Towards Explainable Compositional Reasoning

Isaac Amundson, Amer Tahat, David Hardin, and Darren Cofer
Applied Research and Technology, Collins Aerospace, USA
{first.last}@collins.com

***Abstract*—Formal verification tools such as model checkers have been around for decades. Unfortunately, despite their ability to prove that mission-critical properties are satisfied in both design and implementation, the aerospace and defense industry is still not seeing widespread adoption of these powerful technologies. Among the various reasons for slow uptake, difficulty in understanding analysis results (i.e., counterexamples) tops the list of multiple surveys. In previous work, our team developed AGREE, an assume-guarantee compositional reasoning tool for architecture models. Like many other model checkers, AGREE generates potentially large counterexamples in a tabular format containing variable values at each time step of program execution up to the property violation, which can be difficult to interpret, especially for novice formal methods users. In this paper, we present our approach for achieving *explainable* compositional reasoning using AGREE in combination with generative AI. Our preliminary results indicate this technique works surprisingly well, and have encouraged us to expand this approach to other areas in explainable proof engineering.**

I. Introduction

Formal methods provides a mathematically rigorous means of verification that one would expect for the development of high-assurance systems such as those in the aerospace and defense industries. Certification guidance has even been published on how formal methods can be used to satisfy airworthiness objectives for airborne software in commercial aircraft [1]. However, despite the effectiveness of these powerful proof techniques, their adoption into traditional development processes has been slow and uneven. Reasons for slow uptake include scalability limitations of the underlying algorithms, poorly designed user interfaces and other tool usability factors, and the need for formal training to properly use them [2].

The DARPA Pipelined Reasoning of Verifiers Enabling Robust Systems (PROVERS) program was recently launched with the goal of producing scalable and usable formal methods tools that can be integrated into traditional aerospace and defense development processes. Specifically, a key outcome of the program is that product engineers with minimal formal methods background will be able to benefit from these powerful technologies, further driving their adoption while simultaneously improving product dependability.

To address these challenges, our team is developing the Industrial-Scale Proof Engineering for Critical Trustworthy Applications (INSPECTA) framework[1]. INSPECTA consists of *ProofOps* and *BuildOps* tools and methods that integrate with current aerospace DevOps pipelines and achieve provably correct design and implementation at each level of the system hierarchy. In order to address the key objectives of PROVERS, we pay particular attention to addressing scalability and explainability concerns with respect to the proof tools in our framework.

Within the *ProofOps* workflow, INSPECTA uses the Assume-Guarantee Reasoning Environment (AGREE) [3], a formal compositional reasoning tool for Architecture Analysis and Design Language (AADL) [4] models. Compositional reasoning partitions the formal analysis of a complex system architecture into verification tasks corresponding to the architecture's decomposition. By partitioning the verification effort into proofs about each subsystem within the architecture, the analysis will scale to handle large system designs.

Although AGREE does not suffer from some of the scalability issues inherent in other formal methods frameworks due to the compositional nature of the analysis, generated counterexamples can still be difficult to understand, especially for formal methods novices when the counterexamples contain several steps, each consisting of multiple variables. This problem is not unique to AGREE, but is common to most model checkers in use today [5]. Recently, however, a novel approach to producing explainable counterexamples has emerged in the form of generative AI.

Research on applying generative AI to formal reasoning has already gained significant attention. For instance, OpenAI researchers conducted pioneering work in 2020, leveraging large language models (LLMs) for mechanical theorem proving [6]. This resulted in the development of GPT-f, a proof assistant for Metamath, which achieved a 56% success rate and proved 200 theorems [7]. Other studies have explored LLMs for proof generation and repair. First et al. achieved a 50% success rate in proof repair for Isabelle/HOL [8], using Minerva [9], a model based on Google's PaLM [10]. Research has also examined GPT-3.5 and GPT-4 for Coq theorem proving [11], primarily focusing on diagnosing failed proofs. LLMs have further been applied to discover program invariants [12], [13] and support automated reasoning, as seen in the Clover project by Stanford and VMware, which emphasizes verifiable code generation [14].

