Mining Simulation Metrics for Failure Triage in Regression Testing

Zissis Poulos[1], Andreas Veneris[1]
[1]Dept. of ECE, University of Toronto, Toronto, Ontario M5S 3G4, Canada.

***Abstract*—Design debugging poses a major bottleneck in modern VLSI CAD flows, consuming up to 60% of the verification cycle. The debug pain, however, worsens in regression verification flows at the pre-silicon stage where myriads of failures can be exposed. These failures need to be properly grouped and distributed among engineers for further analysis before the next regression run commences. This high-level and complex debug problem is referred to as failure triage and largely remains a manual task in the industry. In this paper, we propose an automated failure triage flow that mines information from both failing and passing tests during regression, and automatically performs a coarse-grain partitioning of the failures. The proposed framework combines formal tools and novel statistical metrics to quantify the likelihood of specific design components being the root-cause of the observed failures. These components are then used to represent failures as high-dimensional objects, which are grouped by applying data-mining clustering algorithms. Finally, the generated failure clusters are automatically prioritized and passed to the best suited engineers for detailed analysis. Experimental results show that the proposed approach groups related failures together with 90% accuracy on the average, and efficiently prioritizes the responsible design errors for 86% of the exposed failures.**

I. Introduction

Recent technical road-maps stress that regression testing has become an essential and much needed component of modern verification flows [1]. Large regression test suites are often developed to ensure that coverage goals set by the verification plan are met, especially in early design stages. While functional verification generally remains a resource and time-intensive process, the task of managing large regression suites has introduced additional challenges that jeopardize time-to-debug milestones.

One of the major problems associated with regression testing at the pre-silicon stage is that of *failure triage*. Failure triage commences once a regression run finishes with a plethora of failures exposed. Its goal is to sort these failures, and then distribute them to engineers for root-cause analysis and fixing. Particularly, failure triage is performed in two stages. The first stage, called failure binning, identifies and groups together failures that are related. That is, failures that are likely to originate from the same design error. The second stage, referred to as bin distribution, takes these failure groups and determines which ones are of high priority for debug and who is the best suited engineer to further analyze each group.

The complexity of triage is mainly due to the fact that engineering intuition regarding the relationship of the exposed failures is often limited. What further adds to is complexity is a strict requirement that the triage process needs to be highly accurate. If failures are not properly grouped and assigned to the rightful owners, confusion may occur as failures constantly circulate among developer teams, which often span different time-zones. This may lead to scenarios where time-to-debug becomes comparable to debug time itself.

To mitigate such delays in the debug flow, recent work has placed its focus in data-mining methodologies that formulate failure triage as a clustering problem [2,3]. The authors in [2] use SAT-based debugging tools to extract information from error traces [4]. Then, they employ a statistical model of expected behavior for potential error locations. This allows failures to be represented as high-dimensional objects in a metric Euclidean space, where clustering can be performed. In [3], a similar approach is followed, but the method operates only in non-metric spaces. A limitation of these methods is that they exclusively utilize failing tests, while no information regarding the behavior of various design components is extracted from *passing tests*. Further, they make the assumption that each exposed failure is the result of one or more errors in a single design component. As a result, they fall short when managing failures caused by the combined effect of multiple errors occurring in more than one design components.

To address these issues, the work presented here proposes an automated triage framework that leverages information and extracts statistical metrics from failing and passing tests alike. The information mined comes in the form of coverage, specifically signal toggling activity, for various design components during regression runs. These extracted metrics are combined with the results of SAT-based debugging, in order to develop various measures that quantify the likelihood of specific design components being the source of one or more failures. The proposed engine uses components that are potentially erroneous as features that allow failures to be represented by high-dimensional objects in a metric space. Finally, by utilizing the above representation it offers a straightforward methodology to cluster these failures, prioritize them and distribute them to engineers for detailed debugging.

Experiments on four industrial designs show that the proposed work achieves 90% average accuracy for failure binning and efficiently prioritizes the responsible design errors for 86% of the exposed failures.

