Failure Triage in RTL Regression Verification

Zissis Poulos, *Student Member, IEEE*, and Andreas Veneris, *Senior Member, IEEE*

***Abstract*—We propose an automated failure triage framework for Register Transfer Level (RTL) debugging in functional verification regression flows which unifies three critical aspects of the problem: the approximation of the general location of root-cause(s) in the design under verification (DUV), the binning of all related failures generated by regression runs, and the distribution of these binned failures to the proper engineer(s) for detailed analysis. The proposed triage engine entails two novel methodologies. The first is a classification framework that mines information from SAT-based debugging and simulation to probabilistically reason about the relation of root-causes with their respective failing verification traces. This enables the construction of a priority ranking for these root-causes, and can effectively guide debugging by focusing resources on high-priority root-causes. Second, we propose a formulation of failure binning as exemplar-based clustering for grouping and distributing failing traces to the proper engineering team(s). Experiments on industrial designs show that the proposed methodology achieves 84% and 81% accuracy when it comes to failure grouping and distribution respectively, with only a 6.5% runtime overhead over existing debugging state-of-the-art techniques.**

***Index Terms*—Failure Triage, Regression Verification, RTL, Debug, Satisfiability**

I. Introduction

THE semiconductor industry continues to experience an increase in the size and complexity of VLSI circuits. Recent studies proclaim that almost a third of modern designs surpass the 100 million gate mark, and that over half of modern circuits involve one or more embedded processors [1]. These trends have widened the verification gap, a term referring to the disparity between our verification and design capabilities. Indicatively, functional verification today consumes over 50% of the design effort [1].

One of the main reasons behind the strenuous verification cost is the task of design debugging. Design debugging commences after verification exposes an error trace (i.e., a sequence of stimuli leading to a functional failure) and is the task that locates the responsible design error(s) using information provided by that error trace. Accounting for over 60% of the verification effort and as much as 39% of the time for verification engineers [1], it has urged industry and academia into developing automated solutions to reduce the associated cost. Specifically, CAD tools that employ formal engines, such as BDDs, SAT and QBF [2]–[5], have proven to be the most effective in automating the localization of root-causes. Driven by traditional industrial practices these tools are mainly designed to perform debugging on a case-by-case basis, where localization is largely focused on a single functional failure at a time.

However, the rise in design complexity has brought a shift in verification strategies that today are complemented by extensive regression runs with strict coverage goals. The co-existence of multiple design errors at early design stages often results into hundreds of failures exposed by regression, thus hampering the application of conventional formal debugging.

Specifically, in regression settings, performing an early coarse-grain analysis to understand the nature of these failures and eventually distribute them to the proper engineers for detailed analysis constitutes an essential pre-processing debugging step, referred to as *failure triage*. Failure triage is usually performed in three steps: coarse-grain debugging, failure binning and failure bin distribution. Coarse-grain debugging constitutes an early-stage analysis with a dual purpose: (a) approximating the general location of the root-cause for each of the exposed failures (i.e., testbench or specific design modules), and (b) generating a priority sequence for potential root-causes, so that high-priority ones are more likely to correspond to the culprit of the failure. In turn, the goal of failure binning is to leverage information from coarse-grain debugging to discover correlations between the exposed failures and bin together these failures that are likely to share the same root-cause. Finally, failure bin distribution identifies these failures that should be prioritized within each bin, and it assigns each bin to the engineer(s) most familiar with these high-priority failures.

Current studies assert triage as a fast growing regression pain occupying up to 30% of the debugging effort [6]. Despite these projections, triage remains a predominantly manual and *ad hoc* process in the industry [7]. Typically, it is handled by primitive means of high-level debugging, such as grouping similar error logs together or simple waveform inspection. These utilities mainly rely on symptom-related information (e.g., which exception checker captured the mismatch), and often fail to detect failures sharing the same root-cause. These inaccuracies reflect into additional debugging overhead in the detailed analysis stage that follows.

To reduce triage costs and mitigate the risk for time-to-market, we introduce a novel failure triage engine, which combines formal methods with data-mining techniques to automate the process. The contributions of this work pertain to each of the three triage stages previously discussed.

When it comes to coarse-grain debugging, approximating the root-cause location can be efficiently performed by existing formal solutions. Specifically, we assign this process to SAT-based debuggers [3]. These tools identify, for each failure, all possible design locations (suspects) where a modification can rectify the corresponding error trace. However, no existing method is available to generate a priority sequence for these suspect components. As such, we propose a classification-based method to address prioritization. Our goal is to detect suspect components that are likely to correspond to the actual error source of a failure, and segregate those from suspects that are unrelated. We do so by performing feature engineering to select appropriate attributes that can expose a separability between these two entities. Specifically, we introduce two metrics extracted from error trace information, namely the temporal distance and excitation resistance of suspect locations, and we empirically show that these can be used as suspect features to produce accurate suspect priority sequences. The classification task is performed by Support Vector Machines (SVMs).

Further, in this work, the tasks of failure binning and bin distribution are unified under a single data-mining formulation. Particularly, we automate both these stages by encoding them as exemplar-based clustering, which partitions the set of exposed failures into bins and also determines which failures are the most representative of erroneous behavior within each bin. By using the ranked suspects produced by coarse-grain debugging as failure attributes, we introduce a novel high-dimensional representation for each failure. This allows us to map all failures in a joint space, where their similarity can be naturally quantified though known distance metrics. We then employ a belief propagation algorithm, known as Affinity Propagation [8], to bin similar failures together and utilize failures-exemplars to determine the best-suited engineer for detailed debug. Experiments on industrial designs show that our methodology achieves 84% accuracy when it comes to failure grouping, and 81% accuracy for bin distribution, with only a 6.5% overhead in runtime.

The remainder of this paper is organized as follows. Section II offers background regarding the three main triage stages, and discusses prior work in the field. Section III outlines the proposed coarse-grain debugging method, and discusses the metrics that are proposed for classification. The formulation and exemplar-based clustering process for failure binning and failure bin distribution are presented in Section IV. Finally, Section V presents our experimental evaluation and Section VI concludes the work.

II. Triage in RTL Debugging

Failure triage commences immediately after regression is complete and it precedes detailed debugging. Regression in this context refers to the application of simulation test suites, sanity checks and/or formal model checking at the end of some predetermined development stage, to ensure the design performs as expected. We assume that the results of this process are persisted into proper simulation and failure log files. With this assumption, triage essentially constitutes a filtering process that translates an unstructured debugging problem (i.e., randomly pick a failure caught during regression and debug it) into a structured set of multiple debugging instances, where each debugging instance focuses on a group of related failures and has the proper engineering resources allocated to it. A high-level view of a triaged debugging flow is illustrated in Figure 1, where the results of regression verification are translated into K debugging instances (failure groups). This Section elaborates on the steps, notation, requirements and prior work in debugging triage.

A. Regression Data

To aid the presentation we first discuss what type of input the triage engine receives after regression is complete.

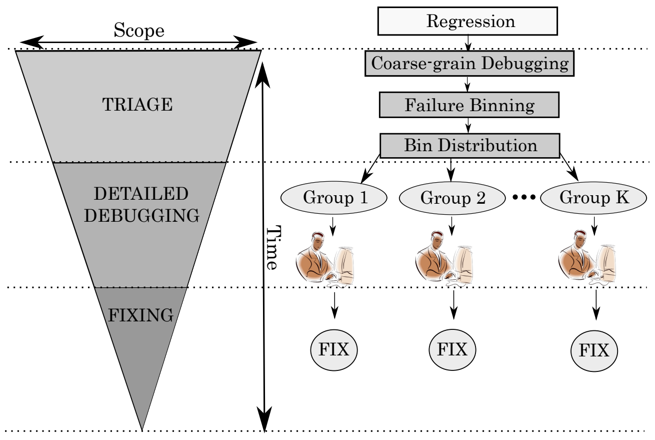


Fig. 1. Triage in design debugging flows

Consider a design undergoing regression verification, with possibly multiple errors at the RTL. When a mismatch between the expected “golden” value(s) (0,1 or X) and the one(s) observed is identified at some observation point (end-to-end checker, exception checker or assertion), we say that a failure occurs. Suppose that N design failures are exposed by the end of regression, and let these failures be denoted as $\mathbf{F} = \{F_1, F_2, \ldots, F_N\}$. In this work we consider scenarios where verification, whether formal or simulation-based, returns error traces, i.e., sequences of stimuli exposing these failures. Thus, it is hereby assumed that, for each failure F_i, its corresponding error trace, denoted E_i, is also available. It is also assumed that several error-trace data are accessible by the triage engine, such as internal signal values and signal/block coverage information.

Finally, for a given regression run, one can assume that several tests in the suite may pass without exposing any failure. Suppose the regression run ends with M passing tests. Then we denote these as $P_1, P_2, \ldots, P_M$, and again assume that simulation data related to these passing tests are accessible by the triage engine.

B. Coarse-grain Debugging

Coarse-grain debugging is the first triage sub-phase, also shown in Figure 1. Here, the goal is to overapproximate the error location for each failure in $\mathbf{F}$. This task entails two parts: (a) identifying a set of design locations that are potential root-causes for each failure, and (b) generating a priority sequence (ranking) of these locations. The latter involves identifying these locations most likely to be the actual error source for any given failure, a mechanism that can dramatically reduce the engineering effort associated with detailed debugging, and which also serves as a means to generate representative failure signatures to aid the other two triage sub-phases that follow.

Recent work uses SAT-based debugging to address the first part of coarse-grain debugging, and several heuristics based on simulation data to produce the necessary rankings [9]. The application of such formal tools to exhaustively identify all potential root-causes is a well-established method in the industry, and thus is also adopted by the work presented here.

In what follows, we revisit basic concepts and notation behind SAT-based RTL debugging, the details of which are given in [3], and we outline how rankings are generated in past work.

