A Survey of Tool-supported Assurance Case Assessment Techniques

MIKE MAKSIMOV, SAHAR KOKALY, and MARSHA CHECHIK, University of Toronto, Canada

Systems deployed in regulated safety-critical domains (e.g., the medical, nuclear, and automotive domains) are often required to undergo a stringent safety assessment procedure, as prescribed by a certification body, to demonstrate their compliance to one or more certification standards. Assurance cases are an emerging way of communicating safety, security, and dependability, as well as other properties of safety-critical systems in a structured and comprehensive manner. The significant size and complexity of these documents, however, makes the process of evaluating and assessing their validity a non-trivial task and an active area of research. Due to this, efforts have been made to develop and utilize software tools for the purpose of aiding developers and third party assessors in the act of assessing and analyzing assurance cases. This article presents a survey of the various assurance case assessment features contained in 10 assurance case software tools, all of which identified and selected by us via a previously conducted systematic literature review. We describe the various assessment techniques implemented, discuss their strengths and weaknesses, and identify possible areas in need of further research.

CCS Concepts: • **General and reference → Surveys and overviews**;

Additional Key Words and Phrases: Assurance case, safety case, tools, assessment, analysis, survey

ACM Reference format:
Mike Maksimov, Sahar Kokaly, and Marsha Chechik. 2019. A Survey of Tool-supported Assurance Case Assessment Techniques. *ACM Comput. Surv.* 52, 5, Article 101 (September 2019), 34 pages.
https://doi.org/10.1145/3342481

1 INTRODUCTION

Systems in safety-critical domains such as the nuclear, medical, transportation, and aerospace domain, whose failure may result in the loss of life or environmental damage, are often required to undergo a stringent safety certification process and gain the approval of an independent entity such as a regulatory body. Various standards such as DO 178C [5] (avionics domain) and ISO 26262 [16] (automotive domain) prescribe guidelines to which manufacturers are required to adhere to, such as processes that need to be used during development, tests that a system must undergo, and so on. These standards also often mandate manufacturers to produce and use assurance cases as a way to argue specific claims about their systems (e.g., that they are acceptably safe), as well as to show that the necessary steps and guidelines were followed during development. The term *assurance case* is used to describe a documented body of evidence, constructed as a

The work reported in this article has been funded by General Motors and NSERC Canada.
Authors' addresses: M. Maksimov, S. Kokaly, and M. Chechik, Department of Computer Science, University of Toronto, 40 St. George Street, Toronto, ON M5S 2E4, Canada; emails: {maksimov, skokaly, chechik}@cs.toronto.edu.

0360-0300/2019/09-ART101 $15.00
https://doi.org/10.1145/3342481

compelling argument, with the purpose of demonstrating that a particular claim about a system holds. These documents are typically created to support claims in a variety of areas, e.g., safety (referred to as a *safety case*), security, dependability, and so on, and must present or reference the necessary evidence supporting the justification of those claims, e.g., that a system is acceptably safe.

Assurance cases tend to grow excessively in size and can become very complex [23]. For example, an assurance case for an air traffic control system may consist of over 500 pages and 400 referenced documents [24]. Furthermore, the content of these documents is predominantly expressed in natural language, much of which is not amenable to automated assessment. This means that the evaluation of an assurance case largely depends on human insight and experience, making the review and assessment of such a document a very difficult and time-consuming process. How one determines whether an assurance case has sufficient strength or is sound in its argument for a particular claim is a non-trivial matter and an active area of research. The current lack of systematic methods in the assurance case assessment process may also lead to inconsistencies in their evaluation, in part due to faults in human judgment.

In response to the difficulties outlined above, various software tools (referred to as *assurance case tools*) have been developed to support developers and assessors in the assurance case assessment process. The development of these tools has been enabled by the introduction of formal assurance case syntaxes, such as the Goal Structuring Notation (GSN) [7], the Claims-Arguments-Evidence (CAE) notation [10], and more recently, the Structured Assurance Case Metamodel (SACM) notation [8]. In our previous work [29], we conducted a systematic literature review identifying a significant number of assurance case tools developed over the past 20 years and evaluated them based on six distinct functionality categories.

In this article, we examine tool support for assessing structured (graphical) assurance cases and build upon our previous work by conducting an in-depth survey of the various assessment features contained in 10 assurance case tools, all of which were identified by us as exhibiting notable assessment capabilities [29]. We focus on researching and conveying, in detail, the state-of-the-art assessment approaches currently available in assurance case tools, and do not compare or rank the tools featured in our study against one another. Thus, we categorize and describe tool support for two primary types of assurance case assessment, namely, assessment of an assurance case's structure and content.