The remainder of this paper is organized as follows. Section II discusses prior work in failure triage for design debugging. Section III describes the proposed failure triage methodology. Finally, Section IV discusses experiments and Section V concludes the paper.

II. Preliminaries and Prior Art

Consider an erroneous design with a single or multiple errors in the RTL that undergoes regression testing. We say that a failure occurs, when a mismatch between the expected "golden" value(s) (0,1 or X for unknown) and the observed one(s) is identified at some observation point (primary output,

probed internal signal or the output of an assertion). Suppose that regression testing exposes N design failures, denoted $\mathbf{F} = \{F_1, F_2, \ldots, F_N\}$.

The first triage stage, failure binning, aims to produce a complete partition of the failure set $\mathbf{F}$ into K disjoint clusters. Ideally, failures that are due to the same RTL error are placed into the same cluster, and into distinct clusters otherwise. For this partitioning to be accurate, there are two key components that should be addressed. First, pairwise similarity between the exposed failures needs to be quantified, and second, an appropriate number of clusters, K, needs to be selected.

Finally, the goal of failure bin distribution is to allocate each of the K clusters to engineers that are most familiar with failures within that cluster. The key requirement for this process is to develop a robust method to identify and prioritize those design components that are possibly the root-cause of all failures in a particular cluster. This way, the cluster can be passed to the engineer responsible for those high-priority design components.

A. Data Collection Methods

In failure triage, the quality of the required metrics that correlate and prioritize failures heavily relies on the availability and type of data that a triage tool collects from regression runs. Especially concerning failure similarities, recent work in [2] has shown that data collected from SAT-based debuggers and logic simulators can generate proper failure similarities. In what follows we discuss the basic information that can be collected by these tools and that are also used in the triage flow presented here.

1) SAT-based Debugging: SAT-based debugging forms a powerful method for diagnosis, as it drastically reduces the number of design components that ought to be analyzed to track down the root-cause of a failure [4]–[7]. The input to a SAT-based debugger is an error trace ET_i (sequence of stimuli of length $|ET_i|$ cycles) that exposes failure F_i during regression testing. For each failure F_i, the automated debugger outputs a set of design components (RTL blocks or signals), denoted $S_i = \{s_1, s_2, \ldots, s_{|S_i|}\}$. Components $s_1, s_2, \ldots, s_{|S_i|}$ and the set S_i are commonly referred to as *suspects* and *suspect set*, respectively. These suspects constitute all possible design locations that can be responsible for the observed failure, or in other words, that can be rectified to fix failure F_i for the particular error trace ET_i. Due to its exhaustive nature, SAT-based debugging guarantees that the design location responsible for some failure F_i is included in suspect set S_i. As such, S_i can be viewed as a "signature" that characterizes failure F_i.

Additionally, SAT-based debuggers can effectively address scenarios where a failure is caused by the combined effect of multiple errors in the RTL. A parameter, referred to as *error cardinality*, and denoted by E, determines the number of design components that can be simultaneously rectified to fix the observed failure. In those cases, each suspect is considered to be a tuple of E components, commonly referred to as *block suspects*. Specifically, for failure F_i and error cardinality E, the suspect set is denoted as $S_i^{(E)}$ and each suspect $s_j \in S_i^{(E)}$ is returned as a tuple of block suspects $s_j = \{b_1, b_2, \ldots, b_E\}$. In this case we say that suspect blocks $b_1, b_2, \ldots, b_E$ can be simultaneously rectified to fix failure F_i. Note that, for $E = 1$, each suspect s_i corresponds to a single suspect block.