1) SAT-based Debugging: When executing a baseline SAT-based debugging pass for a given failure $F_i \in \mathbf{F}$, the automated debugger returns a set of design components (RTL signals, blocks or modules), denoted as $S_i = \{s_1, s_2, \dots, s_{|S_i|}\}$. These design elements, $s_1, s_2, \dots, s_{|S_i|}$, are referred to as *suspects*, and constitute all possible design locations that can be responsible for failure F_i. Broadly speaking, automated debuggers can perform the task at the gate-level, but in this work we focus on RTL debugging, which offers a lower resolution analysis. This is desirable since state-of-the-art debuggers operate at the RTL. Additionally, a low resolution analysis is more suitable for triage purposes, since decisions are made based on blocks higher in the design hierarchy. Along these lines, a suspect can be an `always` block, an `if` statement, a module definition or instantiation etc.

In practice, the SAT-based debugger can be run with an additional parameter, referred to as the cardinality constraint, which is a positive integer. For a cardinality constraint equal to $c > 1$ the debugger is forced to discover locations where exactly c bugs need to interact to produce the observed mismatch. However, when operating at higher RTL hierarchy, a single suspect component will cover the case of higher cardinality suspects at any level below.

In this paper we work with suspects of single cardinality. Specifically, we leave the cardinality constraint free, and for any suspect of cardinality $c > 1$, we traverse the hierarchy upwards until we encounter the single cardinality suspect that contains all c suspect locations below. In extreme cases this includes the top-level design and/or the testbench as suspects (the latter when one or more suspects are found to be in the stimuli). It should be noted that the debugger rarely returns suspects of cardinality greater than 2, unless it operates at the gate level.

With this assumption and due to its exhaustive nature SAT-based debugging guarantees that the design location responsible for some failure F_i will be included in suspect set S_i. In this context, suspect set S_i can be viewed as a "signature" that characterizes failure F_i. Finally, SAT-based debuggers can point to the exact cycle where an erroneous value is excited within some suspect component, and can also return the erroneous value's propagation path, which constitutes the circuit elements that it traverses to reach the point of mismatch.

Example 1. *To demonstrate the concepts introduced above consider an error trace E_1 leading to the exposure of failure F_1, as depicted in Figure 2. The sequential behavior of the circuit for that trace is shown using its Iterative Logic Array (ILA) representation [3]. Suppose an error at component s_2 is excited in cycle $n-2$ and propagates to cause a failure (F_1) at an observation point in cycle n. The error trace is then passed to an automated debugger. The result is a suspect set $\mathcal{S}_1 = \{s_1, s_2, s_3\}$ of design components that can explain the mismatch. Suspects s_1, s_2 and s_3, where erroneous values are excited in cycles k, $n-2$ and $n-1$ respectively, along with propagation paths of these erroneous values are also shown in the figure. Note that the erroneous component is included in S_1 as suspect s_2.*

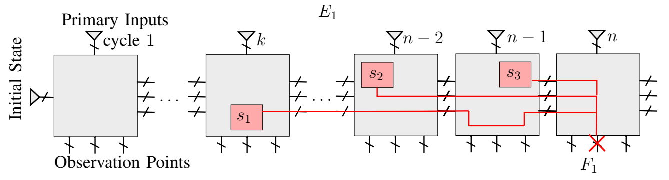


Fig. 2. Error trace and suspect components

2) Suspect Ranking: In practice, a SAT-based debugger tends to return many design locations apart from the one responsible for the failure, since there are often multiple unrelated design blocks where a change can be made to mask the failure. A *spurious suspect* refers to such a result *i.e.,* a suspect location returned by the debugger that is not the actual or an equivalent error source. We note, the term "spurious" in this work is different from the one used in traditional abstraction and refinement debugging [10], where it refers to non-viable solutions due to over-approximation of the solution space during abstraction.

For various reasons, it is sensible to determine which design locations are spurious suspects. For instance, during detailed debug the engineer must spend time investigating each of the suspects. Therefore, knowing which suspects are spurious can accelerate the process, as it reduces the number of locations that need to be examined. Additionally, identifying spurious suspects early can serve as a filtering process that reduces "noise" in the debugging data that is collected.

Of course, no known method exists that can identify spurious suspects with certainty. Thus, a mechanism is required that properly ranks all suspect components based on their likelihood of being an actual design error. This ranking process constitutes the second sub-phase of coarse-grain debugging.

Ideally, one needs to identify and promote suspects that exhibit behavior similar to that of typical human-introduced design errors. Recent work has experimentally shown that metrics extracted from simulation data for each suspect component can aid this process. Specifically, the authors in [9] use signal toggle coverage measured within each error trace to identify behavior that is representative of human-introduced errors, but atypical of spurious suspects. However, one can point to three main drawbacks in the above method. First, signal toggle coverage may be of limited use when handling suspects that correspond to larger design blocks. Second, coverage information is extracted only from error traces, and the potential of leveraging the coverage of passing tests has largely been left unexplored. Finally, the simulation metrics that are used have no statistical support, in that there exists no analysis to empirically confirm that they expose a separability between spurious suspects and design errors. In this work, we address all these points, as it will be shown in the next Section.

Along similar lines, a recent thread of work on bug localization and triage has successfully applied data-mining techniques combined with coverage event analysis [11] and high-level instruction set simulation [12]. These works mainly focus on probabilistically inferring the location of a bug, without explicitly addressing the problems of failure binning and distribution in regression settings. Data-mining tools have also been developed in related verification fields, and prominently for automatic assertion generation [13].

C. Failure Binning

Failure binning is inherently an unsupervised learning task (clustering), based on some descriptive model that associates failures in $\mathbf{F}$ and quantifies how similar any two failures in the set are. Its goal is to produce a complete partition of the failure set $\mathbf{F}$ into K disjoint clusters (bins). Ideally, failures that originate from the same RTL error are placed into the same cluster, and into distinct clusters otherwise. The process requires a pairwise failure similarity measure to be quantified based on the above desired relationship.

There can be several ways to quantify similarity between failures F_i and F_j. In [9], similarity is computed as a weighted version of the Jaccard Coefficient. The metric quantifies the overlap between suspect sets of F_i and F_j, while taking into account the ranking of the suspects within these sets. Binning is then performed via hierarchical clustering algorithms. On the post-silicon verification front, where triage has also been attracting interest, the authors in [14] compute similarity by extracting features from the failure reports using a topology-aware approach based on graph partitioning. For the binning task they employ k-means clustering.

Another significant parameter in failure binning is the number of clusters to be formed, which generally has to be "guessed" *a priori*. This is an inherently difficult task in failure triage, since it assumes prior knowledge from the engineer's side regarding the number of design errors responsible for the set of failures $\mathbf{F}$. Since this type of knowledge is limited and often absent in typical triage scenarios, the authors in [9] employ heuristics based on cluster density, a mechanism that appears to be the bottleneck of their method in terms of binning accuracy. On the other hand, the work in [14] for post-silicon triage, handles this issue by performing k-means clustering multiple times and outputting the partition with minimal overall within-cluster similarity. This incurs a severe cost in the overall running time, while accuracy becomes highly sensitive to the initial seeding of the algorithm.

D. Failure Bin Distribution

The final phase of triage is failure bin distribution, which typically commences immediately after the binning stage. Suppose that failure binning generates K failure clusters, $C_1, C_2, \ldots, C_K$. Failure bin distribution aims at allocating each of these K clusters to engineers that are most familiar with failures within that cluster. In past work this allocation is done by aggregating all suspects components that appear in failures of a particular cluster, sorting them by rank and passing that cluster to the engineer(s) that are best-suited to analyze the high-ranked suspects in that cluster.

This heuristic is, however, highly dependent on the quality of each cluster. Specifically, failures that are falsely placed in a cluster may misguide this allocation, as it is entirely possible that their suspects are ranked high, but are completely unrelated to the error responsible for that bin. In this work, we conjecture that it is naturally more suitable to allocate the cluster based on the failure that is most representative of the erroneous behavior within each bin, rather than by aggregating and sorting all suspects. The mechanic through which we determine such failures is presented in Section IV.

III. Coarse-grain Debugging

The proposed triage flow begins with the application of SAT-based root-cause analysis on each failure $F_i \in \mathbf{F}$, resulting in a set of N suspect sets $S_1, S_2, \ldots S_N$, as described in Section II. Recall that the suspect ranking process that follows requires us to produce an ordering of all suspects $s_j \in S_i$ for all suspect sets S_i. For a particular set S_i, we propose that the ordering can be generated by sorting suspects in S_i based on their likelihood of being the actual error source responsible for failure F_i. In this Section we describe a process for carefully selecting features related to simulation, which can be used by a predictive model to generate these likelihoods.

The process, which largely pertains to feature engineering, entails several aspects. Across a wide range of possible suspect features, such as location, type (signal, operand, type of statement), coverage information etc., we need to identify those that can efficiently aid in discriminating between spurious suspects and human-introduced errors (suspects-errors); for example, characteristics that are widely present in the former and rare in the latter, or vice versa. Additionally, these features need to be cost-efficiently mined, i.e., without requiring complex computations, rerunning simulation and/or SAT-based debugging. Finally, the number of suspect features should preferably be much smaller than the number of suspects to be classified, so as to mitigate "curse of dimensionality" issues. In this work, we conjecture that signal controllability and observability can give rise to effective simulation metrics that distribute differently between spurious suspects and suspects-errors, and thus become the basis of a predictive model.

With such directives, we focus on a particular probabilisitic error behavior model, which forms the motivational factor behind Bounded Model Debugging (BMD) [15], and which has been extended in [9] to serve similar purposes as the ones pursued here. The model embeds the concepts of excitation probability and propagation probability into a single expression for the probability of a design location causing a failure at some observation point. Excitation probability in this model relates to signal controllability, while propagation probability relates to signal observability. We restate the model and related assumptions below [9].

