Proof-Guided Testing: an Experimental Study

Guillaume Lussier, Hélène Waeselynck, Karim Guennoun
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse Cedex 4 - France
Email: {glussier,waeselyn,kguennou}@laas.fr

Abstract

Proof-guided testing is intended to enhance the test design with information extracted from the argument for correctness. The target application field is the verification of fault-tolerance algorithms where a paper proof is published. Ideally, testing should be focused on the weak parts of the demonstration. The identification of weak parts proceeds by restructuring the informal discourse as a proof tree and analyzing it step by step. The approach is experimentally assessed using the example of a flawed group membership protocol (GMP). Results are quite promising: (1) compared to crude random testing, the proof-guided method allowed us to significantly improve the fault revealing power of test data; (2) the overall method also provided useful feedback on the proof and its potential flaw(s).

1 Introduction

Suppose you find in the literature an algorithm that could address your needs (under some assumptions, some properties are ensured). But the algorithm is published with an informal paper proof, and you would like to get stronger evidence for correctness. Before embarking formal development, you may consider a lighter approach: make a rapid prototype of the algorithm, test it, and gain confidence that the algorithm has a chance to be correct.

Since testing is a partial verification technique, a key issue is the selection of relevant test inputs. Ideally, you would like to take advantage of the fact that a paper proof exists. For example, a proof by cases might suggest test cases that are potentially significant to the correctness of the algorithm. Identification of the most complex and less convincing parts of the proof might suggest that some input subspaces be sampled more stringently than others. Intuitively, you would expect revealing inputs to be somehow related to weaknesses in the proof.

This paper investigates how the idea of *proof-guided testing* can be put into practice, and whether it is effective for revealing flaws. This calls for providing an analysis scheme to extract information from realistic proof examples, and for studying the relevance of extracted information. The target algorithms we consider are fundamental Fault-Tolerance (FT) algorithms, used to build dependable architectures. Their correctness is a critical concern, and most of them are published with an informal paper proof.

Preliminary work in that direction was initiated in [3]. We used an FT task scheduling algorithm as an example of a flawed algorithm "proved" by informal demonstration. The results were deceiving. Analysis of the proof quickly revealed flaws of reasoning. But the flaws were too high-level to supply useful information for testing. Our conclusion was that the viability of proof-guided testing would have to be studied on more convincing proof examples.

The group membership protocol (GMP) in [1] provides such an example. Its informal proof is much better crafted than the previous one. The authors used model-checking of an instance of the protocol to consolidate their paper demonstration for the general case. Still, the protocol was found flawed after publication, which makes it a challenging example for our purpose.

Section 2 provides an overview of our approach. It retains a principle introduced in [3]: the analysis of paper proofs is supported by a structured representation of the informal discourse which gives a clear view of the reasoning steps. Section 3 introduces the GMP example. Sections 4 and 5 illustrate the process of extracting information from its proof so as to define the test input domain, selection criteria to generate inputs from this domain, and the oracle checks determining correctness of the test outputs. Test results are shown in Section 6. In Section 7, the results obtained from both proof analysis and testing provide feedback on the original demonstration.

2 Overview of the approach

Figure 2 provides an overview of the approach investigated in the paper. It is based on four steps.

