PREPRINT - Accepted at the 42nd AIAA/IEEE Digital Avionics Systems Conference (DASC), 2023.

AACE: Automated Assurance Case Environment for Aerospace Certification

Zamira Daw*, Chanwook Oh†, Matthew Low†, Timothy Wang*, Isaac Amundson‡, Alessandro Pinto*, Massimiliano Chiodo*, Guoqiang Wang*, Saqib Hasan‡, Ryan Melville*, Pierluigi Nuzzo†

*Raytheon Technologies Research Center, Berkeley, CA, USA
†University of Southern California, Los Angeles, CA, USA
‡Collins Aerospace, Minneapolis, MN, USA

***Abstract*— A certification process evaluates whether the risk of a system is acceptable for its intent. Certification processes are complex and usually human-driven, requiring expert evaluators to determine software conformance to certification guidelines based on a large number of development artifacts. These processes may result in superficial, biased, and long evaluations. In this paper, we propose a computer-aided assurance framework, called Automated Assurance Case Environment (AACE), enabling synthesis and validation of assurance cases (ACs) based on a system's specification, assurance evidence, and domain expert knowledge captured in AC patterns. A commercial aerospace case study shows that the generated ACs are meaningful, and numerical results show the efficiency of AACE.**

***Index Terms*—Assurance cases, certification, synthesis, validation**

I. Introduction

Certification processes help ensure aspects of product dependability including safety, security, and functional correctness. However, typical certification efforts contribute to a considerable portion of the product cost (40% for product changes) and delays the introduction of new technologies. Therefore, there is a need for methodologies that make certification processes more agile and automated. Current methodologies and tools require a significant amount of manual effort, and do not incorporate advanced reasoning technologies to support automated validation. Certification is generally based on guidelines and standards that provide best practices to develop high-assurance systems. However, technology innovation outpaces the evolution of those standards. In case of new technologies where standards are not yet available, *assurance arguments* can help applicants present novel means of compliance in a structured manner.

Assurance cases (ACs) provide a means to make explicit an argument that dependability properties have been acceptably addressed in a system. More specifically, an assurance case is defined as a "reasoned and compelling argument, supported by a body of evidence, that a system, service, or organization will operate as intended for a defined application in a defined environment" [1].

At present, however, ACs are generally created through manual processes and limited to facilitating discussion by collecting related arguments and documenting the structure of the arguments. Difficulties in utilizing ACs for purposes beyond manual documentation partly stem from the absence of formal semantics. Existing tools for AC visualization and manipulation suffer from notational gaps, leaving room for disagreeing interpretations and misunderstandings among developers. Unfortunately, the creation of rigorous and interpretable arguments is non-trivial, and only a small number of attempts have been made toward formalisms and tools that can assist in this task [2]–[4].

The construction and validation of ACs is further complicated by the following factors:

- Increasing complexity of modern cyber-physical system designs
- Heterogeneity of supporting evidence
- Necessity to assess argument quality when faced with quantitative and qualitative sources of doubt
- Requirement to produce interpretable outcomes.

This paper addresses these challenges by presenting a framework for the automated synthesis and validation of ACs. We use a commercial aerospace industrial example as a case study, showing that certification arguments generated by AACE are understandable and plausible, and the performance of the tool can support industrial applications. Overall, AACE reduces manual validation tasks done by certification authorities using automation while maintaining a high level of assurance by using argumentation patterns. This objective is achieved by combining formal methods and domain knowledge encapsulation in a novel manner. Furthermore, the formalization of the patterns reduces ambiguity in the argument and allows future analysis regarding cost and schedule optimizations.

Distribution statement "A" (approved for public release, distribution unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA), contract FA875020C0508. The views, opinions, 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. The authors wish to also acknowledge the partial support by the National Science Foundation (NSF) under Awards 1846524 and 2139982, the Office of Naval Research (ONR) under Award N00014-20-1-2258, the Defense Advanced Research Projects Agency (DARPA) under Award HR00112010003, and the Okawa Research Grant.

II. Background

We provide an overview of assurance cases as well as assume-guarantee contracts and Bayesian networks, which are the underlying formalisms used in this paper.

A. Assurance Cases

An *assurance case* (AC) is a hierarchical argument that maps a top-level *claim* to *evidence* through strategies and intermediary claims to establish that a system satisfies the requirements in its operative environment [4], [5]. Structured language [6] or graphical notations such as Claims-Arguments-Evidence (CAE) [7] and Goal Structuring Notation (GSN) [1] are often employed to describe ACs. The AC representation is then a directed acyclic graph that maps the system specification (the top-level claim) at the root to the leaf nodes representing the supporting evidence.

B. Assume-Guarantee (A/G) Contracts

Assume-guarantee (A/G) contracts offer effective mechanisms to analyze system requirements and behaviors in a modular way [8]–[11]. By using A/G contracts as a specification formalism, we can represent both the claims about the system as well as the contexts under which the claims must hold.

An A/G contract C is a triple (V, A, G) where V is the set of variables, A (assumptions) is a specification for an environment over V, and G (guarantees) is a specification for an implementation over V, representing the set of promised behaviors given that the environment satisfies A. We omit V when this is clear from the context.

C. Bayesian Networks

We model sources of doubt in ACs using Bayesian networks (BNs), which have been successfully used in many application domains to incorporate subjective notions of probability or belief and to quantitatively reason about the confidence in assertions affected by uncertainty [12]–[14]. BNs can efficiently encompass both aleatoric and epistemic uncertainty, producing more compact models than other probabilistic reasoning frameworks for uncertainty quantification [15].

III. Formalizing Assurance Cases

We use the following running example to demonstrate the workflow of the AC generation framework (see Figure 1). For illustration purposes, the patterns are represented using elements of the GSN.

Pattern *P1* in Figure 1 establishes the sufficiency of the test cases for checking system correctness if the tests exercise all requirements and cover the entire source code. *P2* argues that a set of tests covers the entire source code if the decision coverage (DC) and modified condition/decision coverage (MCDC) metrics are satisfied. *P3* argues that a set of test cases fully exercises the requirements if the tests cover the normal and robust ranges of the requirements. Confidence in this statement can be boosted by adding evidence from falsification tools.