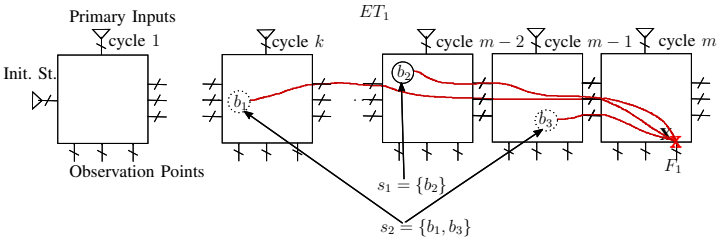

Fig. 1. Error trace and suspect components

Finally, modern SAT engines allow debuggers to return the exact cycle where an error is excited at some block suspect to cause a failure [8]. That is, for each suspect set S_i and error cardinality E, there is an associated excitation set $T_i = \{t_1, t_2, \ldots, t_{|S_i|}\}$, with $t_j = \{t_{b_1}, t_{b_2}, \ldots, t_{b_E}\}$, where t_{b_k} is the excitation cycle for suspect block $b_k \in s_j$. As an additional benefit, these tools return *error propagation paths* in the circuit that show how an erroneous value propagates from a block through consecutive cycles to reach the failing output [8].

Example 1: *To demonstrate the above concepts consider an error trace, as depicted in Figure 1. In that figure we show the sequential behavior of the circuit for that trace using its Iterative Logic Array (ILA) representation [4]. In more detail, an error at design block b_2 is excited in cycle $m-2$ and propagates to cause a failure (F_1) at an observation point in cycle m. The generated error trace ET_1 of length $|ET_1| = m$ is then passed to an automated debugger. The result is a suspect set $\mathcal{S}_1 = \{s_1, s_2\}$ of design components that can explain the wrong output. Suspect $s_2 = \{b_1, b_3\}$ is a tuple of two block suspects found when debugging is performed with cardinality parameter $E = 2$, while suspect $s_1 = \{b_2\}$ involves a single block suspect (E=1). Block suspects b_1, b_2 and b_3, excited in cycles k, $m-2$ and $m-1$ respectively, along with their propagation paths are illustrated in Fig. 1. Note that the erroneous component is included in the set S_1 as suspect $s_1 = \{b_2\}$. In this example, suspect $s_2 = \{b_1, b_3\}$ can explain the failure through the propagation of errors from blocks b_1 and b_3, but does not include the actual error location.*

2) Simulation Metrics: As it is shown in [3] and [2], coverage metrics that are extracted by regression simulation logs can be successfully employed to determine which suspect components should be treated with higher priority. Broadly speaking, knowing whether a suspect component was rigorously exercised or not, provides a measure of how reliably it can be actually considered erroneous or error-free. In this work we focus on *toggle coverage* as a measure that provides such information.

Definition 1: *Given an error trace ET_i, error cardinality E, suspect set S_i and a suspect $s_j = \{b_1, b_2, \ldots, b_E\}$, with $s_j \in S_i$, we define the* toggling frequency $f_j^i(b_k)$ *of suspect block $b_k \in s_j$ with respect to error trace ET_i, to be the average toggling across all the input(s) of b_k as it is measured in error trace ET_i.*

More precisely, the toggling frequency for each suspect can be measured within error trace windows that span specific simulation cycles of interest. In fact, the work in [2] has shown that it is reasonable to measure toggling frequencies between consecutive excitation cycles for each suspect block. In this paper, we measure frequencies in the exact same manner.

III. Proposed Triage Flow

The factor that has the greatest impact on failure triage quality is the accuracy of the failure binning step. To generate an accurate clustering of the exposed failure set, there are three major steps required. Namely, data collection, data weighting and the generation of pairwise failure similarities. In what follows, we provide the details of our methodology for each of the aforementioned tasks.

A. Data Collection

Before failure binning commences, it is necessary to collect all the relevant information for each failure exposed by regression. To this end, we follow the standard approach of performing SAT-based debugging for each failure F_i and corresponding error trace ET_i. In our methodology, SAT-based debugging for each failure F_i is performed iteratively with the error cardinality parameter E selected from the set $\{1, \ldots, E_{max}\}$, where E_{max} is determined by the engineer. At the end of each iteration with cardinality E we collect suspect set $S_i^{(E)}$, and once all the iterations are completed we form a set that includes all the returned suspects across all iterations. This set is denoted as $\mathbf{S_i}$ and includes suspect components that range from single block suspects ($E = 1$) to tuples of E_{max} block suspects.

$$\mathbf{S_i} = \bigcup_{E=1}^{E_{max}} S_i^{(E)} \tag{1}$$

Further, for each suspect block we also maintain the list of excitation cycles along with its error propagation paths, as these have been defined in Section II.