The rest of the article is organized as follows: Section 2 introduces background information on assurance cases and their notations. Section 3 summarizes relevant results from our initial pre-study. In Section 4, we describe the methodology used for selecting and surveying the assurance case tools found in this work. An introduction of common underlying techniques utilized for assurance case assessment is presented in Section 5. Sections 6 and 7 present our findings for tool supported assessment of an assurance case's structure and content, respectively. Related work is discussed in Section 8. We conclude our survey and present directions for possible future work in this area in Section 9.

2 BACKGROUND

In this section, we present background information necessary for the understanding of assurance cases. This includes basic assurance case principles, definitions, and forms of representation.

2.1 Assurance Cases

Assurance cases (ACs) are created to justify that a certain claim (or claims) about a system hold true. These claims typically argue about a specific feature, e.g., a system's security, safety, or dependability. A safety case, for example, argues that a system is safe to operate in a given context. A

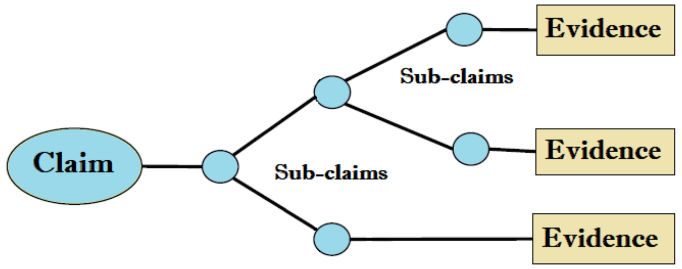


Fig. 1. An example assurance case structure. Inspired by Reference [30].

security case, however, will argue the security properties of a given system. In the GSN Community Standard [7], an assurance case is defined as:

> "A reasoned and compelling argument, supported by a body of evidence, that a system, service, or organization will operate as intended for a defined application in a defined environment."

In assurance case terms, an argument takes the form of a series of connected claims. The argument starts with an overall claim about the system or object of interest, which is then decomposed into a series of sub-claims intended to support the overall claim. Depending on the complexity of the object or system being argued for, these claims often need to be decomposed into further supporting sub-claims until eventually the claims are supported by evidence justifying those claims. Figure 1 presents the basic structure of an assurance case.

The history of assurance cases can be traced back to the nuclear industry and the Windscale nuclear reactor accident in 1957, where a significant amount of radiation was released due to a fire hazard (32 people killed and hundreds of cancer cases) [34]. This incident led to a number of safety reforms in the nuclear site licensing process in the following years, introducing the requirement of safety cases as a way to argue about safety. Similarly, the Clapham rail disaster and the Piper Alpha oil rig catastrophe in 1988, and other such disasters served to further motivate the need for more stringent safety assurance in their respective industries.

Assurance cases are now a popular way of conveying safety-critical information due to their structured argumentation, and have become common practice in many industries (e.g., nuclear, medical, off-shore drilling, automotive, avionics, and others). Their use is often recommended or required by different standards such as the ISO 26262 [16] functional safety standard for road vehicles, or the standard for software considerations in airborne systems, DO 178C [5]. For the purpose of certifying a system in regulated safety-critical domains such as the ones mentioned above, assurance cases are often presented to independent entities such as a regulatory body. A number of generic and publicly available assurance case projects can be found online [2–4, 6].

2.2 Assurance Case Notations

Significant research has been conducted to ensure that assurance cases can be clearly and successfully presented, discussed, and assessed. Various notations have been created for representing assurance cases, most of which inspired by Toulmin's argumentation model [39]. These include the Goal Structuring Notation (GSN) [7], the Claims-Arguments-Evidence (CAE) notation [10], the TRUST-IT [18] argumentation model, and more recently the Structured Assurance Case Metamodel (SACM) notation [8]. The Goal Structuring Notation is currently the most used one, and our previous survey showed that out of the 37 assurance case tools reviewed, 32 supported GSN [29]. The notation consists of six core elements: goal, strategy, context, solution, justification,