We formalize an AC into a *hierarchical contract net* (HCN) that specifies the logic of the argument and a *confidence network* that captures the sources of doubt of the logical argument. An HCN is a tree-like structure that connects a top-level contract, which specifies the AC claim, to a contract network that consists of all premises and supporting evidence of an AC. Formally, the connection is defined by a refinement relation between the top-level contract and the composition of contracts within the network. Details on HCNs can be found in [16]. Figure 2 shows the HCN corresponding to P3 in Figure 1. Note that the claim and the premises of P3 are transformed into guarantees in the contracts.

A confidence network captures sources of uncertainty in the premises of the AC (formalized by an HCN). A confidence network is implemented as a Bayesian network (BN), where the nodes of the BN correspond to the sources of uncertainty modeled as random variables, while the edges and the associated conditional probabilities represent the dependencies between the sources of uncertainty. For example, a method for

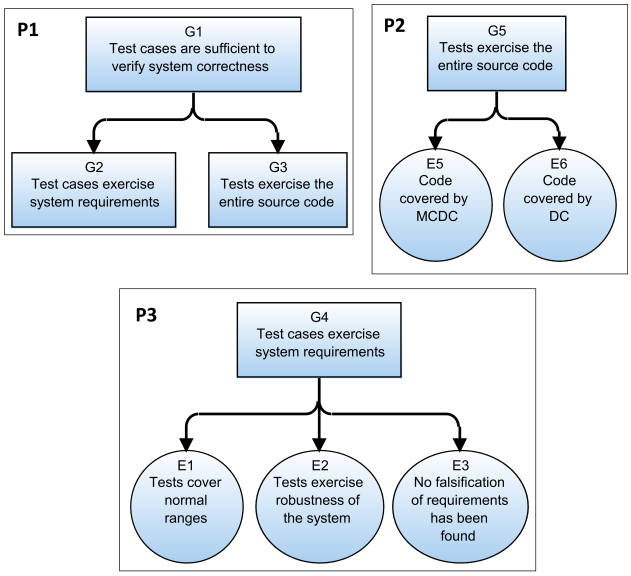


Fig. 1. AC pattern library used in the running example throughout the paper. Rendered as a rectangle, a *goal* node G represents a claim forming a part of the argument while an *evidence* (or *solution*) node E, rendered as a circle, represents a reference to an item of evidence.

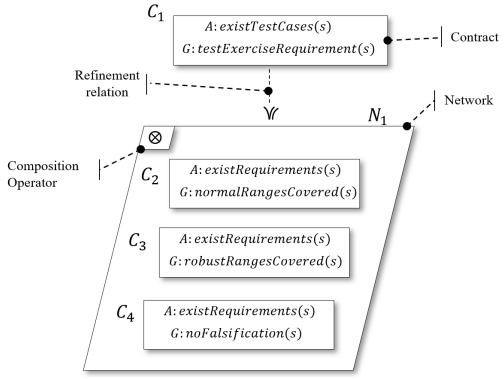


Fig. 2. Formalization of P3 in Figure 1 as an HCN.

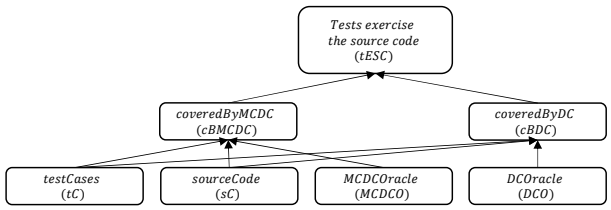


Fig. 3. Bayesian model captures sources of uncertainty and their relationship allowing analysis of the impact of the source of doubt on the goal.

ensuring that high-level software requirements satisfy system-level requirements is through a validation activity. The result of this validation activity is captured as a part of the logical argument and the lack of independence in the validation is captured as a possible source of doubt. Details on confidence networks can be found in [17].

The BN in Figure 3 can be associated with pattern *P2* in Figure 1. BNs can incorporate aleatoric and epistemic uncertainty, and allow combining multiple evidence items. For example, via the conditional probability $P(tESC|cBMCDC, cBDC)$, the BN pattern in Figure 3 can capture the dependencies among the confidence in the claim "tests exercise the source code," the result of the test (a source of aleatoric uncertainty) and the belief (*subjective* probability) in the quality of the test oracle (a source of epistemic uncertainty).

IV. AACE Framework

AACE is an assurance generation and validation tool for aerospace certification (Figure 4). AACE automatically translates argumentation patterns, written in Resolute [18], into HCNs and BNs. Based on a problem definition and pattern library, the generation algorithm creates an HCN that represents all possible assurance cases, which we refer to as *AC candidates*. Although this process guarantees the correctness of the generated assurance cases by construction, the generation algorithm does not consider the available evidence, which typically relates to the leaf premises in an assurance argument modeled as an HCN. Therefore, some candidates may be eliminated later in the validation process due to the lack of evidence or confidence. We denote by evidence a set of objective facts collected, for example, via tests, analyses, or proofs, that can be used in support of or against a claim. The validation algorithm coordinates logic and probabilistic reasoning to validate the soundness of an HCN and quantify its confidence level based on the available evidence. AACE also provides intuitive user interfaces to guide pattern specification as well as evaluation of the generated assurance cases.

Stakeholders: A high-level representation of AACE capabilities and its interaction with its stakeholders is shown in the use case diagram in Figure 5. The AACE stakeholders are *Domain Expert*, *Program Manager*, *Certification Authority*, *Evidence Curator*, and *Evidence Extractor*, as further detailed below:

- Argumentation Team: models domain-specific AC patterns that provide a general argument of how a given claim can be achieved in a domain and under specific conditions. The argumentation team is composed of a domain expert and an argumentation expert. Having a team ensures that the arguments are understandable, unambiguous, logical, and correct according to the domain.
- Program Managers: set top-level claim, which includes the target system, the certification goal, and the metric on which ACs can be optimized (e.g., confidence, cost, schedule). The Program Manager can trigger the generation of optimal ACs, which include validation and optimization of ACs by taking into account the available evidence.
- Certification Authorities: evaluate the resulting ACs for a given system and certification target. Certification authorities can refute ACs in which the argument is determined to be unsatisfactory. They could also go a step further and revise the applied pattern (e.g., by adding missing required evidence or tuning confidence models).
- Evidence Curator: provides evidence upon request.
- Evidence Extractor: extracts additional evidence in case there is still missing evidence.

