Computer-Aided Generation of Assurance Cases

Timothy E. Wang[1], Chanwook Oh[2], Matthew Low[2], Isaac Amundson[3], Zamira Daw[4], Alessandro Pinto[5], Massimiliano L. Chiodo[1], Guoqiang Wang[1], Saqib Hasan[3], Ryan Melville[1], and Pierluigi Nuzzo[2]

[1] Raytheon Technology Research Center, Berkeley, CA, USA
[2] University of Southern California, Los Angeles, CA, USA
[3] Collins Aerospace, Cedar Rapids, IA, USA
[4] University of Stuttgart, Stuttgart, DE
[5] NASA Jet Propulsion Laboratory, Pasadena, CA, USA

Abstract. Assurance cases (ACs) have gained attention in the aerospace, medical, and other heavily-regulated industries as a means for providing structured arguments on why a product is dependable (i.e., safe, secure, etc.) for its intended application. Challenges in AC construction stem from the complexity and uniqueness of the designs, the heterogeneous nature of the required supporting evidence, and the need to assess the quality of an argument. We present an automated AC generation framework that facilitates the construction, validation, and confidence assessment of ACs based on dependability argument patterns and confidence patterns capturing domain knowledge. The ACs are instantiated with a system's specification and evaluated based on the available design and verification evidence. Aerospace case studies illustrate the framework's effectiveness, efficiency, and scalability.

Keywords: Assurance case, contracts, synthesis, validation, confidence

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 and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

1 Introduction

Assurance about certain properties of mission-critical systems, such as safety, security, and functional correctness, is essential throughout the product development lifecycle. Certification standards (e.g., DO-178C in the aerospace industry) tend to be prescriptive in nature, enforcing a rigid, often costly, evaluation process. Assurance evaluation is currently performed mostly manually, and due to the substantial amount of evidence that needs to be examined, can lead to incomplete or biased assessments.

The trend in research is shifting from a prescriptive certification process to argument-based certification (e.g., based on overarching properties (OPs) [1]), which offers more flexibility to certify emerging technologies, including systems enabled by artificial intelligence (AI). However, this approach expands the scope of the evidence evaluation process, beyond checking the *sufficiency* of evidence, to validating whether the proposed system development activities and outcomes are appropriate to establish safety, security, or standard compliance for a given application. In this context, automated methods and tools that support the creation, maintenance, and evaluation of arguments are deemed as necessary for a streamlined assurance process.

Assurance cases (ACs) are explicit arguments that desired properties have been adequately established for a given system. ACs are typically created manually and, at present, are mostly used to document the argumentation structure. There is a lack of consensus on a systematic approach for creating and validating ACs. Tools for AC visualization and manipulation exist, but notations are often not completely defined, leaving room for interpretation and misunderstanding [2]. For example, the Goal Structuring Notation (GSN) specifies the visual syntax and semantics of its elements, such as goals, strategies, justifications, and solutions. However, the standard [3] primarily relies on natural language for expressing claims, which can be ambiguous and open to misinterpretation. Formalisms and tools that can assist in the creation of rigorous and interpretable arguments are limited to only a few attempts [4–6].

This paper addresses some of the challenges in constructing and validating ACs, including (a) the complexity of modern safety and security-critical systems, (b) the heterogeneous nature of the evidence, and (c) the need to assess the argument quality, i.e., quantify the persuasiveness of the argument given the sources of doubt. We present an *automated, end-to-end* framework that synthesizes and validates ACs. The framework includes (i) an efficient synthesis algorithm that automatically creates AC candidates using a pattern library of *pre-order relations* linking claims with their supporting (sub)-claims and (ii) a validation algorithm that uses logic and probabilistic reasoning to effectively identify a set of *logically valid* and *most persuasive* AC candidates. The framework also includes a visualization tool for traversing the generated ACs in multiple formats at different levels of the argument hierarchy.

Related Work. Several tools [7–10] support manual or partly automated AC creation, instantiation, management, and analysis. Approaches based on the AMASS platform [11–14] use contracts for automated AC creation to enable compositionality and reuse of argumentation patterns. Beyond automated and compositional AC generation with contracts, our framework also provides quantitative assessment of the persuasiveness of an AC, based on design artifacts and evidence, by leveraging Bayesian reasoning [15] to compute confidence values.

A few tools [16, 17] also support automated AC generation and confidence assessment, albeit not within a contract-based, compositional framework. A possible limitation of existing approaches to confidence quantification stems from the fact that missing or contrary information may be “masked” by an overall high confidence score at the top-level claim, leading to insensitivity to critical local evidence [18]. We address this issue via a compositional and hierarchical approach that combines local probabilistic reasoning with global logical reasoning to derive the confidence of the top-level claim [19]. Our assessment algorithm, consisting of uncertainty quantification steps followed by decision-making steps, allows appropriately weighing different evidence items and early discounting of unreliable evidence with low confidence levels, thus making the overall process of propagating confidence values to the top-level claim and determining their sufficiency more robust to inaccuracies in the confidence models.