In previous work [15], [16], Tahat et al. developed a copilot for large-scale proof repair using multi-shot conversational learning. The approach achieved a 97% success rate across 58 theorems from a repository containing 20,000 lines of Coq code from the Copland proofbase. Additionally, they

[1]https://loonwerks.com/projects/inspecta.html

introduced an evaluation framework to assess the convergence of dialogues toward predefined proof sets.

In this paper, we present our current work on using generative AI to provide clear and concise explanations of counterexamples generated by AGREE. Although using generative AI for explainable formal verification has been explored in other works (e.g., [17]), to the best of our knowledge, this is the first application of applying generative AI for producing explainable counterexamples from compositional reasoning over architecture models. Our initial results indicate this approach is well-suited for providing clear explanations of root cause, as well as suggestions for addressing the contract violations.

II. Explainable AGREE

A. Overview

AGREE provides a formal contract language for specifying *assumptions* (i.e., expectations on a component's input and the environment) and *guarantees* (i.e., bounds on a component's behavior). Because AGREE is implemented as an AADL *annex* in the Open Source AADL Tool Environment (OSATE), the contracts are specified directly on components in the AADL model. AGREE then uses a k-induction model checker to prove properties about one layer of the architecture using properties allocated to subcomponents. The analysis proves correctness of (1) component interfaces, such that the output guarantees of each component must be strong enough to satisfy the input assumptions of downstream components, and (2) component implementations, such that the input assumptions of a system along with the output guarantees of its subcomponents must be strong enough to satisfy its output guarantees.

When a contract violation is found (i.e., when an assumption is determined to be invalid or a guarantee is unsupported), AGREE produces a counterexample consisting of values for each system variable at each execution step. A sample counterexample is depicted in Figure 1. Currently, OSATE includes the AADL Simulator tool that can accept an AGREE counterexample as input and walk through the trace in the graphical editor, but it is of limited help when it comes to identifying the root cause of the contract violation.

B. Making Counterexamples Actionable

We therefore desire AGREE counterexamples that are *actionable*; that is, an explanation of the violation in terms that will quickly lead to a passing analysis (e.g., by making changes to the model or formal contract). To achieve this, we implemented an interactive conversational copilot powered by GPT-4o (omni) multi-modal generative AI, specifically developed to assist AGREE users in identifying the root causes of counterexamples and to support the subsequent model repair process. It was designed to be user-friendly and integrates with the OSATE IDE (see Figure 2).

In the remainder of this section, we detail our methodology and present our key findings using the `Integer_Toy` and `Car` models included with the AGREE distribution.

Counterexample

Variables for the selected component implementation

Variable Name	0	1	2
Inputs:			
{Target Speed.val}	121	0	0
{Target Tire Pitch.val}	0	1/5	0
State:			
{[G car 1] actual speed is less than constant target speed}	true	true	false
{_TOP.AXL..ASSUME.HIST}	true	true	true
{_TOP.CNTRL..ASSUME.HIST}	true	true	true
{_TOP.SM..ASSUME.HIST}	true	true	true
{_TOP.THROT..ASSUME.HIST}	true	true	true
{const_tar_speed}	true	false	true
Outputs:			
{Actual Speed.val}	11	10	100/11
{Actual Tire Pitch.val}	0	1/5	0
{State Signal.val}	0	0	0

Variables for AXL

Variable Name	0	1	2
Inputs:			
{AXL.Speed.val}	46	46	46
{AXL.Target Tire Direction.val}	0	1/5	0
State:			
{AXL..ASSUME.HIST}	true	true	true
Outputs:			
{AXL.Actual Tire Direction.val}	0	1/5	0

Variables for CNTRL

Variable Name	0	1	2
Inputs:			
{CNTRL.Actual.val}	11	10	100/11
{CNTRL.Target.val}	121	0	0
State:			
{CNTRL..ASSUME.HIST}	true	true	true
{CNTRL.e}	110	-10	-100/11
{CNTRL.e_dot}	-110	120	-10/11
{CNTRL.e_int}	110	100	-210/11
{CNTRL.u}	110	-10	-100/11
Outputs:			
{CNTRL.Actuator_Input}	110	-10	-100/11

Variables for SM

Variable Name	0	1	2
Inputs:			
State:			
{SM..ASSUME.HIST}	true	true	true
Outputs:			
{SM.State_Out.val}	0	0	0

Variables for THROT

Variable Name	0	1	2
Inputs:			

Fig. 1: AGREE counterexample generated from the `Car` model.