V. Capturing Domain Expert Knowledge

AACE proposes the use of argumentation patterns to capture the means of compliance and their justification. An argumentation pattern represents a generalizable, modular, and reusable assurance case. Patterns undergo vetting by domain experts to ensure their completeness and correctness. AACE provides a pattern editor, as shown in Figure 6. Domain expert knowledge is encapsulated within the argumentation pattern. The elicitation process is led by the argumentation expert, who engages with the domain expert (e.g., a software engineer) to inquire about the required premises for achieving a given goal. In the running example, we employed DO-178C to extract domain knowledge. It is noteworthy that we did not directly map the objective; instead, our focus was on understanding the reasoning behind the objectives in order to create the patterns, as depicted in Figure 1. Sources of doubt can be added as *defeaters* using the editor.

After the pattern is created, the editor generates a `Resolute` file (Figure 7) that combines a human-understandable justification (to facilitate evaluation by certification authorities) with a formal specification to enable automation.

If sources of doubt are identified in the pattern creation, AACE generates a `json` file containing a BN, as shown in Figure 3. Domain experts can use a BN editor to add/remove sources of doubt, interconnections between sources of doubt, and modify the impact of the sources of doubt on the goal. BNs can be constructed based on domain expert knowledge, e.g., via an elicitation and calibration process. The BN in Figure 3 models a *confidence pattern* encapsulating notions from testing, to express the key correlations between the outcome of a coverage test and the quality of the artifacts (e.g., source code, test oracle) produced during software development and testing.

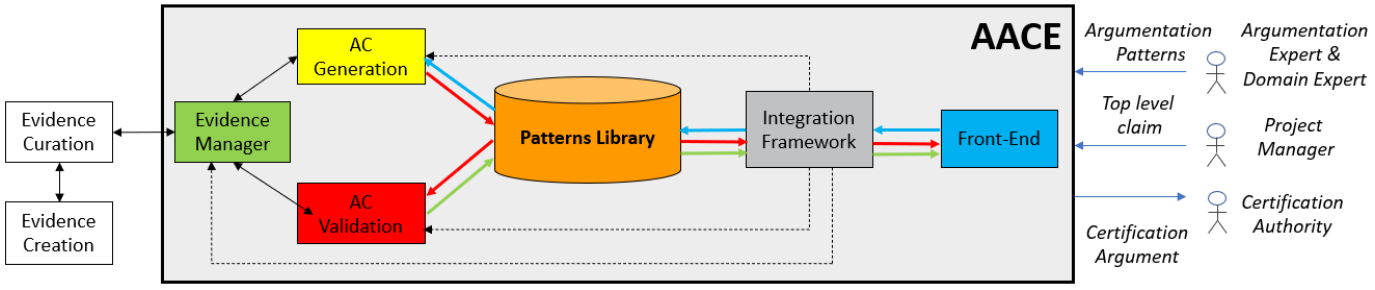


Fig. 4. AACE architecture.

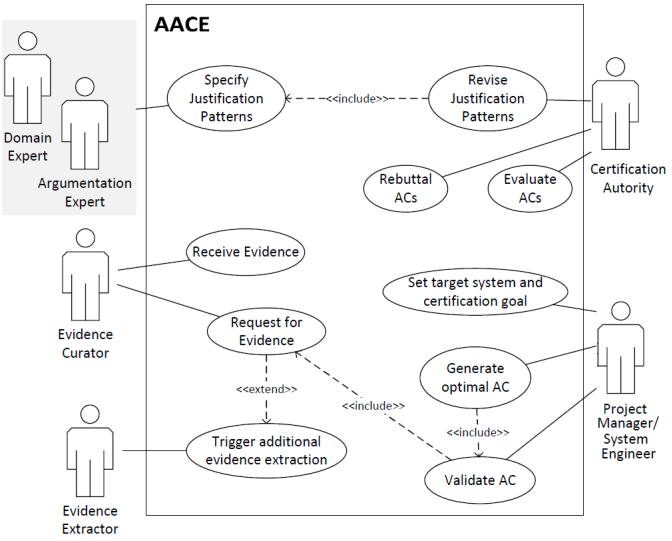


Fig. 5. Stakeholders of argumentation-based certification.

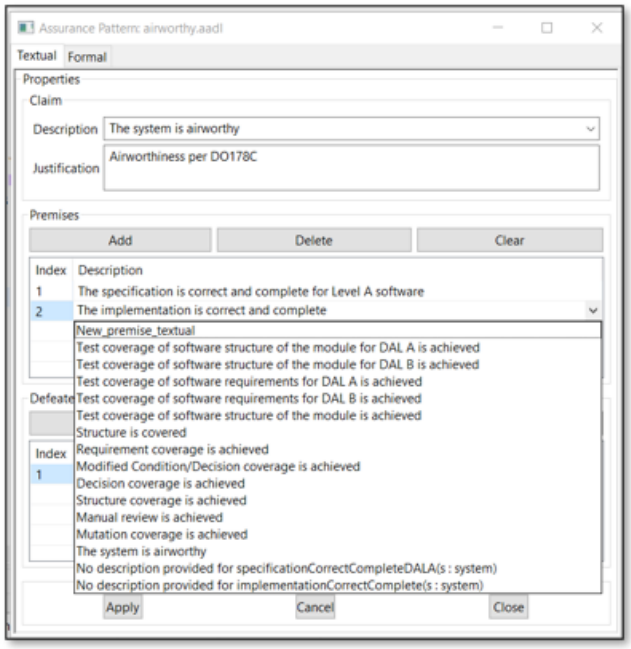


Fig. 6. Pattern editor supports the creation of patterns by providing a search mechanism for existing claims and by separating the argument creation from the argument formalization.

VI. Reasoning About Lifecycle Artifacts

Evidence produced during the entire lifecycle is captured in an ontology database. Such evidence is linked to real artifacts. The evidence ontology defines key software certification concepts such as Component, Requirement, and Test that are widely accepted by industry domain experts. In addition, the ontology captures data types and their relationships. Utilizing ontological reasoning between the assurance case generation and the evidence extraction allows the incorporation of varied sources of evidence and enables automated pre-analysis to identify inconsistencies and conflicts. Currently, we use the Rapid Assurance Curation Kit (RACK) [19] as our evidence database. Alternatively, evidence may be retrieved from other sources such as an architecture model of the system under development (e.g., in AADL).

