Internship Report

Mohammed Bekkouche

April 12, 2024, to April 22, 2024

1 Introduction

This report presents an overview of the work carried out during my internship at the Department of Computer Science, Sapienza University, under the supervision of Professor Enrico Tronci.

During a 10-day advanced research internship at the Department of Computer Science at Sapienza University in Italy, funded by the Higher School of Computer Science in Sidi Bel Abbès, Algeria, I conducted research to further our work, which involves combining model checking and testing techniques to enhance error localization in an imperative program.

Debugging software can be time-consuming, particularly when pinpointing faulty instructions. Accelerating fault localization can significantly reduce costs and enhance software quality. Current state-of-the-art research primarily revolves around two categories of approaches: spectrum-based and model-checking-based methods for fault localization. We have introduced a novel approach that integrates model checking and testing to enhance fault localization effectiveness. Our approach employs model checking to minimize the number of suspicious instructions identified from each failing test (counterexample), thereby reducing coverage of non-faulty statements. The fundamental concept is to narrow down failing test cases using model checking prior to spectrum-based fault localization.

The objective of the research are as follows: To conduct experiments on an approach integrating model checking and spectrum-based fault localization.

The scope of the research involves:

- Addressing the research within the context of software fault localization and improvement methodologies.
- Developing and experimenting with our approach that integrates model checking and spectrum-based fault localization.
- Conducting experiments on the TCAS benchmark from the Siemens test suite.
- Using multiple counterexamples in the experiments to evaluate the effectiveness of the proposed approach.
- Comparing the proposed approach with traditional spectrum-based fault localization approaches.
- Showcasing the importance of using multiple counterexamples compared to using a single counterexample in our approach.

2 Background

The Department of Computer Science at Sapienza University, known as "Dipartimento di Informatica" in Italian, is one of the leading institutions in Italy and Europe in the field of computer science research and education. Located in Rome, Italy, Sapienza University itself is one of the oldest and largest universities in Europe, dating back to 1303.

During my internship, I conducted experiments on our approach integrating model checking and spectrum-based fault localization on the TCAS benchmark from the Siemens test suite [7], using multiple counterexamples. It is worth noting that in our paper [2] for the CSA 2024 conference held on April 23-24, 2024, in

Algiers, Algeria (`http://www.emp.mdn.dz/events/csa/index.html`), we experimented with our approach integrating model checking and spectrum-based fault localization, but considering only one counterexample for each erroneous version in the TCAS benchmark. We compared our approach with traditional spectrum-based fault localization approaches.

I worked under the supervision of Prof. Enrico Tronci, who serves as the head of the Department of Computer Science at Sapienza University. Prof. Tronci has extensive experience in Formal Verification of Cyber-Physical Systems and Model Checking. Throughout my internship, Prof. Tronci provided me with guidance, mentorship, and support.

3 Related Work

Numerous studies have been conducted to aid in fault localization within imperative programs. We present key methodologies in model checking and testing.

Model Checking-based Fault Localization Model checking assesses program adherence to a specification by traversing states. Various techniques have been proposed, including Explain [11, 10], BugAssist [14], SNIPER [15, 16], LocFaults [4, 3], and others. Most model checking-based approaches simplify fault localization to diagnosing an inconsistent system of constraints.

Spectrum-based Fault Localization Spectrum-based approaches utilize ranking metrics to assess the suspicion levels of program elements. Several techniques have been proposed, including Tarantula [13, 12], Ochiai [1], FLCN [20], ConsilientSFL [17], and others. These approaches generally follow the same principle, with the main difference being the ranking metrics used.

4 Results

To evaluate our approach combing Model Checking and Testing, experiments were conducted using Siemens' TCAS benchmark suite, which is well-known for bug localization studies [18, 12, 9, 14, 16, 5, 21, 8, 17]. This suite simulates an aircraft collision avoidance system, consisting of 173 lines of code spread across 41 versions, each containing realistic faults. Versions v33 and v38, which exhibited array index out-of-bounds errors, were excluded from the analysis. These errors differ from postcondition violations and have not been addressed. The TCAS benchmark includes a total of 1068 test cases.