Once all debug sessions are completed and all suspects collected into set $\mathbf{S_i}$, then toggle frequencies for each of these blocks are extracted from simulation logs. Unlike the work in [2], which only collects this information from error traces, we also measure the frequencies of these block suspects from traces of passing tests in the regression suite. Specifically, if at the end of the regression run there are N_p traces denoted $P_1, P_2, \ldots, P_{N_p}$ that expose no failures, then those are also parsed to collect frequencies for those blocks that are returned as suspects by the debugging sessions performed earlier. For these frequencies, we follow the definition and notation given below:

Definition 2: *For each suspect $s_j = \{b_1, b_2, \ldots, b_E\}$, with $s_j \in \mathbf{S_i}$, we define the* passing toggling frequency $\hat{f}_j^i(b_k)$ *of suspect block $b_k \in s_j$, to be the average toggling across all the input(s) of b_k as it is measured in all passing traces $P_1, P_2, \ldots, P_{N_p}$.*

B. Data Weighting

As discussed in Section II, each suspect component provides some guidance to the general error location related to each failure. However, some of the suspect locations (i.e. reset signals, primary inputs, dangling logic, bit-flips etc.) can explain the failure but may be irrelevant to the erroneous module or signal responsible for it. Thus, all collected suspect components need to be appropriately weighted to quantify the likelihood of being an actual design error.

Ideally, we need to identify and promote suspects that exhibit behavior similar to that of typical design errors. Recent work has experimentally shown that there are two properties often observed in such suspect components. The first is temporal proximity to the observed failure [9]. That is, typical design errors are expected to be excited only a few cycles before the failure is observed, since they can quickly propagate to observation points in most cases. Second, these locations are expected to exhibit low toggling frequency measured between consecutive excitation cycles [2]. The argument behind the latter is that typical RTL errors are relatively "easy" to excite in the majority of cases.

Temporal proximity of a suspect block b_j to the failing observation point can be expressed by the number of cycles between the excitation cycle t_{b_j} and the cycle where failure F_i is observed. We assume that each error trace ET_i begins at cycle 1, therefore the failure is observed at cycle $|ET_i|$. Hence, for suspect block b_j, temporal proximity with respect to F_i is quantified by computing $|ET_i| - t_{b_j}$.

Regarding the toggling frequency of block b_j with respect to error trace ET_i, this has to be compared against all other frequencies of suspect blocks that are returned as suspects for failure F_i. Thus, when we say that a suspect block b_j has a low toggling frequency, this is relative to the block of maximum frequency that explains the same failure as b_j.

Once toggling frequency and temporal proximity are calculated for every block suspect $b_j \in s_i$, where $s_i \in \mathbf{S_k}$, a weight, denoted as $l_i^k(b_j)$, quantifies the likelihood of that block suspect being the responsible error for failure F_k. The weight $l_i^k(b_j)$ is given as follows:

$$l_i^k(b_j) = \frac{1}{2}\Big[\Big(1 - \frac{|ET_i| - t_{b_j}}{|ET_i|}\Big) + \Big(1 - \frac{f_i^k(b_j)}{\max\limits_{s_m \in \mathbf{S_k}, b_n \in s_m} f_m^k(b_n)}\Big)\Big] \tag{2}$$

In Eq. 2, the first term promotes the likelihood (weight) of suspect block b_j with respect to failure F_k when its excitation cycle is observed close to the cycle where the failure is exposed, and penalizes it otherwise. On the other hand, the second term penalizes high frequencies, thus reducing the weight of suspects that are hard to excite. Note that in the second term, the denominator $\max\limits_{s_m \in \mathbf{S_k}, b_n \in s_m} f_m^k(b_n)$ is used to normalize over the maximum toggling frequency observed in any other suspect block for failure F_k. The overall weight is the average of both quantities.

