Presented at 42nd Digital Avionics Systems Conference (DASC), Barcelona, Spain, October 2023

CLARISSA: Foundations, Tools & Automation for Assurance Cases

Srivatsan Varadarajan[1], Robin Bloomfield[25], John Rushby[3], Gopal Gupta[4],
Anitha Murugesan[1], Robert Stroud[2], Kateryna Netkachova[2], Isaac Hong Wong[1]
[1]Honeywell Aerospace, *srivatsan.varadarajan@honeywell.com*
[2]Adelard (NCC), *robin.bloomfield@nccgroup.com*
[3]SRI International, *rushby@csl.sri.com*
[4]University of Texas at Dallas, *gupta@utdallas.edu*
[5]City, University of London *r.e.bloomfield@city.ac.uk*

***Abstract*—Assurance cases are gaining traction as a means of certification in Aerospace and other safety and security critical industries. In this paper, we first introduce a rigorous Assurance 2.0 framework that eliminates ad-hoc construction of assurance cases with emphasis on the validity and soundness of the argumentation process, confidence of the claims/arguments/evidences and the systematic exploration of defeaters. We next describe the tools and automation support for Assurance 2.0 that was developed in the Clarissa project and finally highlight the key capabilities in the context of an illustrative example.**

***Index Terms*—Assurance Case, Certification, Property-based Compliance, Integrated Security and Safety Analysis.**

I. Introduction

Certification in regulated industries like Aerospace, Nuclear, Automotive, etc., has traditionally relied on a *compliance driven approach* with highly *prescriptive requirements* on system development processes to guarantee both safety and security. For example, the entire avionics development is regulated through ARP-4761 [1] for safety, DO-326A [2] for security, and ARP-4754A [3] for system development via guidelines like DO-178C [4] for software and DO-254 [5] for hardware. Notwithstanding a proven track record, although such a compliance-driven approach offers clarity to stakeholders on exact criteria to meet, the quality of compliance and the degree of confidence in the system is hard to judge. The rationale for prescriptions is hidden from the system developers, while the detailed design knowledge imbalance is skewed negatively against the regulators. This hinders adoption of development innovations that improve system flexibility and predictability, which could lower costs and time incurred in certification phase that have historically dominated the product life-cycle.

In contrast, we argue that an *assurance case approach*, which is more *objective, outcome-driven, property-based, automated, and systematic*, can overcome these limitations and lower the safety and security risks posed by the system over its lifetime. In this paper, we highlight our novel contributions to assurance case foundations and their realizations in tools developed under the *Consistent Logical Automated Reasoning for Integrated System Software Assurance (CLARISSA)* project for the DARPA Automated Rapid Certification Of Software (ARCOS) program [6]. Clarissa assurance cases are built upon a more rigorous and demanding Claims Argument Evidence (CAE) [7] methodology called *Assurance 2.0* [8]. This simplifies the development and assessment of cases because issues that were previously treated in an ad-hoc manner and were subject to contention, misinterpretation and challenges are now made explicit and treated systematically.

Clarissa tools provide the Assurance 2.0 foundational building blocks and automation support that ensures focus on *positive claims* while actively searching for negative *defeaters* that could invalidate a claim and *propagate refutations* through the case. Evidence is weighed deliberately using *confirmation measures* that carefully distinguish between facts established by the evidence (*claims about something measured*) and inferences drawn from it (*claims about something useful*). Assurance 2.0 requires that the completed assurance case should be *indefeasible* whereby no credible new information would change the judgment and evaluation of the case i.e., no unresolved doubts and defeaters.

Assurance 2.0 simplifies the assessment of assurance cases by being clear about what is developed within the assurance argument and what is referenced and integrated by it through the application of external *theories*, *models* that describe system behaviors (e.g. requirements, code architectures, etc.), and *evidence assemblies*. Theories are reusable assurance sub-case templates with self-contained justifications, which provides an opportunity for well-established theories to be *pre-certified* based on criteria for their correct instantiations. Further, we define a simple formalism for Claims, Evidence, and Theory specifications to overcome ambiguities associated with natural language and allow for automated semantic checks (e.g., consistency and completeness) and reasoning. Theories reduce cognitive load and improve understanding and efficiency by opening the door to assurance case synthesis. The overall assurance case can easily be built in an error-free manner, within Clarissa, by instantiating various theories along with integrating arguments and substantiating evidence.

To demonstrate the practicality of Clarissa's approach and tools, we use the ArduCopter system, an open-source platform, as an illustrative case example. The objective is to construct an assurance case that establishes justifiable confidence in the flight-critical ArduCopter software's fitness to carry out autonomous surveillance missions within geofenced areas while meeting safety and security criteria. The structure of the assurance case is based on an *Overarching Properties (OP)* [9] approach that satisfies three primary objectives: Intent, Correctness, and Acceptability. OP is gaining FAA recognition as an alternative means of compliance, especially for next-generation AI/ML-based avionics and autonomous software, which do not lend themselves to certification using only traditional DO-178C processes.

II. Assurance 2.0 Foundations

Assurance is the process of collecting evidence about a system and its environment and developing claims and an argument that use these to justify (or reject) deployment of the system; an assurance case is a way of organizing and presenting this information, together with other relevant facts and knowledge in a manner that facilitates overall comprehension and assessment.

