*TPLP* **24** (4): 805–824, 2025. 

doi:10.1017/S1471068424000425

# *Automating Semantic Analysis of System Assurance Cases Using Goal-Directed ASP*

ANITHA MURUGESAN and ISAAC WONG
*Honeywell Aerospace, Plymouth, MN, USA*
(*e-mails:* anitha.murugesan@honeywell.com, isaachong.wong@honeywell.com)

JOAQUÍN ARIAS
*CETINIA, Universidad Rey Juan Carlos, Madrid, Spain*
(*e-mail:* joaquin.arias@urjc.es)

ROBERT STROUD
*Adelard (part of NCC Group), London, UK*
(*e-mail:* robert.stroud@nccgroup.com)

SRIVATSAN VARADARAJAN
*Honeywell Aerospace, Plymouth, MN, USA*
(*e-mail:* srivatsan.varadarajan@honeywell.com)

ELMER SALAZAR and GOPAL GUPTA
*University of Texas at Dallas, Dallas, TX, USA*
(*e-mails:* elmer.salazar@utdallas.edu, gupta@utdallas.edu)

ROBIN BLOOMFIELD
*Adelard (part of NCC Group), London, UK*
*City, University of London, London, UK*
(*e-mail:* robin.bloomfield@nccgroup.com)

JOHN RUSHBY
*SRI International, Menlo Park, CA, USA*
(*e-mail:* Rushby@csl.sri.com)

*submitted 20 August 2024; accepted 13 September 2024*

**Abstract**

Assurance cases offer a structured way to present arguments and evidence for certification of systems where safety and security are critical. However, creating and evaluating these assurance cases can be complex and challenging, even for systems of moderate complexity. Therefore, there is a growing need to develop new automation methods for these tasks. While most existing

assurance case tools focus on automating structural aspects, they lack the ability to fully assess the semantic coherence and correctness of the assurance arguments.

In prior work, we introduced the Assurance 2.0 framework that prioritizes the reasoning process, evidence utilization, and explicit delineation of counter-claims (defeaters) and counter-evidence. In this paper, we present our approach to enhancing Assurance 2.0 with semantic rule-based analysis capabilities using common-sense reasoning and answer set programming solvers, specifically s(CASP). By employing these analysis techniques, we examine the unique semantic aspects of assurance cases, such as logical consistency, adequacy, indefeasibility, etc. The application of these analyses provides both system developers and evaluators with increased confidence about the assurance case.

*KEYWORDS:* automated assurance reasoning, semantic analysis, answer set programming

## 1 Introduction

Certification of systems in regulated industries, such as aerospace, nuclear, and healthcare, necessitates authorities to determine whether a system assuredly complies with domain standards such as security and safety, by evaluating the evidence of system assurance provided. However, as system complexity increases, the evidence presented using traditional, highly prescriptive, and process-driven approaches (such as DO-178C in aerospace) often becomes overwhelming and largely unstructured, which makes their compilation and review time-consuming and cumbersome. Hence, there has been a growing interest in recent years in developing *Assurance Cases*, as an alternative means to present evidence and establish confidence in system compliance.

The assurance case approach advocates for hierarchically structuring persuasive arguments, backed by a well-organized body of evidence, to effectively substantiate the top-level claim about the system, such as its compliance with standards. Many methodologies emphasize creating and presenting the assurance case using graphical formats. This systematic and visually engaging approach is increasingly recognized as a preferred choice for both organizations seeking certification and certifying agencies (Holloway and Graydon (2018)). While assurance cases address challenges associated with systematically presenting large bodies of evidence, the descriptions used within their arguments and evidence are predominately natural language statements. Thus, interpreting the meaning of those statements to assess the arguments' coherence and evidence's relevance remains an intellectually demanding, manual task for assurance case authors and evaluators.

Consequently, there is a significant drive towards researching and developing strategies to automate the creation and evaluation of assurance cases, aiming to reduce human effort and enhance confidence in the cases. While most current assurance tools and methodologies primarily concentrate on verifying the structural soundness of assurance cases, there is not adequate support to reason about the *semantics* or the meaning conveyed by the natural language statements used within the case.

To that end, DARPA's research program, Automated Rapid Certification of Software, was intended to explore and address the challenges associated with the generation and evaluation of assurance cases. As part of this initiative, we have developed a comprehensive approach and tool suite named Consistent Logical Automated Reasoning for

Integrated System Software Assurance (CLARISSA), which is founded on our assurance methodology called Assurance 2.0 (described in Bloomfield and Rushby (2020)). While the complete details of CLARISSA are elaborated in Varadarajan et al. (2024a); Murugesan et al. (2023); and Bloomfield and Rushby (2024), we provide only brief and informal introductions here.

In this paper, we present a subset within our CLARISSA approach that automates the analysis of the semantic aspects of Assurance 2.0, by leveraging the power of commonsense reasoning and answer set programming (ASP) Brewka et al. (2011); Gelfond and Kahl (2014) solvers, namely s(CASP) Arias et al. (2018). s(CASP) is a novel goal-oriented, non-monotonic reasoner capable of efficiently handling logical reasoning (see Section 3.2) tasks essential for semantic analysis, which is central to our research. The goal-directed execution of s(CASP) automates commonsense reasoning based on general rules and exceptions (through default negation). That is, it enables inferences to be drawn from a set of logical rules that formalize the assurance case. Additionally, s(CASP) performs *deductive and abductive reasoning*, which is essential for proving whether a top-level claim can be deduced based on the arguments, evidence, and/or assumptions in assurance cases, and make it possible to define invariants (as *global constraints*) that allows the analysis of scenarios that violate these invariants.

In a nutshell, we leverage the general pattern of each assurance statement – that intuitively is nothing but assertions of *properties* of *objects* in a certain *environment* – to perform semantic analysis. We first capture crucial lexical components – namely objects, properties, and environment – within the assurance case statement using minimal formalized semantics. Subsequently, we automatically translate the formalized elements of the entire assurance case into corresponding logical rules under ASP semantics. Using various semantic properties of interest expressed as rules and queries, we use s(CASP) to provide the proof that these properties (represented as predicates) hold in the assurance case.

To the best of our knowledge, this methodology is not only the first attempt to tackle the automated semantic analysis of system assurance cases, it also establishes a new paradigm for explainable, knowledge-assisted assurance reasoning. This is thanks to the top-down solving strategy utilized by s(CASP), which produces concise and human-understandable justifications (Arias et al. (2020)). These justifications are essential to precisely identify the reasons for the success (positive queries) or failure (negative queries) of the assurance so that we can resolve the issues raised. Furthermore, s(CASP) supports various forms of negation, each with unique applications in assurance reasoning. As we mentioned before, we use *default negation* to derive detailed justifications for why a given claim cannot be proven, while *classical* or strong negation can be used to impose specific restrictions (see the book by Gelfond and Kahl (2014) for details).