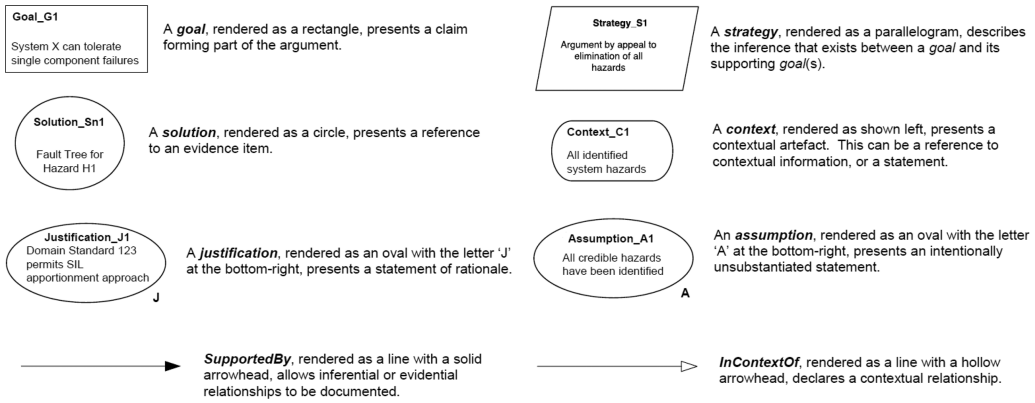


Fig. 2. The basic elements of GSN. Adapted from Reference [17].

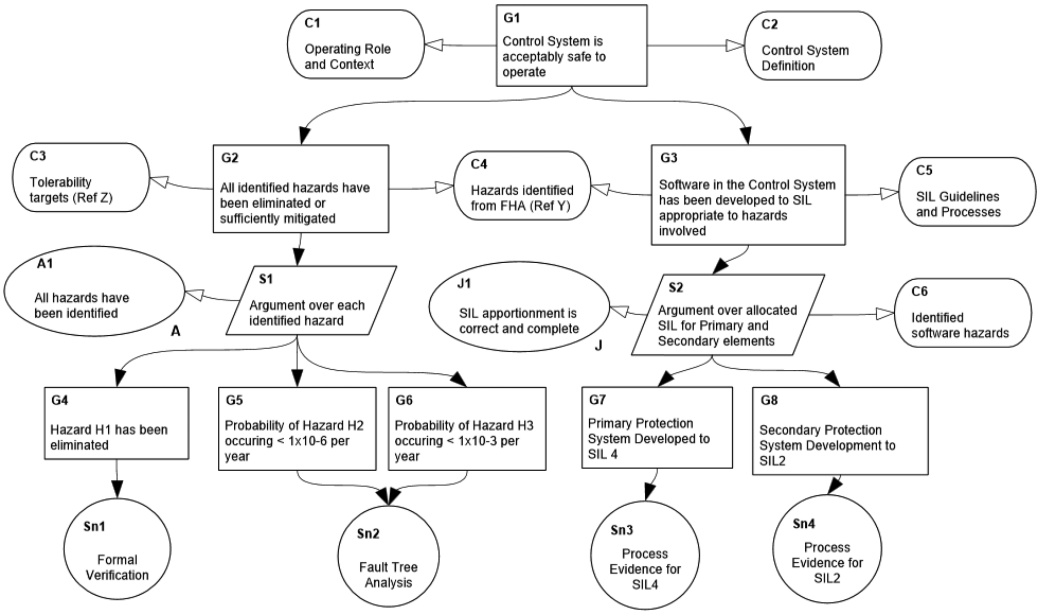


Fig. 3. An example of a GSN assurance case. Taken from Reference [17].

assumption, and two linking elements that represent types of relationships, "SupportedBy" and "InContextOf." Figure 2 presents the supported GSN elements and a brief description of their intended use.

The supported elements are used to construct assurance cases that take the form of a tree structure. The parent node (root "Goal" element) represents the overall claim made about the system or object. This goal is then decomposed via strategies into a number of progressively more specific sub-goals (this process is repeated a number of times depending on the complexity of the system), until eventually all sub-goals lead to solution nodes that reference evidence artifacts justifying the claims made. The elements "context," "justification," and "assumption" are used to provide explanations or other necessary information. Figure 3 presents an example GSN assurance case that argues the safety of a control system.

Table 1. Publication Venues Used for the Manual Search

Name	Abbreviation	Type
International Conference on Computer Safety, Reliability, & Security	SAFECOMP	Conference
International Symposium on High Assurance Systems Engineering	HASE	Conference
International Symposium on Model-Based Safety and Assessment	IMBSA	Conference
International Symposium on Software Reliability Engineering	ISSRE	Conference
Reliability Engineering & System Safety	-	Journal
International Conference on Computers, Software & Applications	COMPSAC	Conference

3 PRE-STUDY

In our previous work [29], we conducted a systematic literature review (SLR) of assurance case tools (which support structured (graphical) assurance cases) that have been developed over the past 20 years, producing a comprehensive list of 46 unique tools. Using the gathered literature, we categorized and evaluated the abilities of 37[1] of those tools. This section reports on the methodology used for compiling the initial list of assurance case tools (from which we extracted the subset of tools for this survey), as well as general findings relevant to tool support for assurance case assessment.

