Suresh, A.T., Tyagi, H., 2017. Estimating Renyi entropy of discrete distributions. IEEE Transactions on Information Theory 63, 38–56.

Ammann, P., Offutt, J., 2017. Introduction to Software Testing. 2nd ed., Cambridge University Press.

Androutsopoulos, K., Clark, D., Dan, H., Hierons, R., Harman, M., 2014. An analysis of the relationship between conditional entropy and failed error propagation in software testing, in: 36th Int. Conf. on Software Engineering, ICSE'14, ACM Press. pp. 573–583.

Ayala, D., Borrego, A., Hernández, I., Ruiz, D., 2020. A neural network for semantic labelling of structured information. Expert Systems with Applications 143, 113053.

Boreale, M., Paolini, M., 2014. On formally bounding information leakage by statistical estimation, in: 17th Int. Conf. on Information Security, ISC'14, LNCS 8783, Springer. pp. 216–236.

Cañizares, P.C., Núñez, A., Lara, J.d., 2019. An expert system for checking the correctness of memory systems using simulation and metamorphic testing. Expert Systems with Applications 132, 44–62.

Cañizares, P.C., Núñez, A., Merayo, M.G., 2018. Mutomvo: Mutation testing framework for simulated cloud and HPC environments. Journal of Systems and Software 143, 187–207.

Cavalli, A.R., Higashino, T., Núñez, M., 2015. A survey on formal active and passive testing with applications to the cloud. Annales of Telecommunications 70, 85–93.

Chan, A., Winter, S., Saissi, H., Pattabiraman, K., Suri, N., 2017. IPA: error propagation analysis of multi-threaded programs using likely invariants, in: 10th Int. Conf. on Software Testing, Verification and Validation, ICST'17, IEEE Computer Society. pp. 184–195.

Chatterjee, K., de Alfaro, L., Henzinger, T.A., 2004. Trading memory for randomness, in: 1st Int. Conf. on Quantitative Evaluation of Systems, QEST'04, pp. 206–217.

Chothia, T., Kawamoto, Y., Novakovic, C., 2014. Leakwatch: Estimating information leakage from java programs, in: 19th European Symposium on Research in Computer Security, ESORICS'14, LNCS 8713, Springer. pp. 219–236.

Clark, D., Feldt, R., Poulding, S.M., Yoo, S., 2015. Information transformation: An underpinning theory for software engineering, in: 37th IEEE/ACM International Conference on Software Engineering, ICSE'15, pp. 599–602.

Clark, D., Hierons, R.M., 2012. Squeeziness: An information theoretic measure for avoiding fault masking. Information Processing Letters 112, 335–340.

Clark, D., Hierons, R.M., Patel, K., 2019. Normalised Squeeziness and Failed Error Propagation. Information Processing Letters 149, 6–9.

Coppik, N., Schwahn, O., Winter, S., Suri, N., 2017. TrEKer: tracing error propagation in operating system kernels, in: 32nd IEEE/ACM Int. Conf. on Automated Software Engineering, ASE'17, IEEE Computer Society. pp. 377–387.

Cover, T.M., Thomas, J.A., 1991. Elements of Information Theory. Wiley Interscience.

de Mesquita Sá Junior, J.J., Correia Ribas, L., Martinez Bruno, O., 2019. Randomized neural network based signature for dynamic texture classification. Expert Systems with Applications 135, 194–200.

Delgado-Pérez, P., Rose, L.M., Medina-Bulo, I., 2019. Coverage-based quality metric of mutation operators for test suite improvement. Software