We have evaluated our approach on multiple industrial-strength case studies, particularly in the fields of Avionics and Nuclear Reactors (refer Varadarajan et al. (2024a)). In this paper, we use one of the case studies, the ArduCopter system, an open-source platform (see Section 4), to illustrate the approach and highlight the contributions:

- Methodically identify and capture the vocabulary within each assurance case statement that contributes to the lexical significance of the assurance case. The main steps of this methodology are outlined in Section 5.

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

- Systematically translate the assurance case into an answer set program that is amenable for execution by the s(CASP) engine. To facilitate this, we have enhanced the Assurance and Safety Case Environment (ASCE) tool with a plugin that captures the vocabulary and automatically transforms the use case into logical notation. The new plugin is described in Section 5.1, and the transformation mapping is explained in Section 5.2.
- Identify and formalize properties that are unique and essential for assessing the semantic rigor of assurance cases. These properties are elaborated in Section 5.3.
- Leverage the capabilities of s(CASP) tool to analyze semantic properties within the assurance case, as described in Section 5.4. In particular, our use of *Negation as Failure* involves deriving negative conclusions from the absence of positive information, to automatically identify defeaters. Additionally, s(CASP) possesses the ability to perform *non-monotonic reasoning* that allows revision of conclusions in light of new information. This feature is particularly valuable for incrementally assessing and improving the strength of assurance cases during the authoring process.

## 2 Motivation

Consider the scenario, used in Holloway (2015), where an assurance case is constructed with the intent to convince Jon's father that Tim (a college student known to the family) is a safe driver to take Jon (a teenager not yet of driving age) to a football game in Tim's car. The assurance case, as shown in Figure 1 (uses Assurance 2.0 format i.e., elaborated in the next section), has a top-level claim *"Tim is a safe driver"* (top-most blue ellipse), that is supported by several fine-grained sub-claims (lower level blue ellipses) and corroborating evidence (purple rectangles at the leaf level) that establishes Tim's capability to drive safely. Jon's father must evaluate the assurance case and determine if he is convinced. If not, he must explain.

Evaluating assurance cases is a complex process that involves several critical steps. It requires analyzing the semantics or meanings of the assurance statements to verify that sub-claims are articulated accurately and consistently, that they collectively support the overarching claim, and that the provided evidence is relevant and sufficiently substantiates the claims. Any semantic discrepancies or gaps discovered during the evaluation undermine confidence in the overall claim. For example, if testimonials (E17 in the Figure) indicate Tim's good driving but also mention a minor road incident not formally recorded by the DMV, it contradicts the sub-claim (C13 ellipse in the Figure) asserting Tim has not been involved in any accidents. Similarly, if Tim's driver's license was issued in a state different from where the football game took place, this discrepancy indicates that the context or environment in which the evidence is presented does not adequately support the top-level claim. Furthermore, the absence of claims and evidence concerning the condition of Tim's car in the assurance case, which is crucial for fully convincing Jon's dad, further diminishes confidence. As exemplified using a few semantic gaps, the evaluation process involves interpreting the meaning of statements both individually and in conjunction with other statements from various perspectives.

Currently, no tools or techniques are available to support evaluators of assurance cases, such as Jon's dad, necessitating manual and labor-intensive evaluation processes that are

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

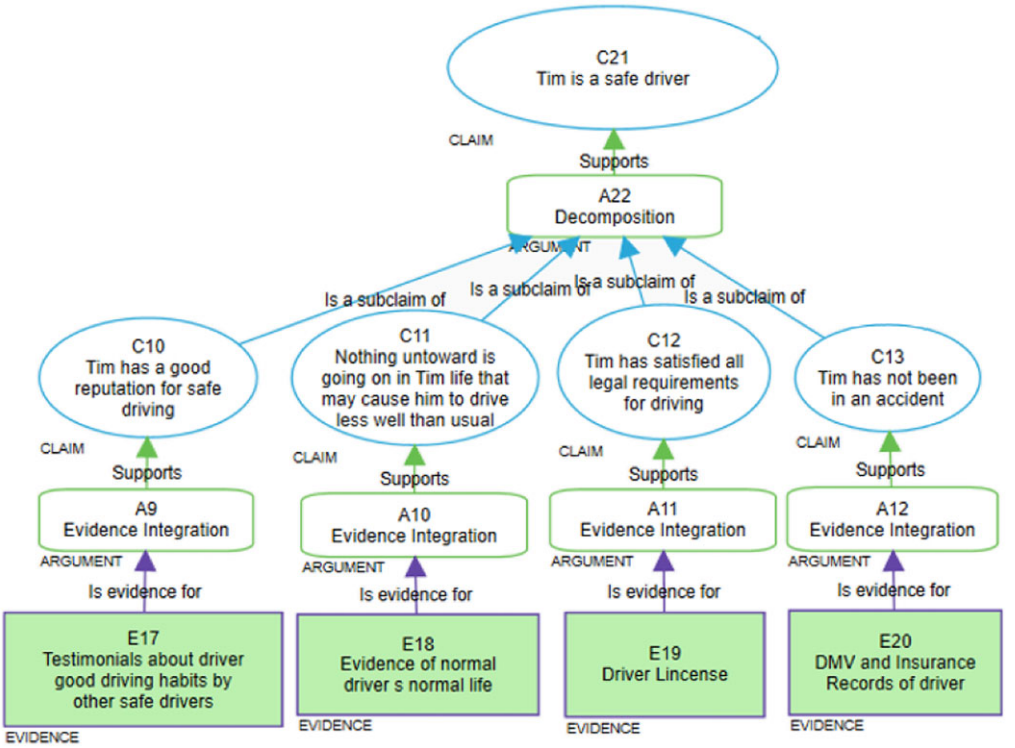


Fig. 1. Safe driver - a motivating example.

prone to errors. The development of automation to aid in the semantic analysis of assurance cases–by identifying gaps such as inconsistencies among statements, inadequacies in claims, the validity of evidence, and deficiencies in the assurance strategy–would be highly advantageous. Evaluating the semantic aspects of assurance cases for even moderately complex real-world systems is both time-consuming and intellectually demanding. Automated semantic analysis tools would significantly alleviate the cognitive load on evaluators and could also assist authors in enhancing the reliability of their assurance cases. The remainder of this paper outlines our approach to providing automated support for the semantic analysis of assurance cases developed based on Assurance 2.0 principles.

## 3 Background

In this section, we present a brief introduction to: (i) Assurance 2.0, the framework/methodology used to create assurance cases of engineering systems, and (ii) s(CASP), the reasoning engine upon which the semantic analysis of our proposal relies.