C. Contextual Prompt Constraints Problem

The GPT-4o generative multi-modal model exhibits significant power in translating human instructions into code and vice versa, particularly when the language in question has been part of its pre-training data and there exists a substantial open-source code base, such as C or Python. However, this capability comes with the drawback of potential hallucinations. Since AGREE is not as widely adopted as languages like C or Python, this problem is exacerbated. Consequently, the lack of relevant context is a significant challenge for generating explainable AGREE counterexamples.

To mitigate the contextual prompt constraints problem, we implemented a dynamic Retrieval-Augmented Generation (RAG) system, allowing it to adjust its context based on user inquiries.

Despite GPT-4o's 128k token capacity, which we estimate can accommodate several thousand lines of AADL in a single prompt, uploading an entire repository's contents can be prohibitively expensive and may well exceed the prompt token limitations. We therefore implemented a practical two-

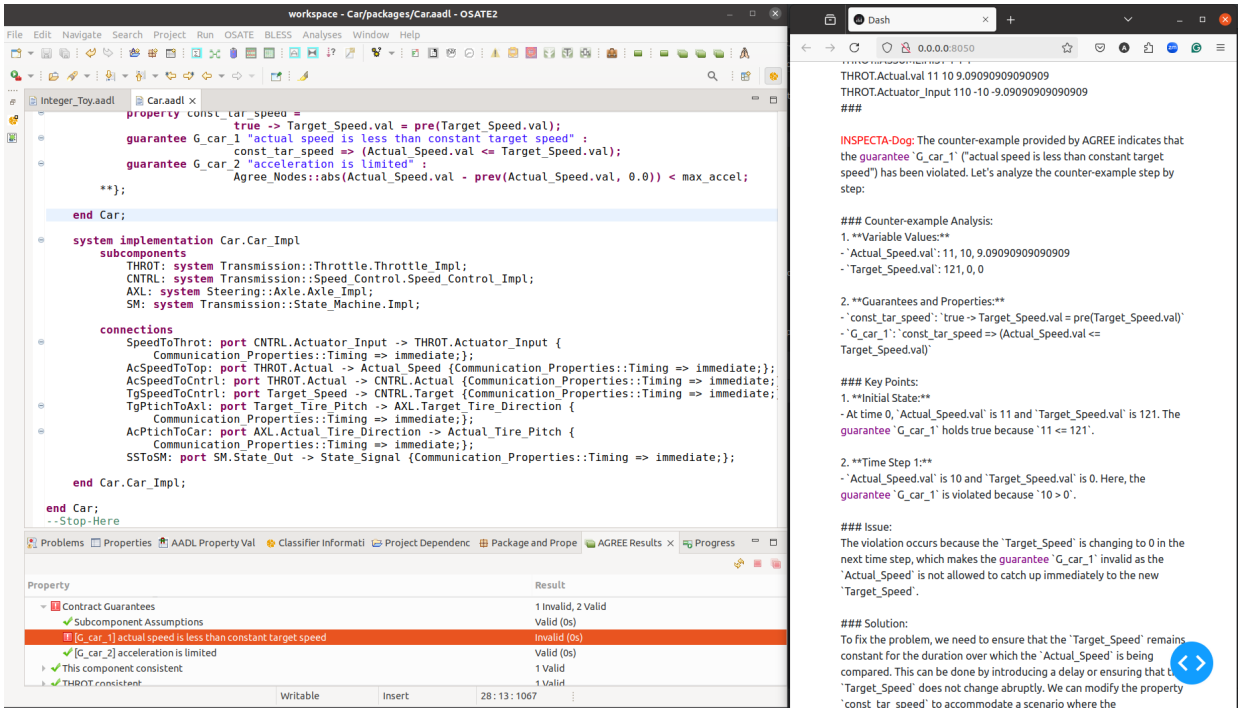

Fig. 2: AGREE copilot in OSATE provides an explanation for a counterexample generated on the `Car` model.

step optimization technique to meet our current needs.