In our implementation, BugAssist [14] identifies suspicious instructions in the faulty program using multiple counterexamples. This constitutes the primary contribution of our work in the internship compared to the research presented at the CSA 2024 conference [2], which employs only a single counterexample for each erroneous TCAS version. By leveraging multiple counterexamples, BugAssist is executed for each TCAS version, utilizing each of its counterexamples to calculate suspicious instructions for every erroneous version, for every counterexample. The counterexamples for each erroneous version are obtained by assessing the equality of the output of the erroneous version with that of the correct version across all test cases (a total of 1068 test cases). If the output is not equal, the respective test case is identified as a counterexample; otherwise, the test case is considered successful.

The instructions of each erroneous version are then classified using suspicious instructions obtained from each counterexample, successful test cases and Ochiai spectral metric, assigning suspicious degrees based on their coverage. We compared ours result with those from a testing-only approach using Ochiai Tarantula, and EWSR spectral formulas. Only one faulty instruction is presented for debugging. Instructions are ranked based on suspicion degree, with tie elements addressed by considering the average position using Equation 1 [19]:

$$MID = S + \left(\frac{E-1}{2}\right) \tag{1}$$

The spectral data is used with a formula (e.g., Tarantula, Ochiai, EWSR) to quantify suspicion level for each potentially faulty element (see Table 1).

Table 1: Ochiai and Tarantula, and EWSR Spectral Formulas Along with Their Equations

Ochiai [1]	$\frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$
Tarantula [13, 12]	$\frac{\frac{e_f}{e_f + n_f}}{\frac{e_f}{e_f + n_f} + \frac{e_p}{e_p + n_p}}$
EWSR [2]	$\frac{e_f \times n_p + n_f \times e_p}{e_f \times n_p + n_f \times e_p + e_f \times e_p + n_f \times n_p} \times \frac{1}{\sqrt{(e_f + e_p) \times (n_f + n_p)}}$

Frequently, a formula is articulated using four counters, which are computed from the spectra in the following manner:

- e_f: the count of statement executions (e) in failed tests.
- e_p: the count of statement executions (e) in passed tests.
- n_f: the count of statement non-executions (n) in failed tests.
- n_p: the count of statement non-executions (n) in passed test cases.

Our approach calculates the program spectrum using both failed and successful test cases. For failed cases, covered code lines are those in the set of suspicious instructions. Spectral metrics prioritize faulty code lines, aiding developers in debugging.

Results from our TCAS benchmark experiments using multiple counterexamples are shown in Figure 1. Each cluster of bars represents individual versions. The first three bars in each cluster denote average positions of the considered faulty instruction using Ochiai, Tarantula, and EWSR spectral formulas for pure spectrum-based fault localization. The fourth bar (MC+TEST) represents the average position using our integrated approach combining model checking and testing, incorporating Ochiai spectral formula.

Across nearly all versions, Ochiai consistently outperforms Tarantula, indicating its superior accuracy in fault localization. The Ochiai spectral formula typically yields moderate to high average positions for the faulty instruction. Notable exceptions include versions v25, v36, and v41, where the average position is notably lower. The Tarantula formula exhibits similar trends to Ochiai but performs less effectively, with moderate to high average positions across most versions. Versions v25 and v36 stand out with uniformly low average positions, indicating potentially less severe faults or fewer identified suspicious code lines.

The EWSR formula demonstrates relatively higher average positions compared to Ochiai and Tarantula, suggesting a more conservative approach to fault localization. Similarly, versions v25 and v36 consistently exhibit low average positions with EWSR, aligning with the observations from Ochiai and Tarantula.