2 Background

Assurance Cases. An *assurance case (AC)* is an argument constructed to establish that a system satisfies the requirements in its operative environment by means of hierarchical steps that map a *claim* to *evidence* via strategies and intermediary claims [2, 6]. ACs are often described using a structured language [20] or graphical notations such as the Goal Structuring Notation (GSN) [21] and the Claims-Arguments-Evidence (CAE) notation [22]. An AC can then be represented as a directed acyclic graph mapping the system specification (the top-level claim) from the root node to the leaf nodes representing the evidence.

Software tools such as AdvoCATE [7] and DS-Bench [23] can be used to specify GSN (or CAE) pattern libraries, providing a limited degree of automation for representation and validation of ACs. However, some semantic elements in these tools are not well-defined in their respective conventions, leaving room for an individual developer to clarify them [2]. The argumentation steps between claims often lack rigor, opening the door for confirmation bias [6]. We address these concerns by leveraging contract operations to solidify the relationship between claims, allied with Bayesian reasoning to assess their strength.

Assume-Guarantee Contracts. Assume-guarantee (A/G) contracts offer effective mechanisms to analyze system requirements in a modular way [24, 25]. We use contracts as a specification formalism to represent claims about the system and the development process as well as the contexts under which the claims hold. We represent a contract C as a pair of logic formulas (ϕ_A, ϕ_G), where ϕ_A (assumptions) specifies the context under which the claim holds and ϕ_G (guarantees) specifies the promise of the claim in the context of ϕ_A. We can reason about the replaceability of a contract (or a claim) by another contract via the *refinement* relation. We say that C_2 refines C_1, written $C_2 \preceq C_1$, if and only if C_2 has weaker assumptions and stronger guarantees than C_1. When this is the case, we can replace C_1 with C_2. Contracts C_1 and C_2 can also be combined, e.g., using *composition* (denoted by $C_1 \otimes C_2$) to construct more complex arguments from simpler claims.

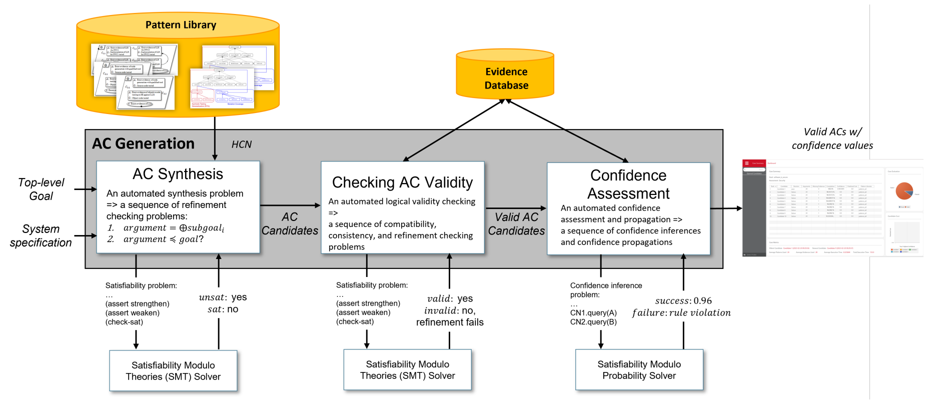

Pattern Library
Evidence Database
AC Generation
HCN
Top-level Goal
System specification
AC Synthesis
An automated synthesis problem => a sequence of refinement checking problems:
1. $argument = \oplus subgoal_i$
2. $argument \preccurlyeq goal$?
AC Candidates
Checking AC Validity
An automated logical validity checking => a sequence of compatibility, consistency, and refinement checking problems
Valid AC Candidates
Confidence Assessment
An automated confidence assessment and propagation => a sequence of confidence inferences and confidence propagations
Valid ACs w/ confidence values
Satisfiability problem: … (assert strengthen) (assert weaken) (check-sat)
$unsat$: yes
sat: no
Satisfiability problem: … (assert strengthen) (assert weaken) (check-sat)
$valid$: yes
$invalid$: no, refinement fails
Confidence inference problem: … CN1.query(A) CN2.query(B)
$success$: 0.96
$failure$: $rule\ violation$
Satisfiability Modulo Theories (SMT) Solver
Satisfiability Modulo Theories (SMT) Solver
Satisfiability Modulo Probability Solver

Fig. 1. Overview of the AC generation framework with summary of output candidates.

3 AC Generation Framework

As shown in Fig. 1, our framework automatically generates and validates ACs given a top-level goal from a pattern library, the evidence database, and the system specification. We refer to the synthesis and validation processes collectively as *generation*. Validation itself coordinates logic and probabilistic reasoning to *select* valid AC candidates and *assess* their confidence level.

AC patterns are reusable AC templates that can be instantiated using system specific information to construct larger ACs. The patterns capture, in the form of generic argumentation steps, domain knowledge about what evidence is required to achieve specific goals. We employ hierarchical contract networks (HCNs) to formalize AC patterns and use Resolute [26] as the underlying language for their specification. Uncertainty in assertions is instead modeled using Bayesian networks (BNs). Given a top-level claim from the pattern library and a system under evaluation, the synthesis algorithm automatically selects and instantiates the patterns and assembles the instantiated patterns together into an HCN that represents a set of *AC candidates*.