First, the RAG system reads the top-level AADL file. It then parses the file's import chain, extracting only the files in the model workspace that are specified on this chain. This step significantly reduces the initial prompt size. The second optimization addresses another practical requirement: handling parts of the repository that may have been included in the model's pre-training data, such as core libraries. To manage this, the RAG system applies a filtering technique to the file names, guiding the system to ignore certain files, such as standard libraries, and retain user-defined files. This approach further reduces the initial prompt size to include only files that the model has not previously encountered. Finally, user inputs are automatically incorporated into the extracted context from the current file and its import chain, allowing for more accurate responses to user inquiries.

This approach significantly mitigates contextual constraint-based hallucinations that can arise from the absence of AADL model specifications. However, it does not address the absence of the counterexample itself or the lack of guidance on the critical system requirements that should be preserved during the model repair process.

D. Model Repair Problem

Given that counterexamples are generated interactively and may not be included in the initial context, we dynamically extend the RAG system. This allows users to upload an exemplar AADL/AGREE model along with a corresponding counterexample (in text or CSV format). Upon submission, the copilot provides a detailed, step-by-step explanation of the counterexample, identifies its root cause(s), and suggests potential solutions, as shown in Figure 2.

Summary:
The key issue was the mismatch between the requirement for the top-level system's output (`< 50`) and the guarantee in the model (`< 70`). By aligning the guarantee in the top-level system with the requirement, we ensure consistency and eliminate the counterexample.

Fig. 3: Refined explanation using a requirements file for the `Integer_Toy` model.

However, a significant challenge encountered was that these explanations and suggested alternatives could include two types of hallucinations, both syntactic and semantic. The former are typically minor and can be detected and resolved using a multi-shot approach. The latter are more problematic, as the copilot might suggest altering a component's guarantee, which could successfully remove the counterexample but risk violating core system requirements that should remain unchanged. We refer to this as the *Model Repair Problem*.

1) Requirements for Counterexample Explanations: To mitigate the Model Repair Problem, we configured the tool to generate solutions that conform to a predefined set of system requirements written in natural language, which are uploaded via a CSV file or directly included in the context.

As a result, the tool was able to more accurately identify the root cause and suggest appropriate solutions, as demonstrated in Figure 3. This refinement significantly enhanced the accuracy of the recommendations.

E. Preliminary Results and Conversational Quality Assessment Problem

Our initial evaluations were conducted manually, focusing on the copilot's ability to accurately identify the root cause of the counterexamples, repair the model, and ensure compliance with the requirements. The system was evaluated on two case studies. The first case study involved the `Integer_Toy` model, while the second dealt with a larger model that imports several files, totaling 7 files and approximately `380` lines of AADL. The copilot successfully identified the root cause of all counterexamples for the specified guarantees (13 out of 13) on the first attempt, demonstrating a high degree of accuracy. However, these manual evaluations highlight the need for greater automation; consequently, we plan to develop a more automated evaluation system to enable testing on more realistic and complex use cases.

We are in the process of selecting a golden set of examples from a formally verified library we developed previously, and constructing a testing set by introducing deliberate violations. These examples will be used by the copilot to evaluate its ability to correctly identify the root causes. While we have demonstrated initial success in this area, model repair remains a more complex challenge. This is because repairing models can result in multiple solutions, particularly for more intricate use cases. One of the key limitations is the tool's ability to consistently remove counterexamples while ensuring compliance with the specified requirements. To address these issues, we are developing a toolset aimed at measuring convergence towards the correct semantics of the golden examples, within a few-shot learning context. This remains an ongoing challenge that we will address in our future work.