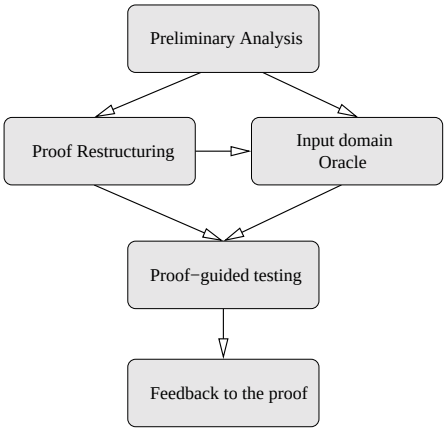


Figure 1. Proposed Approach

- A *Preliminary Analysis* aims at gaining an understanding of the FT algorithm and its requirements: under certain assumptions, some key properties are to be fulfilled. The assumptions include a model of the faults to be tolerated, as well as other environmental assumptions. From their identification, a definition of the algorithm's test input domain is derived. The key properties yield a specification of the test oracle checking acceptance or rejection of the test results. The understanding of the algorithm must be sufficient to initiate development of the prototype to be tested.
- *Proof Restructuring* is the corner stone of the approach. It consists of rewriting the informal discourse as a proof tree. The aim is to obtain a clear view of the proof structure, by interpreting reasoning steps in terms of inference rules of a basic calculus, e.g. *sequent calculus*. The tree reformulation is a useful guide to question the soundness of the proof in a systematic manner, step by step, and to assign a subjective assessment to the various proof branches. Note that the analysis approach is much lighter than formal reworking, and would not allow us to establish the correctness of the algorithm. Still, the approach should be sufficient to make a quick assessment of the proof, and to point out its less convincing parts. The identified weaknesses are used to define test criteria guiding the selection of input sequences. They may also be used to specify additional oracle checks, so as to observe the validity of intermediate lemmas.
- Accordingly, *Proof-Guided Testing* experiments are conducted. We use *statistical testing*, as defined by [7], to automate the generation of input sequences. Statistical testing is a probabilistic approach for generating test data based on selection criteria. Generally speaking, it aims to compensate for the imperfect connection of common test criteria with the flaws to be revealed: the cases identified by a criterion have to be exercised several times with different random test data. As regards proof-guided testing, it seems reasonable not to expect a perfect match between doubtful proof steps and revealing inputs (the easy flaws would not have escaped the informal proof). Using a statistical testing approach, we make a weaker assumption: the information extracted from the proof is sufficiently relevant to increase the program failure probability.
- In addition to providing experimental evidence on the algorithm's behavior, the approach provides some *Feedback to the proof*. The results of proof restructuration and testing may provide hints on how to rework the original demonstration.

3 A Group Membership Protocol

In a distributed system, a group membership service allows non-faulty processors to agree on their membership and to exclude faulty ones. The studied algorithm is a variant of the membership service offered by the *Time-Triggered Protocol* (TTP) [2]. This variant, proposed in [1], was intended to minimize overhead for bandwidth-constrained networks: it requires only one acknowledgment bit per broadcast.

This GMP involves n processors attached to a broadcast bus. Execution is synchronous, with a global time t increased by one at each step. At time t, processor t mod n is the only one that can broadcast messages. This defines broadcast *slots* owned by this processor. The goal of the GMP is to maintain a consistent record of non-faulty processors. Only two types of faults are considered.

- Send fault: a processor fails to broadcast when its slot is reached.
- Receive fault: a processor fails to receive a broadcast.

Faults can be intermittent: a faulty processor may operate correctly in some steps and manifest its fault in others. Only one non-faulty processor can become faulty in any $n+1$ consecutive steps, and there are always at least two non-faulty processors in the system.

Each processor maintains a local view of the membership set, that may be updated at each step. The update rules are intended to ensure three key properties, yielding the three theorems of the paper proof.

- *Theorem 1*: The local membership sets of all non-faulty processors are always identical and contain all nonfaulty processors.
- *Theorem 2*: A faulty processor is removed from the membership sets of non-faulty processors in the step following its first broadcast while faulty.
- *Theorem 3*: A newly faulty processor will remove itself from its local membership set (and thereby diagnose itself) when the slots of at most two non-faulty processors have been passed.

Actually, it is possible for Theorem 3 to be violated. The flaw manifests itself only when there are three processors in the membership[1].

The subsequent sections describe the application of the proposed testing approach to the GMP example. A more detailed account of this work can be found in a research report [5].

4 Preliminary Analysis

The detailed description of the local update rules in [1] makes it straightforward to implement a GMP prototype. The test oracle is specified to perform three checks, corresponding to the on-line verification of the three properties. These checks require that all local membership sets are observed at each step. The definition of the test input domain is parametrized by the number *n* of processors, with $n > 2$. Valid test sequences consist of any sequence of faults satisfying the assumptions presented in Section 3 (e.g., at most $n-2$ processors can become faulty). At this stage of preliminary analysis, we were able to implement a crude random profile generating valid test sequences for systems from 3 to 20 processors, a range targeted by the Time-Triggered architecture hosting the GMP.