The evidence assessment is executed by a query creator and ontology reasoner. The query creator receives evidence requests from the generation and validation modules. The evidence request is then evaluated, and one or multiple SPARQL queries are created and sent to the ontology reasoner. The current interface design uses the REST (REpresentational State Transfer) protocol.

VII. Generating Planned Certification Arguments

Based on a library of patterns and the specification of a given system, AACE generates all possible assurance cases (see Figure 9), termed AC candidates. AC candidates can then be ranked according to several metrics, such as confidence, cost, and schedule, allowing project managers to make informed decisions regarding system certification strategies.

The generation algorithm uses Satisfiability Modulo Theory (SMT) solving for instantiating and connecting different argumentation patterns to build a certification argument. This process guarantees the correctness of the generated assurance cases by construction. Once the AC candidates are created, AACE assesses if the arguments are sufficiently supported by the available evidence. We refer to evidence as a set of facts collected, for example, via tests, analyses, or proofs, that can be used in support of the argument. Given the AC pattern library shown in Figure 1 and the top-level claim *"test*

```
goal G18_High_Quality_Automata(reqs : {requirement}) <=
    ** "Requirements are high quality" **
    justification: "Partial fulfillment of DO-333 objectives";
    claim: High_Quality_Requirements(reqs);
    defeater G19_Soundness: "Formalization is sound" : E6_Manual_Review(reqs);
    G20_Accurate_Consistent(reqs)

goal G25_High_Quality_SL(reqs : {requirement}) <=
    ** "Requirements are high quality" **
    justification: "Partial fulfillment of DO-331 objectives";
    claim: High_Quality_Requirements(reqs);
    G22_Accurate_Consistent(reqs) and G23_Comply_With_Guidelines(reqs) and
        G24_Formalization_Correct(reqs)

goal G12_High_Quality_NL(reqs : {requirement}) <=
    ** "Requirements are high quality" **
    justification: "Partial fulfillment of DO-178C objectives";
    claim: High_Quality_Requirements(reqs);
    G14_Comply_With_Guidelines(reqs) and G15_Accurate_Consistent(reqs)
```

Fig. 7. Argumentation patterns are defined in Resolute that can be understood by humans with some training and automatically translated in formal models.

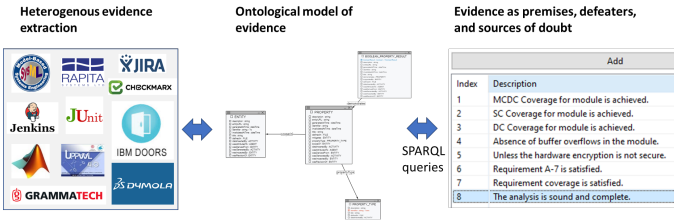


Fig. 8. Evidence produced during the entire product lifecycle is captured in an ontology database allowing automatic reasoning about system and process evidence.

cases are sufficient to verify system correctness," the synthesis algorithm generates the AC shown in Figure 10, where the contract networks defined in *P2* and *P3* refine the two premises of *P1*.

VIII. Assessing Certification Arguments

In the current certification process, validation is performed manually. A certification authority picks a handful of development products (e.g., system level requirements) and traces them through the development lifecycle to validate that the proposed plan has been executed. In contrast, AACE automates the validation process by systematically checking all the supporting evidence and assessing the confidence in the ACs using a combination of logic and probabilistic reasoning (see Figure 11). To quantify the persuasiveness of claims supported by evidence, we utilize BNs to compute confidence values. These confidence values are then classified into coarsely grained, qualitative levels, such as `Low`, `Medium`, or `High`. To determine the sufficiency of an intermediate claim supported by sub-claims, we combine the qualitative confidence levels based on pre-defined rules. For instance, a simple rule might require that all sub-claims must have a `High` confidence level. This combination provides a way to reason about evidence that is required (e.g., results of a validation process) and evidence that increases or reduces the level of confidence (e.g., independence in the validation). The required level of confidence can vary with the criticality level. The values in green attached to each claim in Figure 12

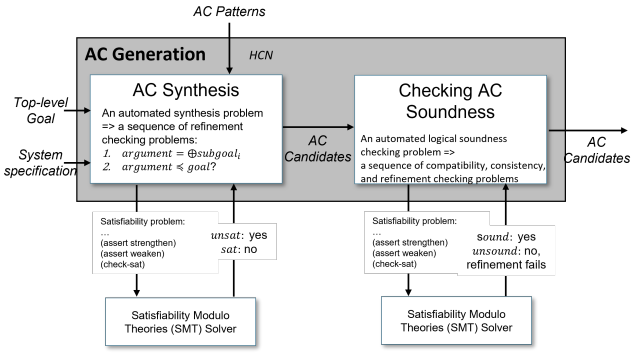


Fig. 9. Synthesis and independent validation of soundness of assurance case candidates.