Suppose that an error exists in the design and that simulation begins at cycle 1. Let p_{ex} be the probability that the error is excited at cycle i. Also, let p_{pr} be the probability that the error propagates from cycle i to cycle $i+1$, and p_{ob} be the probability that an erroneous value reaches some observation point at some cycle i, given that the error has propagated to that cycle. Finally, assume that the input vector sequences are temporally independent and stationary random. Then, the probability $p_{k,n}$ that the error is excited at cycle k and propagates to an observation point at cycle $n \geq k$ is

$$p_{k,n} = (1-p_{ex})^{k-1} \cdot p_{ex} \cdot p_{pr}^{n-k} \cdot (1-p_{ob})^{n-k} \cdot p_{ob} \quad (1)$$

An analytic derivation of this formula is provided by the authors in [9]. An informal explanation is as follows: each cycle i in the ILA expansion of the error trace is viewed as a single trial to excite the design error. Since the design error is excited with probability p_{ex} at any cycle i, then the probability that it is excited at cycle k can be naturally modelled as a geometric distribution $(1-p)^{k-1} \cdot p$, where $p \leftarrow p_{ex}$; that is, the probability that the first $k-1$ trials fail to excite the design error and there is a success at cycle k. Similarly, the observation process is also modelled as a geometric distribution, with the difference that the first trial is indexed by k which is the first successful trial for the excitation process. Then, $n-k$ failed observation trials follow until the error is finally observed at the n-th trial. The propagation process, on the other hand, requires $n-k$ consecutive successes, from cycle k to cycle n which gives rise to the product p_{pr}^{n-k}.

Note that, modelling the propagation and excitation process by geometric distributions assumes that p_{ex} and p_{ob} remain the same across cycles. Despite this simplifying assumption, it has been empirically shown in [9] that the model produces good approximation of the general error behavior (i.e., expected propagation and excitation times). For the purpose of our work, the model encapsulates two parameters that are of interest. The first is the exponent $n-k$ of the propagation probability p_{pr}. This quantity relates to the temporal closeness between the excitation cycle k and the observation cycle n of an erroneous value, and it can be cost-effectively mined from the error trace of a failure, since the debugger points to the exact cycle where an erroneous value is excited at a suspect location. The second parameter is the excitation probability p_{ex}, itself. There is arguably a relation between the coverage of a suspect location (number of times the corresponding RTL segment is exercised) and the excitation probability. One can say that a suspect block with high coverage score measured between the simulation origin (cycle 1) and the excitation cycle k, potentially contains a bug that is harder to detect. In other words, it takes several coverage counts for the bug in that suspect to be sensitized. Such coverage measurement is also cost-effective given the simulation logs that are available to the triage engine after regression.

However, if we are to use metrics related to these parameters as suspect features, we first need to ensure that they can statistically expose a disparity between spurious suspects and suspects-errors. To this end, our goal is to determine the existence, if any, of underlying continuous distributions that can characterize these two parameters of the error propagation model. Knowing these distributions, enables us to perform a model fitting process, which can quantify to what extent spurious suspects and suspects-errors obey these distributions and if a separability can be determined. Our goal is not to perform exact inference on each suspect's parameter, but rather to infer how these parameters distribute across different suspects. To some extent this allows us to tolerate the accuracy loss introduced by the propagation model when we assume that p_{ex} is fixed across cycles.

When it comes to the distance $n-k$ between excitation cycle and observation cycle, one can see that it has a decaying exponential effect on $p_{k,n}$ in Eq. 1. Thus, we hypothesize that exponential distributions can serve as continuous characterizations of this parameter. On the other hand, for a fixed $n-k$ distance, $p_{k,n}$ decays towards 0 as the excitation probability p_{ex} tends to its limits, 0 and 1. Finally, when the distance $n-k$ varies we are interested in values for p_{ex} where $p_{k,n}$ obtains its maxima. Since we are interested in maxima of $p_{k,n}$ and p_{ex} has a finite lower limit, extreme value theory [16] suggests that the Weibull distribution is a candidate for characterizing the distribution of p_{ex} across various suspects. The probability density function of the Weibull distribution is given below:

$$f_{WB}(x;\lambda,\kappa) = \frac{\kappa}{\lambda}\left(\frac{x}{\lambda}\right)^{\kappa-1} e^{-(\frac{x}{\lambda})^{\kappa}} \quad (2)$$

where $\lambda > 0$ and $\kappa > 0$ are scale and shape parameters, respectively.

To verify our hypothesis, we prove that the product $(1-p_{ex})^{k-1}p_{ex}$ in Eq. 1 is in the domain of attraction of the Weibull, by showing that the Gnedenko [17] criteria are satisfied. This essentially proves that, for various probabilities p_{ex} across different suspects, the samples drawn from the density $(1-p_{ex})^{k-1}p_{ex}$ will converge to some known Weibull distribution.

Theorem 1. *Assuming the validity of the model in Eq. 1, with positive constants k, n, p_{pr} and p_{ob}, then $F(p_{ex}) = (1-p_{ex})^{k-1}p_{ex}$ is in the domain of attraction of the Weibull distribution.*

Proof. It has been shown by Gnedenko [17] that a distribution $F(x)$ belongs to the domain of attraction of the Weibull, if the following two criteria are satisfied:

$$\text{Criterion 1:} \quad x_l = \inf\{x|F(x) > 0\} > -\infty$$
$$\text{Criterion 2:} \quad \lim_{h\downarrow 0} \frac{F(hx - x_l)}{F(h - x_l)} = x^{\gamma}\ ,\ \gamma > 0$$

The lower bound of $F(p_{ex}) = (1-p_{ex})^{k-1}p_{ex}$ is clearly zero, thus $x_l = 0$ and Criterion 1 is met. To validate that Criterion 2 is satisfied, we use l'Hopital's rule and compute the relevant limit (using $x_l = 0$):

$$\lim_{h\downarrow 0} \frac{F(hp_{ex} - x_l)}{F(h - x_l)} = lim_{h\downarrow 0} \frac{\frac{\vartheta}{\vartheta h}F(hp_{ex} - x_l)}{\frac{\vartheta}{\vartheta h}F(h - x_l)} =$$
$$= \lim_{h\downarrow 0} \frac{\frac{\vartheta}{\vartheta h}hp_{ex}(1-hp_{ex})^{k-1}}{\frac{\vartheta}{\vartheta h}h(1-h)^{k-1}} =$$
$$= \lim_{h\downarrow 0} \frac{p_{ex}(1-hp_{ex})^{k-1} + hp_{ex}(k-1)(1-hp_{ex})^{k-2}}{(1-h)^{k-1} + h(k-1)(1-h)^{k-2}} =$$
$$= p_{ex} \lim_{h\downarrow 0} \frac{(1-hp_{ex})^{k-1} + h(k-1)(1-hp_{ex})^{k-2}}{(1-h)^{k-1} + h(k-1)(1-h)^{k-2}} =$$
$$= p_{ex}^{\gamma}$$

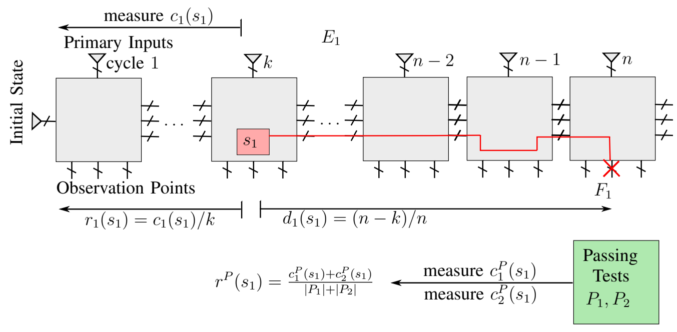


Fig. 3. Simulation metrics for suspect $s_1 \in S_1$ related to failure F_1. Temporal distance is measured between the excitation cycle k and the last cycle in E_i. Excitation resistance with respect to E_i is measured in the trace window staring at cycle 1 and ending at cycle k. Excitation resistance across passing tests is measured using coverage logs for passing tests P_1 and P_2.

where $\gamma = 1$. Hence Criterion 2 is also satisfied. $\square$

With the exponential and Weibull distributions established as candidate models for the two parameters of interest, we now define two metrics that will be used as features to represent suspect locations. These metrics have to be quantities that are tied to the two parameters at hand, and distribute similarly. Specifically, we propose a metric referred to as *temporal distance*, which is associated with the exponent $n-k$ in the model, and a metric referred to as *excitation resistance* in connection to the excitation probability p_{ex}.

Temporal distance refers to the number of cycles between the excitation of the suspect and the observation of the failure divided by the length of the error trace. As such, the temporal distance $d_j(s_i)$ of suspect s_i with excitation time t_i in an error trace E_j of length $|E_j|$ cycles is given as follows:

$$d_j(s_i) = \frac{|E_j| - t_i}{|E_j|} \tag{5}$$

Temporal distance assumes real values in the range $[0 \dots 1]$. Here, the smaller the distance is, the easier it is for the error excited at the relevant RTL block to propagate at the observation point.

Excitation resistance relates to the number of times a portion of the circuit (e.g., an RTL block) is covered by a test in the regression suite. Given an error trace E_j, a suspect block s_i, and its excitation cycle t_i, we first define the *coverage count* $c_j(s_i)$ of s_i with respect to trace E_j as follows:

$$c_j(s_i) = \langle \text{\# of times } s_i \text{ is covered in } E_j \text{ in cycles } [1, \dots, t_i] \rangle$$

Then, the excitation resistance $r_j(s_i)$ of a suspect RTL block s_i with respect to E_j is defined as the quotient

$$r_j(s_i) = \frac{c_j(s_i)}{t_i} \tag{7}$$

Excitation resistance also assumes values in the range $[0 \dots 1]$. Intuitively, the smaller the excitation resistance of a suspect is the easier it is for an error in the relevant RTL block to be excited by the test vectors and eventually propagate to the observation point. The metric is normalized over the number of cycles (t_i, assuming simulation starts at cycle 1) that it takes for the error to be sensitized.