The SLR search consisted of three phases: (1) we established a *quasi-gold standard* (QGS) [41] by manually searching various publication venues (used to construct a search string for the subsequent automated search), (2) we conducted an automated literature search on a variety of different digital libraries, e.g., ACM Digital Library, and (3) a web-based search, using a popular search engine, with the goal of uncovering unpublished and/or commercial tools.

We identified five conferences and one journal that published considerable research on the topic of assurance cases (see Table 1). The proceedings of the aforementioned venues, including their workshops, were manually searched for the period of 2015–2017 inclusive and yielded 10 relevant assurance case tool papers.

Upon examining the manually gathered papers, we defined the search string for the automated literature search to be: **"***("Safety Assurance" OR GSN OR SACM OR "Safety Case" OR "Safety Cases" OR "Assurance Case" OR "Assurance Cases" OR "Safety Compliance") AND (Editor OR Tool OR Editors OR Tools OR Toolset OR Toolsets)***"** [29]. Using this search string, a literature search was carried out on Springer Link, ACM Digital Library, Engineering Village, and IEEE Xplore, with the only criteria being that the searched papers be published after the year 1998, and are written in English.

Springer Link, ACM Digital Library, Engineering Village, and IEEE Xplore returned 80, 21, 739, and 112 papers, respectively, for a total of 952 papers. Duplicate, inaccessible, and irrelevant papers (based on a manual review of their abstracts or the full text) were filtered and discarded. After the completion of the filtering process, 82 papers remained. Alternatively, the web-based search used Google as the search engine and was carried out using the same search string as in the literature search.

[1]The remaining nine tools didn't have enough information published for a proper evaluation to be conducted.

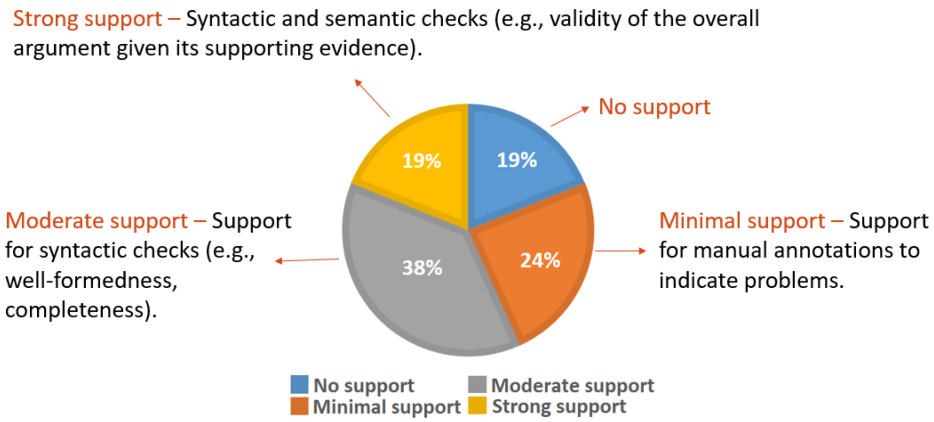


Fig. 4. Overall AC tool support for assessment, as observed in Reference [29].

After reading and examining the materials obtained by our searches, we established six distinct recurring tool functionalities, namely, *assessment*, *creation*, *collaboration*, *integration*, *maintenance*, and *reporting*, which were used as the basis for our evaluation. Four levels of tool support were assigned to each functional category, ranging from A (strong support) to D (no support), thus creating our grading criteria [29].

Figure 4 displays the overall results for assessment support among the 37 evaluated tools. We observed that the majority of the tools (38%) are limited to assessing the structure of an assurance case diagram, while only 19% provide support for some form of structural and content assessment. The remaining 43% of tools either have no assessment capabilities or are limited to very basic annotations that allow the user to write notes about problems found during manual inspection.

4 METHODOLOGY

In this section, we discuss the methodology used for choosing the tools featured in this work, as well as the procedures followed for conducting the survey.

The tools we selected for this study were chosen from the initial list of 37 assurance case tools in our pre-study [29]. We evaluated every tool in the initial list for its general assessment capabilities, as discussed in Section 3 above. For this work, we chose to survey the assessment techniques of every tool that had been previously evaluated as having moderate assessment support or higher, i.e., the tool had some form of support for assessing a graphical assurance case's structure, content, or both. This choice excluded tools that have no assessment capabilities and tools that only have support for manual user annotations (minimal support), since those tools were not of interest. The subset of tools meeting this criterion were then checked for availability[2] for hands-on usage and testing to be performed. Table 2 contains the list of the 10 tools meeting the above selection criterion and are thus the focus of this survey. Information about each tool's supported notations, assessment types, and whether it is commercial or free to use is also presented. For example, the tool ACedit is an Eclipse plugin that supports the GSN and ARM notations, and is limited to assessing the structure of an assurance case.