### *3.1 The Assurance 2.0 framework*

The Assurance 2.0, defined by Bloomfield and Rushby (2020), is a modern framework that supports reasoning and communication about the behavior and trustworthiness of

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

engineered systems and their certification. It provides a framework that separates out the deductive and inductive reasoning combined with the use of a practical indefeasibility criterion for justified belief. This frames the notion of defeaters, both *undercutting* and *rebutting*, and motivates construction of arguments that are predominately deductive, an approach known as "Natural Language Deductivism." Details on *defeaters* and *eliminative argumentation* are provided by Bloomfield et al. (2024). We also advocate the use of *Confirmation Measures* to evaluate the strength of evidence and arguments. We reduce confirmation bias through active search for defeaters and a methodology for doing so by means of counter-claims and counter-cases. We argue in Bloomfield and Rushby (2024) that confidence cannot be reduced to a single attribute or measurement. Instead, we draw on three different perspectives: positive, negative, and residual doubts. Our work also provides details of the approach to logical evaluation and soundness.

This framework adopts a Claims-Arguments-Evidence approach with an increased focus on the evidence, doubts/objections, and reasoning and overall semantics of a case. The building blocks of a typical Assurance 2.0 case are claims (and sub-claims) that assert the properties of objects, such as 'The train is safe' evidences which are artifacts establishing trustworthy facts directly related to a claim; arguments that serve as bridging rules connecting what is known or assumed (sub-claims, evidence) to the claim under investigation; side claims which offer additional justification or assumptions to support the argument; defeaters that capture doubts and objections that challenge the validity of claims, arguments, or evidence; and Theories defined as reusable templates that can be instantiated in concrete assurance cases as sub-cases. Assurance 2.0 cases are authored using the ASCE tool developed by Adelard LLP (2024). Figure 2 in Section 4 shows an exemplar of an assurance case authored in ASCE tool using Assurance 2.0 principles.

### *3.2 s(CASP), a non-monotonic reasoner*

To conduct the semantic analysis, we leverage the advances in the field of logic programming. In particular, we use ASP, a paradigm rooted in logic programming (see paper by Brewka et al. (2011) for details), which integrates reasoning methods that automates commonsense reasoning capturing and managing incomplete information, cyclical reasoning, and constraints.

Among the different ASP solvers, we chose s(CASP), a goal-directed ASP system that executes answer-set programs in a top-down manner. The goal-directed execution of s(CASP) is particularly well-suited for reasoning about assurance cases, because it generates partial stable models including only the relevant information needed to support (or decline) a given claim. Assurance cases are commonly structured in a way that makes sense to human interpretation. *Deductive and abductive reasoning*, supported by s(CASP), is essential for proving whether a top-level claim of an assurance case can be deduced based on the arguments, evidence, and assumptions. Both forms of negation are supported by s(CASP), and they have diverse applications in assurance reasoning. For instance, we use *default negation* to derive detailed justifications for why a given claim cannot be proven. Additionally, we are exploring the use of *abducibles*, which involves *even loops over negation* and makes it possible to derive negative conclusions from the absence

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

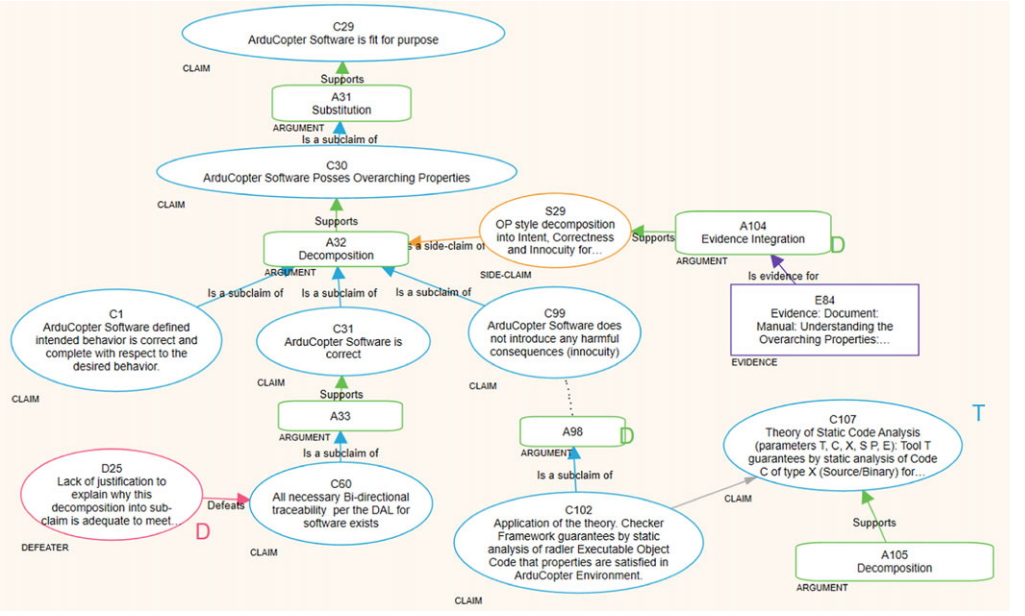


Fig. 2. Arducopter case fragment in assurance 2.0.

of positive information, and automatically identify defeaters. s(CASP) supports reasoning about *global constraints* and `classical negation`, so it is able to analyze scenarios that violate these constraints. Furthermore, the top-down solving strategy employed by s(CASP) generates concise, human-understandable *justifications* (see work by Arias et al. (2020) for details). These justifications play a crucial role in precisely identifying the reasons for assurance failure and resolving concerns. Lastly, s(CASP) possesses the ability to perform *non-monotonic reasoning*, allowing for the revision of conclusions in light of new information (due to the presence of negation). This feature is particularly valuable for incrementally assessing and improving the strength of assurance cases during the authoring process.

## 4 Illustrative Case Study: The ArduCopter System

To demonstrate the concepts of our approach and tools, we will utilize the open-source ArduCopter system as a case study in the remainder of this paper. This system was previously utilized to assess the CLARISSA methodology. Additional evaluations conducted on industrial-strength case studies from the avionics and nuclear sectors are detailed in Varadarajan et al. (2024a).

The ArduCopter, derived from the open-source ArduPilot autopilot platform (Ardupilot 2024), is an unmanned aerial vehicle designed to oversee a diverse range of avionic vehicles, enabling them to perform various autonomous tasks. Our focus with ArduCopter is to construct an assurance case using the Assurance 2.0 methodology,

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

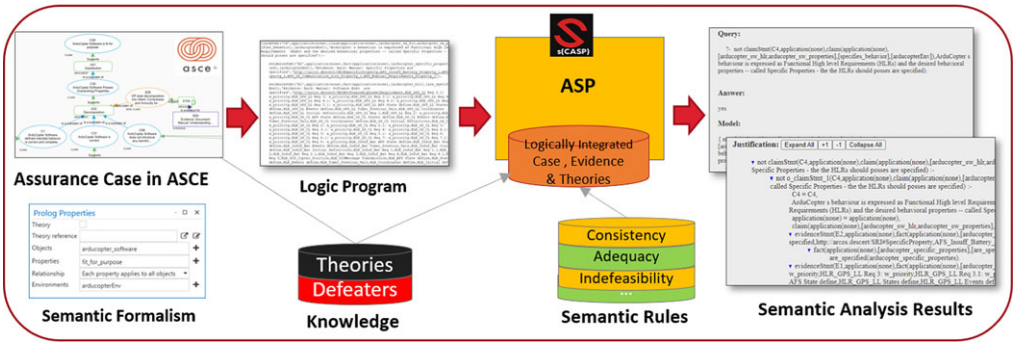


Fig. 3. Semantic analysis approach.

establishing justified confidence in its ability to conduct autonomous surveillance missions while adhering to safety and security standards. The complexity of the available details, including the concept of operations, system architecture, and other development and verification artifacts for the ArduCopter system, was sufficient to construct a realistically sized assurance case with an appropriate level of complexity. This allowed us to explore the associated challenges and assess the effectiveness of our approach. We will use excerpts from our evaluation of this system to illustrate the principles of our approach.