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 evidence is maintained according to an *evidence ontology* in a database that is accessible by the AC generation framework. The availability of evidence could be considered by the synthesis algorithm for eliminating certain AC candidates early. However, our framework also targets an assurance-driven development process with objectives (e.g., cost and development time) taken into account during certain stages of the product lifecycle. Particularly, in the early stages of product development, when the system design is incomplete, we are interested in exploring the set of all potential evidence items and assurance arguments, to analyze cost and benefit trade-offs. The *validation* algorithm selects the AC candidates with the lowest number of missing supporting evidence. The selected ACs undergo confidence level quantification, which assesses the ACs' persuasiveness. Finally, our framework returns a set of valid ACs and their confidence scores, ensuring that the most convincing and well-supported ACs are presented to system designers and certification authorities.

Overall, by leveraging our previous results on generic HCN synthesis [27] and AC validation [19], our framework supports the first end-to-end automated methodology that seamlessly integrates contract-based synthesis and validation of ACs, and makes it accessible, via intuitive user interfaces, to system designers and certification authorities.

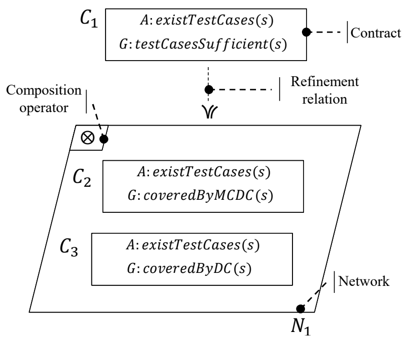
3.1 AC Pattern Formalization

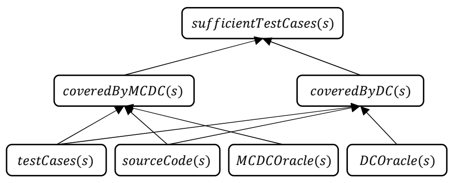
We formalize an AC pattern as a pair consisting of an HCN (assurance pattern) and a set of BNs (confidence patterns).

Hierarchical Contract Networks. An argumentation step involving a concatenation of claims can be represented as a graph of interconnected contracts, termed *contract network* (CN). A network N of m contracts is equivalent to the contract $C_N = C_1 \parallel \cdots \parallel C_m$, where $\parallel \in \{\otimes, \wedge\}$ denotes a contract operation [24]. Stepwise refinements of higher-level claims into lower-level claims can then be captured by a tree of contract networks, that is, a *hierarchical contract network* (HCN) using contract refinement [19, 27]. Fig. 2a shows the HCN corresponding to the pattern *P2* in Fig. 3. The patterns in Fig. 3 and 4 are depicted in GSN only for illustrative purposes. Every statement in the pattern is formalized as a logical predicate, which forms the contract assumptions and guarantees. The goal is translated into contract C_1, while the premises are translated into contracts within the contract network N_1.

Confidence Networks. We use *confidence networks* to capture the sources of uncertainty in the claims of an HCN. In this paper, confidence networks are implemented as BNs, which have been used in the past to incorporate subjective notions of probability or belief and quantitatively reason about the confidence in assertions affected by uncertainty [28]. BNs can encompass aleatoric and epistemic uncertainty and tend to produce more compact models in the number of parameters than other probabilistic reasoning frameworks for uncertainty quantification [29]. BNs can be constructed based on domain expert knowledge, e.g., via an elicitation and calibration process, and can leverage notions from Bayesian epistemology to account for lack of knowledge.

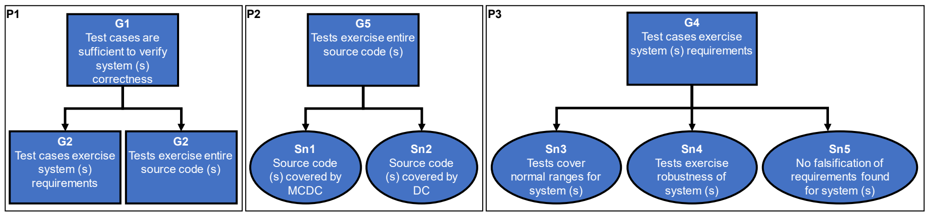
Example 1. Assume an AC pattern library that partially addresses the *correctness* goal, shown in Fig. 3. *P1* argues that test cases are sufficient for demonstrating system correctness if the tests exercise all requirements and cover the entire source code. *P2* argues that a set of tests cover the entire source code based on documented coverage metrics. *P3* argues that requirements are covered by a set of test cases if the tests cover the normal and robust ranges of requirements. The BN in Fig. 2b models a *confidence pattern* expressing the





(a) HCN (assurance pattern) (b) BN (confidence pattern)

Fig. 2. AC pattern for *P2* in Fig. 3.