It should be noted here that excitation resistance relates to the maximum likelihood estimate for $1-p_{ex}$ in the error propagation model. Let p_{MLE} be the maximum likelihood estimate for $1-p_{ex}$ in the geometric distribution $(1-p_{ex})^{k-1}p_{ex}$. Then it is known that p_{MLE} is given by:

$$p_{MLE} = \frac{\sum_{j=1}^{n} (k_j - 1)}{\sum_{j=1}^{n} k_j} \tag{8}$$

where n is the number of samples drawn and k_j is the total number of trials in each sample j. Note that in each sample j there is one successful trial and $k_j - 1$ failed trials. For a suspect with excitation time t_i that would translate into an estimate of $(t_i-1)/t_i$, since we only have one sample, which is the error trace itself and t_i trials, one for each cycle. However, this assumes that each cycle before t_i is a failed trial and may produce a rather pessimistic estimate when taking into account real conditions beyond the model's assumptions. For example, in reality not each cycle should be strictly viewed as a failed trial, especially when the suspect location is not covered in that cycle. Thus we only consider failed trials to be the ones where the location is at least exercised in that cycle, as this rises from Eq. 7. This means that excitation resistance $r_j(s_i)$ is upper bounded by the maximum likelihood estimate p_{MLE} for the suspect's probability of not being excited at some cycle, since $r_j(s_i) = c_j(s_i)/t_i \leq (t_i - 1)/t_i$.

Excitation resistance can also be measured in passing tests. One would expect that suspect locations with high coverage scores across passing tests are less likely to contain bugs that can be easily excited; these locations are covered multiple times but no error is captured. Recall that we assume there are M passing tests in the suite, $P_1, P_2, \dots, P_M$, the lengths of which (in terms of number of cycles) are denoted $|P_1|, |P_2|, \dots, |P_M|$. Then, we define the passing coverage count $c_j^P(s_i)$ of a suspect location s_i to be as follows:

$$c_j^P(s_i) = < \text{\# of times } s_i \text{ is covered in passing test } P_j > \tag{9}$$

Then, the excitation resistance $r^P(s_i)$ of a suspect RTL block s_i with respect to passing tests $P_1, P_2, \dots, P_M$ is defined as:

$$r^P(s_i) = \frac{\sum_{j=1}^{M} c_j^P(s_i)}{\sum_{j=1}^{M} |P_j|} \tag{10}$$

Observe that the excitation resistance of a suspect location with respect to passing tests is identical between suspects that correspond to the same design location, even if they appear in different suspect sets. In contrast, the excitation resistance with respect to each error trace can be potentially different between these suspects to account for their variability within these error traces. From a probabilistic standpoint, $r^P(s_i)$ is an approximation of p_{MLE} for suspect s_i viewing all M passing tests as samples and the sum of all coverage counts over these tests as failed trials, whereas $r_j(s_i)$ refines the approximation regarding suspect s_i by considering its presence in error trace E_j. Figure 3 shows an example of how these metrics can be extracted for a particular suspect location.

The model fitting process described in our experiments (Section V) confirms that the exponential and Weibull distributions are a good fit for these two metrics, respectively. Using these models, we show that suspects-errors are generally governed by lower temporal distance and lower excitation resistance compared to spurious ones. An informal justification of the above is that we expect human-introduced errors to be relatively easier to excite and easier to propagate compared to spurious suspects, which often correspond to obscure ways of justifying a mismatch. This, in turn, statistically justifies the suitability of these metrics for discriminating spurious suspects and suspects-errors by means of a predictive utility, such as a classifier.

In our methodology we choose Support Vector Machines (SVMs) [18] to undertake the binary classification task, where our goal is to predict whether a suspect is a suspect-error (class 1) or a spurious suspect (class 0). There is a twofold rationale behind choosing SVMs. First, we are not interested in estimating the marginal probability of a suspect component being an error. Rather, we desire some measure of confidence based on which we can produce an ordering of the suspects in the suspect raking process. SVMs offer exactly this type of prediction. For each suspect location that is passed as a new input to the classifier, the SVM will output a *confidence score* indicating how likely it is that the new input belongs to one of the two classes. Second, SVMs do not require training sets (suspects passed as training inputs) to be much larger than the number of features. Given that the number of suspects that can be identified is not always in our control, since it largely depends on the complexity of the design and type of errors that the suite can detect, we desire a model that can operate with training sets of small size without jeopardizing accuracy.

The specifics of how SVMs optimize the boundary hyperplanes that separate classes in the feature space are outside of the scope of this paper, so they are omitted here. However, we declare some important meta-parameters of the SVM model used: (a) the Radial Basis Kernel is used as a non-linear kernel function for feature mapping, (b) the hinge loss function is used as a soft margin, and (c) the classification strategy is "one-against-one". The training set for the SVM model is a collection of past debugging results, i.e., identified root causes, spurious suspects, and relevant simulation metrics that are aggregated across previous regression epochs. Maintaining a comprehensive database ensures that enough data will be available for training purposes. In case that the design process is in very early stages, implying the absence or limited access to training sets, then one can resort to heuristics that are based on the same simulation metrics that we introduce in this work; for example, conducting suspect ranking based on empirical thresholds on these metrics. Such heuristics have also been discussed in previous works [9].

Assuming that historical debugging data are indeed available, the inputs to our SVM model are suspect objects represented using the three features described before; temporal distance, excitation resistance with respect to the error trace, and excitation resistance across passing tests. Suspects that are labelled and thus belong to the training set are accompanied by their class indicator, which is set to 0 if the suspect is spurious and set to 1 if the suspect is the actual error source. Thus, if a suspect $s_i \in S_j$ in the training set is spurious, then it is represented as $s_i :=< d_j(s_i), r_j(s_i), r^P(s_i), 0 >$, and if it is a suspect-error, then it is represented as $s_i :=< d_j(s_i), r_j(s_i), r^P(s_i), 1 >$. Suspects that are unlabelled, that is, the ones we need to make predictions for, are represented identically, but without the class indicator.

In any case, if we denote the SVM module as $SVM(x, \vartheta)$, where x is a new input and ϑ is the set of trained model parameters, then the confidence score of $s_i \in S_j$ for some failure F_j in terms of being in class 1 or class 0, is denoted as $score_i^j(class)$, such that:

$$score_i^j(class) \leftarrow SVM(< d_j(s_i), r_j(s_i), r^P(s_i) >, \vartheta) \quad (11)$$

where $class$ assumes the corresponding class indicator. For each suspect s_i, $score_i^j(class)$ assumes real values between $-\infty$ and $+\infty$. The greater the value is for a particular class, the more confident the engine is that s_i belongs to that class.

To order suspects based on likelihood of being the actual error source we need to assign weights to these suspects, such that ones strongly related to class 1 assume a larger weight. Specifically, the weight of $s_i \in S_j$ for some failure F_j, denoted w_i^j, is given as:

$$w_i^j = \begin{cases} score_i^j(class) & \text{, if } \; class = 1 \\ -score_i^j(class) & \text{, if } \; class = 0 \end{cases}$$

With this weighting scheme, we sort each suspect set S_j in non-increasing order of suspect weights. Ties are broken randomly. Essentially, suspects with larger weight, which are more likely to be suspects-errors based on our assumed model, appear earlier in the ordered set. Apart from guiding bin distribution as it is later discussed, these weights form a basic element in how we define pairwise failure similarities for the failure binning process that follows.

IV. Automating Failure Binning and Bin Distribution via Belief Propagation

Once all suspect weights are computed, the engine proceeds to construct pairwise failure similarities. Recall that the similarity between two failures F_i and F_j quantifies to what extent these two failures possibly share the same root cause.

Definition 1. *Given any two failures $F_i, F_j \in \mathbf{F}$ with $i \neq j$, we denote their pairwise failure similarity as $s(i, j) \in \mathbb{R}_{\leq 0}$. We say that for three distinct failures, F_i, F_j, F_k, if $s(i, j) > s(i, k)$, then F_i is more likely to originate from the same error source as F_j, than it is to originate from the same error source as F_k.*

To compute similarities, we propose a mechanism that represents failures as high-dimensional objects, enabling a mapping of these into a joint space. Particularly, we construct a feature-based representation for each failure $F_j \in \mathbf{F}$, as follows. Suppose $s_1, s_2, \dots, s_D$ are all the distinct suspect components in $\bigcup_{i=1}^{N} S_i$. Then, failure F_j is represented by a D-dimensional real-valued feature vector $\vec{F_j} = [x_1^j, x_2^j, \dots, x_D^j]$, where each feature x_i^j is assigned the weight (significance) of suspect s_i with respect to failure F_j, if s_i appears in S_j, or obtains a value of 0 otherwise:

$$x_i^j = \begin{cases} w_i^j & \text{, if } \; s_i \in S_j \\ 0 & \text{, if } \; s_i \notin S_j \end{cases} \tag{12}$$

Failures are then mapped into a D-dimensional space, where similarity $s(i,j)$ can be naturally defined as the negative squared error (Euclidean distance) between $\vec{F_i}$ and $\vec{F_j}$:

$$s(i,j) = -||\vec{F_i} - \vec{F_j}||^2 \tag{13}$$

The squared error in Eq. 13 is negated to abide to similarity semantics. Note that pairwise similarities $s(i,j)$ are non-positive real values, as desired. If two failures F_i and F_j have multiple common suspects of similar weight, then their vectors are expected to appear closer in the joint space, thus obtaining a relatively large pairwise similarity. For the binning and distribution stages that follow, these pairwise failure similarities are given in the form of a $N \times N$ similarity matrix $\mathbf{S}$.

A. Problem Formulation

Unlike prior work, our methodology generates solutions to failure binning and distribution simultaneously, in a unified manner. This is achieved by casting the whole process as exemplar-based clustering. Exemplar-based clustering partitions the data into clusters, but also identifies the most representative member of each cluster, also referred to as *exemplar*. An exemplar is the cluster member that exhibits maximum overall similarity to all other members in the cluster. In our context it can be viewed as the failure that is the most representative of the erroneous behaviour associated with all other failures in the same bin. This failure-exemplar along with its suspect locations can thus determine how to distribute the failure bin it belongs to.