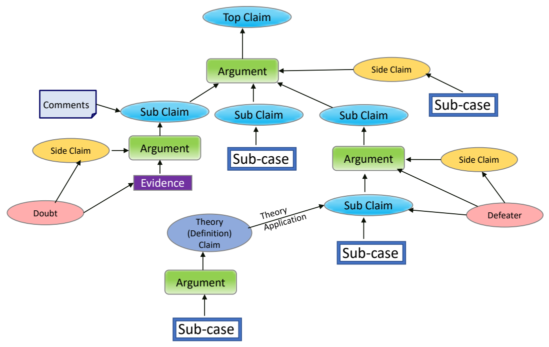


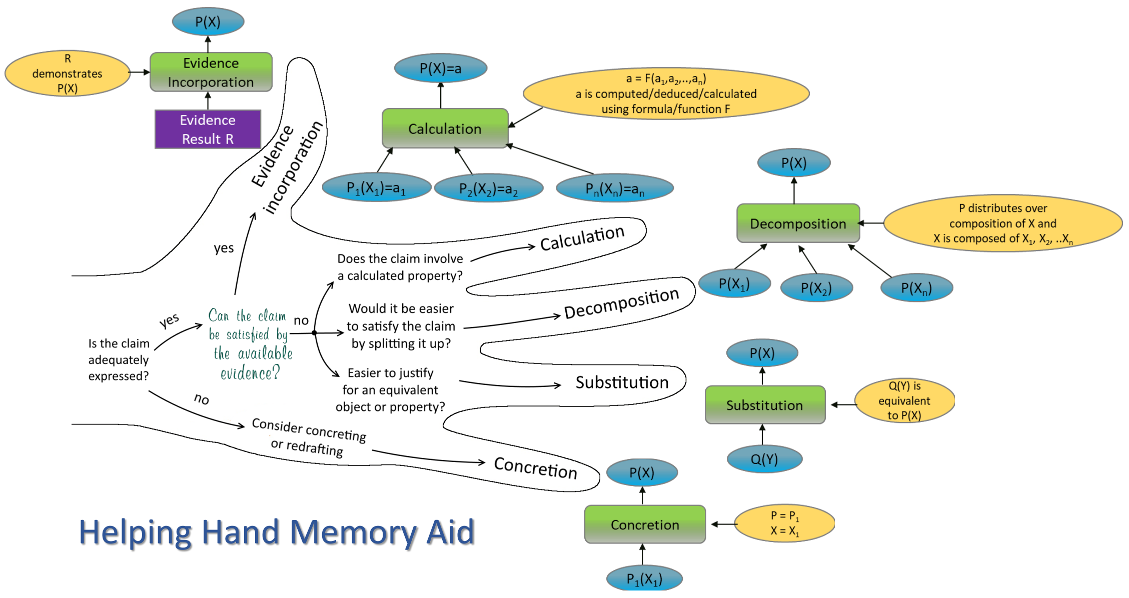
Fig. 1. Example Clarissa Assurance Case

Overview: Assurance cases in Clarissa follow the approach presented as Assurance 2.0 [8], [10], which builds on the earlier "Claims, Arguments, Evidence" or CAE [7], where the main component of an assurance case is a structured argument represented as a tree of claims linked by argument steps, and grounded on evidence. A pictorial example illustrating various concepts discussed in this section is shown in Figure 1. Assurance 2.0 emphasizes *"simplicity through rigor"* whilst constructing assurance cases that comprise arguments, models, theories, evidence assemblies:

- *Models*: Descriptions of behavioral attributes or properties of the system from some point of view (timing, power, functional) at some level of abstraction. Includes requirements, architecture and code.
- *Theories*: Definition, analysis, and evaluation of individual topics, as assurance sub-cases, that justify some claims and allows compositional assurance reasoning. e.g., hazard analysis, MC/DC testing[11], formal verification, static code analysis.
- *Evidence and its assembly*: Evidence must support a claim, otherwise it is just data. Individual items often need assembly to create evidence e.g., tests, assumptions, tooling, provenance, and analysis.
- *Structured arguments* over claims: tie the models, theories and evidence together using a CAE approach. The underlying argument interpretation is *Natural Language Deductivism (NLD)* which involves: (i) Informal application of formal deductive logic whereby sub-claims deductively entail their parent claim as with logical reasoning in mathematics, science, and engineering and (ii) Arguments are both *valid* and *sound*. Argument validity simply requires that all the reasoning steps "fit together" correctly making their claims logically true. Arguments are sound if all steps are well justified and can be accepted as correct/reasonable: i.e. when all evidence incorporation steps cross some threshold for credibility, and all interior or reasoning steps have *indefeasible* justifications. Narrative justifications for soundness are provided for each argument step, often supported by *side-claims* that establish necessary assumptions. Narrative justifications use natural language but may reference some external theory, calculation, proof, or mechanized analysis with tools.

Clarissa Building Blocks: An argument within assurance 2.0 has rather limited scope and can take a restricted form with just a few basic steps for structuring and organizing references to external models, theories and evidence. Thus, Clarissa uses only five different kinds of argument steps and these are called (building) blocks [12], [13], [8] and they are utilized, with side-claims justifications as shown in Figure 2, and comprise:

1) *Concretion*: support one claim (e.g., "is correct") by another more precise, measurable and less abstract claim (e.g., "satisfies DO-178C")
2) *Substitution*: support claim property P about object X by a similar claim property Q about object Y.



P(X)
R demonstrates P(X)
Evidence Incorporation
Evidence Result R
P(X)=a
a = F(a1,a2,..,an) a is computed/deduced/calculated using formula/function F
Calculation
P1(X1)=a1
P2(X2)=a2
Pn(Xn)=an
P(X)
Decomposition
P distributes over composition of X and X is composed of X1, X2, ..Xn
P(X1)
P(X2)
P(Xn)
Evidence incorporation
yes
Does the claim involve a calculated property?
Calculation
Is the claim adequately expressed?
yes
Can the claim be satisfied by the available evidence?
no
Would it be easier to satisfy the claim by splitting it up?
Decomposition
Easier to justify for an equivalent object or property?
Substitution
P(X)
Substitution
Q(Y) is equivalent to P(X)
Q(Y)
no
Consider concreting or redrafting
Concretion
P(X)
Concretion
P = P1 X = X1
P1(X1)
Helping Hand Memory Aid

Fig. 2. Clarissa Building Blocks

Usually, either X and Y or P and Q are held constant. For example, MC/DC testing is "equivalent" to demonstrating absence of unreachable code.

3) *Decomposition*: claim that a conclusion about the whole object, process, property or function can be deduced from the claims or facts about constituent parts including potentially over time. For example, if a property holds for all sub-systems in a system then the property holds for the overall system.
4) *Calculation*: claim that the value of a property of a system in a given environment can be computed from the values of related properties of other objects in that environment. E.g., average data retrieval time from a database can be calculated from the probability that the data is in the cache (property of one sub-system) and the time of data retrieval if it is not in the cache (property of the other sub-system).
5) *Evidence Incorporation*: supports claims of *"something measured" then "something useful"*. It is used at the edge of the case tree (leaf nodes) to incorporate the evidence elements and demonstrates that a sub-claim is directly satisfied by its supporting evidence.