P1
G1 Test cases are sufficient to verify system (s) correctness
G2 Test cases exercise system (s) requirements
G2 Tests exercise entire source code (s)
P2
G5 Tests exercise entire source code (s)
Sn1 Source code (s) covered by MCDC
Sn2 Source code (s) covered by DC
P3
G4 Test cases exercise system (s) requirements
Sn3 Tests cover normal ranges for system (s)
Sn4 Tests exercise robustness of system (s)
Sn5 No falsification of requirements found for system (s)

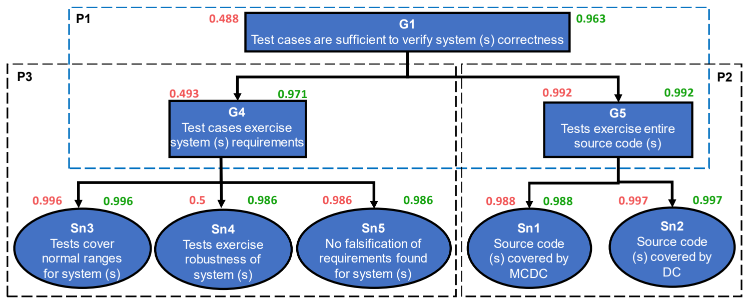
Fig. 3. AC pattern library used in the running example. Evidence items are represented as solution nodes (Sn); claims are represented as goal nodes (G).

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. The BN in Fig. 2b can be associated to pattern *P2* in Fig. 3.

3.2 AC Synthesis Algorithm

Given a contract network N_0, an AC pattern library $\mathcal{L}$, and a set of formulas specifying the context about the system, the synthesis algorithm automatically generates a set of AC candidates in terms of an HCN H with N_0 as the top node. By leveraging the synthesis algorithm first introduced for generic HCNs [27], our synthesis procedure proceeds as follows. The top-level goals N_0 are *instantiated* with the system under evaluation. N_0 is a contract network containing one or more goals. For each instantiated contract $\tilde{C}$ of $\tilde{N}_0$, the algorithm searches the AC pattern library $\mathcal{L}$ for contract networks that are potential refinements. Any contracts that are not refined by any other contract networks in the library are either *evidential*, indicating that they can only be established by evidence from the lifecycle activities, e.g., requirements review, test results, code reviews, formal methods, or they are undeveloped goals.

The returned refinements are also instantiated with the system under evaluation. After instantiation, the validity of potential refinements is determined by



P1
0.488
G1
Test cases are sufficient to verify system (s) correctness
0.963
P3
P2
0.493
0.971
G4
Test cases exercise system (s) requirements
0.992
0.992
G5
Tests exercise entire source code (s)
0.996
0.996
Sn3
Tests cover normal ranges for system (s)
0.5
0.986
Sn4
Tests exercise robustness of system (s)
0.986
0.986
Sn5
No falsification of requirements found for system (s)
0.988
0.988
Sn1
Source code (s) covered by MCDC
0.997
0.997
Sn2
Source code (s) covered by DC

Fig. 4. Generated AC based on the AC pattern library in Fig. 3. Dotted boxes highlight the instantiated pattern. Confidence values when all evidence items are available are shown in green, while red values indicate confidence with missing evidence for Sn4 ("Tests exercise robustness of the system").

converting contract properties (consistency, compatibility, and refinement) into satisfiability problems that are passed to a satisfiability modulo theory (SMT) solver. The algorithm checks that a refinement holds with respect to the system contexts, which are configurations, properties, and operating contexts of the system, encoded as assertions, e.g., "the software component targets the highest safety integrity level," "the wireless communication in component A is disabled." If the validity of the potential refinement is established, then an edge is added between $\tilde{C}$ and $\tilde{N}$ to form an HCN H. The procedure is recursive, i.e., it starts again with the contract network $\tilde{N}$ and terminates when all the leaf contracts are either evidential or undeveloped goals. Since a contract could be linked to more than one contract network, the output of the algorithm, a hierarchy of contracts, is a set of one or more AC candidates that connects the top-level claim to evidence items through multiple argumentation levels. An example output AC candidate is shown in Fig. 4.

3.3 AC Validation and Assessment

Among the AC candidates generated from AC synthesis, our selection process prioritizes candidates that are fully supported by evidence items within the evidence library. In addition, we select a set of AC candidates that are partially supported. For these candidates, we determine the minimum number of additional evidence items required to fully support their claims. We then proceed to assess the confidence level associated with each selected candidate, which accounts for the uncertainty associated with the evidence.

Selecting AC Candidates. AC validation and assessment for a realistic system can be time-consuming, particularly when dealing with a large number of candidates (e.g., 10^6), which can also require significant memory. To address this issue, we employ a depth-first-search approach to select AC candidates that