To solve the problem under this setting we apply an algorithm known as Affinity Propagation (AP) [8]. The algorithm is derived as an instance of max-product loopy belief propagation on graphical models [8]. AP does not explicitly search for exactly K clusters, but instead offers a trade-off between the number of clusters and the within-cluster similarity that is obtained. As such, it is not required that the number of clusters is specified *a priori*. Instead, the algorithm allows the engineer to specify failures that are believed to be of high importance. For that purpose, a parameter called *preference*, denoted p_i, is associated with failure F_i. When preference p_i is set to higher values, then F_i is internally promoted by the algorithm to become an exemplar. Preferences are provided as input in the form of a N-dimensional vector $\mathbf{p} = [p_1, p_2, \dots, p_N]$. These values are then assigned to the self-similarities in the diagonal of matrix $\mathbf{S}$. Specifically, $s(k,k) = p_k, \forall k \in \{1, \dots, N\}$.

Supposing that the algorithm takes $\mathbf{S}$ and $\mathbf{p}$ as input, then, a set of N^2 binary random variables $h_{ij} \in \{0,1\}$ is defined, such that $h_{ij} = 1$ if and only if failure F_i has chosen F_j as its exemplar. In this setting, if $h_{jj} = 1$, then F_j has in fact been chosen as an exemplar by other failures. The objective function is then to maximize the sum of all similarities between failures in the cluster to their failure-exemplar, while also maximizing the total preferences. The associated constrained optimization problem is given as:

$$\max_{\{h_{ij}\}} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} s(i,j) h_{ij} \tag{14a}$$

subject to

$$\sum_j h_{ij} = 1 \qquad \forall i \tag{14b}$$

$$h_{jj} = \max_i h_{ij} \quad \forall j \tag{14c}$$

Eq. 14b ensures that each failure chooses exactly one other failure as its exemplar, while Eq. 14c ensures that an exemplar is never choosing another exemplar. The goal is to find settings of $\{h_{ij}\}$ that maximize the sum-product in Eq. 14a. AP finds solutions through an iterative message-passing process [8]. Upon convergence it returns a set of exemplar failures denoted as $\mathcal{F}_{ex}$:

$$\mathcal{F}_{ex} = \{F_j \in \mathbf{F} : h_{jj} = 1\} \tag{15}$$

Due to the blocking constraint in Eq. 14c, each exemplar defines exactly one cluster of failures and is itself a member of the cluster. Hence, the number of clusters, K, equals the number of exemplars $|\mathcal{F}_{ex}|$. Further, for each non-exemplar failure F_i there exists an exemplar failure F_j which is the most similar to F_i across all other exemplars in $\mathcal{F}_{ex}$. The set of non-exemplars that are most similar to exemplar F_j is therefore

$$\{F_i \in \mathbf{F} : i = \underset{F_j \in \mathcal{F}_{ex}}{\operatorname{argmax}}\, s(i,j)\} \tag{16}$$

Each non-exemplar failure is then assigned to the same cluster C_k as its most similar exemplar failure F_j. Ultimately, during the bin distribution stage, cluster C_k is passed to the engineer(s) that are responsible for the high-ranked suspect locations in S_j associated with failure F_j, where F_j is the exemplar in cluster C_k.

The are several benefits offered by the proposed formulation. First, it does not enforce any constraints on similarities, apart from requiring that they are non-positive real values. Similarities can be either metric or non-metric, symmetric or asymmetric, with no impact on the formulation, unlike prior work which only operates with non-metric similarities [9]. Second, the number of clusters emerges algorithmically from the message-passing process, and does not need to be “guessed”. Moreover, bin distribution is guided by suspects that correspond to failures-exemplars, with low risk of false positive failures misguiding the process. Finally, this enables engineers to focus resources and perform detailed debugging on each exemplar failure and its suspect set. Assuming that a failure-exemplar is indeed representative of the design error responsible for its cluster and that ranking pushes the actual error towards the top of the ranked suspect list, fixing it is likely to eliminate most or all other failures in the same cluster.

B. Triage with Prior Belief

An additional benefit is that the method offers flexibility and user-control considering various triage scenarios. Although it is generally difficult to have an accurate estimate on the number of design errors responsible for all failures, there are cases where engineers, based on experience, target specific failures around which they wish $\mathbf{F}$ to be partitioned. These are failures that the engineer believes should serve as exemplars of erroneous behavior. For example, an engineer may suspect that a bug triggering a specific assertion also manifests at multiple other locations either as exception catching or golden value mismatches, but without explicit knowledge of which these locations are. In that case, they may wish to identify at least one cluster with this failing assertion being the exemplar. To accommodate such scenarios, triage can be performed under two different settings.

The *uniform setting* refers to cases where no assumptions are made regarding to what extent specific failures should be targeted; they are all treated as equally likely to become exemplars. Generally this translates into fixing all preferences $p_k \in \mathbf{p}$ to be the median of all similarities:

$$p_k = \frac{1}{N^2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} s(i,j), \quad k \in \{1, \dots, N\} \tag{17}$$

Conversely, the *non-uniform setting* refers to scenarios where the engineer selects specific failures to promote as exemplars. If failure F_k is targeted then p_k is set to 0. Otherwise p_k is set to the median of similarities, as in Eq. 17. Setting higher preferences affects the number of clusters to be produced, but the number also rises from the message-passing procedure. It is therefore entirely possible that the number of clusters formed at the end will not match the number of promoted failures, if this number does not reflect a reasonable partition. However, if the "guess" is close to ground truth, then the algorithm can produce higher quality solutions. Note that this feature is not offered by any of the existing methods, as the only parameter that is externally enforceable is the number of clusters K and, possibly, the initial seeds for k-means. Such constraints, however, cannot guarantee that $\mathbf{F}$ is eventually partitioned around targeted failures.

To conclude, Figure 4 illustrates the triage process outlined in the previous Sections.

V. Experimental Results

This Section presents experimental results for the proposed triage framework. All experiments are conducted on a single core of an Intel Core i5 3.1 GHz workstation with 8GB of RAM. A total of eight designs are used, from *OpenCores* [19] and in-house development. The underlying automated debugging tool used for extracting the suspect locations is implemented based on [3]. A platform coded in Python is developed to parse debugging and simulation results. For each design, a set of different errors is injected each time by modifying the RTL description. In total, thirty two regression simulations are run, generating a different number of failures each time, caused by a different set of errors. The types of the injected

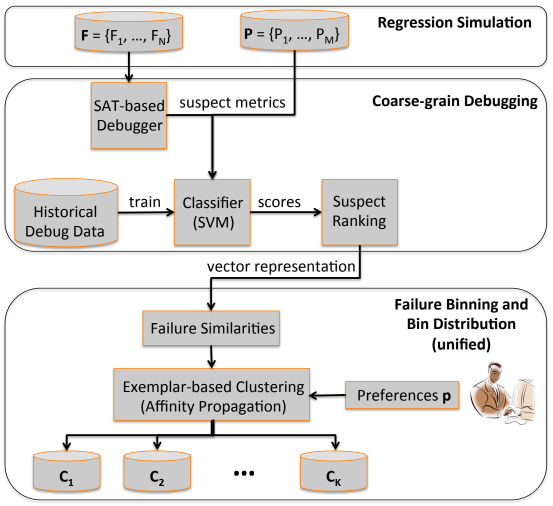


Fig. 4. Proposed Triage Flow

TABLE I
Benchmarks and Regression Statistics

Ckt. (# logic elem.)	Instance No.	# errors	# fail. (N)	# distinct susp. (D)
vga (109797)	vga-1	4	45	36
	vga-2	8	97	61
	vga-3	10	106	129
	vga-4	13	121	155
fpu (83303)	fpu-1	3	19	28
	fpu-2	7	30	29
	fpu-3	9	83	60
	fpu-4	11	125	111
ddr (55069)	ddr-1	5	28	46
	ddr-2	6	51	82
	ddr-3	8	54	79
	ddr-4	9	72	113
mem. ctrl (46767)	mem. ctrl-1	5	32	45
	mem. ctrl-2	7	31	29
	mem. ctrl-3	8	66	94
	mem. ctrl-4	11	95	137
tate pairing (106786)	tate pairing-1	5	51	37
	tate pairing-2	7	55	40
	tate pairing-3	8	61	92
	tate pairing-4	10	95	157
pkt. fwd (40197)	pkt. fwd-1	5	21	35
	pkt. fwd-2	6	25	28
	pkt. fwd-3	7	86	90
	pkt. fwd-4	10	104	196
fdct (277444)	fdct-1	3	22	83
	fdct-2	7	38	100
	fdct-3	9	66	91
	fdct-4	9	90	165
scam_core (509304)	scam_core-1	4	32	19
	scam_core-2	5	44	50
	scam_core-3	7	71	55
	scam_core-4	10	68	79

RTL errors resemble typical human-introduced errors that lead to non-trivial triage scenarios. Types of injected bugs include:

- Type A: missing pipeline stages, incorrect assignments, operands, function replacements
- Type B: incorrect testbench stimuli, incorrect reference model (bug is outside of DUV)
- Type C: bit flips, incorrect padding, and stuck-at faults

For each design, we separate its regression runs into a training set and a testing set. The training set is used to train an SVM model that we associate with each design, and to select proper model parameters (discussed in Section III) via 5-fold cross validation. The testing set is used to evaluate suspect ranking, failure binning and bin distribution accuracy. In order to emulate how a design evolves during a typical verification process, the injected errors used to produce the training dataset are first fixed, and then new ones are introduced to produce the testing set. Finally, errors in the design are captured by checkers and/or System Verilog Assertions.