Evaluation and Assessment of Assurance Cases: An assurance case is intended to provide justifiable confidence in the truth of its top claim, which typically concerns safety or security. A natural question is then "how much" confidence does the case provide? We argue that confidence cannot be reduced to a single attribute or measurement. Instead, we suggest it should be based on attributes that draw on three different perspectives:

(i) *Positive Perspective*: considers extent to which the evidence and overall argument make a positive case to justify belief in its claims. *Primary positive measure* is *soundness*, which interprets the argument as a logical proof by NLD and delivers a yes/no measurement based on logical validity (checkable) plus credibility of evidence and reasoning. Credibility of evidence is "weighed" by *confirmation measures* based on (possibly qualitative) subjective probabilities indicating our confidence in the claim, given the evidence. Weights for each evidence assembly should cross some appropriate threshold and narrative justifications should establish the soundness of each reasoning step. Defeaters are used to indicate when these are considered inadequate.

Secondary positive measure is *probabilistic confidence* or *doubt* (i.e., 1 - probabilistic confidence) which indicate how *strongly* we believe the assessment of soundness. These can be aggregated from evidence through the steps of the argument using various kinds of probability logic. Their purpose is to support *graduated assurance* where less costly (and presumably weaker) evidence and argument are used for elements that pose less risk (as with the Design Assurance Levels - DALs of DO-178C).

(ii) *Negative Perspective*: involves active search for and resolution of *doubts* and *defeaters*. We refer to any concern about a case as a doubt; as we explore the origin and nature of a doubt we will refine it to a defeater that invalidates a claim through a specific (counter)claim or challenge that can be attached to a particular point in the argument. A defeater to an assurance case is rather like a hazard to a system: that is a conjecture why things might go wrong. This defeaters-cognizant approach ensures positive perspective does not become optimistic and helps guard against *confirmation bias*.

Defeaters can be investigated via their own assurance sub-cases. Those that are sustained indicate something is wrong with the primary case or with the system itself, and these must be repaired. Refuted defeaters are retained as commentary on the case that can help evaluators or subsequent developers, who may have similar doubts of their own. Assurance 2.0 values quality and diligence of defeater nomination and exploration, but not their sheer quantity.

Defeaters can also be employed for *eliminative argumentation* [14], [15], which is a contrary approach to assurance, favored by some, that uses a negative argument to refute all reasons why the top claim could be false (i.e., rather than show the system is safe, we show that it is not unsafe).

(iii) *Residual Risk Perspective*: It is possible that not all doubts and defeaters are fully resolved at the time of decision by the evaluators. In this consideration, we explore consequences and likelihoods of unresolved doubts and defeaters and thereby assess risk (their product). Some of these residual risks may be judged unacceptable and thereby prompt a review of the evidence, argument, or system (e.g., additional mitigation), but others may be considered acceptable or unavoidable. We support detailed examination of these issues and can propagate probabilistic valuations in several ways to assist different stakeholders. The purpose is not to deliver a verdict but to support and record thoughtful assessment.

Confidence in Evidential Support for Claims: The argument of an assurance case is *logically valid* when all its leaf claims are supported by evidence and the claims supporting and delivered by interior steps "match up" to provide a coherent graph structure. We say the argument is *fully valid* when, in addition, all its steps are (judged to be) deductive, and there are no unresolved defeaters other than those marked as residual risks. A fully valid argument is *sound* when human evaluators attest that the residual risks are negligible, that the justification for each interior step is indefeasible, and that the weight of each evidential step is sufficient to justify its *"something useful"* claim.

Confirmation Measures are attached to Evidence Incorporation argument blocks to indicate weight of confidence in evidence justifying leaf level claims in a CAE graph. A natural measure of confidence in a claim C given the evidence E is the subjective posterior probability $P(C|E)$, which may be assessed numerically or qualitatively (e.g., "low", "medium", or "high"). However, confidence in the claim is not the same as confidence that it is justified by the evidence. For the latter, while we explore many possibilities in [10], we limit to primarily two confirmation measures:

1) $Keynes(C,E)$: how much does the evidence increase my confidence in the claim indicated by $log\frac{P(C|E)}{P(C)}$
2) $Good(C,E)$: how well does this evidence distinguish between the claim and counterclaim indicated by $log\frac{P(E|C)}{P(E|\neg C)}$

Confidence Propagation: the assessment criterion or "stopping rule" for an assurance case in Assurance 2.0 is indefeasible soundness (which includes full validity). Indefeasibility means we must be confident that no new information would change our assessment or, in other words, that we have thought of everything. But even so, we must have some threshold on the evidence and reasoning that we use to show that everything has been dealt with satisfactorily: e.g., if we use testing to establish some claim, how much testing is enough? In Clarissa, we use subjective probabilistic assessments of *confidence* to record and propagate these judgements.

The purpose of confidence is not to question or reinforce indefeasibility, but to help apportion effort suitably. Confidence is propagated from evidence, and we may generally assume that more confidence requires more or better evidence and that more or better evidence costs more money and effort. Thus, within a single argument, we will generally want to see approximately equal confidence across all claims or, perhaps, an allocation according to risk. Across different cases, we will likely want to see confidence calibrated according to perceived risk posed by the system concerned, thereby supporting *graduated assurance* as seen in the DALs of DO-178C.

As mentioned above, when assessing soundness of evidential claims we use a confirmation measure $Keynes(C,E)$ or $Good(C,E)$ rather than the posterior probability $P(C|E)$ because we wish to evaluate the discriminating power, or "weight" of the evidence, and confirmation measures do this. But once we have assessed soundness, it is reasonable to use the posterior as our measure of probabilistic confidence in the claim C and it is this that will be propagated through the probabilistic valuation of the case. The propagation rule we favor is *sum of doubts* which estimates the *probabilistic doubt* (i.e., 1 - probabilistic confidence) of a parent claim as the sum of probabilistic doubts over its sub-claims and side-claim. This is conservative but requires only weak assumptions. More detailed discussions are available in [10] for different methods of probabilistic confidence propagation and their application to each of the five Clarissa argument building blocks.