Algorithm 1: ASSESSCONF(T, $\mathcal{E}$, $\mathcal{W}$, $\mathbf{R}$)

```
input : Top-level refinement $T = (C_r, \varphi_r, N_r)$; library of confidence networks
        $\mathcal{E}$; library of decision rules $\mathcal{W}$; set $\mathbf{R}$ of HCN refinements.
output: Confidence $\mathcal{P}(C_r)$, $\perp$ if assessment fails; infeasibility certificate cert.
1  $cert \leftarrow \perp$ and $\mathcal{P}, Q \leftarrow \perp$
2  for $C \in N_r$ do
3      $R_{curr} \leftarrow \perp$
4      for $R = (C_u, \varphi, N_l) \in \mathbf{R}$ do
5          if $C = C_u$ then $R_{curr} \leftarrow R$; break;
6      if $R_{curr} = \perp$ then
7          $\mathcal{P}(C) \leftarrow$ INFERCONF($C$, $\mathcal{E}$)
8      else
9          $\mathcal{P}(C), certificate \leftarrow$ ASSESSCONF($R_{curr}$, $\mathcal{E}$, $\mathcal{W}$, $\mathbf{R}$)
10         if $\mathcal{P}(C) = \perp$ then return $\perp$, cert
11 $Q \leftarrow$ DECIDECONF ($\mathcal{P}$, $\mathcal{E}$, $\mathcal{W}$)
12 if $Q = \perp$ then return null, cert
13 else return PROPAGATECONF($\mathcal{P}$, $R_{curr}$), cert
```

can be supported with no or minimum number of additional evidence items. We employ the following completion metric to rank and select the AC candidates:

$$Cpl(H) = \frac{\#\text{ of evidential contracts in } H \text{ with no missing evidence}}{\#\text{ of total evidential contracts in } H}. \quad (1)$$

The completion metric measures the percentage of evidential contracts in H that have complete evidence, thus prioritizing AC candidates with higher completion levels. We then compute the confidence values for these AC candidates.

Assessing AC Candidates. Given the selected AC candidates as a set of HCNs and a library $\mathcal{E}$ of confidence networks, we employ a combination of probabilistic and rule-based automated reasoning [19] to quantify the confidence associated with an HCN candidate H as summarized by Algorithm 1. The algorithm recursively traverses H using a depth-first-search approach, propagating confidence values up from lower-level CNs to higher-level ones. Specifically, for each evidential contract encountered during traversal, the algorithm uses one or more corresponding networks in $\mathcal{E}$ to calculate its confidence value. If a contract is not evidential, the confidence value is propagated from its lower-level CN by recursively calling ASSESSCONF. However, if any confidence propagation rule is violated (as evaluated by DECIDECONF), the validation process terminates with a failure. For example, the decision-making step in our algorithm can apply a simple rule requiring that the majority of the premises in an evidential contract must have a high confidence level according to a pre-determined threshold. Further details about the AC validation and decision making processes can be found in our previous publication [19].

Example 2. Given the AC pattern library in Fig. 3 and the top-level claim *"Test cases are sufficient to verify system correctness,"* the synthesis algorithm generates the AC in Fig. 4, where the contract networks defined in *P2* and *P3* refine the two premises of *P1*. The logic validity of the AC is established by checking the compatibility and consistency of each claim and the refinement relationships between the claims. The confidence level can be assessed using Bayesian inference on the confidence networks associated with the pattern, such as the one in Fig. 2b. Confidence assessment produces the color-coded values in Fig. 4.

4 Evaluation

In this section, we present two case studies for arguing the security of (1) the Advanced Fail Safe (AFS) module for the ArduCopter rotorcraft [30] given a set of evidence items [31] and (2) an industrial-level aerospace system.

Advanced Fail Safe Module of ArduPilot. ArduPilot is an open source platform for controlling vehicles including rovers, fixed-wing aircraft, and rotorcraft. It is a software stack that performs estimation, control, and communication between the software and hardware. In this experiment, we focus on the ArduPilot modules used for the ArduCopter [30] rotorcraft. We created a library consisting of 17 patterns that incorporate system development best practices for compliance with DO-178C, DO-333, and vetted security arguments from domain experts. For example, Fig. 6 shows a pattern for arguing specification quality via three supporting premises.

Security of an Industrial Aerospace System. Our approach was also validated on an industrial aerospace system to argue that the system meets certain security requirements. The results are provided in Table 1 under the name *Industrial Case Study*.

4.1 AC Generation Framework in Action

Synthesis. Given a top-level argumentation goal, "the ArduCopter software s is acceptably secure", the pattern library, and a set of desired security properties ℓ, with $|\ell|$ as the number of properties, a total of $3^{|\ell|}$ AC candidates were generated, as shown in Table 1. Fig. 5 compactly represents the set of all possible ACs for an arbitrary $|\ell|$. The top-level goal is supported by the completeness (C_3 of N_1) and correctness (C_4 of N_1) of the software specification, i.e., the set of requirements that must be satisfied, including a set of security properties, the correctness of the software implementation with respect to the security requirements (C_2 of N_1), and the satisfaction of the security properties of interest, e.g., by eliminating or mitigating certain system-specific security hazards, (C_1 of N_1). Parts of the HCN for supporting C_2, C_3, and C_4 of N_1 are omitted due to the limited space.