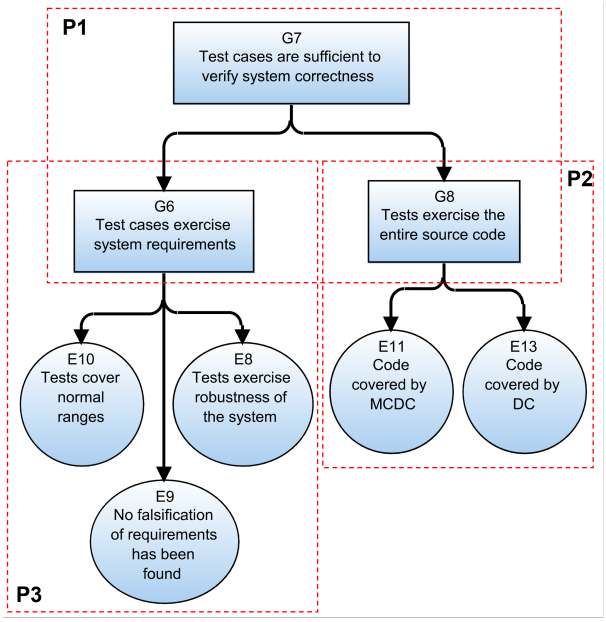


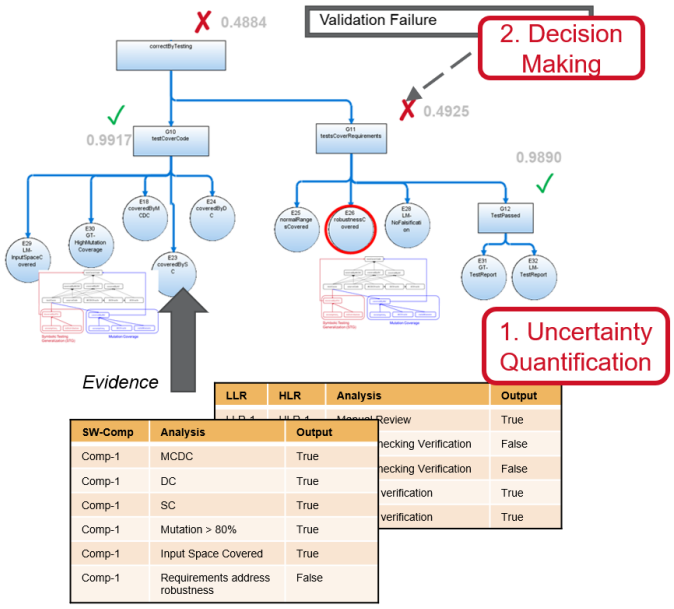
Fig. 10. Generated AC based on the AC pattern library in Figure 1. Dotted boxes highlight the instantiated pattern.

represent the confidence values when all the evidence items are available. On the other hand, the values in red represent the confidence values with missing evidence to support the claim "tests exercise robustness of the system."

IX. Case Studies

We evaluated our framework using two synthetically generated and an industrial-level aerospace case studies on an Intel core i7 processor with 16-GB RAM. The performance metrics were averaged over three runs. We first evaluated the performance and scalability of the proposed synthesis and validation algorithms.

In *Experiment 1*, we conducted an assessment to evaluate the influence of the AC pattern library size (Sys_Lib) and the size of the synthesized HCN (Val_Lib) on the execution time of the synthesis and validation plugins. To perform this evaluation, we executed the framework using input pattern libraries



SW-Comp	Analysis	Output
Comp-1	MCDC	True
Comp-1	DC	True
Comp-1	SC	True
Comp-1	Mutation > 80%	True
Comp-1	Input Space Covered	True
Comp-1	Requirements address robustness	False

LLR	HLR	Analysis	Output
		Manual Review	True
		hecking Verification	False
		hecking Verification	False
		verification	True
		verification	True

Fig. 11. Assessment of the support of evidence.

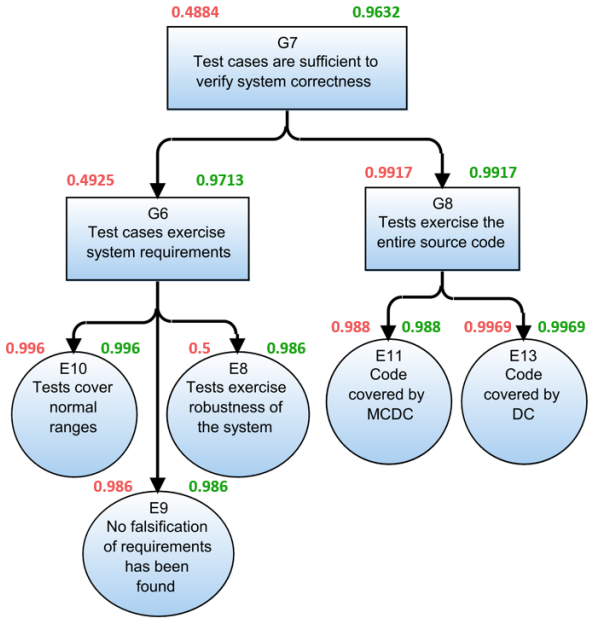


Fig. 12. Assurance case with confidence calculation.

of different sizes, which were generated synthetically. Additionally, we used several different top-level claims, resulting in HCNs of varying sizes. The size of the pattern library ranged from 121 to 29,524 (Sys_Lib=121 to Sys_Lib=29,524), as indicated by the three different colored markers at the bottom of Figure 13. The size of the synthesized HCNs, measured by the number of the patterns that are instantiated for constructing the HCNs, ranged from 3 to 1,200. As depicted in Figure 13, the results demonstrate that the size of the library has no discernible effect on the validation time and only a negligible impact on the execution time of the synthesis process. On the other hand, both the synthesis and validation execution times display a linear increase as the size of an HCN grows.

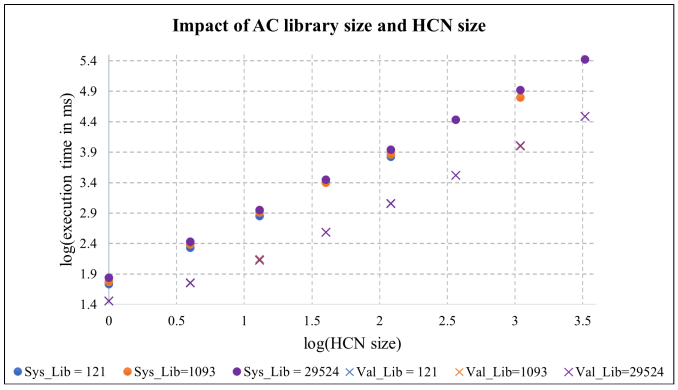


Fig. 13. Performance of the synthesis and validation plugins with respect to the size of the AC pattern library and the size of the generated HCN (x-axis).

In *Experiment 2*, we assessed the impact of the number of the AC candidates on the execution time of the synthesis and validation plugins. Similarly to experiment 13, the synthesis plugin was executed given input libraries of different sizes and top-level claims. The plugin produced multiple AC candidates supporting a single top-level claim.

Multiple AC candidates were created whenever a claim could be supported by more than one argument pattern, e.g., since an implementation can either be verified using a set of testing or a formal method, two AC candidates can be created in this case.

The candidates were constructed such that they have the same size (measured by the number of patterns used in the synthesis). In this experiment, the number of generated AC candidates varied from 1 to $1,094$. As shown in Figure 14, the execution time of the synthesis plugin was not significantly impacted by the number of candidates. In contrast, the execution time of the validation plugin increased as the number of candidates increased.