Figure 2 depicts a snippet of the ArduCopter assurance case constructed using the ASCE tool. The overall structure of the assurance case aims to demonstrate that the ArduCopter exhibits the three fundamental overarching properties–intent, correctness, and innocuity –that are indispensable for ensuring the safety and security of the system. The top-level *claim* of ArduCopter (the uppermost blue ellipse) is broken down into detailed *sub-claims* (blue ellipses) through the use of *arguments* (green rounded rectangles), which are substantiated by *evidence* (purple rectangles) at the leaf level. The rationale behind these refinements is documented in *side-claims* (yellow ellipses). Moreover, any doubts, concerns, or counter-claims regarding any aspect of the case are captured as defeaters (red ellipses), described by Bloomfield et al. (2024). Several theories (blue ellipses adorned with a "T" symbol) were employed in formulating the ArduCopter assurance case, such as the theory of static analysis (ID C107) and its application in a sub-claim (ID C102). The complete details of this case are available in Varadarajan et al. (2024a).

## 5 Semantic Analysis Approach

To a large extent, assurance cases heavily rely on unstructured, free-form natural language despite their structured graphical representation, which is not naturally conducive to automation. On the contrary, complete formal notations allow for automated analysis, but they present challenges in authoring and reviewing without a steep learning curve and also have expressibility limitations. Seeking a middle ground, we defined our approach that facilitates semantic analysis by capturing essential lexical components of assurance statements in an intuitive and "minimally" formal manner.

Our approach to semantic analysis, shown in Figure 3, has the following steps:

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

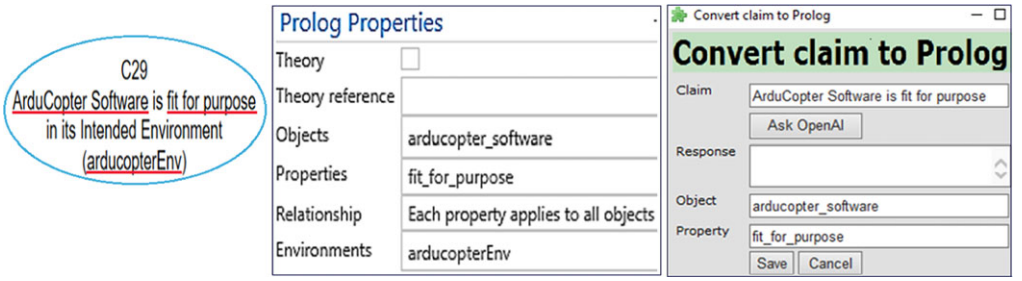


Fig. 4. Objects-properties-environments Formalism and LLM support.

1. **Formalism of Assurance Statements**: The crucial language components used to articulate statements within Assurance 2.0 cases are categorically grounded and formally captured within the ASCE tool interface.
2. **Transformation into ASP**: The formalized language along with the assurance case structure is automatically transformed and exported into equivalent logical predicates in ASP.
3. **Modeling Semantic Properties**: Various properties that ensure semantic rigor of Assurance 2.0 cases are modeled as rules.
4. **Semantic Reasoning using s(CASP)**: The predicate form of the assurance case is systemically analyzed for the various properties using s(CASP) engine and results are reported in a user-friendly manner.

While Figure 3 is aimed to provide a comprehensive high-level overview of our approach, in the following subsections we elaborate on each step and offer details of the text presented in smaller font within the figure.

### *5.1 Formalism of assurance statements*

Assurance cases consist of blocks or "nodes" that delineate the properties or relationships relevant to objects within a specific environment. The "Properties," "Objects," and "Environment" for each node are identified and defined, enabling us to formally articulate statements like "Object O satisfies property P in environment E." For instance, the assertion "ArduCopter Software is fit for purpose in its Intended Environment" can be broken down into (i) "Object = ArduCopter Software," (ii) "Property = fit for purpose," and (iii) "Environment = Intended Environment (arducopterEnv)."

Currently, the ASCE assurance case authoring tool allows users to explicitly record objects, properties, and environments as formal semantics, along with their relationships as depicted in Figure 4. Besides offering a field for natural language descriptions, which aids in the manual inspection of the assurance case, it also allows manual entry of these formal semantics by users. These Objects, Properties, and Environments will be crucial for the next step of transforming the assurance case into a logic program. Because it can be tedious and time consuming to enter these semantics for every node, we have enhanced the ASCE tool with open-source LLMs to parse the natural language description and

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

Table 1. *Concept mapping between assurance 2.0 and first order logic*

| Assurance 2.0 Concept | Mapping into first order logic |
|---|---|
| Top-level Claim | In the assurance case, the top-level claim aligns with the concept of *queries*, which can be subjected to logical entailment analysis. |
| Sub Claims and Side Claims | Sub-claims and side claims are analogous to **predicates** serving the purpose of capturing the relationships between objects. |
| Argument | Arguments that establish relationships among sets of objects-properties-environment can be equated to **rules** where logical implications are utilized to describe these relationships. |
| Evidence | Evidence artifacts that establish the truth about the system are **facts**. |
| Defeater | Defeaters are counter-claims to provide support for not believing that claim, which is the same as **negated goals** in formal logic. |
| Binding | Theory definitions require the specification of variables for objects and the environment, that will be instantiated with terms when applied to a specific assurance case. This process aligns with the concept of **substitution**, which involves dynamically mapping variables to terms. |
| Reasoning | Assurance case reasoning involves demonstrating that a top-level claim is entailed based on its arguments and evidence, akin to **proofs** in logic, where the validity or truth of a claim is established through logical reasoning. |

automatically extract these semantics. Although the current LLM support is limited, we are investigating methods to improve their use.

### *5.2 Transformation into ASP*

Based on the formalism captured for each node of the assurance case, we have augmented the ASCE tool with a plugin to automatically transform the entire assurance case into equivalent logical predicates in ASP. The choice to transform into ASP is based on the fact that each concept in Assurance 2.0 and the intended analysis can be readily mapped to corresponding concepts in first order logic. The mapping between these concepts is illustrated in Table 1.