Theories: Definition and their Application: Science and engineering are fundamentally built around *theories* and *models* and assurance cases constructed in Clarissa follows that approach. In assurance of safety/security critical aerospace systems, we tackle many different topics and use different models with different techniques and different justifications, e.g., hazard analysis, requirements based testing, structural code coverage, reliability growth, human reviews, tracing, static analysis, architecture, time-space partitioning, fault-tolerance, etc. Many of these topics are relatively self-contained and can have their own assurance theory that includes model plus technique with underlying justification. Theories are essentially an *assurance sub-case*, defined as a *reuseable template* but with *semantics* and *justifications*, that can be applied to various assurance cases. The motivation is that an overall assurance case for a complex system can then mostly be built by instantiating various theories plus integrating arguments. Theories then provide a method to control the size of the argument through *compositional reasoning* and aids development of the overall assurance case in a *modular* fashion. Self-contained models and theories, with objective analysis that are better developed, analyzed, justified, and evaluated (i.e., pre-certified) separately without cluttering the main assurance case, can be separately managed by experts and presented and assessed by the scientific and engineering methods traditional to their fields. This approach is akin to function definitions and function calls in computer programming.

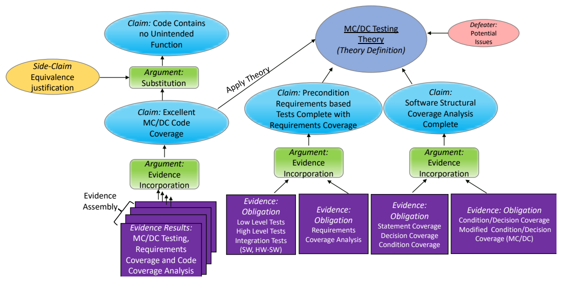

Claim: Code Contains no Unintended Function
Side-Claim Equivalence justification
Argument: Substitution
Claim: Excellent MC/DC Code Coverage
Apply Theory
MC/DC Testing Theory (Theory Definition)
Defeater: Potential Issues
Claim: Precondition Requirements based Tests Complete with Requirements Coverage
Claim: Software Structural Coverage Analysis Complete
Argument: Evidence Incorporation
Argument: Evidence Incorporation
Argument: Evidence Incorporation
Evidence Assembly
Evidence Results: MC/DC Testing, Requirements Coverage and Code Coverage Analysis
Evidence: Obligation Low Level Tests High Level Tests Integration Tests (SW, HW-SW)
Evidence: Obligation Requirements Coverage Analysis
Evidence: Obligation Statement Coverage Decision Coverage Condition Coverage
Evidence: Obligation Condition/Decision Coverage Modified Condition/Decision Coverage (MC/DC)

Fig. 3. MC/DC Testing Theory Example

The properties that can directly be stated and observed about the system and its low-level models (e.g., percentage of objects correctly identified by a vision system under test) will generally be far removed from the properties about which we seek assurance: those will generally concern emergent properties stated about highly abstract models (e.g., safety of an autonomous car). It follows that a central task of an assurance case is to connect properties of the system and its low level models to those of high level models, and this is accomplished by argument steps that iterate through a series of intermediate models that generally align with steps in the design and development of the system, as in the classical "V" Diagram. A typical step of this kind will seek to justify, which is an opportunity for a "*theory*" that a property A of some model P ensures property B of a next higher model Q.

An example theory is that underlying Modified Condition/Decision Coverage (MC/DC) for requirements-based tests[11]: such a *"MC/DC Testing Theory"* must explain this method of testing and coverage evaluation, how is it performed, why is it useful, what issues need to be considered, and what claims it can support as shown in Figure 3. The theory could then be used and applied within a context of side-claim to the parent argument which explains how MC/DC coverage of executable code can justify a claim that the code contains no unintended functions. Alternatively, we could have *composed* another theory for side-claim justification *layered* along with MC/DC Testing Theory. When applying a theory defined elsewhere (in the current assurance case or outside of it), the argument, where theory is (re)used, must provide justification that it is suitable and credible, and that it is applied appropriately. Theory and its definition is not required to be present as part of the argument: it merely references the previously defined theory and instantiates it for a particular application within the context of the arguments being made in the given assurance case. For example, the MC/DC theory may be referenced within an assurance case demonstrating compliance to DO-178C DAL C structural code coverage objectives.

Defeaters and their beneficial utility: The primary criterion for a satisfactory assurance case in the Assurance 2.0 methodology is that it should justify *indefeasible confidence* in its top claim, meaning that in addition to confidence that the claim is true, we must also be confident that there are no *overlooked or unresolved defeaters* that could change that judgement. A *doubt* or *defeater* node indicates concern about an argument and contains a claim indicating the nature of the concern (e.g., "I think there is something wrong here") and points to the argument node in question. At some point, we must return to investigate the nature and origin of the doubt and will either dismiss it as unwarranted, or refine and sharpen it into a *defeater* with a possibly more specific counter-claim (e.g., "the justification for this step is inadequate") and whose investigation is recorded in a sub-case attached to the defeater. Thus, a doubt is simply a defeater that has not yet been investigated (i.e., has no sub-case) and so we will generally refer to both as defeaters. Detailed treatment of defeaters can be found elsewhere [15], [10] but we briefly summarize it here.

Investigation and resolution of defeaters serves two purposes: firstly, it is the primary means to avoid confirmation bias and drive the case toward soundness and indefeasibility; secondly, it helps reviewers comprehend the case as they find their own doubts have been anticipated and answered. Furthermore, recording defeaters and their resolutions in an active manner for an evolving assurance case indicates: *developer diligence* through defeaters proposed and examined during development and *assessor diligence* during assessment of the refuted defeaters and residual risks posed by sustained defeaters.

If a defeater is supported by an assurance sub-case that is adjudged to be sound, so that its claim is true, then the defeater is said to be confirmed or *sustained* and the primary case, and possibly the system it is about, must be modified to overcome the flaw that has been identified. The defeater sub-case can be retained as commentary in the revised primary case, but it should no longer be sustained and may be hard to interpret. Instead, we suggest it should be adjusted to become a refuted defeater (see below) for the revised case. Alternatively, the sustained defeater may be explicitly accepted as a residual doubt, provided it is judged suitably insignificant and retained with commentary in the original primary case.

If a defeater is a *false alarm* or is one that has been overcome by modifications to the original case (as above), then its sub-case should be *refuted* and the primary case can remain unchanged. One way to refute a defeater is to provide a second defeater that targets the first one or some part of its sub-case. If the assurance sub-case for that second defeater is adjudged to be good, then the first defeater is said to be refuted and it and its sub-case play no part in the interpretation of the primary case, but both first and second level defeaters can be retained as a commentary to assist future developers and evaluators who may entertain doubts similar to those which motivated the original defeater. Another way to initiate refutation of defeater is by means of *counter-evidence*: that is, evidence that contradicts the claim it is meant to support. We have extended Clarissa tools to include *refutational reasoning* that handles defeaters at several levels.