In *Experiment 3*, we aim to demonstrate how the framework can be used to generate overarching properties (OP) certification arguments for the navigation subsystem of an autopilot module. The autopilot has a flight stack, which performs the estimation and control of a drone, and a middleware, that supports communication and hardware integration. The navigation subsystem sends an input to the actuators based on multiple sensors (e.g., GPS, IMU, camera control, distance sensor, optical flow) and the target position provided by the remote control. The navigation subsystem possesses 230 high-level requirements, 376 low-level requirements, and 66,000 software components. High-level requirements (HLRs) are written in natural language, and low-level requirements (LLRs) are modeled in Simulink. Requirement-driven testing was utilized to evaluate the correctness of the implementation.

We created an assurance case library containing 50 AC patterns incorporating best system development practices inspired by compliance with the RCA DO-178C, DO-331, and DO-333 standards.

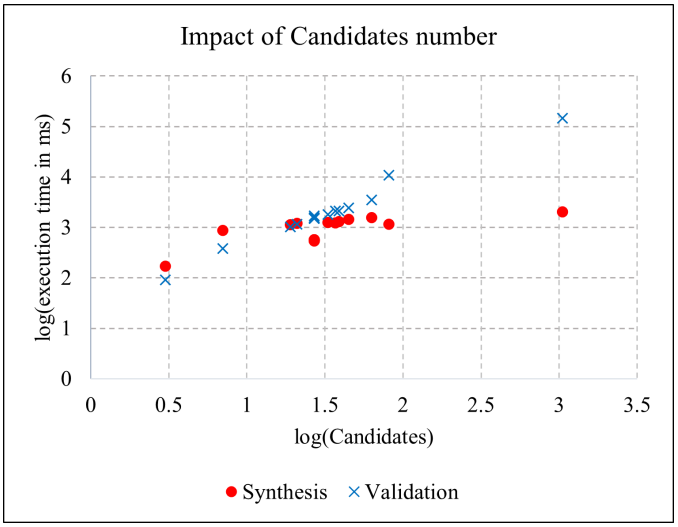


Fig. 14. Performance of the synthesis and validation plugins with respect to the amount of generated AC candidates.

One of the generated ACs is shown in Figures 15 and 16. This AC consists of 44 claims and has a top-level goal stating that "the navigation subsystem is suitable for flying." The top-level goal is satisfied by showing that the navigation system satisfies the *Intent*, *Correctness*, and *Innocuity* properties. For a detailed illustration of the AC, we refer to the Appendix.

Once the AC candidates were generated, the validation algorithm queried the evidence database containing the artifacts from the lifecycle processes (test cases, reviews) and the system specifications on the software components, code generation, criticality level, etc. The AC generation framework took 2.92 seconds to generate and assess the AC for the navigation subsystem.

By employing argumentation patterns repeatedly, our framework achieved enhanced efficiency, particularly in the synthesis plugin. In this experiment, we were able to instantiate over 606 claims within an AC utilizing just a few of argumentation patterns. This approach not only saved memory but also reduced the effort required for logic validation. However, we encountered a bottleneck when the number of evidence items in the database grew, resulting in longer querying times and subsequently increased validation time.

X. Related Work

Some existing tools (AdvoCATE [20], DS-Bench [21]) support limited automation for AC representation and validation with GSN or CAE pattern libraries. However, the semantics of these tools are not fully characterized, e.g., the nature of refinement (inductive or deductive) between claims is not specified [5]. This gap can undermine the strength of an argument through confirmation bias [4]. Our tool removes these ambiguities by modeling claims as contracts and leveraging the well-defined contract algebra [8]. Additionally, we consider Bayesian reasoning to assess the relative strength of available arguments. The AMASS platform provides tools for AC automation using contracts but does not support quantitative assessment of an argument's confidence. Other tools [22], [23] enable AC automation and also support confidence assessment, however, they do not utilize a contract-based, compositional framework. Our tool [24] is the only one to combine rigorous, contract-based AC automation with logically monitored Bayesian reasoning for confidence assessment. We additionally provide an accessible and clear interface [25] to guide domain experts and certification authorities from pattern specification to argument evaluation in a cohesive, end-to-end framework.

XI. Conclusion

We presented the Automated Assurance Case Environment (AACE), a framework for automatic generation and validation of ACs, leveraging domain knowledge-based assurance patterns. ACs are formalized as hierarchical contract networks, which allow for efficient, modular synthesis and validation of arguments. We proposed an AC synthesis approach that uses SMT for instantiating and composing unit argumentation patterns to build a complete AC and an AC validation approach that combines logic and probabilistic reasoning to propagate confidence calculations throughout the AC. We validated the quality of the generated ACs and the performance of our algorithms on a commercial aerospace case study.

We plan to support sensitivity analysis to facilitate the evaluation of BN confidence patterns. We also plan to add cost information to the patterns so that we can investigate cost-effective solutions during synthesis and validation.

References

[1] The Assurance Case Working Group, "Goal structuring notation community standard (version 3)," 2021.

[2] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly, "Weaving an assurance case from design: a model-based approach," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE, 2015, pp. 110–117.

[3] J. Rushby, "Formalism in safety cases," in *Making Systems Safer*. Springer, 2010, pp. 3–17.

[4] R. Bloomfield and J. Rushby, "Assurance 2.0: A manifesto," *arXiv preprint arXiv:2004.10474*, 2021.

[5] J. Rushby, "The interpretation and evaluation of assurance cases," Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-15-01, 2015.

[6] C. M. Holloway, "Explicate'78: Uncovering the implicit assurance case in DO-178C," *Safety-Critical Systems Symposium*, 2015.

[7] Adelard LLP, *Claims, Arguments and Evidence (CAE)*, 2019, https://www.adelard.com/asce/choosing-asce/cae.html.

[8] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[9] S. S. Bauer, A. David, R. Hennicker, K. Guldstrand Larsen, A. Legay, U. Nyman, and A. Wakowski, "Moving from specifications to contracts in component-based design," in *Fundamental Approaches to Software Engineering*. Springer, 2012, pp. 43–58.

[10] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[11] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2104–2132, 2015.