Note that, both of the above criteria apply only to information offered by error traces. In our methodology, we wish to exploit information from passing traces as well. Intuitively, we claim that if a block suspect has, on average, a high toggle frequency across passing traces $P_1, P_2, \ldots, P_{N_p}$ during regression, then its likelihood of being an actual design error should be lower. This is because, although the block is rigorously exercised by these passing traces, it did not lead to a failure (since passing traces do not expose any). For that purpose, we maintain an additional weight (likelihood) denoted $p_i^k(b_j)$, for each block suspect $b_j \in s_i$, where $s_i \in \mathbf{S_k}$. The weight is given by:

$$p_i^k(b_j) = \frac{1}{N_p} \sum_{n=1}^{n=N_p} \frac{\hat{f}_i^n(b_j)}{\max\limits_{s_m \in \mathbf{S_k}, b_w \in s_m} \hat{f}_m^n(b_w)} \tag{3}$$

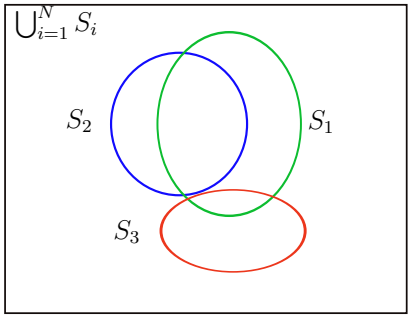

Fig. 2. Suspect set overlap

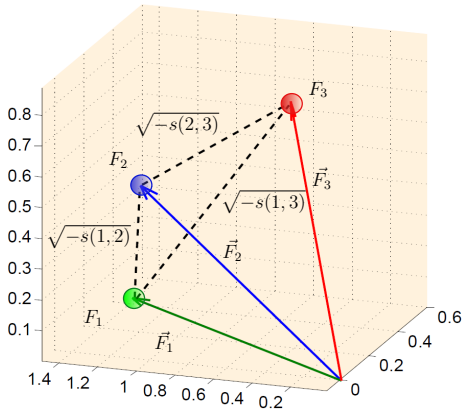

Fig. 3. Failure representations

Note that in Eq. 3, frequencies are again normalized over the maximum toggling frequency.

When both likelihoods $l_i^k(b_j)$ and $p_i^k(b_j)$ are calculated for block b_j, then the overall likelihood for block b_j, denoted $w_i^k(b_j)$ is given as the following product:

$$w_i^k(b_j) = l_i^k(b_j) \times p_i^k(b_j) \tag{4}$$

Eq. 4 is given rise by our requirement that suspect b_j satisfies the criteria with respect to error traces and passing traces at the same time. Moreover, in our method we assume that these two criteria are independent.

C. Pairwise Failure Similarities

As discussed in Section II, failure binning is defined as a complete disjoint partition of failures $F_1, F_2, \ldots, F_N$, which can be naturally formulated as a clustering problem, since failures constitute unlabeled objects. In order to form clusters of related objects, the similarity between each pair of failures needs to be determined.

To this end, we construct a feature-based representation for each failure. Then we map failures to data points into a high-dimensional Euclidean space where pairwise failure similarity can be naturally expressed by the negative squared Euclidean distance between the corresponding data points [10].

When constructing a feature-based failure representation our goal is to express each failure F_i in terms of its suspect components and their weights.

Suppose $\{s_1, s_2, \ldots, s_M\}$ is the set of all distinct suspect components in $\bigcup_{k=1}^{N} \mathbf{S_k}$. Since a suspect component s_i may involve more than one block suspects, its weight with respect to failure F_k, denoted w_i^k, is given by:

$$w_i^k = \prod_{b_j \in s_i} w_i^k(b_j) \tag{5}$$

To compute the above weight, again we follow the rule of product, since suspect s_i has a high likelihood of corresponding to actual multiple design errors if all of its block suspects individually have a high likelihood of being actual design errors.

Once these weights are computed, we associate each failure F_k with a feature vector, denoted as $\vec{F_k} = [x_1^k, x_2^k, \ldots, x_M^k]$, where:

$$x_i^k = \begin{cases} w_i^k & , s_i \in \mathbf{S_k} \\ 0 & , s_i \notin \mathbf{S_k} \end{cases} \tag{6}$$

is a variable obtaining the weight of suspect s_i with respect to failure F_k or a value of 0 if s_i does not appear in the suspect set of failure F_k. With this model, each feature encodes the existence (or absence) of specific suspect components and their corresponding significance, which is computed through the weighting scheme.