Table 2 shows the rule to transform each node type to predicate form. The term `ClaimPredicate` refers to a ASP predicate represented as the `claim([O],[P],[E])`, where `[O]`, `[P]`, and `[E]` represent comma-separated lists of "Objects," "Properties," and "Environments" associated with each assurance node. As we formalize the properties, the relationships specified between lists of objects and properties are preserved. This preservation ensures precision in the analysis. ASCE tool allows 3 different

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

Table 2. *Assurance 2.0 node mapping into answer set programming*

| Node | Translation into ASP |
|---|---|
| Claim | `claimStmt(Claim_ID, Application, ClaimPredicate, Description) :-`<br>`  [claimStmt \| evidenceStmt \| side_ClaimStmt], ... , [-Defeater],`<br>`  ClaimPredicate,`<br>`  theory(Theory_ID, Application, Claim).`<br>`ClaimPredicate :-`<br>`  PropertyList.`<br>`theory('Theory_ID', Application, ClaimPredicate) :-`<br>`  [claimStmt \| evidenceStmt \| side_ClaimStmt], ... ,`<br>`  ClaimPredicate.` |
| Evidence | `evidenceStmt(Evidence_ID, Application, ClaimPredicate, Artefact, URI) :-`<br>`  [-Defeater], ClaimPredicate.`<br>`ClaimPredicate :-`<br>`  PropertyList.` |
| Side Claim | `side_ClaimStmt(Side_Claim_ID, Application, ClaimPredicate,`<br>`  Justification) :-`<br>`  [claimStmt \| evidenceStmt \| side_ClaimStmt], ... , [-Defeater],`<br>`  ClaimPredicate.`<br>`ClaimPredicate :-`<br>`  PropertyList.` |
| Defeater | `defeater(Defeater_ID, Application, defeats(Node_ID),`<br>`  ClaimPredicate, Description) :-`<br>`  [claimStmt \| evidenceStmt \| side_ClaimStmt], ... , [-Defeater],`<br>`  [theory('Theory_ID', Application, Claim)].`<br>`ClaimPredicate :-`<br>`  PropertyList.`<br>`PropertyList.` |

types of relationship specification, in addition to the 'Off' to indicate no relationship. Consequently, the `PropertyList` is derived from the `ClaimPredicate` based on the generic object-property relationship definition, as described in Figure 5. If the Relationship property is Off, which is its default value, the PropertyList is empty and the ClaimPredicate is asserted as an ASP fact, without expanding the property list. For example, if properties such as *consistent* and *verifiable* are specified for both high and low-level requirements specification artifacts, the property list is formulated as `consistent(high-level-requirements, low-level-requirements)` and `verifiable(high-level-requirements, low-level-requirements)`. On the other hand, if the properties such as *traceable to design* and *traceable to test cases* are asserted for specific objects of high and low-level specification artifacts, respectively, the property list will expand to `traceable-to-design(high-level-specification)` and `traceable-to-test-cases(low-level-specification)`. Hence, depending on the specified relationship between the properties and objects, the `ClaimPredicate` is

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

| Relationship | Property List |
|---|---|
| Each property applies to all objects | P(X,Y),Q(X,Y) |
| Each property applies to each object | P(X),P(Y),Q(X),Q(Y) |
| Each property applies to a specific object | P(X),Q(Y) |

Fig. 5. Object-property relationships.

```
claimStmt('C29', application(none), claim( application(none),[arducopter_software],
                    [fit_for_purpose],[arducopterEnv] ), 'ArduCopter Software ...') :-
  claimStmt('C30', application(none), claim( application(none),[arducopter_software],
                    [posses_overarching_properties],[arducopterEnv] ), ...),
  side_ClaimStmt('S30', application(none), claim( application(none),[overarching_properties],
                    [explored_as_means_of_compliance],[arducopterEnv] ), 'The use of ...'),
  claim(application(none), [arducopter_software], [fit_for_purpose], [arducopterEnv]).
```

Fig. 6. Snippet of export of arducopter assurance case into ASP.

expanded in ASP syntax and used for the analysis. Furthermore, to maintain the structure of the case during export and ensure traceability, certain metadata (such as node identifiers, descriptions, etc.) is also included within the exported predicates. Figure 6 shows an example of the exported predicates of Arducopter assurance case.

When the ASCE tool exports the assurance case as ASP predicates, they are categorically saved in separate files: (i) the top-level claim is saved as a query, (ii) the negation of the top-level claim is saved as a negative query, (iii) the body of the assurance case are saved together as rules and facts, (iv) theory definitions are saved separately, and (v) defeaters are saved as integrity constraints that counter the claims. In the following section, we elaborate on various properties of interest and their analyses.

### *5.3 Modeling semantic properties*

Semantic properties are rules about the contextual meaning of the concepts used in the statements within assurance cases such as consistency among claims, correctness of arguments, adequacy of evidence, etc. Although these properties are inherent in the minds of most authors and evaluators of assurance cases, to the best of our knowledge, they have not yet been systematically defined, let alone automatically checked. We outline some of the categories of properties that we have identified as crucial for evaluating the semantic rigor of assurance cases below.

1. *Indefeasibly Justified*: This property implies (a) the top-level claim is sufficiently supported by well-founded arguments and evidence, ensuring *justification* and (b) there are no unresolved defeaters that could potentially alter the decision regarding the top-level claim called *indefeasibility*. This semantic property is fundamental to an assurance case and in simple terms, it means the top-level claim can be indisputably deduced given the arguments and evidence.
2. *Theory Application Correctness*: Theories in Assurance 2.0 are reusable assurance fragments that can be independently specified, and 'pre-certified' that could be

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

applied to concrete assurance cases as sub-cases. However, when using theories, it is critical to ensure that they are instantiated appropriately and that all the necessary properties and evidences obligated by the theory are provided in the concrete case. This property guarantees the correct use of the theories.

3. *Property-Object-Environment Consistency*: Claims, arguments, and evidence assert properties of objects in specific environments. In extensive, hierarchically defined assurance cases, it is crucial to ensure no conflicts or contradictions exist among these assertions. While some conflicting terms are universally recognized, such as 'X is safe' and 'X is hazardous,' others, like 'X has no vulnerabilities' and 'X has residual security risks,' are domain-specific. We call these sets of inconsistent combinations of properties, objects, and environments as consistency rules for assurance.
4. *Adequacy*: Assessing the sufficiency of sub-claims and evidence supporting the stated claims is a crucial aspect scrutinized in assurance cases. For example, for the Arducopter case we defined rules that required properties of evidence such as:

   `requirements_testcas_coverage_achieved`,
   `requirementsbased_testcases_passed` and
   `structuralcoverage_of_requirementsbased_tests_achieved`

   to be present to meet the claim with property:

   `do178C_requirements_test_conformance_achieved`

   Similarly, `meets_intent`, `is_correct`, and `innocuous` properties were required for the top-level claim with the `overarching` property. Any violation of these rules shows inadequacy in the assurance case. Though these adequacy rules require a deeper understanding of the domain and context, unlike consistency rules, once defined, these adequacy rules will enable easy, rigorous, and recurrent validation of subsequent versions of large and complex assurance cases.
5. *Completeness*: Completeness of assurance cases refers to the state of encompassing all the necessary elements in the domain of objects, properties, and environments for the system in consideration. While adequacy property is defined to find the presence of desired properties of a certain object, completeness concerns the presence of the same property for all the objects of a certain type. For instance, in the Arducopter case example, we defined rules that required the process of assessment to be completed (`process complete`) for all types of `assessments` performed on the Arducopter system.
6. *Harmonious Coexistence of Theories*: As outlined in Section 3, Assurance 2.0 permits the incorporation of theories into an assurance case. However, employing multiple theories concurrently poses a risk of conflicts and contradictions stemming from disparities in their definitions or application methods, despite each theory being flawless on its own. Since these conflicts are mainly semantic, defining incompatible combinations as rules allows automatically checking for their presence.