To inquire about the assessment capabilities of each tool, the tools were acquired and tested. The 10 tools range from commercial products requiring a license, to free to use or open-source plugins

[2]Some of the previously evaluated tools from the literature no longer exist, no longer work due to being outdated, or are private and not available for public use. Authors and developers of private tools were contacted and asked for permission to acquire their tools for testing purposes.

Table 2. Surveyed Tools

Tool name	Supported notations	Supported assessment type	Product type
ACedit https://github.com/arapost/acedit	GSN, ARM	Structure	Eclipse plugin (Open source)
AGSN https://github.com/AGSNeditor/development	GSN	Content	Eclipse plugin (Open source)
ASCE https://www.adelard.com/asce/choosing-asce/index/	CAE, SACM, GSN	Structure & Content	Commercial (Full trial license)
Astah GSN http://astah.net/download	GSN, ARM, SACM	Structure	Commercial (Paid license)
AutoFOCUS 3 https://af3.fortiss.org/download/	GSN	Structure & Content	Eclipse plugin (Open source)
CertWare https://nasa.github.io/CertWare/	ARM, CAE, GSN, EUROCONTROL	Structure & Content	Eclipse plugin (Open source)
D-Case Editor http://www.jst.go.jp/crest/crest-os/osddeos/en/tech.html	GSN, SACM	Structure & Content	Eclipse plugin (Open source)
NOR-STA https://www.argevide.com/purchase/assurance-case/	TRUST-IT Argument Representation	Structure & Content	Commercial (Full trial license)
Resolute https://github.com/smaccm/smaccm	Tool-specific notation	Content	Eclipse plugin (Open source)
SafeEd http://cs-gw.utcluj.ro/~adrian/tools/safed/gsn/gsn.html	GSN	Structure	Eclipse plugin (Free use)

built on top of pre-existing software such as Eclipse. All companies with commercial tools asked were kind enough to provide a full trial license of their product. Each tool from the list was used and tested along with the guiding help of published literature, tool manuals, and tool documentations. All tools except ACedit and SafeEd came with predefined examples. These examples were a useful component in surveying the various assessment abilities that each tool provided during the hands-on testing. The tool providers of AGSN and NOR-STA also gave us live demonstrations of the abilities of their respective tools.

Last, it is important to note that our work does not compare the featured tools against one another with the goal of determining which is better, nor does it rank them in any way. The majority of the tools in our study have their own unique focus, making them unsuitable for direct comparison.

5 COMMON UNDERLYING ASSESSMENT TECHNIQUES

The majority of the tools in our study utilize different core approaches for assessing the content of structured assurance cases. For example, the D-Case Editor relies on type-checking techniques to validate the consistency of a structured assurance case's content, while NOR-STA implements linguistic appraisal scales that internally use Dempster-Shafer Theory (among others) to quantitatively reason about arguments, as well as to capture uncertainty due to a lack of evidence.

This section provides brief background on the underlying techniques used by the tools in our study.

5.1 Type Checking

Type systems include rules to verify that data is properly represented throughout a system/program [33]. Types describe the possible values that a structure (e.g., a variable) can take. *Type checking* is the mechanism for verifying and enforcing these type constraints (e.g., that the value

```
int a = 23;
int b = 5;
Boolean c = true;
int sum;

sum = a + b;
```

a)

```
int a = 23;
int b = 5;
Boolean c = true;
int sum;

sum = a + c;
```

b)

Fig. 5. An example of type checking in Java: (a) valid action, (b) type error.

of a variable matches what is expected by its type). This verification process can occur during compile time (static type checking) or at runtime (dynamic type checking). Performing an illegal action results in a type error that notifies the user that an attempt has been made to carry out an operation on a data type that it is not meant to occur on.

Figure 5 presents an example of a valid and invalid type operation in the Java programming language. Both examples have the variables *a*, *b*, and *sum*, of type integer, and the variable *c*, of type Boolean. Figure 5(a) depicts a valid operation that adds together the integers *a* and *b* and stores them in the variable *sum* also of type integer. In Figure 5(b), the same operation is invalid and leads to a type error due to an attempt to add two variables of different types, i.e., the integer *a* with the Boolean *c*.