By using the above representation, each failure F_i can be mapped to an individual data point into an M-dimensional Euclidean space. Then, the similarity between two failures F_i and F_j, denoted $s(i,j)$, can be expressed as the negative squared error (Euclidean distance) between vectors $\vec{F_i}$ and $\vec{F_j}$:

$$s(i,j) = -||\vec{F_i} - \vec{F_j}||^2 \tag{7}$$

Generally, we expect a small distance to indicate large similarity and vice versa. Intuitively, failures that share many suspects in common with similar weights are expected to appear close to each other, thus having a large similarity. The absence of shared suspects between two failures and/or large variations in suspect weights indicate a smaller similarity and these failures are mapped to data points that are relatively distant.

Figure 2 illustrates a hypothetical example of failures F_1, F_2, F_3 with corresponding suspect sets S_1, S_2 and S_3 that overlap. Failures, such as F_1 and F_2, that have suspect sets with proportionally large overlap (many shared suspects) are expected to be strongly related and vice versa. In our work the contribution of the overlap is refined based on suspect weights and is implicitly represented into a metric space, as shown in Figure 3. Note that pairwise similarities $s(i,j)$ are non-positive real values. In both cases, the larger $s(i,j)$ is, the stronger the relation between F_i and F_j is considered to be.

D. Failure Binning and Bin Distribution

Once the feature-based failure representation is constructed, then clustering can be performed. The goal is to form clusters such that failure similarities (negative squared Euclidean distances) per cluster are maximized. A widely-adopted algorithm for this task is k-means clustering [10]. One of the

bottlenecks in triage, however, is that the number of clusters, K, is not known *a priori*, and is usually hard to predict. In the worst case, we need to try partitions that range from a single cluster including all failures to ones that involve N clusters, one for each of the N exposed failures. The former case can be interpreted as a scenario where all failures are caused by a single suspect component, while the latter one as a scenario were each single suspect causes a single failure. Reality however, almost always falls between those two extreme cases. A straightforward approach to output a partitioning that involves a reasonable number of clusters is to run k-means for every possible number of clusters with various seeds selected at random each time, and output the partition with the maximum silhouette average across all datapoints [10]. The silhouette measure indicates how dense the identified clusters are. Therefore, by obtaining the partition with maximum silhouette average we achieve well-separated and dense clusters.

Finally, for the bin distribution step, the proposed method decides how to pass clusters of failures to engineers based on suspect components that appear among the failures of each cluster. Particularly, for each cluster C_i from the set of clusters $C_1, C_2, \ldots, C_K$ that is generated, we compute the intersection $\bigcap_{F_j \in C_i} \mathbf{S_j}$. This set gives us all the mutual suspect components between failures in cluster C_i. Then we pass cluster C_i to the engineer who is most familiar with the suspect components of highest weight in that intersection. Highest weight components are selected because, based on the proposed weighting scheme, these are locations of greater significance and should be the ones targeted first by the engineer.

IV. 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. Four *OpenCores* [11] designs are used for the evaluation (`vga`, `fpu`, `spi` and `mem_ctrl`). The SAT-based debugger used to extract suspect locations is implemented based on [4]. A platform coded in Python is developed to parse debugging and simulation data, calculate the appropriate failure similarities and cluster the failure set via the k-means algorithm. For each design, a set of different errors is injected each time by modifying the RTL description. The injected RTL errors resemble typical human-introduced errors (missing pipeline stages, incorrect read/write from/to FIFO, bad stimulus etc.) that lead to non-trivial triage scenarios [12]. In total, twenty regression tests are run, generating various numbers of failures each time, caused by a different set of errors.

For each design, a pre-generated set of test sequences is used that is stored in vector files. Each regression run involves hundreds to thousands of input vectors. For the purpose of capturing failures we use end-to-end "golden model" checkers that compare the expected value for various operations, exception checkers and various assertions.

