Supporting Assurance Case Development Using Generative AI

Torin Viger, Logan Murphy, Simon Diemert, Claudio Menghi, Alessio Di, Marsha Chechik

Supporting Assurance Case Development Using Generative AI

Torin Viger[1], Logan Murphy[1], Simon Diemert[2], Claudio Menghi[3], Alessio Di Sandro[1], and Marsha Chechik[1]

[1]University of Toronto, Canada
[2]Critical Systems Labs Inc., Canada
[3]University of Bergamo, Italy

***Abstract*—As the use of cyber-physical systems in safety-critical domains continues to rise, assurance cases have become a widely adopted approach for justifying the safety of these systems. During assurance case development, errors can occur such as logical fallacies and argument incompleteness which can lead to the deployment of unsafe systems. Methods such as Eliminative Argumentation have been proposed to improve confidence in assurance cases by identifying potential doubts in the argument (called *defeaters*) and arguing that they have been appropriately mitigated; however, using these methods does not guarantee that engineers will identify all relevant defeaters. In this paper, we propose our vision for using generative artificial intelligence to aid in the identification of defeaters in assurance cases to improve their reliability.**

I. Introduction and Motivation

As cyber-physical systems become increasingly used in safety-critical domains, ensuring that these systems function correctly and safely has become essential. To mitigate the risks these systems pose, many critical domains have adopted *Assurance Cases* (ACs) as a method to justify why critical systems are sufficiently safe and reliable. ACs are structured arguments, often represented graphically as a tree using notations such as Goal Structuring Notation [5], where a high-level claim about a system is decomposed by *decomposition strategies* into more refined subclaims supported by evidence.

ACs play an essential role in ensuring that complex systems are dependable and trustworthy; however, they are also prone to many errors during their development [4]. Incomplete arguments, logical errors, and incorrect claims/evidence can cause false confidence and may result in the deployment of unsafe systems, such as the RAF Nimrod MR2 [6]. *Eliminative Argumentation* (EA) [3] is a methodology that helps mitigate these errors by explicitly identifying and reasoning about defeaters (i.e., reasons to doubt the argument); however, relevant defeaters may still be overlooked even if EA is used. Consequently, there is a need for further support to ensure that ACs are trustworthy.

Recent advances in generative artificial intelligence (GAI) are revolutionizing software engineering. Software engineers are already leveraging GAI to aid in software implementation, and efforts are currently underway to evaluate whether GAI can aid in other software engineering processes, such as modeling, testing, and verification [2]. Naturally, engineers may wonder whether GAI can support the creation of ACs.

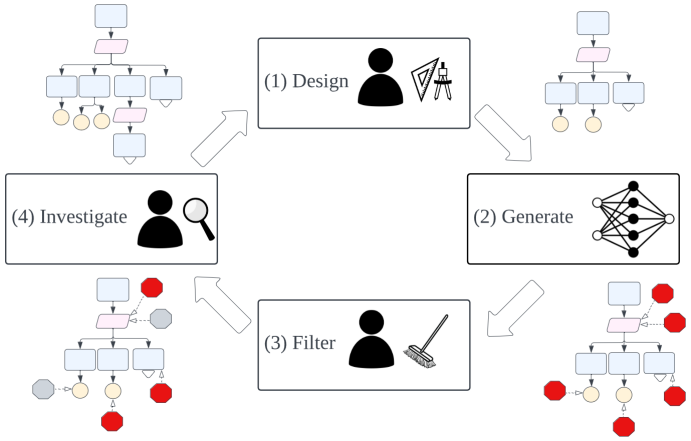

Fig. 1. Our vision for AI-enabled Eliminative Argumentation.

From an assurance perspective, GAI presents a significant challenge; most GAI models are black boxes, making it difficult to understand why their output artifacts were generated. This is particularly concerning in AC development, as a GAI *hallucination* (i.e., generation of an incorrect artifact) could create false confidence in unsafe system safety.

Using GAI to generate AC claims and strategies may be problematic for the above reasons. Instead, *we believe that GAI can be used in conjunction with existing techniques to increase confidence in ACs by helping to identify defeaters in the argument*. The risks posed by using GAI to identify defeaters are significantly lower than in generating AC claims and strategies. This is because incorrect defeaters identified by GAI (i.e., generated defeaters not corresponding to legitimate weaknesses of the argument) may create false doubts in the system, but will not create false confidence. Furthermore, GAI has already been shown as an effective tool for supporting the hazard analysis process [1].

II. Vision

Our vision for *AI-enabled Eliminative Argumentation* (AEA) is summarized in Fig. 1. The workflow consists of four stages: *Design*, *Generate*, *Filter*, and *Investigate*. In the *Design* stage, the AC is developed using established processes. In the *Generate* stage, the AC and relevant background information about the system are provided to the GAI model, and the model is prompted to generate a set of potential defeaters to the