5 Restructuration of the Paper Proof

In [1], Theorem 1 is proved by induction on time. In order to establish the induction step, model checking of a 4-processors instance was used to conduct a repeated strengthening of the invariant until an inductive one was reached. The final invariant contains 8 conjuncts. The first step of the proof is to show that the invariant holds in the initial state (time $t = 0$). Then the authors assume the validity of the 8 conjuncts until time t and prove each conjunct for time $t + 1$. Theorem 1 follows trivially from the invariance of the first two conjuncts. The proofs of Theorem 2 and Theorem 3 are based on the assumption that the 8 conjuncts are invariant. Hence, the construction of this invariant was the foundation of the whole proof.

[1]See the corrected version of [1] downloadable at `http://www.csl.sri.com/papers/wdag97`

Quoting from [6]: " The informal proof of inductiveness of the conjoined invariants is long and arduous ". For such intricate paper proofs, careful reading is likely to be insufficient to identify the weak parts: this is the rationale for proof restructuration.

5.1 Principle

We used sequent calculus to restructure this informal proof in the form of proof trees, and analyze them step by step. A *sequent* is written in the form $\Gamma \vdash P$, where Γ is a list of hypotheses, and P is a conjecture to be proved under these hypotheses. An intuitive interpretation is that the conjunction of the hypotheses should imply P. As an example, sequent :

$$algo, fault\ model, Inv(Th1) \vdash Theorem\ 3$$

captures the fact that the authors are trying to prove Theorem 3 under three classes of hypotheses: the update rules of the local views (noted $algo$), the $faultmodel$, and the invariance of the 8 conjuncts established by the inductive proof of Theorem 1 ($Inv(Th1)$). The sequent is not formal, since notations $Theorem\ 3$, $algo$, etc., are not formally defined.

A proof is then represented as a tree of sequents: the main goal is placed at the root (bottom) of the tree, and the proof tree is constructed upwards from the root by applying legal *inference rules* like:

$$\frac{\Gamma \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A}\ \text{cut} \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee\vdash$$

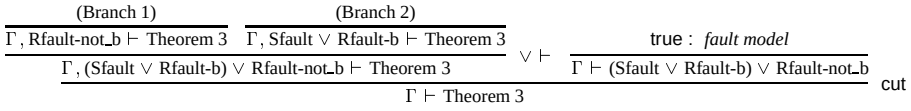
The cut rule is used to introduce new lemmas, i.e. the proof of A is split into two branches, the proof of B and the proof A assuming B. Rule $\vee\vdash$ is used for case splitting: the proof of C is split into two branches, depending on whether A, or B, is assumed.

Back to the example of Theorem 3, we have the following informal discourse:

> " If a processor p becomes send-faulty, ... Similarly, if p just became receive-faulty in the expected broadcast before its slot, ... [*argument to address both of these cases*]
> If a processor p becomes receive-faulty ... but p is not the next expected broadcaster ... [*argument to address this latter case*]. "

This is clearly the pattern of a proof by cases, which can be represented by the successive application of cut (to introduce the appropriate disjunction of cases in the hypotheses) and $\vee\vdash$ (to perform the splitting according to those cases), as shown in Figure 2.

A proof tree is complete when all its proof branches end with `axiom` rules. For example, there is an axiom stating



Proof notations :

faults	Sfault : p suffers a send fault in t_f, Rfault_b : p suffers a receive fault in t_f and is the next expected broadcaster, Rfault-not_b: p suffers a receive fault in t_f and is not the next expected broadcaster
hypotheses	$\Gamma \equiv algo, fault\ model$, Inv(Th1)

Figure 2. Root of the Proof Tree for Theorem 3

that a formula is true when the goal to be proved is one of the hypotheses. In most cases, the informal discourse is not low-level enough to explicitly refer to axioms. Hence we use labels to denote our *subjective* assessment of the validity of the terminal steps of the branches. For parts of the proof that we consider conclusive, we use label true. This label may be complemented by an indication of the hypotheses from which we think the pending goal can be easily derived: for instance, in Figure 2, the completeness of the decomposition into cases (rightmost branch) is obvious from the *fault model*. Pending proof branches that we consider missing, or too complex to be easily discharged, are labeled $\perp$. Label false is used in case we are able to establish that the current goal does not hold under the considered hypotheses.