AC synthesis considers three methods to assess the satisfaction of a security property $p_i \in \ell$ (C_i of N_2), namely, architecture analysis ($N_{6,1}$), model checking ($N_{6,2}$), and static analysis ($N_{6,3}$). These methods are denoted by the conditional refinements ($R_{2,1,1}$, $R_{2,1,2}$, and $R_{2,1,3}$) represented by dotted lines in Fig. 5. Not all the methods can support every security property. AC synthesis only provides

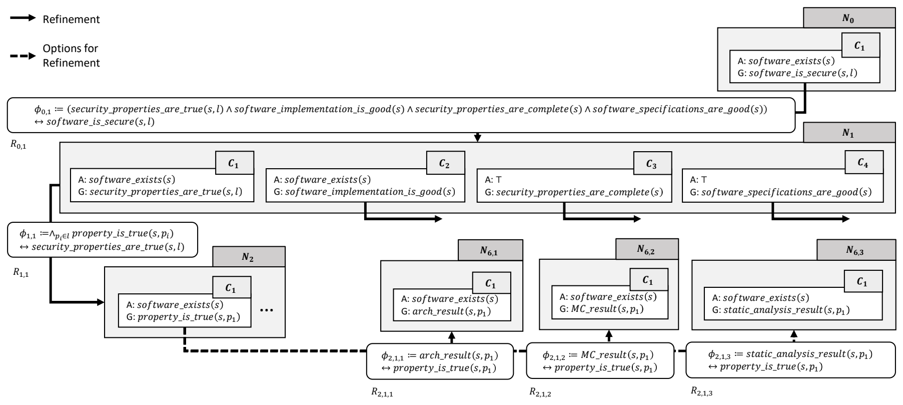
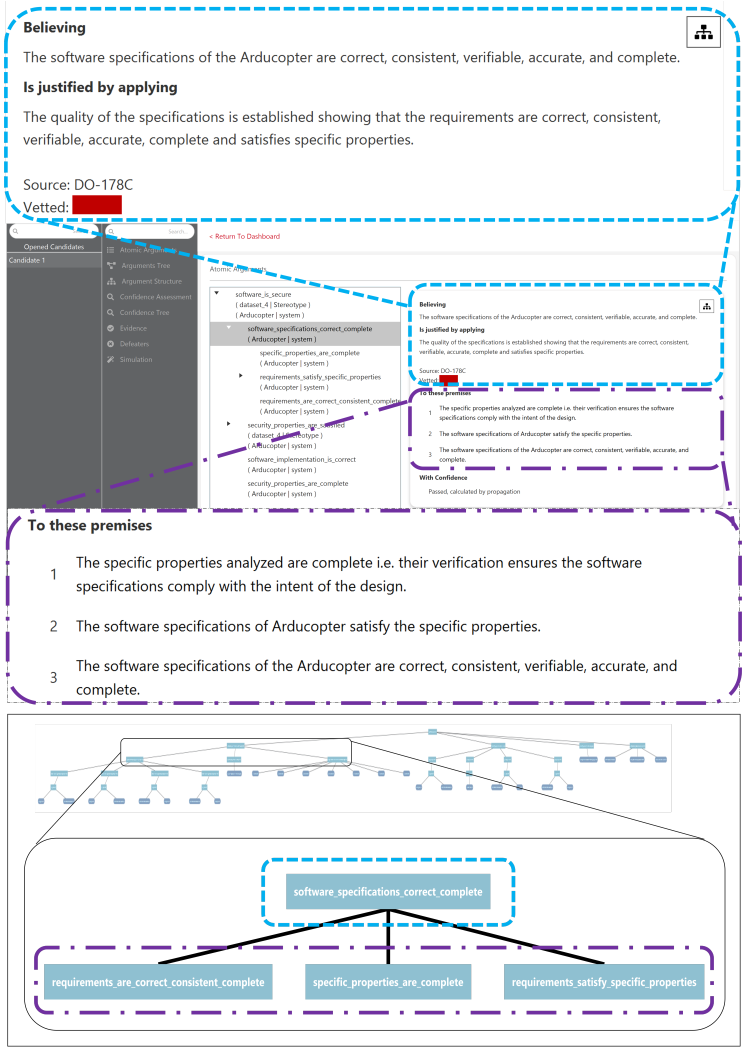

Refinement
Options for Refinement
N_0
C_1
A: $software\_exists(s)$
G: $software\_is\_secure(s,l)$
$\phi_{0,1} \coloneqq (security\_properties\_are\_true(s,l) \land software\_implementation\_is\_good(s) \land security\_properties\_are\_complete(s) \land software\_specifications\_are\_good(s))$
$\leftrightarrow software\_is\_secure(s,l)$
$R_{0,1}$
N_1
C_1
A: $software\_exists(s)$
G: $security\_properties\_are\_true(s,l)$
C_2
A: $software\_exists(s)$
G: $software\_implementation\_is\_good(s)$
C_3
A: T
G: $security\_properties\_are\_complete(s)$
C_4
A: T
G: $software\_specifications\_are\_good(s)$
$\phi_{1,1} \coloneqq \land_{p_i \in l}\, property\_is\_true(s,p_i)$
$\leftrightarrow security\_properties\_are\_true(s,l)$
$R_{1,1}$
N_2
C_1
A: $software\_exists(s)$
G: $property\_is\_true(s,p_1)$
...
$N_{6,1}$
C_1
A: $software\_exists(s)$
G: $arch\_result(s,p_1)$
$N_{6,2}$
C_1
A: $software\_exists(s)$
G: $MC\_result(s,p_1)$
$N_{6,3}$
C_1
A: $software\_exists(s)$
G: $static\_analysis\_result(s,p_1)$
$\phi_{2,1,1} \coloneqq arch\_result(s,p_1)$
$\leftrightarrow property\_is\_true(s,p_1)$
$R_{2,1,1}$
$\phi_{2,1,2} \coloneqq MC\_result(s,p_1)$
$\leftrightarrow property\_is\_true(s,p_1)$
$R_{2,1,2}$
$\phi_{2,1,3} \coloneqq static\_analysis\_result(s,p_1)$
$\leftrightarrow property\_is\_true(s,p_1)$
$R_{2,1,3}$