The integrated approach consistently achieves lower average positions compared to pure spectrum-based fault localization across all versions. This suggests that combining model checking and testing effectively enhances fault localization accuracy. Notable improvements are observed in versions where pure spectrum-based approaches yield high average positions, indicating that model checking contributes to enhancing spectrum-based fault localization.

We recall that in Figure 1, we presented the position of the erroneous instruction for each erroneous version in TCAS in the ranking returned by our integrated approach using multiple counterexamples. To assess the significance of this improvement compared to what was presented in the CSA 2024 paper [2], which involved using only one counterexample for each erroneous version, we calculated the position of the erroneous instruction in the ranking obtained by our approach but using only one counterexample at a time. Then, we calculated the average positions obtained for all counterexamples of each TCAS version (see bar 2 in each bar cluster in Figure 2 and the blue points in Figure 3). We compared this result with our approach considering all counterexamples for each erroneous TCAS version at once (see bar 1 in each bar cluster in Figure 2 and the red points in Figure 3). In Figure 3, we also showed the standard deviation for each erroneous version, which quantifies the amount of spread or dispersion of the values (positions of the faulty instruction in the output of our approach by considering one counterexample each time) around the average.

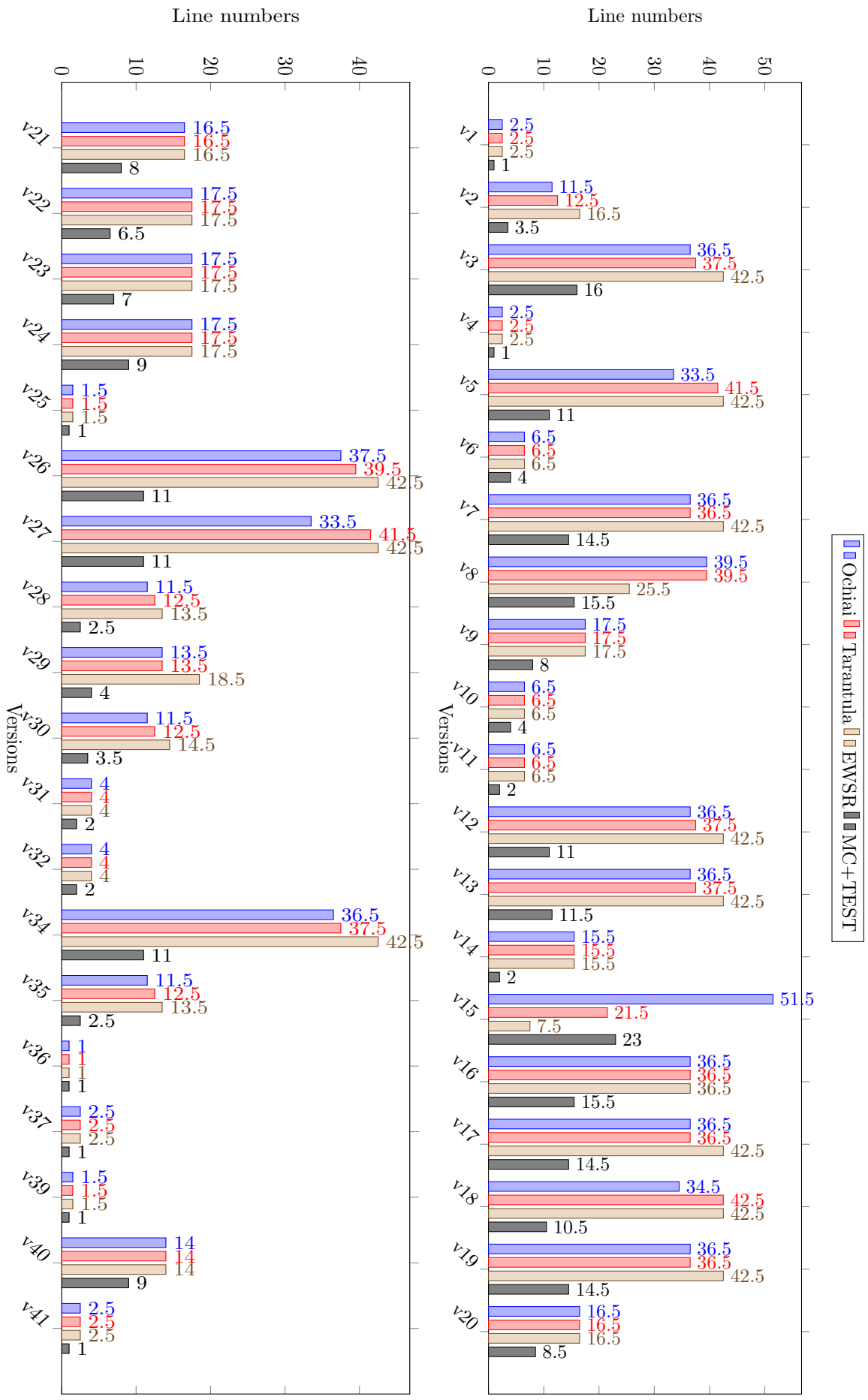

Figure 1: The Average Position of the Faulty Instruction in Each Program Version of the TCAS Suite from Siemens using Multiple Counterexamples