The properties listed above represent only a fraction of potential assurance case properties. We view this effort as an initial step in identifying essential property categories

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

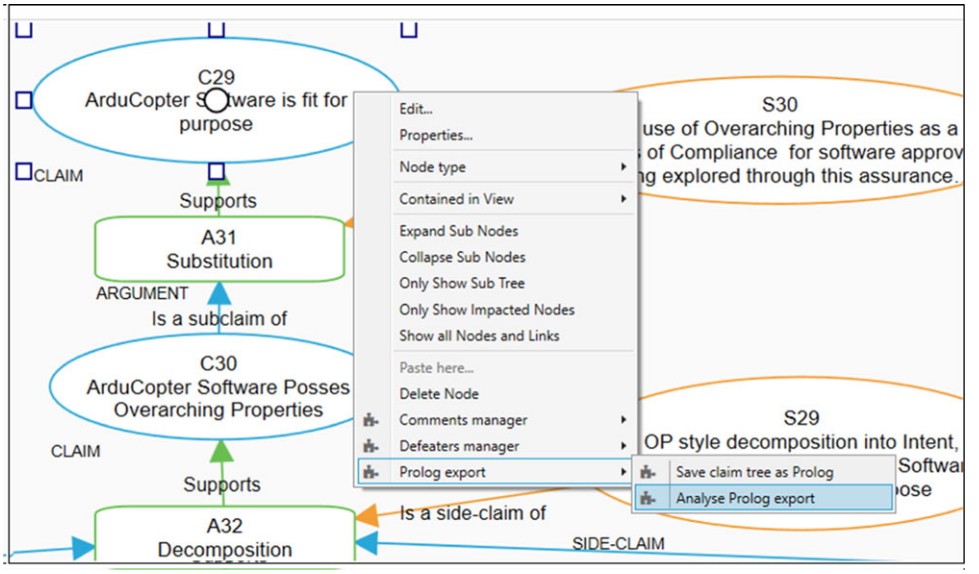


Fig. 7. Semantic analysis option in ASCE Interface.

for evaluating assurance case rigor. In the following section, we formally define these properties and discuss their automated analysis.

### *5.4 Semantic reasoning using s(CASP)*

In this section, we outline how we use s(CASP) to automatically analyze these properties using the exported ASP program of the assurance case. The ASCE tool is enhanced with semantic analysis capabilities, allowing users to invoke s(CASP) and perform semantic analysis through the ASCE interface, as illustrated in Figure 7.

In essence, the s(CASP) system rigorously analyzes whether a specific query can be deduced in the provided ASP program. A successful execution yields the display of a "model," offering a detailed explanation as to why the query is entailed. Conversely, if the query cannot be deduced, executing the negation of that same query enables retrieving an explanation for the cause of the failure. Our objective is to harness this capability of s(CASP) to analyze the exported program of the assurance case for various semantic rules. Furthermore, for enhanced usability, we utilized the `--html` flag option of s(CASP) that displays the justification tree via an interactive HTML page, allowing the display of analysis results and models in a web browser. To address scalability concerns with the s(CASP) system, we enhanced the s(CASP) system 1 by implementing a more efficient and robust search, adding a debugger, and incorporating several builtins.

In the remainder of the section, we delve into the details of analyzing each of the previously outlined properties using s(CASP), illustrated with examples from the Arducopter system. Although the specific properties and results are particular to this case example, the general methodology is widely applicable to a broad range of assurance cases.

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

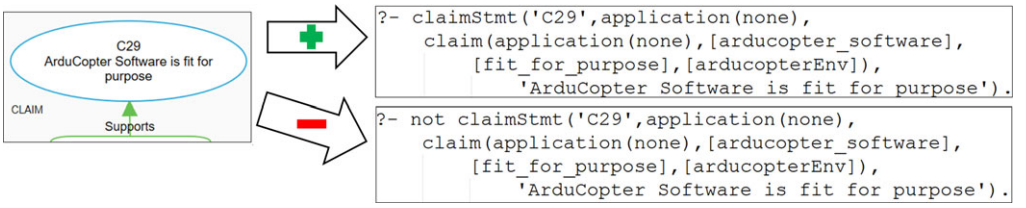


Fig. 8. Positive and negative query exported from top-level claim node.

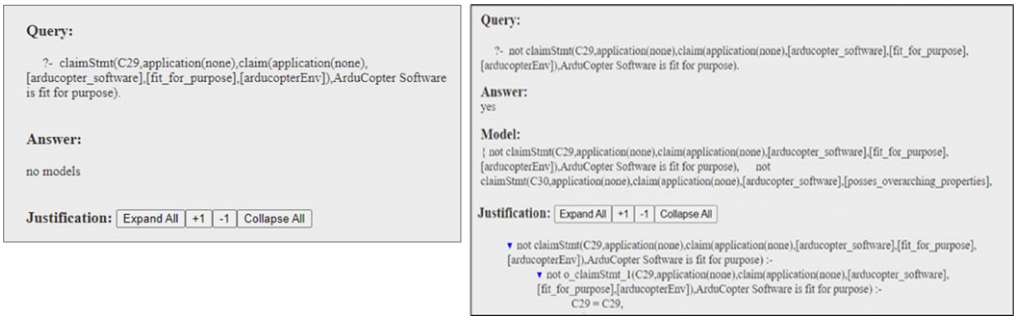


Fig. 9. Example of s(CASP) output of failed positive query and successful negative query.

### 5.4.1 Indefeasibly justified

As mentioned earlier, this property concerns indisputable entailment of top-level claim. Formally, we represent this property as a query that includes the top-level claim. We have enhanced the ASCE tool to automatically and separately export the top-level claim in ASP as both positive and negative queries in addition to exporting the entire assurance case along with defeaters. The snippet of the exported query of ArduCopter assurance case's top-level claim is shown in Figure 8.

When the positive query is successfully executed in s(CASP) and an explanation is provided, it signifies that the assurance case indeed possesses this property. Conversely, if the negative query is successfully executed, s(CASP) returns the unresolved defeaters as violations, as illustrated in Figure 9. While the specifics of the model and justification depend on the system being analyzed, Figure 9 primarily shows how the results of positive and negative queries will be displayed.