Complete restructuration of the GMP proof did not exceed one week of work. This is due to the fact that the approach is much lighter than formal reworking. The goal is not to establish the correctness of the algorithm. Rather, it is to make a quick identification of the less convincing parts of the proof. A clear view of the proof structure is extracted from the informal discourse, by adopting a concise notation in sequent style and by making the inference steps explicit.

5.2 Problems Identified

The proof trees were a convenient support to analyze the inductive proofs of the 8 conjuncts, as well as the proofs of Theorems 2 and 3. We identified several problems.

A general problem, affecting all the proofs, is that the authors did not properly account for all possible fault patterns: in the argumentation, the occurrence of permanent or intermittent faults is ignored. For example, after a first receive fault, the faulty processor is assumed to be able to receive (or broadcast) subsequent messages if it decides to do so (except in the proof of one conjunct, where the possibility of not receiving two consecutive expected broadcasts is mentioned).

Besides this, we found additional problems affecting respectively Theorem 3 and Conjunct (5). Both of them illustrate the intricacy of the reasoning on time.

Conjunct (5) corresponds to the following property:

> " If a processor *p* became faulty less than *n* steps ago and *q* is a nonfaulty processor, either *p* is the present broadcaster or the present broadcaster is in *p*'s local membership set iff it is in *q*'s. "

This formulation is clearly not easy to handle in order to establish the induction step from t to $t+1$. In the demonstration, text fragments such as "n steps ago", "since then and up until the next step" have to be interpreted. They define time windows whose bounds depend on whether the reference step is t or $t+1$. We found problems related to the treatment of the boundary cases, and three proof branches were labeled false. This does not mean that Conjunct (5) itself is false, but its proof relies on a false lemma (used in three branches).

As regards the proof of Theorem 3, informal discourse is dense, making the logical structure difficult to extract. Part of the difficulty comes from the fact that different time steps intervene in the argumentation (four different time identifiers are distinguished in the proof tree). Also, Branch 2 aggregates two fault configurations, *Sfault* and *Rfault-b* (see Figure 2), for which the distinguished time steps may involve different properties. At the end of the assessment, Branch 1 of the proof tree is considered satisfactory, while the development of Branch 2 contains two $\perp$ labels. The first label corresponds to a terminal sequent whose goal trivially holds for an *Sfault* configuration, but not for an *Rfault-b* one. The second label corresponds to a sketchy part of the proof, too complex to be easily discharged for both *Sfault* and *Rfault-b*.

5.3 Feedback for Testing

The results obtained from proof restructuration now have to be analyzed from the perspective of testing. Three problems have been identified. Two of them may potentially affect the whole GMP proof, while the third one is localized in the proof of Theorem 3.

The first problem implies that the GMP behavior has to be tested in the case of intermittent or permanent faults.

The second problem concerns the false lemma used in the proof of Conjunct (5). The lemma appears at several

places, and it is not possible to identify safe subsets of the GMP input domain. Since the invariance of Conjunct (5) is assumed in the proofs of Theorems 1, 2 and 3, no part of the GMP proof structure can be safely considered as conclusive. All the functional cases that can be extracted from the proof trees are candidate cases to be covered during testing, as they are potentially significant to the correctness of the GMP. These cases are:

- the activation conditions of the different update rules of the algorithm ($algo$ labels appearing in the various proof trees),
- cases *Sfault* / *Rfault-b* / *Rfault-not_b* (proof tree of Theorem 3),
- transient faults not followed by any subsequent fault of the affected processor (all the proof trees),
- two consecutive receive faults (proof tree of one conjunct).