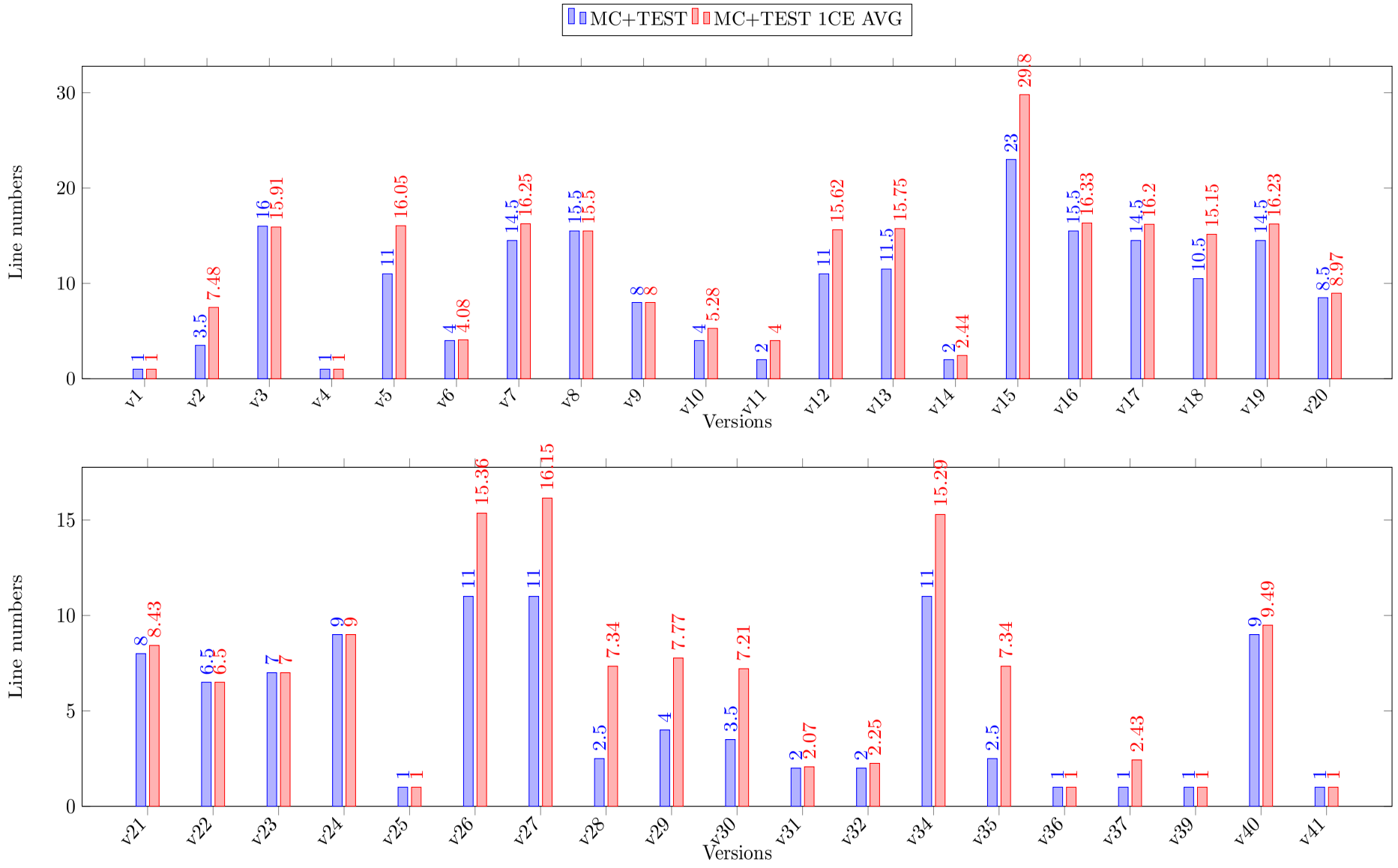

Figure 2: Average Position of Faulty Instructions in Each TCAS Suite Program Version by Siemens: Utilizing Multiple Counterexamples and Calculating the Average Position Across Each Counterexample

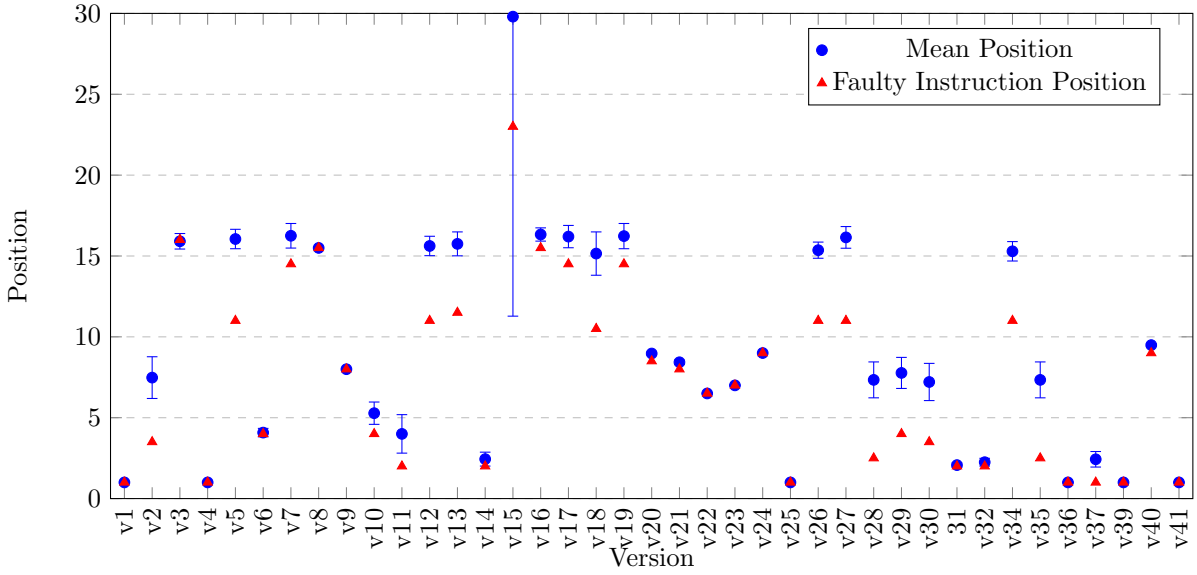

Figure 3: Plot of Positions with Error Bars (Standard Deviation)

The figures 2 and 3 show that for most erroneous versions considered in our approach, using multiple counterexamples at once yields a more accurate (inferior) ranking than considering only one counterexample at a time. Figure 3 demonstrates that the standard deviation for each erroneous version is very small (except for the case of v15) for the data representing the positions of the faulty instruction when considering one counterexample at a time. From this figure, these positions are very close to the average (except for v15) and then remain higher than the case when multiple counterexamples are used at once.