5.2 Bayesian Belief Networks

Bayesian Networks (BNs), commonly referred to as *belief networks*, represent the conditional dependencies between a set of variables through a directed acyclic graph (DAG) and are used to compute probabilities through the use of Bayesian inference [19]. Nodes in the graph represent the variables of interest (e.g., the occurrence of an event) and have set probabilities (weights) ranging from [0, 1]. Edges in the graph present causal/conditional dependencies between nodes, meaning that nodes that are not connected by a path are conditionally independent of each other. Each node is associated with a probability function (or probability distribution) for specifying the probabilistic relationships between them and the other nodes in the graph. This distribution can be either conditional, in the case that the node has a parent, or marginal, in the case that it is a root node. Once a Bayesian network has been fully specified, it can be used to represent a joint probability distribution, as well as to compute posterior probabilities of variables when given new information (evidence). The latter is done by combining prior weights with new evidence, thus creating new weights and updating beliefs.

Figure 6 presents an example of a simple causal Bayesian network used to describe the causal relationships among the state of the sky (Cloudy), whether the sprinkler is on (Sprinkler), whether it has rained (Rain), and whether the grass is wet (Wet Grass). When new evidence is added, such as setting certain variables to true or false, the posterior probabilities of the remaining variables in the graph are computed to reflect the new observations. In our example, if the variables for a cloudy sky and wet grass are set to true, the probability of the grass being wet as a product of rain is computed to be higher than the probability of the grass being wet due to the sprinkler being on.

5.3 Dempster-Shafer Theory

Dempster-Shafer Theory (DST), also referred to as the *theory of belief functions*, was created by Arthur P. Dempster [14] and later developed by Glenn Shafer [36] into a mathematical framework for reasoning with uncertainty. The framework has strong appeal in areas where uncertainties

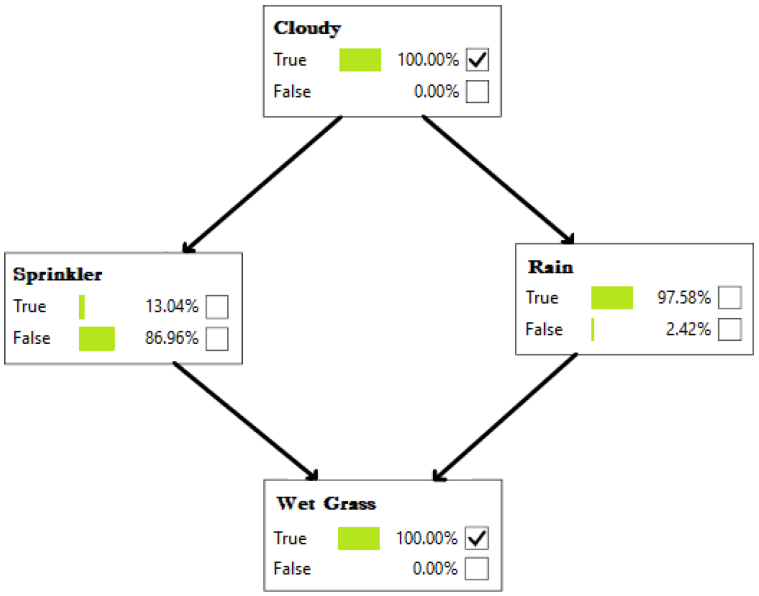


Fig. 6. An example of a graphically represented Bayesian network.

cannot be easily mapped by standard probability theory, such as Bayesian reasoning. Both DST and Bayesian reasoning set weights ranging from [0, 1] to events to reason about them. Where DST differs is that it supports the assignment of weights (called a mass) to a combination of possible events, rather than only assigning weights (probabilities) to each individual event, as is done in standard probability theory. Another key difference in DST is that there is an explicit distinction between uncertainty and lack of confidence, taking the stance that uncertainty is not the same as lack of confidence and should therefore be reasoned with differently. Ultimately, the DST framework allows evidence from multiple sources to be aggregated together via various combination methods, forming a degree of belief that summarizes the corpus of all the available evidence.

For example, suppose we have an election with two candidates: X and Y. A survey asked 100 people about their voting choice. The results showed that 40% of the people will vote for candidate X, 30% will vote for candidate Y, and 30% of the people stated that they are still undecided. In standard probability theory, the 40% probability that candidate X will win means that candidate Y should have a 60% chance of winning by default, since these are the only two possible outcomes and their combination should add to 1. This is not what the observed data is saying, however, and it doesn't take into account the 30% of people who are still undecided. Traditional probabilistic analysis would apply the uniform distribution function justified by the Principle of Insufficient Reason [35]. That would equally distribute the 30% uncertainty between the two possible outcomes in our example, making the probability of winning for candidate X and candidate Y, 55% and 45%, respectively. This, however, is an artificial distribution and it articulates the challenges faced when modeling uncertainty using standard probability methods.