Prior to the introduction of defeaters, Clarissa considered only positive arguments and did so using NLD. Assessment of logical validity and soundness is fairly straightforward in such cases. But this determination becomes a little more complicated when we have to account for defeaters, and also wish to allow incompletely developed arguments. In these cases, we cannot always expect the top claim to be adjudged true; instead, we can ask which claims are *true* and which are *unsupported* (i.e., have contested or incomplete sub-cases), and we can do this by propagating logical assessments upward from the leaf nodes. In refutational reasoning with defeaters, we need to consider the additional possibility that claims may be *false* in order to handle counter-evidence and multi-level defeaters. The Clarissa tools use a three-valued logic to support these analyses.

Assurance 2.0 also allows support for *eliminative argumentation* [16] whereby instead of allowing argumentation to *sustain a positive claim*: e.g., "system is safe", we instead attempt to *refute a negative claim*: "system is unsafe". A successful refutation will establish the negation of that claim, namely "the system is not unsafe". As an approach to arguing that claim, one could consider all reasons the system could be unsafe, and then provide arguments to eliminate each one. This encourages a more skeptical viewpoint and reduces confirmation bias. Eliminative Argumentation requires similar reasoning to refutation of defeaters as discussed above. To do this, we introduce *exact defeaters*, which a) point to a node that is either a claim or another defeater that is b) lacking a subcase, and c) whose own claim is the *negation* of the one pointed to. The previous kind of defeater is now called an *exploratory defeater* and differs from an exact defeater in that its claim need not negate the claim it points to (and it may point to nodes other than claims and defeaters), but merely call it into question. Clarissa's support for refutational reasoning is extended to deal with exact defeaters, and its notation is augmented by introduction of a new *disjunctive decomposition argument block* to complement the *conjunctive decomposition argument block* discussed earlier as one of the five building blocks. Figure 4 illustrates a lot of the defeaters concepts through an example.

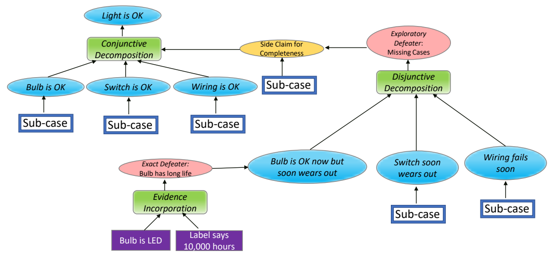


Fig. 4. Example Illustrating Defeaters

III. Clarissa Tools and Automation

Clarissa Tools Architecture is shown in Figure 5, which consists of: (i) *Assurance and Safety Case Environment (ASCE)* [12] which is the most widely adopted commercial software for the creation and management of safety and security assurance cases, and (ii) a goal-directed top-down *solver for Constraints Answer Set Programs s(CASP)* [17], [18] for reasoning about assurance cases using an enhanced *Prolog* engine. ASCE has full support of Assurance 2.0 framework and enforces the methodology while it also facilitates systematic creation of Assurance 2.0 cases leveraging theories, ensuring the validity and soundness of the logical arguments with justifications while enabling active search for defeater and either sustaining or refuting them. Libraries of theories and defeaters are maintained as active repository of knowledge and known vulnerabilities. ASCE also performs *structural analysis* to ensure their *correct and complete* construction while automatically analyzing specific *syntactic* elements of assurance cases including adherence to notations, grammar/spell-checks within natural language descriptions. ASCE also automatically converts the assurance case to an *equivalent logic program* to support systematically reasoning with the s(CASP) engine. The s(CASP) engine reasons over the *semantics* or underlying meaning of the claims, arguments, and evidence presented in assurance cases which includes various properties of the assurance case such as *consistency* (i.e. absence of logical contradictions), *indefeasibility* (i.e. absence of defeaters) and *completeness* (i.e. state of encompassing all the requisite elements), etc.

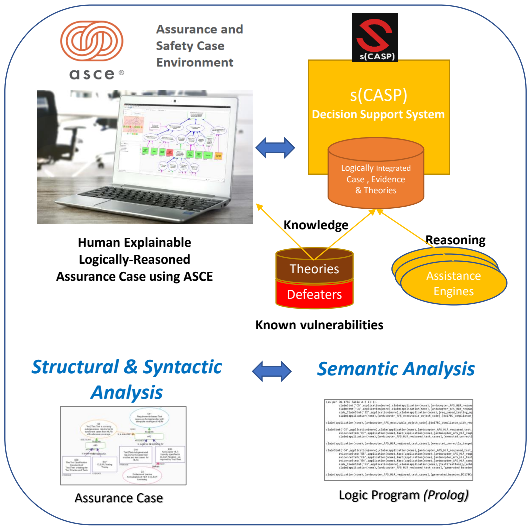

asce
Assurance and Safety Case Environment
s(CASP)
s(CASP) Decision Support System
Logically Integrated Case, Evidence & Theories
Knowledge
Reasoning
Theories
Defeaters
Assistance Engines
Human Explainable Logically-Reasoned Assurance Case using ASCE
Known vulnerabilities
Structural & Syntactic Analysis
Semantic Analysis
Assurance Case
Logic Program (Prolog)

Fig. 5. Clarissa Tools Architecture

Claims-Evidences Formalisms with Prolog Export: To a large extent, assurance cases traditionally rely primarily on free-form natural language to document claims, arguments and evidence. Hence, despite being well-structured and syntactically correct, ensuring semantic correctness (the top-level claim logically follows from its sub-claims, arguments, and evidence), and that there are no logical inconsistencies or fallacies is intellectually demanding for both authors and evaluators, due to the inherent ambiguity, and inconsistency of natural language. Consequently, automating semantic reasoning would greatly reduce human effort and enhance the quality and confidence in these cases. While free-form natural language is not conducive to such automation, fully formal notations are impractical to represent the vagaries of claims-arguments-evidences descriptive texts. Hence, we define a balanced approach that presents details in an intuitive and *"minimally"* formal way that is also easily *extensible*. We take a two-step approach[19], where we first categorically specify the terms used in the descriptions – in terms of *objects* of the system, *properties* they possess, and optional *environments* in which the properties of the given objects are valid. e.g., the claim: *"software is correct w.r.t requirements"* can be expressed in the object-property-environment formalism as *object(software), property(correct_wrt_requirements), environment(env)* and *claimstmt(software,correct_wrt_requirements,env))*. Then, in the second step, the assurance case is transformed into a logic program notation (see bottom of Figure 5), that can be subject to various formal analyses at the back-end using the s(CASP) system. Every Assurance 2.0 concept in an assurance case can be mapped to an s(CASP) constructs. e.g, $Argument \equiv Rule$, $Claim \equiv Predicate$, $Evidence \equiv Fact$, $Justification \equiv Proof$, $Defeaters(variants) \equiv classical\, negation(\neg)\, or\, negation\, as\, failure\, to\, prove(not)$, $Validity \equiv Querying$. The claim/evidence language formalisms and the assurance case transformation to logic programs are detailed in [19].