Table I summarizes benchmark information and statistics per regression run. From left to right, columns show the circuit name and number of gates, an enumeration for regression runs, the number of input vectors, the number of co-existing RTL errors, the number of observed failures (N), and finally the number of distinct suspect components (M) generated by SAT-based debugging per regression run. Note that for each regression run the generated failure similarity matrix is of size $N \times M$, as shown in the last column of Table I.

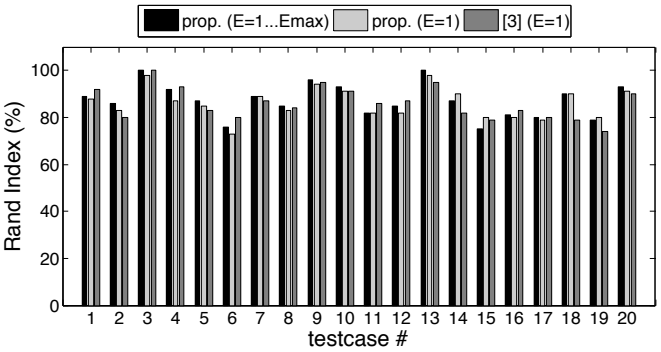

Fig. 4. Failure binning accuracy

Failure binning is performed by multiple k-means executions as described in Section III. Specifically, we run k-means approximately 50 times per chosen number of clusters, with different randomly selected seeds each time. We only show results based on the partition that gives the maximum silhouette average, which is also the output of the triage engine.

To evaluate failure binning accuracy, we use the `Rand Index` (`R.I.`) measure [10]. This metric compares the estimated clustering against a reference failure binning, with the latter corresponding to an ideal partition where all failures are grouped with 100% accuracy; that is, all failures belonging to the same cluster are caused by the same design error, and all failures in separate clusters are caused by different errors. The metric ranges from 0 to 1 and represents the fraction of correct clustering decisions. Accuracy is hence measured as $100 \times$ `R.I.`%.

Figure 4 demonstrates a comparison in terms of binning accuracy between the proposed framework when run with all possible error cardinalities selected from $1 \ldots E_{max}$ [`prop. (E=1 ... Emax)`], when run with cardinality always con-

TABLE I
Benchmarks and Regression Statistics

Ckt. (# gates)	Test No.	# vectors	# errors	$\lvert\mathbf{F}\rvert$ (N)	$\lvert\bigcup_{i=1}^{N}\mathbf{S_i}\rvert$ (M)	# matrix entries ($N \times M$)
vga (72292)	1	25206	4	45	55	2475
	2	25206	7	62	92	5704
	3	25206	8	97	104	10088
	4	31870	10	106	140	14840
	5	31870	13	121	191	23111
fpu (83303)	6	17365	3	19	39	741
	7	17365	7	30	177	5310
	8	20094	7	55	103	5665
	9	41759	9	83	101	8383
	10	41759	11	125	139	17375
spi (1724)	11	4573	3	13	50	650
	12	4573	5	28	56	1568
	13	4573	6	51	104	5304
	14	5019	8	39	226	8814
	15	5019	9	72	144	10368
mem_ctrl (46767)	16	10834	3	17	35	595
	17	10834	5	32	82	2624
	18	10834	7	31	62	1922
	19	13370	8	66	129	8514
	20	13370	11	95	169	16055

strained to $E = 1$ [`prop. (E=1)`], and the method described in [2] [[3] `(E=1)`]. Recall that the method in [2] is by construction limited to error cardinality $E = 1$. The proposed triage engine outperforms the framework in [2] in 11/20 regression runs, when executed with all cardinalities. Precisely, across all regression runs, the proposed engine achieves 90% clustering accuracy on average, compared to 85% by [2], respectively.