[12] F. V. Jensen, *Introduction to Bayesian Networks*, 1st ed. Springer, 1996.
[13] R. Neapolitan, *Learning Bayesian Networks*, ser. Artificial Intelligence. Pearson Prentice Hall, 2004.
[14] C. Hobbs and M. Lloyd, "The application of Bayesian belief networks to assurance case preparation," in *Achieving Systems Safety*. Springer London, 2012, pp. 159–176.
[15] K. Verbert, R. Babuška, and B. De Schutter, "Bayesian and Dempster–Shafer reasoning for knowledge-based fault diagnosis–A comparative study," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 136–150, 2017.
[16] T. E. Wang, Z. Daw, P. Nuzzo, and A. Pinto, "Hierarchical contract-based synthesis for assurance cases," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 175–192.
[17] C. Oh, N. Naik, Z. Daw, T. E. Wang, and P. Nuzzo, "ARACHNE: Automated validation of assurance cases with stochastic contract networks," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2022, pp. 65–81.
[18] A. Gacek, J. Backes, D. D. Cofer, K. Slind, and M. Whalen, "Resolute: An assurance case language for architecture models," in *Proceedings of the ACM SIGAda annual conference on High integrity language technology*, 2014, pp. 19–28.
[19] A. Moitra, P. Cuddihy, K. Siu, B. Meng, J. Interrante, D. Archer, E. Mertens, K. Quick, V. Robert, and D. Russell, "A semantic reference model for capturing system development and evaluation," in *16th International Conference on Semantic Computing (ICSC)*. IEEE, 2022, pp. 173–174.
[20] E. Denney, G. Pai, and J. Pohl, "AdvoCATE: An assurance case automation toolset," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2012, pp. 8–21.
[21] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa, "DS-Bench toolset: Tools for dependability benchmarking with simulation and assurance," in *International conference on dependable systems and networks*. IEEE, 2012, pp. 1–8.
[22] C. Cârlan, V. Nigam, S. Voss, and A. Tsalidis, "ExplicitCase: tool-support for creating and maintaining assurance arguments integrated with system models," in *International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 330–337.
[23] S. Ramakrishna, C. Hartsell, A. Dubey, P. Pal, and G. Karsai, "A methodology for automating assurance case generation," *arXiv preprint arXiv:2003.05388*, 2020.
[24] T. Wang, C. Oh, M. Low, I. Amundson, Z. Daw, A. Pinto, M. Chiodo, G. Wang, S. Hasan, R. Melville, and P. Nuzzo, "Computer-aided generation of assurance cases," in *Computer Safety, Reliability, and Security. SAFECOMP Workshops*. Springer, 2023.
[25] Z. Daw, T. Wang, C. Oh, M. Low, I. Amundson, G. Wang, R. Melville, and P. Nuzzo, "Computer-aided evaluation for argument-based certification," in *42nd AIAA/IEEE Digital Avionics Systems Conference*. IEEE, 2023.

APPENDIX

This section provides details on the AC generated in Experiment 3. The AC represented in Figure 16 shows that the navigation system satisfies the *Intent*, *Correctness*, and *Innocuity* properties. These three properties are sufficient since Overarching Properties (OPs) are used as a means of compliance. In this Figure, the branches related to Correctness and Innocuity are undeveloped for readability. The branch of the AC arguing about Correctness is, instead, shown in Figure 15.

The AC argues that Intent holds if the software system specification satisfies (complies with) the system-level requirements (SLR) (G5), the specification is of high quality (G9), and is fully developed (G13). Since the navigation system is specified using high-level requirements (HLRs), low-level requirements (LLRs), and software architecture, the instantiated pattern achieves G5 by ensuring that system-level requirements are satisfied by HLRs (G6) and that the software architecture and the LLRs satisfy the HLRs in G7 and G8, respectively. The synthesis algorithm instantiates the same pattern to satisfy G6 and G7, showing refinement between two sets of requirements by checking for traceability and compliance. Depending on the type of requirements, compliance can be shown using a manual check-list, especially for natural-language requirements, or formal verification, if both the sets of requirements are specified using formal models. In the navigation example, compliance is shown by manual check-list. By virtue of its modularity, our framework can then support the automatic incorporation of manual check-lists, aiming to evaluate single pieces of evidence, when they are deemed necessary, and enable their

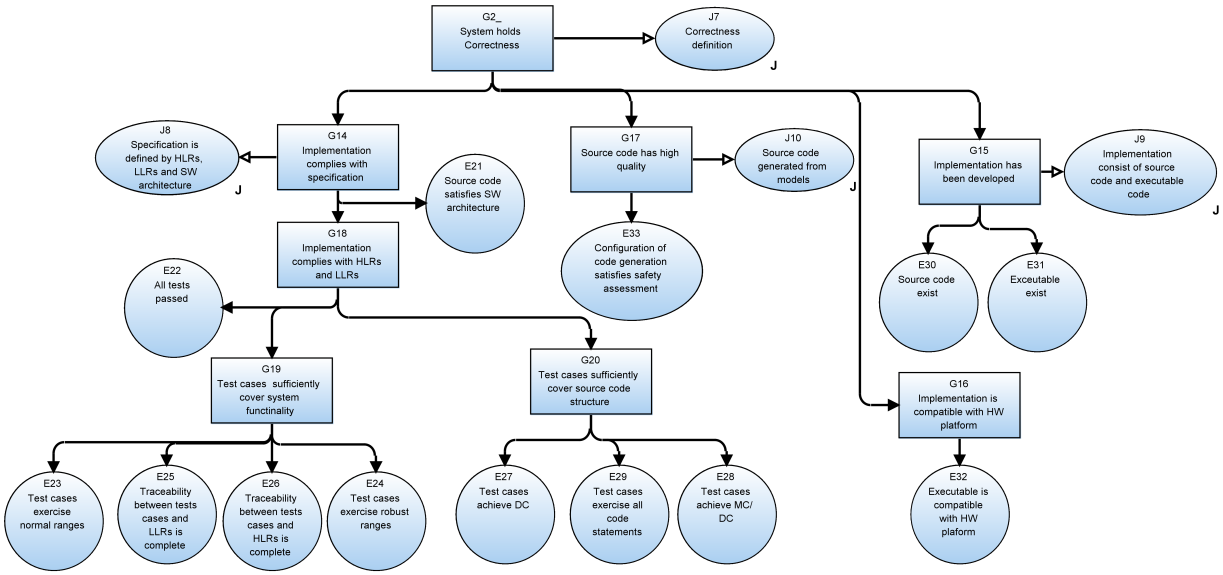


Fig. 15. Part of the generated AC for the certification of a commercial aerospace product corresponding to the *Correctness* property.