Automated Theory and Defeater Synthesis: Initially when architecting a case from scratch or regenerating a case with changes, ASCE provides an *expert guided synthesis approach* for applying pre-defined theories and associated defeater templates. The idea is that we first define specific set of theories with structure that includes: (i) theory preconditions (ii) evidence obligations (iii) defeater templates and (iv) all claims/evidences having object-property-environment *types*. We next define, within the assurance case, specific claim where the desired theory needs to apply to, with appropriate object-property-environment *instances*. ASCE then supports ability to synthesize sub-case of theory application while ensuring object-property-environment instances match with the appropriate types specified in the theory definition and also populates the associated potential defeaters associated with such a sub-case that need to be addressed. This simplifies the construction of assurance cases from reusable theories and defeaters through sub-case templates.

IV. Illustrative Example

To evaluate and demonstrate the concepts and capabilities of our approach we use the open-source Arducopter flight system [20] platform as a case study. Arducopter is an advanced, and reliable autopilot software system, capable of controlling avionic vehicles such as conventional and vertical take off and landing (VTOL) airplanes, multi-rotors, etc. The mission goal for the Arducopter is to perform autonomous surveillance while flying within a pre-configured operational safety perimeter or *geofence*. A subset of this Arducopter system, the safety-critical software component of an autonomous battery-powered Arducopter that performs surveillance missions was implemented and formally verified by a group of researchers as a part of the ARCOS program [21]. Our interest is to construct a rigorous assurance case for that part of Arducopter.

Integrated Safety and Security Assurance 2.0 Case

The goal of developing an assurance case for Arducopter is to investigate if justifiable confidence can be provided for its safety and security for certification purposes given the evidence. The overall structure of the Arducopter assurance case is based on the principles of Overarching Properties (OPs) [9]. The OP approach allows organizing assurance arguments in a way that demonstrates that the system in consideration possesses the three fundamental properties—namely intent, correctness, and acceptability—that are necessary for the safe and secure operation of the system. Hence, we structured our case with three main branches, such that the hierarchical composition of the evidence and the claims deduce that the three overarching properties are met in the Arducopter software, which in turn implies the top-level claim "*Arducopter Software is fit for purpose*". In particular, our focus was on demonstrating how well the safety and security claims are specified and supported by evidence in the assurance case.

To that end, the overall claim of the intent overarching property (that requires the defined intended behavior to be correct and complete with respect to the desired behavior), is supported by sub-claims on the functional and non-functional requirements capturing the desired behavior. While the claim about Safety Requirements is established by showing evidence of compliance with standards such as ARP 4754 and 4761, the claim about security requirements is argued via a rigorous ontology-based approach that was followed by the development team. Similarly, the claim about the correctness of overarching property that requires that the implementation is correct with respect to its defined intended behavior is supported by sub-claims on the correctness of the Functional and Non-Functional specifications, traceability, software architecture as well as the source and executable object code correctly implementing the software requirements and architecture. These sub-claims were adequately supported by various verification artifacts such as testing, formal verification, and review evidence provided by the development team. The claim of the third leg of the assurance case—acceptability branch—requires showing that the software does not have any unacceptable impact. Since our focus was on safety and security aspects, this branch establishes through evidence of various formal analyses that no known hazards and vulnerabilities are present in the software implementation due to adequate safety and security controls.

In all these three branches, unlike the traditional monolithic, composition of arguments and evidence, Assurance 2.0 advocates a principled, integrated approach to assembling a major portion of assurance cases using theories. We have defined several theories and have instantiated them appropriately in the Arducopter Assurance case in all the three main OP branches of the assurance case. For instance, consider one of the theories '*Theory of Static Analysis of Code*' that is used in the acceptability branch. Figure 6, shows its definition (right side of figure) and its instantiation in the Arducopter Case (left). Some texts are abstracted in the figure to cope with the page size, but an interested reader is referred to [22] for complete details. The theory's top-level claim states that 'Tool *T* guarantees by static analysis of Code *C* of type *X* (Source/Binary) for Software Component *S* that the code satisfies property *P* in Environment *E*'. This claim is supported by a set of sub-claims and evidentiary obligations all relating to the type or class of the entities referred to in the top-claim. This theory is used in the Arducopter case to support the claim relating to the results of the static analysis performed by a specific technique called the 'Checker Framework'. Hence, we instantiated the theory Tool *X* to 'Checker framework', Code *C* to 'Arducopter', Type *T* to Binary, etc. Further, in the instantiation, we attached all the necessary Arducopter evidence as required by the theory (shown by green-colored evidence blocks in the leaf nodes in the figure). In the full Arucopter case, we have defined and used a total of four such theories.

Further, since Assurance 2.0 supports active exploration and recording of potential defeaters, we systematically identified and recorded them throughout the Arducopter assurance case. The majority of the defeaters were concerned with the inadequacy of the justifications and the lack of additional evidence for certain claims. All the evidence provided for the Arducopter case was available from various sources such as online resources, information stored in structured repositories, etc. Our assurance case assembles the evidence hierarchically to substantiate the top-level claim.