III. Conclusion

In an effort to make AGREE results more explainable, we have developed a generative AI-based tool that produces natural language explanations from (potentially complex) AGREE counterexamples. Although initial results are encouraging, we will continue to evaluate our approach on increasingly complex models and formal specifications. In addition, we believe usability of other AGREE features can also benefit from generative AI. The most obvious candidates are the formalization of AGREE contracts from natural language requirements and the modification of models to conform to their contracts.

Our prototype implementation is currently loosely coupled with AGREE and OSATE. In order to truly address AGREE usability, tighter tool integration is required, and will be the focus of upcoming work as we continue to refine the tool. We envision an integrated copilot that smartly parses the model abstract syntax tree, interacts with the user, and makes automated updates to the AGREE contracts and AADL model by using features provided by the IDE.

INSPECTA includes a DevOps Assurance Dashboard for displaying development status, analysis results, and progress towards achieving assurance goals. The explainable counterexamples will be accessible from the dashboard, and therefore a mechanism will need to be implemented to retrieve them from the modeling workspace and display them properly. The dashboard will also capture and display tool usage metrics to help us better understand the degree to which a more explainable counterexample aids the user in addressing a requirement or design issue (e.g., by tracking the number of times the model is modified or AGREE is run before producing a passing result). Metrics analysis will in turn drive new usability enhancements in AGREE. We look forward to sharing the outcome of these efforts in the near future.

IV. Acknowledgment

This work was funded by DARPA contract FA8750-24-9-1000. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] RTCA, *DO-333: Formal Methods Supplement to DO-178C and DO-278A*, December 2011.

[2] J. A. Davis, M. Clark, D. Cofer, A. Fifarek, J. Hinchman, J. Hoffman, B. Hulbert, S. P. Miller, and L. Wagner, "Study on the barriers to the industrial adoption of formal methods," in *Formal Methods for Industrial Critical Systems*, C. Pecheur and M. Dierkes, Eds. Springer Berlin Heidelberg, 2013, pp. 63–77.

[3] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods*, A. E. Goodloe and S. Person, Eds. Springer, 2012, pp. 126–140.

[4] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, 1st ed. Addison-Wesley Professional, 2012.

[5] A. P. Kaleeswaran, A. Nordmann, T. Vogel, and L. Grunske, "A systematic literature review on counterexample explanation," *Information and Software Technology*, vol. 145, 2022.

[6] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *arXiv preprint arXiv:2009.03393*, 2020.

[7] N. Megill and D. A. Wheeler, "Metamath: A computer language for mathematical proofs," 2019.

[8] E. First, M. N. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," *arXiv preprint arXiv:2303.04910*, 2023.

[9] A. Lewkowycz, A. Andreassen, D. Dohan *et al.*, "Solving quantitative reasoning problems with language models," *arXiv preprint arXiv:2206.14858*, 2022.

[10] A. Chowdhery, S. Narang, J. Devlin *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[11] S. Zhang, E. First, and T. Ringer, "Getting more out of large language models for proofs," *arXiv preprint arXiv:2305.04369*, 2023.

[12] K. Pei, D. Bieber, K. Shi *et al.*, "Can large language models reason about program invariants?" *Proceedings of the 40th International Conference on Machine Learning*, July 2023.

[13] H. Wu, C. Barrett, and N. Narodytska, "Lemur: Integrating large language models in automated program verification," *arXiv preprint arXiv:2310.04870*, 2023.

[14] C. Sun, Y. Sheng, O. Padon, and C. Barrett, "Clover: Closed-loop verifiable code generation," *arXiv preprint arXiv:2310.17807*, 2024.

[15] A. Tahat, D. Hardin, A. Petz, and P. Alexander, "Proof repair utilizing large language models: A case study on the copland remote attestation proofbase," in *Proceedings of International Symposium On Leveraging Applications of Formal Methods Verification and Validation (AISolA)*, 2024.

[16] ——, "Metrics for large language model generated proofs in a high-assurance application domain," in *High Confidence Software and Systems Conference (HCSS'24)*, 2024.

[17] R. Martins, "Transforming logic into language: Bridging the gap with large language models," in *2nd International Workshop on Explainability of Real-time Systems and their Analysis (ERSA'23)*, December 2023.