Fig. 5. Compact representation of AC candidates for arguing the security of the ArduCopter software in HCN form.

candidates whose conditional refinements connect the properties to the appropriate supporting methods. For example, if a multi-rate, quasi-periodic model of computation is adopted, where processes operate periodically at their individual periods and communicate via bounded latency channels, the satisfaction of a property such as “the processing latency for a message between a pair of communicating AFS subsystems is bounded” can be supported by the evidence generated by an architecture modeling and timing analysis tool geared toward the selected model of computation [32]. On the other hand, the security property that “no denial of service occurs due to triggering of nullness exceptions” can be better supported by evidence generated via static analysis. In our example, with a total of $|\ell| = 4$ properties, it took 12 s to generate 3^4 AC candidates (4 properties with 3 options for refinement), effectively capturing all the possible means of compliance and evaluation strategies.

Validation. To test the selection capability of our framework, we designed the ArduCopter case study to only have one fully supported candidate, out of a total of 81 candidates. However, AC validation also retained 9 AC candidates with the highest completion scores to suggest AC candidates requiring the least additional evidence items. The candidate with the highest completion score was successfully validated with confidence 0.9836 [19]. Its assessment, consisting of 24 refinement steps, took 8 s.

Performance Evaluation. We report the performance and scalability of the AC generation framework by increasing the number of total AC candidates from 9 to 6×10^5 in Table 1. We observed a sub-linear increase in the execution time for the generation and confidence assessment of ACs as the total number of AC candidates increased. The time spent for selecting the ACs grew linearly with the total generated candidates. Larger ACs in the industrial case study resulted in greater execution time for confidence assessment. When applied to an


Believing

The software specifications of the Arducopter are correct, consistent, verifiable, accurate, and complete.

Is justified by applying

The quality of the specifications is established showing that the requirements are correct, consistent, verifiable, accurate, complete and satisfies specific properties.

Source: DO-178C
Vetted: 🟥

software_is_secure
(dataset_4 | Stereotype)
(Arducopter | system)
software_specifications_correct_complete
(Arducopter | system)
specific_properties_are_complete
(Arducopter | system)
requirements_satisfy_specific_properties
(Arducopter | system)
requirements_are_correct_consistent_complete
(Arducopter | system)
security_properties_are_satisfied
(dataset_4 | Stereotype)
(Arducopter | system)
software_implementation_is_correct
(Arducopter | system)
security_properties_are_complete
(Arducopter | system)

With Confidence

Passed, calculated by propagation

To these premises

1 The specific properties analyzed are complete i.e. their verification ensures the software specifications comply with the intent of the design.

2 The software specifications of Arducopter satisfy the specific properties.

3 The software specifications of the Arducopter are correct, consistent, verifiable, accurate, and complete.

software_specifications_correct_complete
requirements_are_correct_consistent_complete
specific_properties_are_complete
requirements_satisfy_specific_properties

Fig. 6. Detailed view of pattern justification and its instantiation in the generated AC for the ArduCopter platform.

augmented version of the industrial case study, our framework could synthesize 10^{24} AC candidates in about 2,100 s from a library of 91 patterns. Overall, our generation framework was able to generate over 6×10^5 AC candidates and recommend those with the highest confidence values and completion metrics in less than 100 minutes on an Intel Core i3 CPU with 32-GB RAM.

Case Study	Pattern Library Size	Security Property Count	Total Candidates	Valid Candidates	Average Claims per Candidate	Synthesis (s)	Validation (s)	
							Pruning	Assessment
ArduCopter	17	2	3^2	1	43	10	18	69
	17	4	3^4	1	50	12	21	82
	17	6	3^6	1	58	16	26	92
	17	8	3^8	1	66	18	41	102
	17	10	3^{10}	1	73	21	152	119
	17	12	3^{12}	1	81	25	3,143	832
Industrial Case Study	91	N/A	6×10^5	10	652	819	1,683	3,322

Table 1. Performance of the AC generation framework for the ArduCopter software and an industrial-level aerospace system.

5 Conclusion

We presented a framework for computer-aided 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. Empirical results show that our framework is able to efficiently generate AC candidates for representative real-world systems in a scalable manner. We intend to improve it by normalizing evidence items from different sources, by ranking candidates using a predictive cost model, and by introducing mechanisms for the elicitation of confidence models. While generating assurance patterns and confidence models may require significant initial effort, we also aim to investigate methods to alleviate this burden by providing templates for computer-aided pattern elicitation, formalization, and validation.