Structural Analysis and Semantic Analysis

At its core, a correct Assurance 2.0 case is one in which the top-level claim is precise and logically follows from the composition of its sub-claims, arguments, and supporting evidence. However, even for systems of moderate complexity such as the Arducopter, the case is large with extensive sets of arguments and a substantial body of evidence, all expressed in natural language. Hence, the task of verifying the correctness of the arguments and evidence as well as ensuring they logically support the stated claims about the system is time-consuming and intellectually challenging for both

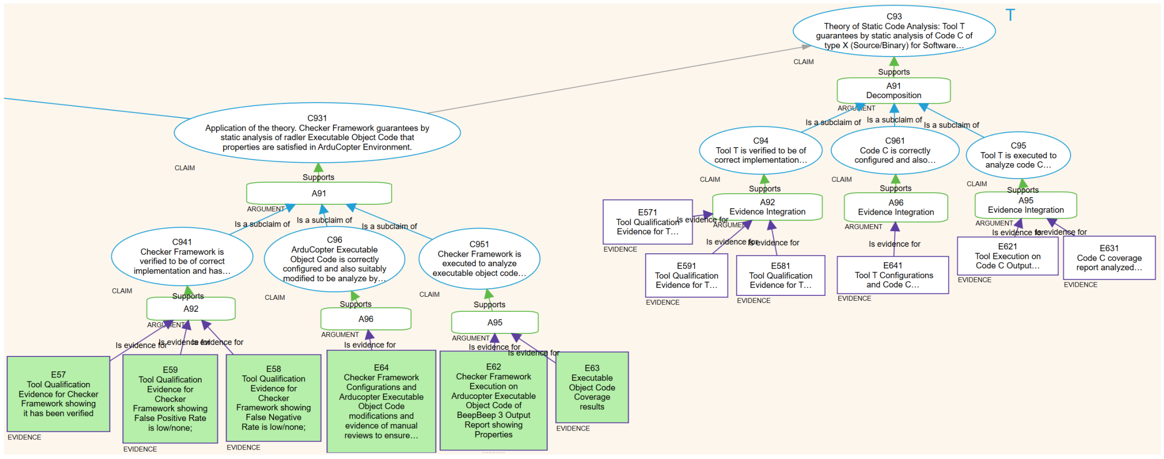

C93
Theory of Static Code Analysis: Tool T guarantees by static analysis of Code C of type X (Source/Binary) for Software…
CLAIM
T
Supports
A91
Decomposition
ARGUMENT
Is a subclaim of
Is a subclaim of
Is a subclaim of
C94
Tool T is verified to be of correct implementation…
CLAIM
C961
Code C is correctly configured and also…
CLAIM
C95
Tool T is executed to analyze code C…
CLAIM
Supports
A92
Evidence Integration
ARGUMENT
Supports
A96
Evidence Integration
ARGUMENT
Supports
A95
Evidence Integration
ARGUMENT
E571
Tool Qualification Evidence for T…
EVIDENCE
Is evidence for
Is evidence for
Is evidence for
E591
Tool Qualification Evidence for T…
EVIDENCE
E581
Tool Qualification Evidence for T…
EVIDENCE
Is evidence for
E641
Tool T Configurations and Code C…
EVIDENCE
Is evidence for
Is evidence for
E621
Tool Execution on Code C Output…
EVIDENCE
E631
Code C coverage report analyzed…
EVIDENCE
C931
Application of the theory. Checker Framework guarantees by static analysis of radler Executable Object Code that properties are satisfied in ArduCopter Environment.
CLAIM
Supports
A91
ARGUMENT
Is a subclaim of
Is a subclaim of
Is a subclaim of
C941
Checker Framework is verified to be of correct implementation and has…
CLAIM
C96
ArduCopter Executable Object Code is correctly configured and also suitably modified to be analyze by…
CLAIM
C951
Checker Framework is executed to analyze executable object code…
CLAIM
Supports
A92
ARGUMENT
Supports
A96
ARGUMENT
Supports
A95
ARGUMENT
Is evidence for
Is evidence for
Is evidence for
Is evidence for
Is evidence for
Is evidence for
E57
Tool Qualification Evidence for Checker Framework showing it has been verified
EVIDENCE
E59
Tool Qualification Evidence for Checker Framework showing False Positive Rate is low/none;
EVIDENCE
E58
Tool Qualification Evidence for Checker Framework showing False Negative Rate is low/none;
EVIDENCE
E64
Checker Framework Configurations and Arducopter Executable Object Code modifications and evidence of manual reviews to ensure…
E62
Checker Framework Execution on Arducopter Executable Object Code of BeepBeep 3 Output Report showing Properties
E63
Executable Object Code Coverage results
EVIDENCE

Fig. 6. Theory of Static Analysis and Its Application

authors and evaluators; the risk lies in the possibility of approving flawed assurance arguments and consequently certifying systems that are unsafe or insecure.

To cope with this challenge, our approach provides automated tool-supported structural and semantic analysis capabilities for Assurance 2.0 cases. While the ASCE tool performs structural analysis based on Assurance 2.0 core concepts, we leverage s(CASP) to reason about semantic properties of the assurance case.

Structural Analysis in ASCE: The ASCE tool is built-in with the capability to perform various structural and syntactic analyses on assurance cases, including:

- *Assurance 2.0 Methodological Enforcements*: ensures that the core Assurance 2.0 concepts are upheld and reports errors in the way claims, arguments, theories, and evidence are specified or associated. For instance, claims not supported by arguments, missing evidence nodes, incomplete declaration of attributes, missing side claims, etc., are all reported by the tool.
- *Indefeasibility*: report all unretired/unresolved defeaters in the case.
- *Natural Language Deductivism*: ensures arguments are valid, sub-claims deductively entail claims, and reasoning steps are logically true and sound.
- *Structural Checks*: A number of structural checks such as the validity of embedded links among nodes is established at the time of assurance case creation; validation of Dynamic Narrative Region elements, that are established to link evidence stored in repositories, etc., are checked by the tool.
- *Confidence*: capability to assess quantitative and qualitative confidence (high, low, medium) at both node level and for the overall case.

Semantic Analysis with S(CASP): While the complete details of the assurance case transformation to a logic program and its analyses are described elsewhere [19], the following are some of the semantic properties of the assurance case that are analyzed.

- *Indefeasibly Justified*: Possessing this semantic property, fundamental to an assurance case, implies that (a) the top-level claim is sufficiently supported by well-founded arguments and evidence, ensuring justification, and (b) there are no unresolved defeaters that potentially alter the decision regarding the top-level claim, establishing indefeasibility.
- *Theory Application Correctness*: This property guarantees that the theories are precisely instantiated in the assurance case; in particular, the properties and the instantiations (objects and environment) align with their respective theory definitions.
- *Property-Object-Environment Definition Consistency*: Inconsistencies within the specifications of properties, objects, and environments in the nodes of an assurance case can pose a problem as they may lead to flawed validation of assurance arguments. Hence, the absence of contradictions in the property-object-environment definitions is a crucial property that is checked in Assurance 2.0 cases.
- *Evidence Validity and Adequacy*: Existence and appropriateness of all evidence to support their respective claim are inevitable in a well-defined assurance case. By defining domain-specific rules about validity and adequacy for classes of evidence (For e.g., test-case artifacts should also always have an associated coverage analysis report), we automatically detect and report the issues such as missing evidence nodes, and incorrect or inadequate evidence within the evidence node.
- *Completeness*: Completeness refers to the state of

encompassing all the necessary elements. By defining say, a "domain completeness" rule that required every property-object-environment combination be used at least once in a claim or evidence node, we can evaluate the completeness of the assurance case.