5 Enhancing program verification efficiency through parallel Model Checking

To detect errors in a computer program and verify its conformity with its specification, the CPBPV tool [6] (a Model Checker) transforms the program into a Control Flow Graph (CFG) and then explores this graph path by path in depth. At the end of each path, CPBPV checks the conformity of the constraint system corresponding to the path with the specification. If it is not conformant, the process ends and provides a counterexample. Otherwise, CPBPV proceeds to the next path. If all paths have been explored without finding any errors, then the program conforms to its specification.

Professor Enrico Tronci proposed integrating parallel computing into the CPBPV tool (Parallel Model Checking) to accelerate the program verification process. Initially, we explore the paths sequentially (without parallel computing) to enumerate the paths of the CFG. Then, we distribute the paths (constraint systems of the paths) across multiple processors, each testing the conformity of the path with the specification in parallel. If at least one processor detects errors, the process ends by providing a counterexample. Otherwise, the program conforms to its specification.

This idea should be implemented and experimented with on several benchmarks to highlight the benefits of parallel programming in model checking.

6 Internship Experience

During my internship at the Department of Computer Science, I had the opportunity to present a seminar titled "Model Checking-Enhanced Spectrum-Based Fault Localization," which showcased the innovative approach described in this report. Presenting this seminar was a highly beneficial experience, as it allowed

me to share our research findings and engage in discussions with peers and faculty members. The feedback received during the seminar helped refine our approach further and provided valuable insights for future research directions.

Moreover, my experience at the Department of Computer Science, Sapienza University, was exceptionally constructive. The department provided a stimulating academic environment, with access to cutting-edge research facilities and resources. Working alongside Professor Enrico Tronci and other researchers was both inspiring and enriching, as I had the opportunity to collaborate on various projects and learn from their expertise. The supportive atmosphere fostered collaboration and innovation, contributing to my professional growth and development as a researcher.

Overall, my internship experience at the Department of Computer Science, Sapienza University, was immensely rewarding. It not only allowed me to contribute to cutting-edge research but also provided a platform for personal and professional growth in a vibrant academic setting. I am grateful for the opportunity and look forward to applying the knowledge and skills gained during this internship to future endeavors.

7 Conclusion

In conclusion, my internship at the Department of Computer Science, Sapienza University, provided valuable insights into advanced research methodologies in computer science. Under the guidance of Professor Enrico Tronci, I conducted experiments on integrating model checking and spectrum-based fault localization on the TCAS benchmark from the Siemens test suite. This research aimed to enhance fault localization accuracy in software debugging.

Throughout the internship, our research findings, initially presented at the CSA 2024 conference in Algiers [2], Algeria, were further developed. We conducted a comprehensive comparison between our novel approach and traditional spectrum-based fault localization methods, utilizing multiple counterexamples. Our analysis revealed the remarkable effectiveness of our integrated method in achieving consistently lower average positions of the faulty instruction across all versions of the TCAS benchmark. Notably, this integration of model checking and testing exhibited promising results, especially in cases where conventional spectrum-based approaches yielded higher average positions.

We highlighted the improvement achieved by using multiple counterexamples simultaneously compared to using a single counterexample at a time in our approach. The results on the TCAS benchmark demonstrated that this enhancement leads to more accurate rankings of the faulty instruction in most cases.

Furthermore, the exploration of parallel Model Checking offers exciting prospects for enhancing program verification efficiency. Professor Enrico Tronci's proposal to integrate parallel computing into the CPBPV tool presents a compelling avenue to expedite the verification process, potentially revolutionizing software engineering practices. By leveraging parallelism to distribute path exploration across multiple processors, this approach has the potential to significantly reduce verification times, particularly for intricate systems.