These problems mean that it is necessary to perform a global test, verifying the behavior of the GMP on a sample drawn from the whole input domain. The sampling profile has to account for the combination of transient, intermittent and permanent fault patterns with the other cases extracted from the proof analysis. To design such a profile, we built an abstract automaton capturing our understanding of the GMP behavior, and making the identified cases explicit. We ended with a decomposition of the input domain into 20 classes of fault scenarios, corresponding to classes of paths in the automaton. The classes were made equally likely under the test profile.

We also decided to strengthen the test oracle so as to check the validity of Conjunct (5). Adding this check does not require increasing observability: as for the other checks, it is sufficient that the local membership sets of the processors are observed at each step.

Concerning Theorem 3, the identified problem suggests two input subdomains to be tested more thoroughly than others: transient *Sfault* and *Rfault-b* with no subsequent fault of the affected processor. Hence, the global test will be supplemented by specific tests directed towards these subdomains.

6 Test Experiments

The test experiments were performed on a C prototype of the GMP algorithm. An overview of results is given in Table 1, for samples of 10^4 sequences. The crude random profile (derived from preliminary analysis, see Section 4) is used to study whether the proof-guided approach did allow us to enhance the revealing power of testing.

Whatever the test profile, the oracle checks for Theorem 1, Theorem 2 and Conjunct (5) never failed. All observed failures affect Theorem 3. They are due to the flaw already mentioned.

Table 1. Test Results

	Failure Rate for Theorem 3
crude random	6.15%
global	7.65%
specific *Sfault*	0%
specific *Rfault-b*	19%

The global profile, ensuring balanced coverage of the 20 classes of paths, is not much more stressing than the crude one for revealing the flaw. But it was intended to reveal flaws related to potentially invalid assumptions that Conjunct (5) holds, or to missing proof cases for intermittent or permanent faults. Its strongest result is that no such problem was found. In particular, the fact that Conjunct (5) never failed suggests that it could be valid, despite the identified weak parts of its proof.

The tests guided by the proof of Theorem 3 involves two specific profiles: one for transient send faults, and one for transient receive faults on the next expected broadcaster. The first one did not detect any failure: this tends to indicate that Theorem 3 should be valid for send faults. The second one reached an average detection rate of 19% for Theorem 3. This is three times the detection rate of the crude random testing and clearly indicates that Theorem 3 is flawed for the *Rfault-b* fault configuration. By closely analyzing the test results, we also noted that every failure was triggered by a fault injected in a 3 processor membership. These observations will be confirmed by Theorem 3 proof reworking.

7 The Proof Revisited

We will now use the test results to gain deeper understanding of the proof.

Since Conjunct (5) never failed during testing, we should be able to redo the proof and discharge the pending weak parts. In the proof, the authors use properties over time windows whose bounds depend on whether the reference step is t or $t + 1$. The weaknesses we found are related to the treatment of the boundary cases (see Section 5.2). By treating separately those boundary cases, we managed to hand-prove the relevant properties and so finish the proof of Conjunct (5).

Our tests specifically directed towards Theorem 3 proof never reported a failure with the *Sfault* configuration. We therefore decided to split the proof of Branch 2 of Theorem 3 into two sub-proofs, one for the *Sfault* configuration and one for the *Rfault-b* configuration.

For an *Sfault* configuration, we were able to discharge the

two $\perp$ labels (see section 5.2), and finish the proof.
For the *Rfault-b* configuration, we were able to hand-prove the first $\perp$ part of Branch 2. However, it was much more arduous to work on the second $\perp$ part. In the corresponding proof tree development, we had to use Conjunct (5), and so had to prove that its precondition was verified: " *p* became faulty less than *n* steps ago ". This precondition is false when only 3 nonfaulty processors remain in the group.

Using the test results, we were able to precisely locate the flaw in the proof of Theorem 3.