- *Harmonious Composition of Theories*: when multiple theories are linked and applied within an assurance case, there is a risk of conflicting properties among their definitions or in their concrete applications. We are currently exploring approaches to define novel rules and will allow s(CASP) to check for the presence of conflicts, or determine that the theories can harmoniously coexist within that assurance case and can be composed correctly.

In summary, the realm of automated structural and semantic analysis relieves humans from repetitive tasks, leading to improved decision-making about assurance cases.

V. Conclusion

Assurance 2.0 framework aims to develop the science underpinning assurance cases and to improve confidence in their construction and assessment which in turn should increase their adoption in different certification regimes. The assurance case tools and automation and analysis support we have built in the CLARISSA project and described in this paper will go a long way towards fulfilling this objective. Our future work, among others, includes developing a *sentencing dashboard* for supporting assessor verdicts of the assurance cases, utilizing Large Language Models (LLM) support for conversion to *object-property-environment* formalism from natural language statements and support for *automated assurance case synthesis* with associated theories and defeaters.

Acknowledgment

The authors thank Elmer Salazar and Joaquín Arias for their s(CASP) contributions. CLARISSA is supported by DARPA under contract number FA875020C0512. Distribution Statement "A": Approved for Public Release, Distribution Unlimited. 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] SAE International, "ARP4761: Aerospace Recommended Practice (ARP) Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Website: https://www.sae.org/standards/content/arp4761/.

[2] Radio Technical Commission for Aeronautics (RTCA), "DO-326A: Airworthiness Security Process Specification," Website: https://www.rtca.org/standards/.

[3] SAE International, "ARP4754A: Aerospace Recommended Practice (ARP) Guidelines For Development Of Civil Aircraft and Systems," Website: https://www.sae.org/standards/content/arp4754a/.

[4] Radio Technical Commission for Aeronautics (RTCA), "DO-178C: Software Considerations in Airborne Systems and Equipment Certification," Website: https://www.rtca.org/standards/.

[5] ——, "DO-254: Design Assurance Guidance for Airborne Electronic Hardware," Website: https://www.rtca.org/standards/.

[6] Defense Advanced Research Projects Agency (DARPA). Automated Rapid Certification Of Software (ARCOS). [Online]. Available: https://www.darpa.mil/program/automated-rapid-certification-of-software

[7] Adelard. Claims Arguments Evidence (CAE). [Online]. Available: https://claimsargumentsevidence.org/

[8] R. Bloomfield and J. Rushby, "Assurance 2.0: A manifesto," *arXiv preprint arXiv:2004.10474*, 2021. [Online]. Available: https://arxiv.org/abs/2004.10474v3

[9] C. Michael Holloway, NASA Langley Research Center, "Understanding the Overarching Properties: First Steps, FAA Final Report," Website: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/TC_Overarching.pdf.

[10] R. Bloomfield and J. Rushby, "Assessing confidence with assurance 2.0," *arXiv preprint arXiv:2205.04522*, 2023. [Online]. Available: https://arxiv.org/abs/2205.04522v3

[11] J. J. Chilenski, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, pp. 193–200(7), September 1994. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/sej.1994.0025

[12] Adelard LLP, "Assurance and Safety Case Environment (ASCE)," Accessed on June 20, 2023. [Online]. Available: http://www.adelard.com/asce

[13] R. Bloomfield and K. Netkachova, "Building blocks for assurance cases," in *ASSURE: Second International Workshop on Assurance Cases for Software-Intensive Systems*. Naples, Italy: IEEE International Symposium on Software Reliability Engineering Workshops, Nov. 2014, pp. 186–191.

[14] J. Goodenough, C. Weinstock, and A. Klein, "Eliminative argumentation: A basis for arguing confidence in system properties," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005, 2015. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=434805

[15] R. Bloomfield, K. Netkatchova, and J. Rushby, "Defeaters and Eliminative Argumentation in Clarissa (To be published)," SRI, Tech. Rep., 2023.

[16] J. B. Goodenough, C. B. Weinstock, and A. Z. Klein, "Eliminative induction: A basis for arguing system confidence," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, pp. 1161–1164.

[17] J. Arias, M. Carro, E. Salazar, K. Marple, and G. Gupta, "Constraint Answer Set Programming without Grounding," *TPLP*, vol. 18, no. 3-4, pp. 337–354, 2018.

[18] G. Gupta, E. Salazar, S. C. Varanasi, K. Basu, J. Arias, F. Shakerin, R. Min, F. Li, and H. Wang, "Automating commonsense reasoning with asp and s(casp)," 2022.

[19] A. Murugesan, I. H. Wong, R. Stroud, J. Arias, E. Salazar, G. Gupta, R. Bloomfield, S. Varadarajan, and J. Rushby, "Semantic Analysis of Assurance Cases using s(CASP)," *To appear at Goal Directed Execution of Answer Set Programs (GDE) Workshop in Int'l Conf. on Logic Programming (ICLP)*, 2023.

[20] "ArduPilot," https://ardupilot.org/, Accessed on June 28, 2023.

[21] D. Bhatt, H. Ren, A. Murugesan, J. Biatek, S. Varadarajan, and N. Shankar, "Requirements-driven model checking and test generation for comprehensive verification," in *NASA Formal Methods: Proc. 14th International Symposium, NFM 2022, Pasadena, CA, USA*. Springer, 2022, pp. 576–596.

[22] Clarissa DARPA ARCOS Team: Honeywell, SRI, Adelard (NCC) and University of Texas - Dallas, "Clarissa ARCOS Phase 2 Tools and ArduCopter Assurance Case Release 12/20/2022," https://github.com/ARCOS-Clarissa/Clarissa-Tools/tree/master/phase-2-1220-release, Accessed on January 30, 2023.