Table I summarizes benchmark information per regression run. From left to right, columns show the design name and total number of logic elements in the synthesized design, the instance number corresponding to each regression run, the number of simultaneously injected RTL errors, the number of exposed failures (N), and finally the number of distinct suspect components (D) generated by SAT-based debugging per instance. Instance names with 1 or 3 as a suffix correspond to those used for training, while instances with 2 or 4 as a suffix are used for testing and evaluation. The SAT-based debugging tool used in our flow returns suspect components hierarchically, from the module level down to the signal level [3]. When collecting suspect locations, we favor suspects that correspond to design modules and/or Verilog/VHDL blocks, rather than suspects at the signal/gate level. Apart from the reasons outlined in Section II-B1, this is also to maintain the dimensionality of each failure at manageable levels, without necessitating dimensionality reduction.

A. Evaluation of Metrics

In this subsection we present our model fitting process for testing the discrimination power of the proposed metrics. Any suspect components and relevant metrics mentioned henceforth are aggregated from the regression runs that belong to the training set, as shown in Table I. For our evaluation we create a histogram $\mathbf{h^d_{err}}$ that captures the frequency of suspects-errors that have temporal distance falling within intervals of length 0.1 in the range $[0.0\ldots1.0]$. Likewise, we create histogram $\mathbf{h^d_{spur}}$ for the temporal distance of spurious suspects, $\mathbf{h^r_{err}}$ for the excitation resistance of suspects-errors, and $\mathbf{h^r_{spur}}$ for the excitation resistance of spurious suspects.

We expect that Weibull explains well our empirical data when it comes to excitation resistance. However, for the sake of rigor we further test the goodness of fit offered by alternative distributions that also exhibit at least exponentially decreasing tails. These distributions are the Log-normal, denoted f_{LN}, and Gamma, denoted f_G. On the other hand, for temporal distance data we solely perform an exponential fit, since it naturally arises from Eq 1.

TABLE II
GOODNESS OF FIT

model	$\mathbf{D_{KL}}$ **divergence**			
	$\mathbf{h^d_{err}}$	$\mathbf{h^d_{spur}}$	$\mathbf{h^r_{err}}$	$\mathbf{h^r_{spur}}$
f_{exp}	**0.0592**	0.0950	N/A	N/A
f_{WB}	N/A	N/A	**0.0443**	0.2822
f_G	N/A	N/A	0.0723	0.1148
f_{LN}	N/A	N/A	0.0548	0.2071

For model fitting we apply multinomial maximum likelihood [20] on the histograms, and we use the Kullback-Leibler (KL) divergence to evaluate the quality of fit. The KL divergence between empirical data $\mathbf{h} \in \{\mathbf{h^d_{err}}, \mathbf{h^d_{spur}}, \mathbf{h^r_{err}}, \mathbf{h^r_{spur}}\}$ and fitted model $\mathbf{f} \in \{f_{WB}, f_{LN}, f_G, f_{exp}\}$ is denoted $D_{KL}(\mathbf{h}|\mathbf{f})$ and given below:

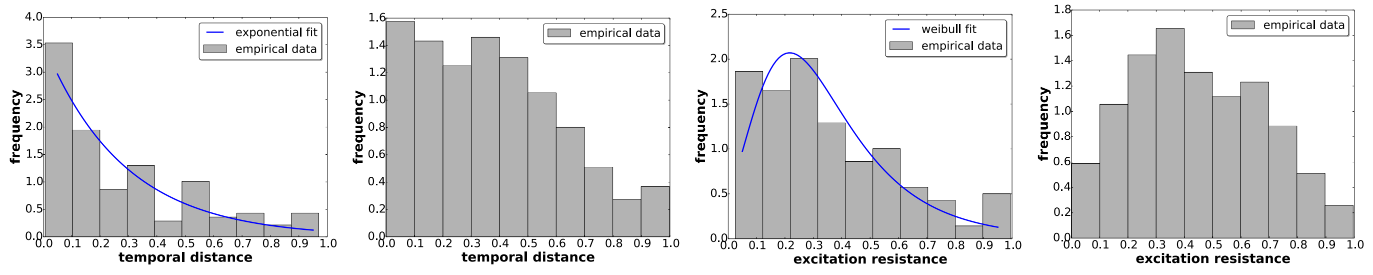
$$D_{KL}(\mathbf{h}|\mathbf{f}) = \sum_d h_d \log \frac{h_d}{f_d} \quad (18)$$

where f_{exp} is the exponential distribution, and d corresponds to the sampling points used for evaluation. KL divergence measures information loss when $\mathbf{h}$ is represented by $\mathbf{f}$. Low KL divergence implies that the model explains the data well, and vice versa.

Both for temporal distance and excitation resistance we find which distribution and under which parameters offers the lowest KL divergence for suspects-errors. We then use that same model to fit our empirical data for spurious suspects, and we obtain the KL divergence as well. The goal is to exhibit that the best model to fit the data for suspects-errors does not offer a good fit for spurious suspects. This, in turn, exposes a separability between these two entities when the metrics at hand are used as the characterization basis, and thus justifies that these metrics can be used for filtering suspects.

Figure 5 illustrates the produced histograms. Particularly, Figure 5(a) shows the aggregate temporal distance histogram for suspects-errors over all circuits ($\mathbf{h^d_{err}}$) and its exponential fit. By inspection, once can say that the distribution of temporal distance for suspects-errors follows the exponential curve, as expected. Similarly, Figure 5(b) illustrates the aggregate histogram $\mathbf{h^d_{spur}}$. It can be seen that temporal distance distributes differently for spurious suspects, with a tail that does not decrease exponentially and a great portion of the frequencies almost uniformly distributed in the range $[0.0\ldots0.5]$. It is worth noting that the majority of spurious suspects exhibit relatively low temporal distance. This skewness can be justified to some extent by considering the existence of spurious suspects that are temporally related to the actual design error when the former reside in the error's fan-out or fan-in cone.

In terms of excitation resistance, the histogram for suspects-errors and the Weibull fit, as well as the histogram for spurious suspects are illustrated in Figure 5(c) and Figure 5(d), respectively. In this experiment, there are apparent differences between the histograms. Human-introduced errors tend to follow a Weibull model, and in fact with a relatively low mean. In contrast, spurious suspects appear to concentrate towards modes further away from small excitation resistance values. This concentration can potentially be a consequence of some



(a) Exponential fit on $\mathbf{h}^{\mathbf{d}}_{\mathbf{err}}$ (b) $\mathbf{h}^{\mathbf{d}}_{\mathbf{spur}}$ (c) Weibull fit on $\mathbf{h}^{\mathbf{r}}_{\mathbf{err}}$ (d) $\mathbf{h}^{\mathbf{r}}_{\mathbf{spur}}$

Fig. 5. Temporal distance and excitation resistance histograms

TABLE III
SUSPECT RANKING QUALITY

Instance No.	suspect set size (mean)	top-20% (top-50%) rank error in [9] (%)	top-20% (top-50%) rank error in proposed (%)				
			failing traces only			failing + passing traces	
			d	r	d+r	r	d+r
vga-2	16	36.8 (22.8)	30.8 (17.3)	40.6 (26.5)	32.2 (17.5)	25.0 (24.0)	**21.2 (14.9)**
vga-4	18	35.1 (24.1)	44.5 (31.3)	39.2 (34.1)	30.5 (24.3)	28.5 (21.7)	**20.3 (12.4)**
fpu-2	9	26.3 (18.3)	24.5 (19.8)	23.9 (18.4)	18.7 (10.5)	19.3 (15.4)	**16.6 (9.8)**
fpu-4	14	34.7 (19.9)	27.6 (22.9)	30.5 (26.2)	25.6 (18.4)	28.8 (27.0)	**22.7 (17.7)**
ddr-2	16	22.8 (13.0)	23.5 (18.1)	22.7 (16.1)	**16.6 (8.1)**	22.3 (16.5)	16.9 (8.3)
ddr-4	21	24.5 (16.3)	20.8 (17.4)	34.9 (26.5)	20.7 (13.9)	19.6 (16.6)	**19.0 (10.2)**
mem. ctrl-2	13	32.9 (20.1)	30.4 (20.1)	41.7 (29.0)	24.7 (12.6)	30.7 (24.1)	**18.5 (9.3)**
mem. ctrl-4	24	34.7 (27.2)	33.8 (24.9)	36.6 (23.5)	29.8 (19.3)	25.3 (22.1)	**25.1 (18.4)**
tate pairing-2	20	21.2 (14.6)	29.5 (19.0)	24.5 (20.0)	**17.5 (13.0)**	24.6 (18.0)	17.6 (11.9)
tate pairing-4	27	18.4 (15.0)	20.6 (15.4)	16.7 (14.4)	11.8 (8.2)	19.9 (18.4)	**9.9 (5.7)**
pkt. fwd-2	13	11.3 (9.8)	19.6 (13.3)	25.6 (20.3)	9.0 (6.1)	10.5 (10.1)	**9.0 (5.9)**
pkt. fwd-4	12	19.6 (14.3)	21.3 (15.2)	23.7 (17.6)	14.2 (7.7)	19.1 (17.6)	**10.4 (7.2)**
fdct-2	20	30.3 (28.4)	33.7 (29.5)	37.1 (29.2)	29.4 (21.2)	30.2 (19.4)	**20.6 (18.8)**
fdct-4	24	31.2 (25.0)	30.4 (22.0)	28.4 (28.0)	23.5 (20.5)	28.1 (23.6)	**18.4 (17.3)**
scam_core-2	13	41.7 (32.6)	35.1 (30.6)	40.1 (35.2)	31.4 (27.2)	33.7 (28.1)	**29.1 (16.8)**
scam_core-4	15	25.8 (22.2)	27.2 (24.5)	26.6 (22.3)	19.7 (19.0)	19.9 (18.7)	**17.5 (14.9)**
Average		27.9 (20.3)	28.9 (21.3)	32.3 (25.6)	22.2 (15.5)	24.1 (20.1)	**19.4 (13.5)**