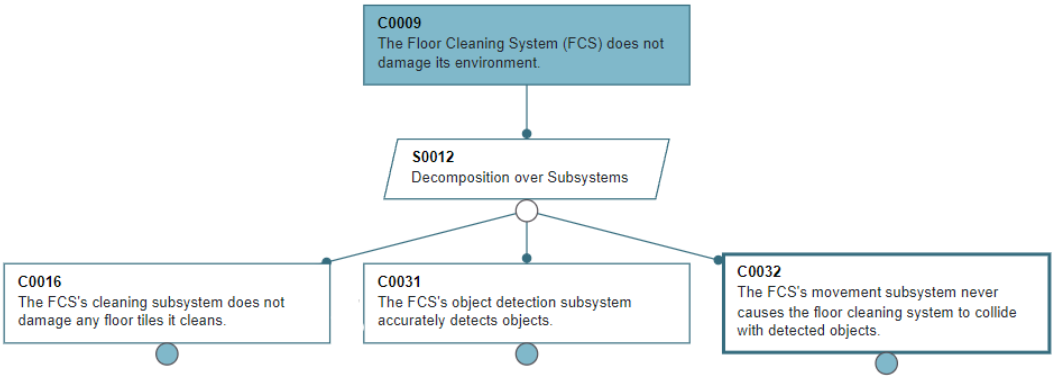

Fig. 2. A fragment of an AC for a floor cleaning system.

AC. In the *Filter* stage, the engineer attempts to identify and filter out defeaters which are inapplicable to the system/AC (i.e., defeaters that cannot correspond to legitimate weaknesses in the system because they are logically inconsistent or contain incorrect information). In the *Investigate* stage, the engineer investigates the remaining defeaters, and refines their original AC to incorporate them and show how they have been mitigated where possible.

To illustrate this workflow, consider a simple Floor Cleaning System (FCS) [9] designed to clean floor tiles without damaging its environment. The FCS consists of subsystems for cleaning, object detection and movement.

(1) Design: The engineer creates the AC fragment presented in Fig. 2, which decomposes a claim that the FCS will not damage its environment into a set of subclaims over each subsystem of the FCS.

(2) Generate: The engineer provides background information describing the FCS, its components and its operating environment to the GAI model. They then provide the argument fragment in Fig. 2 to the model, and prompt the model to identify any potential defeaters in the argument. The model generates the following defeaters:

- Defeater 1: Unless there are system integration failures that could result in unexpected interactions or behaviors that may cause damage to the environment.
- Defeater 2: Unless there are software or firmware bugs that could lead to unintended behaviors causing damage, even if individual subsystems perform as intended.
- Defeater 3: Unless there are unforeseen environmental factors that could override the system's capabilities and result in damage.
- Defeater 4: Unless there is a lack of maintenance or wear and tear that could lead to a degradation in system performance, causing damage to the environment.

(3) Filter: The engineer identifies defeater 1 as an inapplicable to the FCS, and filters out this defeater.

(4) Investigate: The engineer adds defeaters 2, 3, and 4 to their AC, having identified that they represent credible threats to the system. The engineer constructs an argument to show how defeaters 2 and 4 have been mitigated, and leaves defeater 3 in the AC as a residual unmitigated risk.

III. Evaluation Plan

We intend to assess the feasibility of AEA by empirically evaluating the effectiveness of GPT-4 [8] in proposing defeaters for ACs.

To conduct our experiment, we plan to use a baseline of two publicly available industrial ACs developed using EA. We will compare the defeaters generated by GPT-4 against our baseline set with support from domain experts for each case study. In conducting our experiment, it is possible we will find that GPT-4 provides only superficial, repetitive or predictable defeaters, or that the time required to filter irrelevant defeaters is too significant to be practically useful; however, prior work on using GAI to support hazard analysis [1] gives reason to believe that GAI may be an effective tool in creatively identifying reasonable defeaters. We may observe that AEA is able to provide meaningful and novel defeaters, demonstrating the effectiveness of GAI as a tool to support AC development and opening the door for further applications of GAI in assurance engineering. This work is a part of our broader vision to use assurance cases as data [7].

References

[1] Diemert, S., Weber, J.H.: Can large language models assist in hazard analysis? arXiv preprint arXiv:2303.15473 (2023)
[2] Fraiwan, M., Khasawneh, N.: A review of chatgpt applications in education, marketing, software engineering, and healthcare: Benefits, drawbacks, and research directions. arXiv preprint arXiv:2305.00237 (2023)
[3] Goodenough, J.B., Weinstock, C.B., Klein, A.Z.: Eliminative argumentation: A basis for arguing confidence in system properties. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (2015)
[4] Greenwell, W.S., Knight, J.C., Holloway, C.M., Pease, J.J.: A taxonomy of fallacies in system safety arguments. In: 24th International System Safety Conference (2006)
[5] GSN Working Group: GSN Community Standard Version 2, http://www.goalstructuringnotation.info/. York, UK (2011), [Accessed Jan. 28th, 2020]
[6] Haddon-Cave, C.: The Nimrod Review: An Independent Review into the Broader Issues Surrounding the Loss of the RAF Nimrod MR2 Aircraft XV230 (2009)
[7] Menghi, C., Viger, T., Di Sandro, A., Chechik, M.: Assurance case development as data: A manifesto. International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) (2023)
[8] OpenAI: Gpt-4 technical report. arXiv:2303.08774 (2023)
[9] Viger, T., Murphy, L., Di Sandro, A., Menghi, C., Shahin, R., Chechik, M.: The foremost approach to building valid model-based safety arguments. Software and Systems Modeling pp. 1–22 (2022)