8 Summary and Conclusion

Functional testing approaches usually rely on coverage measures, test purposes, or selection hypotheses associated with models of behavior. Such criteria are used to select finite test sets from the models. They always involve assumptions. For example, transition coverage assumes that flaws manifest themselves as simple output or transfer errors. Test purposes represent pieces of behavior that are deemed important to be tested. Uniformity hypotheses are used to group inputs that should be equivalent in their capability of stimulating the system under test. In this paper, we investigated whether an informal argument for correctness can be a useful basis for deriving such assumptions.

Proof-guided testing was effective in the GMP example. Its proof was more convincing than the one we studied in previous work [3]. During restructuration, we were not stopped by imprecise definitions and high-level flaws. As a result, we were able to point at several weaknesses in the proof, one of them being linked to the flaw in the algorithm. This encouraging result suggests that an informal proof may carry relevant information for testing, provided it passes the restructuration step.

Note that, in the GMP example, the extracted information was not sufficient to *a priori* identify the precise 3-processors configuration. We claim that its identification would take very deep insight into the proof. The proposed light analysis approach, coupled to a statistical testing approach, seems more cost-effective. It allowed us to identify an input subspace to be sampled more thoroughly than others, and to obtain a failure rate that was three times the rate under the crude profile. Once the revealing configuration is found by testing, it becomes easier to revisit the proof so as to diagnose the flaw.

Indeed, an interesting result of proof-guided testing is that it supplies feedback not only on the algorithm, but also on its proof. This is useful from the perspective of a system integrator who would like to re-use the proof in the dependability case. If the algorithm is found flawed and a fix is contemplated, then the proof will have to be reworked as well. We believe that the diagnosis of the reasoning flaws, based on the proof tree and the knowledge of revealing scenarios, should facilitate this. The proof can also be improved when no algorithm's flaw is revealed, for consolidation purposes. In the GMP example, we observed no failure for Conjunct (5), and this was an encouragement to rework its pending branches. At the end of the method, if the algorithm never failed during testing, and the proof has been successfully reworked, our confidence in the algorithm's correctness has been significantly increased.

An extension of this work is now to investigate testing from formal, but partial, proofs. As previously, testing would be a means to get confidence that the algorithm is correct, or to help diagnose flaws preventing proof branches to be discharged. First experimental results along these lines have already been obtained in [4].

References

[1] S. Katz, P. Lincoln, and J. Rushby. Low-overhead time-triggered group membership. In M. Mavronicolas and P. Tsigas, editors, *11^{th} Int. Workshop on Distributed Algorithms (WDAG'97)*, pages 155–169, Saarbrücken Germany, Sept. 1997. Springer-Verlag. LNCS 1320.

[2] H. Kopetz and G. Grünsteidl. TTP – a time-triggered protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, Jan. 1994.

[3] G. Lussier and H. Waeselynck. Informal Proof Analysis Towards Testing Enhancement. In *13^{th} Int. Symposium on Software Reliability Engineering (ISSRE'02)*, pages 27–38, Annapolis, MD, USA, Nov. 2002. LAAS research report n° 01580, IEEE Computer Society.

[4] G. Lussier and H. Waeselynck. Deriving test sets from partial proofs. In *15^{th} Int. Symposium on Software Reliability Engineering (ISSRE'04)*, Saint-Malo, France, Nov. 2004. LAAS research report n° 04199. *To appear*.

[5] G. Lussier, H. Waeselynck, and K. Guennoun. Proof Guided Testing: Towards Complementarity of Verification Techniques. LAAS research report n° 03198, 2003.

[6] J. Rushby. Verification diagrams revisited: Disjunctive invariants for easy verification. In E. A. Emerson and A. P. Sistla, editors, *Computer-Aided Verification (CAV'00)*, pages 508–520, Chicago, IL, July 2000. Springer-Verlag. LNCS 1855.

[7] P. Thévenod-Fosse, H. Waeselynck, and Y. Crouzet. Software staistical testing. In H. B. Randell, J-C. Laprie and B. Littlewood, editors, *Predictably Dependable Computing Systems*, pages 253–272. Springer Verlag, 1995.