### 5.4.2 Theory application correctness

As previously explained, a theory is a reusable assurance case fragment applicable to concrete system assurance cases. To ensure reusability, objects and environments in theory nodes are defined as variables (uppercase), while properties are atoms (lowercase). Applying a theory in the concrete case requires the properties in that node and its sub-nodes to match the respective theory node's property. Additionally, the objects and environment of those nodes must be atoms defined as instances of the respective variables in the theory node. When authoring the assurance case, the types of objects and environments, along with their system-specific instantiations, should be predefined by the

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

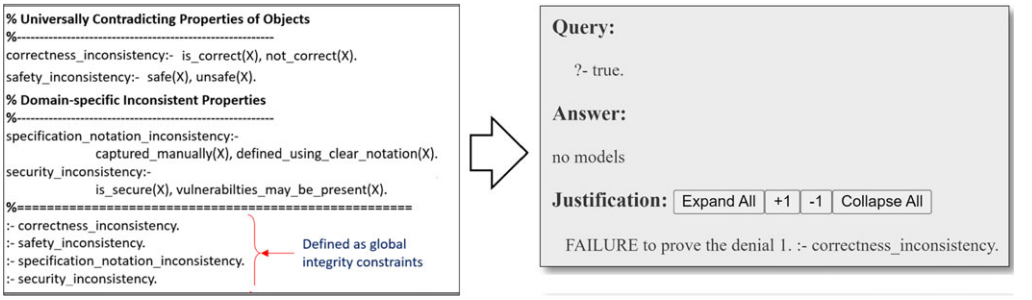


Fig. 10. Consistency rules and s(CASP) analysis output.

author. Using the ASCE interface, the author selects the desired theory and the correct instantiations (from the predefined list of type-instances) for each of the theory.

To automate the verification of correct theory application, we rely on s(CASP) to assess : (a) the direct match between properties outlined in the theory and those in the corresponding application nodes, and (b) the validity of all objects and environments as instantiations of the types specified in the theory nodes. For the analysis, we execute using the *scasp* command along with the exported program of the assurance case, the theory definition, and the same queries used to check the indefeasibly justified property. The analysis results are reported similarly to indefeasibly justified property analysis.

### *5.4.3 Property-object-environment consistency*

We formally expressed consistency rules as global integrity constraints expressed in ASP notation. These constraints are specified in the form of conjunctions of `properties(Objects, Environment)` or `properties(Objects)`, where properties are atoms, whereas Objects and environment are defined as variables. This approach allows us to detect any consistency issues present in any instance of object or environment. The snippet displayed on the right side of Figure 10 illustrates the consistency rules devised for examining the ArduCopter case.

To verify the adherence to these rules, we execute the *scasp* command alongside the exported program of the assurance case, the consistency rules, and the query, which is '?- true'. Essentially, this query prompts the s(CASP) engine to determine whether the assurance case contains instantiations (objects) for the inconsistent set of properties defined by the rules. s(CASP) notifies us of any violations it discovers, as demonstrated on the left side of Figure 10 for the Arducopter case.

### *5.4.4 Adequacy*

The adequacy property is also specified as rules structured as conjunctions of `properties(Objects, Environment)` or `properties(Objects)`, similar to consistency rules. While consistency rules are global integrity constraints, adequacy rules are designed to verify the concurrent presence of properties for the same instance of an object. Therefore, the query posed to s(CASP) is whether all the properties in the conjunction are present in the assurance case. We execute the *scasp* command using the exported program of the assurance case, the adequacy rules, and this query. Essentially, this query

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

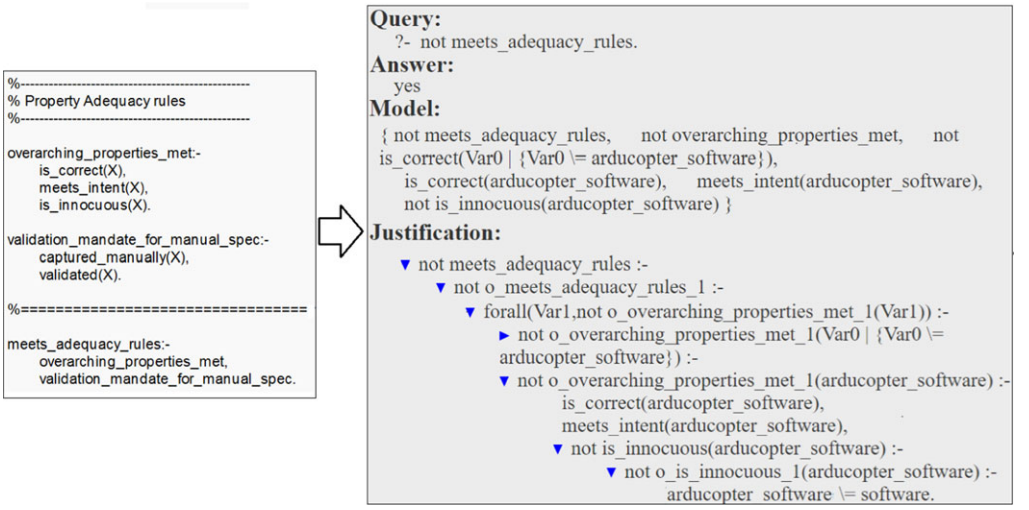


Fig. 11. Adequacy rules and s(CASP) analysis output.

prompts the s(CASP) engine to verify whether the assurance case contains instantiations (objects) with all the properties.

For instance, consider one of the adequacy rules defined for ArduCopter, which checks if the three overarching properties, namely `meets_intent`, `is_correct`, and `innocuous`, are satisfied, as shown in Figure 11. When analyzed using the exported program of the Arducopter case, the s(CASP) engine searches through the case to identify instantiations that fulfill this rule. If this condition is not met, as illustrated on the right side of Figure 11 for explanatory purposes, by negating the query to s(CASP), we get a justification tree, detailing the reason for the 'inadequacy', such as the absence of the `is_innocuous` property for the `arducopter_software` object.

### *5.4.5 Completeness*

Completeness properties concern the domain of the objects, properties and environment within the entirety of the assurance case. Since Assurance 2.0 case creation requires authors to define a global set of objects, properties, and environments for the system under consideration, we utilize this predefined set for specifying and analyzing completeness properties. These completeness property specifications are similar to adequacy. However, instead of verifying if completeness is met, we negate the property and query s(CASP) to determine the reason why the assurance case does not meet the completeness property. This query prompts s(CASP) to search for an instantiation of an object that does not satisfy completeness. This level of detail allows the assurance evaluator to assess if all instantiations of a certain type have certain common properties.