In contrast, Dempster-Shafer theory supports the assignment of weights to all possible combinations of events, as well as allows the explicit representation of uncertainty. Thus, the above example can be expressed as $(X) = 0.4$, $(Y) = 0.3$, and $(X \text{ or } Y) = 0.3$, allowing DST to be more flexible than traditional probability approaches and to better support reasoning in the light of epistemic uncertainty.

5.4 Logic Programming

Logic Programming is a programming paradigm and a way of writing computer programs using languages based on formal logic [27]. Here, logic is used to represent knowledge, which can later be manipulated and used to draw inferences and answer queries, similar to that of a theorem proving process. A program written in a logic programming language is a set of statements that express facts and rules about the world. Due to these characteristics, logic programming languages are considered to be declarative (nonprocedural), meaning that a program specifies what is to be done rather than how to do it.

For example, consider Prolog as the logic programming language of choice. Prolog consists of three basic constructs: facts, rules, and queries. Facts and rules are used to construct knowledge bases, while queries allow the user to ask questions.

Example 5.1. Assume that given information about a person's parents, one would like to determine that same person's grandparents. Also, assume that we have the following knowledge base:

```
mother(susan,joy).        //Susan is the mother of Joy.
father(steve,joy).        //Steve is the father of Joy.
mother(joy,greg).         //Joy is the mother of Greg.
father(georgi,mike).      //Georgi is the father of Mike.

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).   //Rules expressing the grandparent relationship.
```

Given the above knowledge base, one can now write queries to determine the grandparents of certain individuals. For example:

```
?- grandparent(steve,greg).   //Prolog will answer yes.
?- grandparent(georgi,greg).  //Prolog will answer no.
```

6 ASSESSMENTS OF THE ASSURANCE CASE STRUCTURE

We now turn to the analysis of the available AC tool methods for assessing or enhancing the structural correctness of an assurance case diagram. After testing and researching the tools, we established that all of the structural assessment methods supported by them fall into three distinct categories: (1) enforcing structural constraints during argument creation, i.e., cannot produce argument models that do not comply with the predefined meta-model, (2) enabling validation of correctness and completeness of an assurance case's structure (e.g., determining if all claims/goals lead to solutions), and (3) supporting user queries about the argument structure (e.g., queries that display uninstantiated or unconnected nodes in the diagram). Table 3 presents an overview of the structural assessment methods supported by each tool. For example, the tool Astah GSN does not allow the construction of incorrect models, does not feature correctness/completeness checks, and allows syntactic queries.

6.1 Structural Constraints During Argument Creation

The introduction of structural constraints during argument creation is the simplest method for managing the structural correctness of an assurance case diagram. These constraints are not concerned with assessing the validity of an assurance case's structure, but rather aim to minimize the creation of invalid argument models. The implementation of these constraints is typically done with the use of the Epsilon Validation Language (EVL) or Object Constraint Language (OCL), and

Table 3. Table Summary of Tool Support for Assurance Case Structural Assessments

Tool name	Are structural constraints during argument creation available?	Can the model be validated for correctness/ completeness?	Are syntactic queries from the user allowed?
ACedit	Yes	Yes	No
AGSN	Yes	No	No
ASCE	No	Yes	No
Astah GSN	Yes	No	Yes
AutoFOCUS 3	No	Yes	No
CertWare	Yes	Yes	No
D-Case Editor	No	Yes	No
NOR-STA	Yes	Yes	No
Resolute	No	No	No
SafeEd	Yes	Yes	Yes

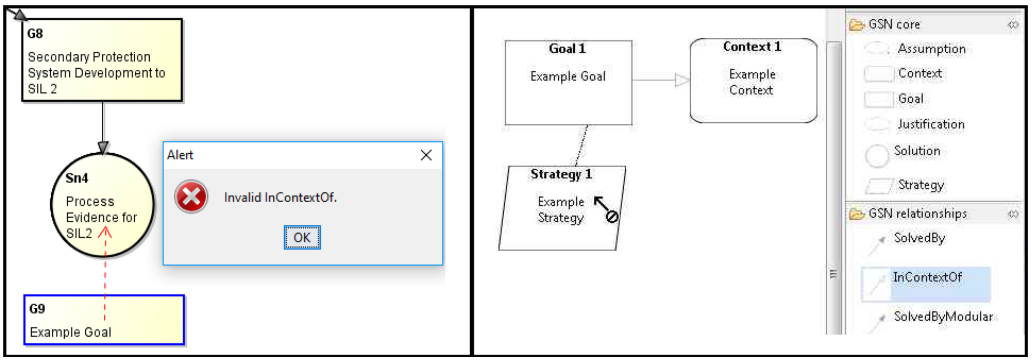


Fig. 7. An example of invalid linking constraints in the tools: (a) Astah GSN, (b) ACedit.