TABLE IV
FAILURE BINNING QUALITY

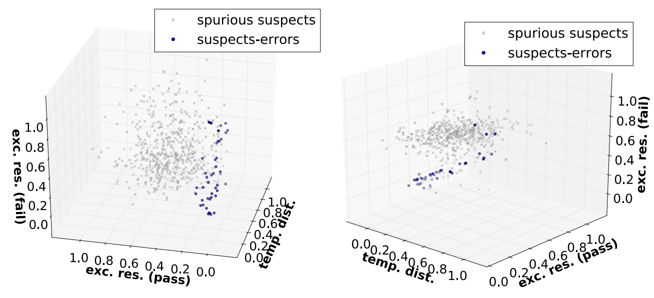
Instance No.	NMI							precision (PPV)		cluster statistics			
	log bins	no scoring		proposed						abs. error		mean size	
			[9]	uniform	impr.	non-uniform	impr.	[9]	proposed	[9]	proposed	[9]	proposed
vga-2	0.37	0.77	0.73	0.82	12.3%	0.83	13.7%	0.77	0.79	2	1	9.7	10.8
vga-4	0.44	0.75	0.79	0.81	2.5%	0.81	2.5%	0.83	0.86	1	1	8.6	8.6
fpu-2	0.42	0.79	0.80	0.79	-2.0%	0.87	8.8%	0.75	0.75	1	1	5.0	5.0
fpu-4	0.58	0.81	0.73	0.86	17.8%	0.92	26.0%	0.66	0.89	3	0	15.6	11.4
ddr-2	0.29	0.85	0.80	0.92	13.5%	0.93	14.8%	0.78	0.90	2	1	12.8	7.3
ddr-4	0.31	0.66	0.68	0.79	16.1%	0.79	16.1%	0.64	0.74	1	0	9.0	8.0
mem. ctrl-2	0.56	0.92	0.91	0.92	1.1%	0.91	0%	0.82	0.82	0	0	4.4	4.4
mem. ctrl-4	0.37	0.90	0.85	0.94	10.5%	1.0	17.6%	0.79	1.0	2	0	10.6	8.6
tate pairing-2	0.23	0.82	0.71	0.82	15.5%	0.84	18.3%	0.63	0.88	2	0	6.1	7.9
tate-pairing-4	0.35	0.70	0.66	0.77	16.7%	0.78	18.2%	0.55	0.83	4	1	15.8	8.6
pkt. fwd-2	0.18	0.64	0.59	0.75	27.1%	0.93	57.6%	0.47	0.94	3	2	8.3	6.3
pkt. fwd-4	0.19	0.68	0.70	0.79	12.9%	0.88	25.7%	0.71	0.85	2	2	13.0	13.0
fdct-2	0.24	0.88	0.81	0.90	11.1%	0.93	3.0%	0.81	0.85	3	2	20.4	22.2
fdct-4	0.39	0.90	0.90	0.92	2.2%	1.0	11.1%	0.84	0.85	1	1	10.1	14.3
scam_core-2	0.48	0.75	0.72	0.81	12.5%	0.82	13.9%	0.67	0.74	2	1	6.8	8.5
scam_core-4	0.12	0.77	0.76	0.78	2.6%	0.79	4.0%	0.68	0.80	2	0	8.2	7.7
Average	0.31	0.79	0.75	**0.84**	**12.0%**	**0.87**	**16.0%**	0.70	**0.85**	1.94	**0.81**	8.87	**9.12**

TABLE V
RANKING ERROR FOR EACH TYPE OF INJECTED BUGS

Error Type	top-20% rank error	top-50% rank error
Type A	18.7%	11.2%
Type B	14.4%	9.2%
Type C	24.8%	20.1%

tests in the suite being tailored to rigorously exercise specific module-level functionality in the DUV.

Table II summarizes goodness of fit results, where we report the KL divergence of the model fitting process for the histograms previously discussed. Recall that lower KL divergence indicates a better fit. Column 1 corresponds to the model to be fitted each time, while columns 2 to 5 show



(a) vga (b) fpu

Fig. 6. 3D mapping of spurious suspects and suspects-errors

KL divergence values for histograms $\mathbf{h^d_{err}}$, $\mathbf{h^d_{spur}}$, $\mathbf{h^r_{err}}$ and $\mathbf{h^r_{spur}}$, respectively. From these results, one can see that the exponential and Weibull distributions are indeed the best fit for temporal distance and excitation resistance. Additionally, the learned parameters for suspects-errors are not appropriate to characterize spurious suspects, hence the discrepancies in KL divergence shown in Table II.

B. Quality of Suspect Ranking

To further support the quality of the suggested metrics, we demonstrate the prediction error that the SVM model achieves for each design, using the features defined in Section III. It is worth stressing that each suspect set usually contains a large number of spurious suspects, whereas it only contains a single suspect-error. If we were to train using all spurious suspects we would introduce a significant bias in class sizes (approximately a $1 : 20$ ratio). To alleviate this issue, we down-sample: for every suspect error we randomly select three spurious suspects from the same suspect set, thus enforcing a $1 : 3$ ratio between the two classes in the training set. This ratio empirically provided a good trade-off between class size bias and training set size.

To perform a direct comparison of prediction quality between our method and the heuristic-based method in [9] we contrast the suspect ranking quality that these two methods produce, since suspect ranking is merely an ordering of suspects based on prediction confidence. To this end we find useful to define prediction error as follows: a suspect-error is misclassified if it does not belong to the top-20% of the ordered suspect list. For example, if a suspect-error appears in a suspect set of size 50, and our SVM-based ranking places it anywhere below the top 10 positions, then the suspect-error is misclassified. Similarly we report prediction error based on the top-50% of the ranked suspect sets. Results are shown in Table III. Here, each table entry corresponds to the average top-20% prediction error, and the average top-50% prediction error in parentheses, for the method in [9] and the proposed method under various configurations. We split results in two major sets; one corresponding to our method's performance when only failing traces are used, and one when both failing and passing traces are utilized. Further, we break-down results based on which features are used; temporal distance only (`d`), excitation resistance only (`r`), or both (`d+r`). Finally, we report the mean suspect set size for each regression.

From Table III, it can be seen that the proposed supervised method is superior to [9], even when only failing traces are used. This is expected, since [9] performs unsupervised ranking without any training. Our method offers a 21% (24%) reduction in average top-20% (top-50%) prediction error. The average errors are 22.2% vs. 27.9% and 15.5% vs. 20.3%, respectively. When information from passing tests is also used, our method's predictive power further improves. We achieve a 19.4% (13.5%) top-20% (top-50%) error, which constitutes an improvement of 31% (34%) over existing methods. One interesting observation is that the discriminating power of temporal distance appears to be stronger compared to excitation resistance when it comes to failing traces. Excitation resistance, on the other hand, appears to offer stronger predictions when passing traces are used; high coverage of specific modules in passing tests may be a strong indicator of bug-free modules. However, the prediction error does not improve significantly when examining the top-50% suspects compared to other settings in the table. This may be explained by the fact that, some times, coverage levels -which affect excitation resistance measurements- are characteristic of the test itself rather than the root-cause, and thus cannot expose abnormalities between root-causes and spurious suspects. In any case, the fusion of all these metrics across failing and passing tests aids the engine to break such uncertainties and achieve superior performance.

One question that arises regarding ranking quality is whether it depends on the type of injected errors. Table V shows accuracy achieved for all three types of injected errors. It can be seen that Type B bugs that lie outside of the DUV are the ones identified as real bugs most effectively by the ranking scheme. Type A bugs are also effectively pushed higher in the ranked lists. On the other hand bugs that are more obscure or deeper, such as the ones of Type C, are more difficult for the engine to promote as actual errors, mainly due to the fact that other equivalent suspects resemble them. This is essentially the case where the discriminating power of excitation resistance and temporal distance is less obvious. Despite that, the ranking error for Type C bugs still ranges between acceptable values.

To visualize how the three features used in this work help us discriminate between suspects-errors and spurious ones, we show two of the 3-dimensional mappings that are produced when training an SVM for `vga` and `fpu` (without down-sampling). Figure 6 illustrates the distinct subspaces that these two classes occupy. Although there exist some overlaps, these are treated properly by training non-linear SVMs.

C. Quality of Failure Binning

The quality of failure binning can be naturally quantified by two factors. The disparity between the number of bins formed and the number of errors present in the design which we refer to as the *absolute cluster error*, and the ratio of correct clustering decisions (clustering accuracy) when failures are grouped together. To evaluate clustering accuracy we use the Normalized Mutual Information (`NMI`) [21], as it is well suited for comparing clusterings with different number of clusters. `NMI` always ranges between 0 and 1, with 1 corresponding to perfect accuracy.

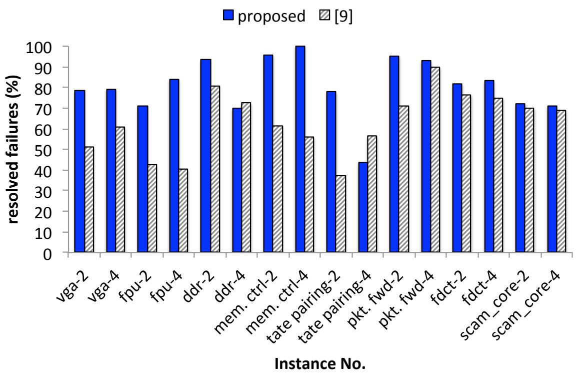


Fig. 7. Failures resolved by bin distribution

Table IV summarizes these results. From left to right, columns indicate the instance name, `NMI` for the method in [9], `NMI` for the proposed method under the uniform setting and a non-uniform setting, the improvements we achieve on `NMI` for both these settings, the average precision, also known as Positive Predictive Value (PPV), the absolute cluster error, and finally the mean cluster size. For reference, we also provide the `NMI` achieved by a simple binning method based solely on the error logs. It can be viewed as a document clustering process, where we apply a common method based on word frequency scores to represent error logs and then apply k-means clustering. Results for this method are shown in the second column. Finally, column 3 shows the NMI achieved when we turn-off the SVM-based scoring function, and instead allow failures to be represented by $0/1$ vectors in Eq. 12. The results obtained in this case offer an estimate on the accuracy of other methods that may replace the SAT engine in our flow. For example, when Binary Decision Diagrams [22], critical path tracing, or simulation may be preferred. These methods can produce sets of possibly buggy locations, but cannot output *exact* excitation and propagation paths.