References

1. C. Holloway, *Understanding the overarching properties*, ser. NASA technical memorandum, 2019.
2. J. Rushby, "The interpretation and evaluation of assurance cases," Computer Science Laboratory, SRI International, Menlo Park, CA, Tech. Rep. SRI-CSL-15-01, Jul. 2015.
3. T. A. C. W. Group, "Goal structuring notation community standard (version 3)," 2021.
4. R. Hawkins, I. Habli *et al.*, "Weaving an assurance case from design: a model-based approach," in *Int. Symp High Assurance Systems Engineering*, 2015, pp. 110–117.
5. J. Rushby, "Formalism in safety cases," in *Making Systems Safer*. Springer, 2010, pp. 3–17.
6. R. Bloomfield and J. Rushby, "Assurance 2.0," *arXiv preprint arXiv:2004.10474*, 2020.
7. E. Denney, G. Pai, and J. Pohl, "AdvoCATE: An assurance case automation toolset," in *Int. Conf. on Computer Safety, Reliability, and Security.* Springer, 2012, pp. 8–21.
8. M. R. Barry, "CertWare: A workbench for safety case production and analysis," in *Aerospace conference*, 2011, pp. 1–10.

9. Y. Matsuno, “D-case editor: A typed assurance case editor,” *University of Tokyo*, 2011.
10. Adelard LLP, “Assurance and safety case environment (ASCE),” 2011. [Online]. Available: https://www.adelard.com/asce/
11. I. Sljivo, B. Gallina *et al.*, “Tool-supported safety-relevant component reuse: From specification to argumentation,” in *Ada-Europe International Conference on Reliable Software Technologies*. Springer, 2018, pp. 19–33.
12. I. Šljivo, G. J. Uriagereka *et al.*, “Guiding assurance of architectural design patterns for critical applications,” *Journal of Systems Architecture*, vol. 110, p. 101765, 2020.
13. J. L. de la Vara, A. Ruiz, and G. Blondelle, “Assurance and certification of cyber-physical systems: The AMASS open source ecosystem,” *Journal of Systems and Software*, vol. 171, p. 110812, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220302120
14. D. Nešić, M. Nyberg, and B. Gallina, “Product-line assurance cases from contract-based design,” *Journal of Systems and Software*, vol. 176, p. 110922, 2021.
15. R. Neapolitan, *Learning Bayesian Networks*, ser. Artificial Intelligence. Pearson Prentice Hall, 2004.
16. C. Cârlan, V. Nigam *et al.*, “ExplicitCase: Tool-support for creating and maintaining assurance arguments integrated with system models,” in *International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 330–337.
17. S. Ramakrishna, C. Hartsell *et al.*, “A methodology for automating assurance case generation,” *arXiv preprint arXiv:2003.05388*, 2020.
18. P. J. Graydon and C. M. Holloway, “An investigation of proposed techniques for quantifying confidence in assurance arguments,” *Safety Science*, vol. 92, pp. 53 – 65, 2017.
19. C. Oh, N. Naik *et al.*, “ARACHNE: Automated validation of assurance cases with stochastic contract networks,” in *Computer Safety, Reliability, and Security*. Springer, 2022, pp. 65–81.
20. C. M. Holloway, “Explicate’78: Uncovering the implicit assurance case in DO-178C,” *Safety-Critical Systems Symposium*, 2015.
21. The GSN Working Group Online, *Goal Structuring Notation*. [Online]. Available: http://www.goalstructuringnotation.info/
22. Adelard LLP, *Claims, Arguments and Evidence (CAE)*, 2019 (accessed October 23rd, 2020), https://www.adelard.com/asce/choosing-asce/cae.html.
23. H. Fujita, Y. Matsuno *et al.*, “DS-Bench toolset: Tools for dependability benchmarking with simulation and assurance,” in *International conference on dependable systems and networks*. IEEE, 2012, pp. 1–8.
24. A. Benveniste, B. Caillaud *et al.*, “Contracts for system design,” *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.
25. S. S. Bauer, A. David *et al.*, “Moving from specifications to contracts in component-based design,” in *Fundamental Approaches to Software Engineering*. Springer, 2012, pp. 43–58.
26. A. Gacek, J. Backes *et al.*, “Resolute: An assurance case language for architecture models,” in *SIGAda annual conference on High integrity language technology*, 2014, pp. 19–28.
27. T. E. Wang, Z. Daw *et al.*, “Hierarchical contract-based synthesis for assurance cases,” in *NASA Formal Methods*. Springer, 2022, pp. 175–192.
28. F. V. Jensen, *Introduction to Bayesian Networks*, 1st ed. Springer, 1996.
29. 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.

30. ArduPilot Dev Team, “Arducopter,” 2023. [Online]. Available: https://ardupilot.org/copter/
31. N. Shankar, D. Bhatt *et al.*, “DesCert: Design for certification,” *arXiv preprint arXiv:2203.15178*, 2022.
32. D. Bhatt, H. Ren *et al.*, “Requirements-driven model checking and test generation for comprehensive verification,” in *NASA Formal Methods*. Cham: Springer International Publishing, 2022, pp. 576–596.