In the Arducopter case, we established a completeness property `assessment(X)` and `not process_complete(X)`. When the negation of this rule was executed in s(CASP), along with the assurance case and the definition of the domain types and their instantiations, the result was a justification explaining the reason for the failure. As illustrated in the Figure 12, the lack of the `process_complete` property in the assurance case for

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

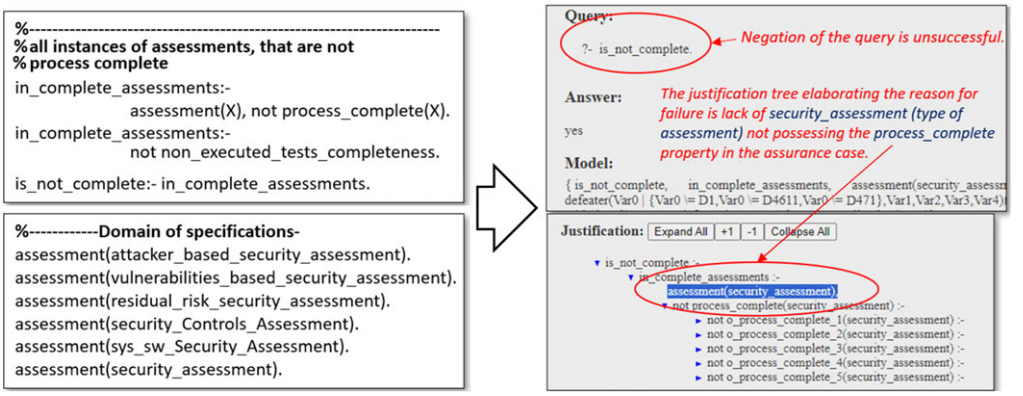


Fig. 12. Completeness rules and s(CASP) analysis outputs.

the `security_assessment` instance of type `assessment` was identified as the cause of failure.

### *5.4.6 Harmonious coexistence of theories*

Inharmonious theory definitions of rules resemble consistency rule definitions. For example, in the Arducopter case, we defined a rule for `inharmonious_DAL_theories`, to identify if theories pertaining different DAL levels coexist, such as `achieves_DAL_C_DO178c_requirement_testing(X)` and `achieves_DAL_A_DO178c_code_coverage(X)`. These are also defined as global integrity constraints, and their analysis is conducted in the same manner as consistency rules. Essentially, this query prompts the s(CASP) engine to determine whether the assurance case contains references to theories that are defined as inharmonious, as defined by the rules. s(CASP) notifies us of any violations it discovers, similar to the way the inconsistencies were reported (as shown earlier in Figure 10).

In sum, the realm of semantic analysis offers a wide range of possible analyses. As part of our ongoing work, we are exploring valuable and powerful properties to analyze. Automating these analyses provides valuable insights and relieves humans from repetitive tasks, leading to improved decision-making regarding assurance cases.

## 6 Conclusion

The Assurance 2.0 framework aims to advance the science behind assurance cases and enhance confidence in their development and assessment across various certification regimes. This paper introduces our method of enriching the Assurance 2.0 framework with semantic analysis capabilities by harnessing the reasoning abilities of s(CASP). Our approach innovatively involves systematically translating Assurance 2.0 cases into ASP notation and formally defining key properties essential for the robustness of assurance cases, thereby enabling semantic analysis via s(CASP). To our knowledge, this methodology represents the first effort to automate the semantic analysis of system assurance cases, establishing a novel paradigm for explainable, knowledge-assisted reasoning. Evaluations

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

conducted on industrial-strength case studies in safety-critical domains such as avionics and nuclear reactors have yielded positive feedback from assurance authors and evaluators. As part of future work, we intend to explore the use of LLMs for semantic property specification and analysis, as well as enhance the tool's capability to analyze large assurance cases assembled using several complex theories. While this paper focuses on the semantic analysis of assurance cases within our CLARISSA approach, interested readers are referred to Varadarajan et al. (2024a) and Varadarajan et al. (2024b) for a comprehensive overview of our work.

## Acknowledgements

CLARISSA is supported by DARPA 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 Dept of Defense or the U.S. Govt.

We are grateful to the anonymous reviewers for their insightful comments and suggestions for improvement.

## References

Ardupilot. 2024. Ardupilot. https://ardupilot.org/. [Accessed on May 5, 2024].

Arias, J., Carro, M., Chen, Z. and Gupta, G. 2020. Justifications for goal-directed constraint answer set programming, In Proceedings 36th International Conference on Logic Programming (Technical Communications), volume 325 of EPTCS, 59–72.

Arias, J., Carro, M., Salazar, E., Marple, K. and Gupta, G. 2018. Constraint Answer Set Programming without Grounding. *Theory and Practice of Logic Programming* 18, 3-4, 337–354.

Adelard (part of NCC Group). 2024. Assurance and safety case environment (ASCE), http://www.adelard.com/asce. [Accessed on November 19, 2024].

Bloomfield, R. and Rushby, J. 2020. Assurance 2.0: A manifesto. arXiv preprint arXiv:2004.10474.

Blommfield, R. and Rushby, J. 2024. Assessing confidence with assurance 2.0. arXiv preprint arXiv:2205.04522.

Blommfield, R., Netkachova, K. and Rushby, J. 2024. Defeaters and Eliminative Argumentation in Assurance 2.0. arXiv preprint arXiv:2405.15800.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Gelfond, M. and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge: Cambridge University Press.

Holloway, C. M. and Graydon, P. J. 2018. DOT/FAA/TC-17/67: Explicate 78: Assurance case applicability to digital systems. Technical report, Federal Aviation Administration (FAA). https://www.faa.gov/sites/faa.gov/files/aircraft/air_cert/design_approvals/air_software/TC-17-67.pdf

Holloway, M. 2015. Understanding assurance cases: An educational series in five parts. Informal report, NASA Langley Research Center.

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press

Murugesan, A., Wong, I. H., Stroud, R., Arias, J., Salazar, E., Gupta, G., Bloomfield, R., Varadarajan, S. and Rushby, J. 2023. Semantic analysis of assurance cases using s(CASP). In *Proceedings of the International Conference on Logic Programming 2023 Workshops (Goal Directed Execution of Answer Set Programs (GDE)). Published by CEUR-WS.*

Varadarajan, S., Bloomfield, R., Rushby, J., Gupta, G., Murugesan, A., Stroud, R., Netkachova, K. and Wong, I. 2024a. Consistent Logical Automated Reasoning for Integrated System Software Assurance (CLARISSA), DARPA ARCOS Final Report. To appear shortly. Technical report, DARPA

Varadarajan, S., Bloomfield, R., Rushby, J., Gupta, G., Murugesan, A., Stroud, R., Netkachova, K., Wong, I. H. and Arias, J. 2024b. Enabling theory-based continuous assurance: A coherent approach with semantics and automated synthesis. In *International Conference on Computer Safety, Reliability, and Security. SAFECOMP 2024 Workshops: DECSoS, SASSUR, TOASTS, and WAISE*, Florence, Italy, vol. 14989 (pp. 173–187). Cham: Springer Nature Switzerland.

https://doi.org/10.1017/S1471068424000425 Published online by Cambridge University Press