The Computer Science Department at Sapienza University provided a collaborative environment where I worked with Model Checking experts. This experience enhanced my skills and theoretical understanding, fostering my growth as a researcher.

8 Acknowledgements

I would like to express my sincere gratitude to Professor Enrico Tronci for providing me with the opportunity to undertake my internship at the Department of Computer Science, Sapienza University of Rome. I am also thankful to École Supérieure en Informatique of Sidi Bel Abbès for financing this internship.

References

[1] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, pages 39–46. IEEE, 2006.

[2] Mohammed Bekkouch. Model checking-enhanced spectrum-based fault localization. In *International Conference on Computing Systems and Applications*. Springer, 2024.

[3] Mohammed Bekkouche. *Combining techniques of bounded model checking and constraint programming to aid for error localization*. PhD thesis, Université Nice Sophia Antipolis, 2015.

[4] Mohammed Bekkouche, Hélene Collavizza, and Michel Rueher. Locfaults: A new flow-driven and constraint-based error localization approach. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1773–1780, 2015.

[5] Arpit Christi, Matthew Lyle Olson, Mohammad Amin Alipour, and Alex Groce. Reduce before you localize: Delta-debugging and spectrum-based fault localization. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 184–191. IEEE, 2018.

[6] Hélene Collavizza, Michel Rueher, and Pascal Van Hentenryck. Cpbpv: a constraint-programming framework for bounded program verification. *Constraints*, 15(2):238–264, 2010.

[7] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10:405–435, 2005.

[8] Arpita Dutta, Krishna Kunal, Saksham Sahai Srivastava, Shubham Shankar, and Rajib Mall. Ftfl: A fisher's test-based approach for fault localization. *Innovations in Systems and Software Engineering*, 17(4):381–405, 2021.

[9] Andreas Griesmayer, Stefan Staber, and Roderick Bloem. Automated fault localization for c programs. *Electronic Notes in Theoretical Computer Science*, 174(4):95–111, 2007.

[10] Alex Groce, Sagar Chaki, Daniel Kroening, and Ofer Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer*, 8:229–247, 2006.

[11] Alex Groce, Daniel Kroening, and Flavio Lerda. Understanding counterexamples with explain. In *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004. Proceedings 16*, pages 453–456. Springer, 2004.

[12] James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 273–282, 2005.

[13] James A Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pages 467–477, 2002.

[14] Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. *ACM SIGPLAN Notices*, 46(6):437–446, 2011.

[15] Si-Mohamed Lamraoui and Shin Nakajima. A formula-based approach for automatic fault localization of imperative programs. In *Formal Methods and Software Engineering: 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings 16*, pages 251–266. Springer, 2014.

[16] Si-Mohamed Lamraoui and Shin Nakajima. A formula-based approach for automatic fault localization of multi-fault programs. *Journal of Information Processing*, 24(1):88–98, 2016.

[17] Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, and Babak Bagheri. Consilientsfl: using preferential voting system to generate combinatorial ranking metrics for spectrum-based fault localization. *Applied Intelligence*, 52(10):11068–11088, 2022.

[18] Manos Renieres and Steven P Reiss. Fault localization with nearest neighbor queries. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, pages 30–39. IEEE, 2003.

[19] Qusay Idrees Sarhan and Árpád Beszédes. A survey of challenges in spectrum-based software fault localization. *IEEE Access*, 10:10618–10639, 2022.

[20] Abubakar Zakari, Sai Peck Lee, and Chun Yong Chong. Simultaneous localization of software faults based on complex network theory. *IEEE Access*, 6:23990–24002, 2018.

[21] Abubakar Zakari, Sai Peck Lee, and Ibrahim Abaker Targio Hashem. A single fault localization technique based on failed test input. *Array*, 3:100008, 2019.

Dr. Mohammed Bekkouche
École Supérieure en Informatique, Sidi Bel Abbès, Algeria

Professor Enrico Tronci
Head of the Department of Computer Science, Sapienza University, Italy