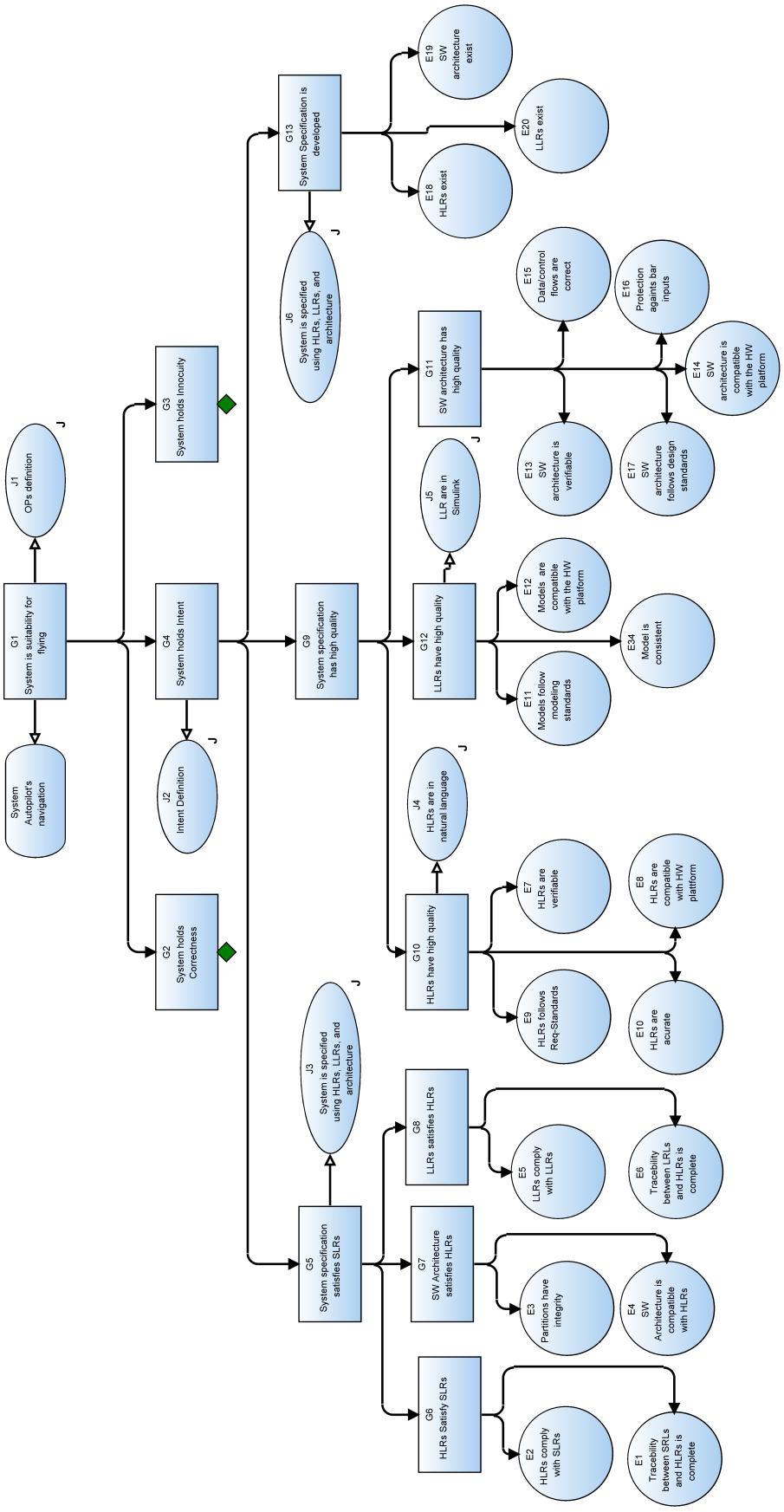


Fig. 16. Generated AC for the certification of a commercial aerospace product. The AC portion corresponding to Correctness is shown in Figure 15. The Innocuity AC is not shown due to space constraints.

assessment in the context of complex argumentation steps and the overall AC.

To illustrate the reasoning capability of our framework for general properties, we selected "high quality" as a property to be supported by the specification. Depending on the type of specification, this property may be interpreted as stating that the requirements are well-written without ambiguity or that the models are formulated at the right level of abstraction. In this respect, the AC pattern itself captures information about its applicability, which is used by the synthesis algorithm during the pattern instantiation process. An example of this situation is offered by G10 and G12, where a different pattern was selected for HLRs written in natural languages versus LLRs specified using Simulink models.

G10 is achieved by checking that textual requirements are accurate (E10), verifiable (E7), follow standards (E9), and are compatible with the specific hardware (HW) platform (E8). Since LLRs are specified as Simulink models and they have defined semantics, accuracy and verifiability do not need to be checked. The instantiated pattern ensures high quality by checking that there is no inconsistency (E14), that models comply with modeling guidelines (E11), and that they are compatible with the target HW platform (E12). Evidence for consistency is supplied by a report from Simulink Design Verifier.

Correctness (G2) is satisfied if the implementation complies with the specification (G14), the source code is high-quality (G17), the implementation is compatible with the target HW platform (G16), and the implementation exists (G15). Since the navigation system code is auto-generated from the model, the synthesis algorithm instantiates a pattern specific to code generation. This pattern (instantiated for G14) argues that the configuration of the code generator needs to undergo a safety assessment in order to avoid optimizations that lead to discrepancies between the behavior of the model and the implementation. This type of information is provided to the user as justification[1]. However, it is not shown in the figure, for the sake of readability.

Testing is used to show that the implementation complies with the specification (G18). This goal is achieved by ensuring that the implementation passed all tests (E22) and that test cases sufficiently cover the system functionality (G19) and the code structure (G20). Coverage of system functionality is ensured by showing traceability (E25, E26) and coverage of the requirements for normal and robust ranges (E23, E24). Because the autopilot system is classified as a high-criticality system (Level A), the synthesis selected a pattern corresponding to that level of criticality (G20). Accordingly, for this pattern, test cases need to achieve three different coverage metrics: Decision coverage (DC), modified condition/decision coverage (MC/DC) and statement coverage.

Two examples illustrating how different artifacts can be grouped in the generated AC are G6 and G8, which instantiate the same pattern for requirement satisfaction, and G10 and G12, which instantiate different patterns to ensure a high-quality specification. This modeling choice avoids the instantiation of the same pattern for 606 requirements and prevents the validation algorithm from querying for the available evidence for each requirement.

[1]Justification nodes are labeled with *J* in GSN.