Quality Journal 27, 823–859.
Díaz, G., Macià, H., Valero, V., Boubeta-Puig, J., Cuartero, F., 2020. An intelligent transportation system to control air pollution and road traffic in cities integrating CEP and colored petri nets. Neural Computing and Applications 32, 405–426.
Feldt, R., Poulding, S.M., Clark, D., Yoo, S., 2016. Test set diameter: Quantifying the diversity of sets of test cases, in: 9th IEEE Int. Conf. on Software Testing, Verification and Validation, ICST'16, IEEE Computer Society. pp. 223–233.
Feldt, R., Torkar, R., Gorschek, T., Afzal, W., 2008. Searching for cognitively diverse tests: Towards universal test diversity metrics, in: 1st IEEE Int. Conf. on Software Testing Verification and Validation Workshops, IEEE Computer Society. pp. 178–186.
García de Prado, A., Ortiz, G., Boubeta-Puig, J., 2017. COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things. Expert Systems with Applications 85, 231–248.
Gaudel, M.C., 1995. Testing can be formal, too!, in: 6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915, Springer. pp. 82–96.
Gómez-Abajo, P., Guerra, E., de Lara, J., Merayo, M.G., 2018. A tool for domain-independent model mutation. Science of Computer Programming 163, 85–92.
Gómez-Abajo, P., Guerra, E., de Lara, J., Merayo, M.G., 2021. Wodel-Test: a model-based framework for language-independent mutation testing. Software and Systems Modeling (in press) .
Goodfellow, I.J., Bengio, Y., Courville, A.C., 2016. Deep Learning. Adaptive computation and machine learning, MIT Press.
Gu, W., Foster, K., Shang, J., Wei, L., 2019. A game-predicting expert system using big data and machine learning. Expert Systems with Applications 130, 293–305.
Gutiérrez-Madroñal, L., García-Domínguez, A., Medina-Bulo, I., 2019. Evolutionary mutation testing for IoT with recorded and generated events. Software - Practice & Experience 49, 640–672.
He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: 15th IEEE Int. Conf. on Computer Vision, ICCV'15, IEEE Computer Society. pp. 1026–1034.
Hierons, R.M., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H., 2009. Using formal specifications to support testing. ACM Computing Surveys 41, 9:1–9:76.
Hierons, R.M., Merayo, M.G., Núñez, M., 2017. An extended framework for passive asynchronous testing. Journal of Logical and Algebraic Methods in Programming 86, 408–424.
Hierons, R.M., Merayo, M.G., Núñez, M., 2018. Bounded reordering in the distributed test architecture. IEEE Transactions on Reliability 67, 522–537.
Hiller, M., Jhumka, A., Suri, N., 2002. PROPANE: an environment for examining the propagation of errors in software, in: 13th Int. Symposium on Software Testing and Analysis, ISSTA'02, ACM Press. pp. 81–85.
Hiller, M., Jhumka, A., Suri, N., 2004. EPIC: Profiling the propagation and effect of data errors in software. IEEE Transactions on Computers 53, 512–530.
Ibias, A., Hierons, R.M., Núñez, M., 2019. Using Squeeziness to test component-based systems defined as Finite State Machines. Information & Software Technology 112, 132–147.
Ibias, A., Núñez, M., 2020. Estimating fault masking using Squeeziness based on Rényi's entropy, in: 35th ACM Symposium on Applied Computing, SAC'20, ACM Press. pp. 1936–1943.
Ibias, A., Núñez, M., Hierons, R.M., 2021. Using mutual information to test from Finite State Machines: Test suite selection. Information & Software Technology 132, 106498.
Isberner, M., Howar, F., Steffen, B., 2015. The open-source learnlib, in: 27th Int. Conf. on Computer Aided Verification, CAV'15, LNCS 9206, Springer. pp. 487–495.
Islam, M.S., Nepal, M.P., Skitmore, R.M., Kabir, G., 2019. A knowledge-based expert system to assess power plant project cost overrun risks. Expert Systems with Applications 136, 12–32.
Jain, A.K., Mao, J., Mohiuddin, K.M., 1996. Artificial neural networks: A tutorial. IEEE Computer 29, 31–44.
Johansson, A., Suri, N., 2005. Error propagation profiling of operating systems, in: 35th Int. Conf. on Dependable Systems and Networks, DSN'05, IEEE Computer Society. pp. 86–95.
Kutner, M.H., Nachtsheim, C.J., Wasserman, W., Neter, J. (Eds.), 2003. Applied Linear Regression Models. 4th (revised) ed., McGraw-Hill.
Laski, J.W., Szermer, W., Luczycki, P., 1995. Error masking in computer programs. Software Testing, Verification and Reliability 5, 81–105.
Macskassy, S.A., Provost, F.J., 2004. Confidence bands for ROC curves: Methods and an empirical study, in: 1st Int. Workshop on ROC analysis in Artificial Intelligence, ROCAI'04, pp. 61–70.
Marinescu, R., Seceleanu, C., Guen, H.L., Pettersson, P., 2015. A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs. Elsevier. volume 98 of *Advances in Computers*. chapter 3. pp. 89–140.
Masri, W., Abou-Assi, R., El-Ghali, M., Al-Fatairi, N., 2009. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization, in: 2nd Int. Workshop on Defects in Large Software Systems, DEFECTS'09, ACM Press. pp. 1–5.
Merayo, M.G., Hierons, R.M., Núñez, M., 2018a. Passive testing with asynchronous communications and timestamps. Distributed Computing 31, 327–342.
Merayo, M.G., Hierons, R.M., Núñez, M., 2018b. A tool supported methodology to passively test asynchronous systems with multiple users. Information & Software Technology 104, 162–178.
Miranskyy, A.V., Davison, M., Reesor, R.M., Murtaza, S.S., 2012. Using entropy measures for comparison of software traces. Information Sciences 203, 59–72.
Myers, G.J., Sandler, C., Badgett, T., 2011. The Art of Software Testing. 3rd ed., John Wiley & Sons.
Neider, D., Smetsers, R., Vaandrager, F.W., Kuppens, H., 2019. Benchmarks for automata learning and conformance testing, in: Margaria, T., Graf, S., Larsen, K.G. (Eds.), Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday. Springer, pp. 390–416.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An imperative style, high-performance deep learning library, in: 32nd Annual Conf. on Neural Information Processing Systems, NeurIPS'19, Curran Associates, Inc.. pp. 8024–8035.
Pattipati, K.R., Alexandridis, M.G., 1990. Application of heuristic search and information theory to sequential fault diagnosis. IEEE Transactions on Systems, Man, and Cybernetics 20, 872–887.
Pattipati, K.R., Deb, S., Dontamsetty, M., Maitra, A., 1990. START: System testability analysis and research tool, in: IEEE Conference on Systems Readiness Technology, 'Advancing Mission Accomplishment', pp. 395–402.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.
Piper, T., Winter, S., Schwahn, O., Bidarahalli, S., Suri, N., 2015. Mitigating timing error propagation in mixed-criticality automotive systems, in: 18th Int. Symposium on Real-Time Distributed Computing, ISORC'15, IEEE Computer Society. pp. 102–109.
Rényi, A., 1961. On measures of entropy and information, in: 4th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, University of California Press. pp. 547–561.
Roldán, J., Boubeta-Puig, J., Martínez, J.L., Ortiz, G., 2020. Integrating complex event processing and machine learning: An intelligent architecture for detecting iot security attacks. Expert Systems with Applications