In terms of `NMI`, the proposed binning method achieves a 12.0% improvement over [9] in the uniform setting, while this extends to a 16.0% improvement in the non-uniform setting. In the latter, to emulate the case of a good -but not perfect- guess we set the preferences in such a way so that one third of the exemplar failures are correctly identified by the engineer. The better the guess is, the higher the accuracy, but identifying many of these exemplars may not reflect a realistic assumption. Our method's precision is also significantly higher (0.85 vs. 0.70). Lower precision indicates higher false positive rates, which confirms that this is the bottleneck of the method in [9], while our method is robust in that sense. False positives are damaging triage effectiveness, as they can misguide bin distribution and "pollute" failure bins with noisy debug data.

We also observe that the proposed method tends to form slightly larger clusters and is characterized by smaller absolute cluster error. The latter implies a better estimation on the actual number of errors present in the design. Finally, the performance of log clustering is poor, as it is seen in the second column. This mainly reveals the shortcomings of exclusively relying on the error logs, and it also exposes the non-trivial nature of our regression data sets.

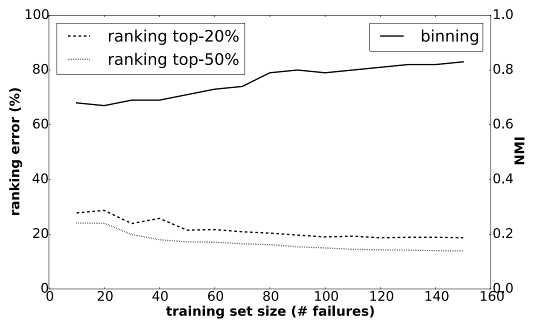


Fig. 8. Ranking and binning accuracy vs. training set size

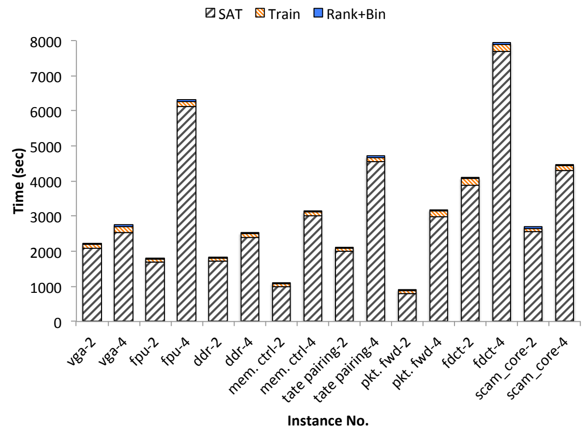


Fig. 9. Triage time consumption break-down

Finally Figure 8 shows how suspect ranking and failure binning accuracy is affected when we change the training set size. We report the average top-20% and top-50% ranking error as in Table III. Suspect ranking is relatively unstable initially, but quickly improves when more failures are added. This is expected for a learning algorithm. For training sets that include 60 or more failures (approximately half of the test set size) then the engine stabilizes to the accuracy levels shown in Table III. Similar conclusions can be drawn for failure binning, although this stage is more robust for small training sets. Failure vectors already include important information encoded by non-zero values in their dimensions when suspects are present into suspect sets. This can produce good baseline clusterings before the SVM scores converge.

D. Quality of Bin Distribution

A direct way of assessing the quality of bin distribution is to identify, for each cluster, whether fixing the root-cause of the exemplar failure eliminates the majority of failures in the same cluster. We report the percentage of failures resolved by fixing the exemplar failure, on a cluster by cluster basis, taking the average per regression run. The method in [9] decides distribution by ranking suspects and assigning the cluster based on the highest-ranked suspect. To compare against our method, we take the same measurements, but assume that the fix happens on that top suspect. Results are shown in Figure 7. It can be seen that the proposed distribution process outperforms the one in [9], with an average of 80.6% failures resolved vs. 63.1% failures resolved across all regression sessions. The effectiveness of this stage suffers the most from false positives (low PPV in Table IV). Our method is, however, significantly more resilient, with the only exceptions being `ddr-4` and `tate pairing-4`. In these cases, the AP algorithm is not as effective in identifying the proper exemplars.

E. Runtime Evaluation

Finally, we measure time consumption for the triage engine and present a break-down of the total time based on the three major steps: the baseline SAT-based debugging session, the SVM training stage, and the suspect ranking phase along with the failure binning/distribution pass. Results are shown in Figure 9. SAT-based debugging is an exhaustive search process and, as expected, dominates the total time (93.8% avg.), with training following (5% avg.). Ranking and binning/distribution together account for only 1.2% of the total time. Since the scope of this work refers to formal verification flows, it should be clarified that the baseline SAT-based debugging session is always performed whether triage is present or not. As such, the proposed triage framework does not incur the time overhead related to this task. It simply requires that it is moved one step higher in the flow, as part of coarse-grain debugging.

VI. Conclusion

In summary, this work introduces a novel automated framework for the growing problem of failure triage in regression debugging. The proposed method harnesses the power of formal debugging engines and combines it with well-known machine learning techniques to achieve high quality results for the major stages of a modern triage process. Because of the way it is constructed, this engine can benefit from future advances in formal tools, coverage and simulation tools, and potentially from the use of additional statistical measures that a regression suite can offer. Finally, it offers fertile ground for exploring novel approaches to intelligently collect relevant training data during the evolution of a design.

References

[1] H. D. Foster, "Trends in functional verification: A 2014 industry study," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[2] O. Sarbishei, M. Tabandeh, B. Alizadeh, and M. Fujita, "A formal approach for debugging arithmetic circuits," in *IEEE Transactions on CAD*, vol. 28, no. 5, May 2009, pp. 742–754.

[3] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Transactions on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[4] S. Mirzaeian, F. Zheng, and K. Cheng, "Rtl error diagnosis using a word-level sat-solver," in *International Test Conference*, 2008, pp. 1–8.

[5] K. Chang, I. Wagner, V. Bertacco, and I. L. Markov, "Automatic error diagnosis and correction for rtl designs," in *Proc. International High Level Design Validation and Test Workshop (HLDVT) 2007*, pp. 65–72.

[6] H.Foster, "From volume to velocity: The transforming landscape in function verification." in *Design Verification Conf.*, 2011.

[7] S.Safarpour, B.Keng, Y.S.Yang, and E.Qin, "Failure triage: The neglected debugging problem," in *Design and Verification Conference*, 2012.

[8] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, pp. 972–976, 2007.

[9] Z. Poulos, Y. Yang, and A. Veneris, "Simulation and satisifiability guided counter-example triage for rtl design debugging," in *Int'l Symposium on Quality Electronic Design*, 2014, pp. 394–399.

[10] B. Keng and A. Veneris, "Path directed abstraction and refinement in sat-based design debugging," in *Design Automation Conf.*, 2012.

[11] M. Farkash, B. Hickerson, and B. Samynathan, "Data mining diagnostics and bug mris for hw bug localization," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. EDA Consortium, 2015, pp. 79–84.

[12] B. Mammo, M. Furia, V. Bertacco, S. Mahlke, and D. S. Khudia, "Bugmd: Automatic mismatch diagnosis for bug triaging," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. ACM, 2016, pp. 117:1–117:7.

[13] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: Automatic assertion generation using data mining and static analysis," in *Design, Automation and Test in Europe*, 2010, pp. 626–629.

[14] R. Angell, B. Oztalay, and A. DeOrio, "A topological approach to hardware bug triage," in *Microprocessor Test and Verification*, 2015.

[15] S. Safarpour, A. Veneris, and F. Najm, "Managing verification error traces with bounded model debugging," in *ASP Design Automation Conf.*, 2010.

[16] R. A. Fisher and L. H. C. Tippett, "Limiting forms of the frequency distribution of the largest or smallest member of a sample," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 24, pp. 180–190, 4 1928.

[17] B. Gnedenko, "Sur la distribution limite du terme maximum d'une serie aleatoire," *Annals of Mathematics*, vol. 44, no. 3, pp. 423–453, 1943.

[18] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[19] OpenCores.org, "http://www.opencores.org," 2007.

[20] R. Jennrich and R. Moore, "Maximum likelihood estimation by means of nonlinear least squares," *ETS Research Bulletin Series*, vol. i, no. 36, 1975.

[21] M. Meila, "Comparing clusteringsan information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873 – 895, 2007.

[22] S.-Y. Huang, "A fading algorithm for sequential fault diagnosis," in *DFT '04: Proceedings of the Defect and Fault Tolerance in VLSI Systems, 19th IEEE International Symposium*. IEEE Computer Society, 2004, pp. 139–147.

Zissis Poulos (S'13) received a Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 2011, and the M.A.Sc degree in Electrical and Computer Engineering from the University of Toronto in 2014. He is currently a Ph.D. candidate at the University of Toronto in the Department of Electrical and Computer Engineering. His research is on formal verification and automated design debugging of digital systems, and applications of data-mining for statistical diagnosis.

Andreas Veneris (S'96-M'99-SM'05) received a Diploma in Computer Engineering and Informatics from the University of Patras in 1991, an M.S. degree in Computer Science from the University of Southern California, Los Angeles in 1992 and a Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1998. In 1998 he was a visiting faculty at the University of Illinois until 1999 when he joined the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Toronto where today he is a Professor. His research interests include CAD for debugging, verification, synthesis and test of digital circuits/systems, and combinatorics. He has received several teaching awards and a best paper award. He is the author of one book and he holds several patents. He is a member of IEEE, ACM, AMC, AAAS, Technical Chamber of Greece, Professionals Engineers of Ontario and The Planetary Society.