It is worth noting that, when the proposed flow is run with cardinality constrained to $E = 1$, then average accuracy drops to 87%. Under this constraint, the collected suspects are the same as in [2]. However, the fact that we leverage information from passing tests allows us to construct a more refined weighting scheme, something that maintains the average accuracy above the one in [2], even by a small margin. Clearly, when error cardinalities vary, then not only we are able to collect more suspect components, but also deal more effectively with scenarios where failures are caused by the combined effect of multiple errors. This advantage is captured by a further increase to accuracy as shown in Figure 4.

As far as the bin distribution step is concerned, its accuracy is evaluated based on the following method. First we identify whether the design error responsible for failures in a particular cluster is included in the intersection of suspects sets as described in Section III. If the design error location is indeed in the suspect list, we obtain its weight, which is already computed during the data weighting process. Finally, we determine if the error location has a high weight compared to other suspect locations in the intersection. This is done by sorting suspects in order of decreasing weight. Ideally, the error location resides among the top 10-20% of suspect locations in the ordered list. In that case the bin is expected to be accurately passed to the right engineer and the responsible design error is effectively prioritized.

Table II summarizes experimental results regarding the bin distribution step. Column 1 indicates the regression run number. Columns 2 to 4 respectively indicate the index of the responsible design error in the sorted list of suspects that is returned to the engineer by bin distribution in [2], the proposed flow when run with cardinality $E = 1$, and when run with all cardinalities selected from $1 \ldots E_{max}$. The indices are normalized over the size of the suspect set intersection each time. Thus, when the index is small, the suspect component that includes the design error appears in the first positions of the list, and vice versa. Each row in the table provides the average design error index per regression run. The last column shows the improvement achieved by the proposed method compared to the bin distribution approach in [2]. From Table II we observe that, on average, the proposed approach prioritizes the responsible design error more effectively compared to [2] in 16/20 regression scenarios. In total, the average improvement achieved compared to [2] is approximately 33% across all regression runs. Overall, out of the total 1,187 exposed failures across all regression runs, the responsible design errors for 1,020 of those (86%) effectively appear as the top 20% suspects in the suspect set returned to the engineer(s).

Finally, concerning time consumption, this is vastly dominated by SAT-based debugging, which, in formal debug flows, is performed whether triage takes place or not. This step consumes from approximately 800 to 7000 seconds per regression test-case. The overhead added due to failure binning and bin distribution is negligible as it is in the range of approximately 50 to 90 seconds per regression run.

TABLE II
BIN DISTRIBUTION PERFORMANCE

Test No	avg. normalized error index			improvement (%)
	[3] (E=1)	prop. (E=1)	prop. ($E = 1 \ldots E_{max}$)	
1	0.13	0.14	**0.09**	31
2	0.27	0.25	**0.11**	59
3	0.18	0.19	0.19	-5
4	0.35	0.28	**0.23**	34
5	0.18	0.15	**0.14**	22
6	0.35	0.36	**0.14**	60
7	0.17	0.20	**0.12**	29
8	0.11	0.12	0.19	-27
9	0.26	0.29	**0.20**	23
10	0.48	0.41	**0.33**	31
11	0.27	0.29	**0.26**	4
12	0.23	0.26	**0.10**	56
13	0.15	0.21	0.16	-7
14	0.26	0.19	**0.13**	50
15	0.19	0.16	**0.16**	16
16	0.30	0.33	**0.25**	17
17	0.09	0.17	0.14	-55
18	0.14	0.11	**0.08**	43
19	0.44	0.14	**0.14**	68
20	0.46	0.22	**0.17**	63
AVG	0.251	0.224	**0.167**	**33**

V. CONCLUSION

To summarize, this work introduces a novel automated framework for the problem of failure triage in regression testing. It proposes the use of information from passing tests to enhance knowledge regarding suspect components generated by SAT-based debugging, and introduces novel metrics that take into account suspects of higher cardinality. Experimental results demonstrate the high accuracy of the proposed triage engine, confirming its practicality and efficiency.

REFERENCES

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

[2] Z. Poulos and A. Veneris, "Clustering-based failure triage for rtl regression debugging," in *Int'l Test Conference*, 2014.

[3] 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.

[4] 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.

[5] 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.

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

[7] K. hui 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.

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

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

[10] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.

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

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