aims to prohibit construction of models that do not comply to the predefined meta-model specifications (e.g., a meta-model following the GSN standard). The structural constraints enforced by each tool listed in Table 3 are respectively dependent on the notation/s that the given tool uses. For example, the tool ACedit, which supports GSN and ARM as assurance case notations, does not allow a user to create structurally invalid diagrams. A list containing the notations supported by each tool is given in Table 2. Figure 7 presents an example of a (GSN) structural constraint during the creation time in the tools Astah GSN and ACedit. The example features two illegal attempts to use the "InContextOf" relationship for connecting a "Goal" node to a: (a) "Solution" node, and (b) "Strategy" node.

This method of structural constraints only avoids the creation of invalid links between elements. It does not account for other structural fallacies, such as circular reasoning among arguments, dangling (unconnected) nodes, or unfinished models (e.g., goals that do not lead to solution nodes), and so on.

6.2 Structural Validation for Correctness/Completeness

This form of structural validation checks may be performed during or after the creation of an assurance case diagram. These checks include meta-model compliance (e.g., that nodes have valid

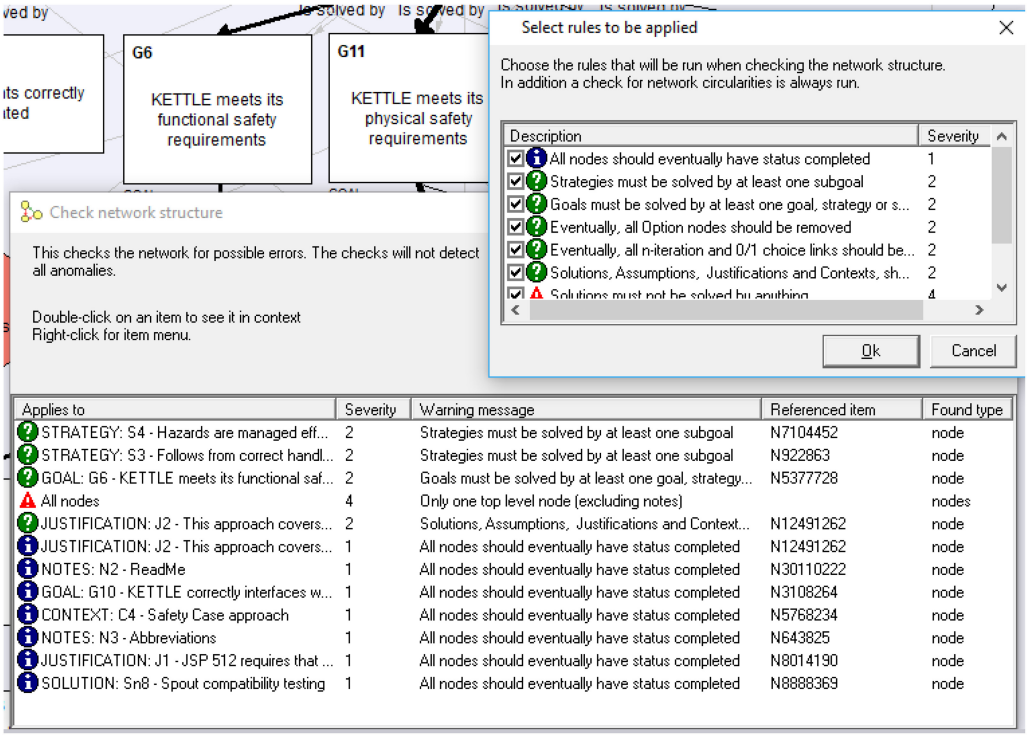


Fig. 8. The rules and results of a structural validation check in the tool ASCE.

connections with respect to the notation used), decomposition validity (e.g., that each goal is solved by at least one sub-goal, strategy, or solution), as well as identify circular argumentation, dangling nodes, nodes without defined properties (e.g., missing ID or description), and the completeness status of nodes. Figure 8 presents the results of a structural validation check in the tool ASCE, as well as the list of available structural checks from which the user can choose. We list each tool's supported structural validation checks below.

- **ACedit.** Includes meta-model validation and checks nodes for missing properties. The tool also offers users "quick fixes" when a violation is found.
- **ASCE.** Supports selectable (can be turned on or off) decomposition rules, checks for dangling nodes, and verifies that all nodes in the assurance case are set as "completed."
- **AutoFOCUS 3.** Validates meta-model correctness, checks for dangling nodes, nodes with undeveloped and/or uninstantiated status, and circular argumentation.
- **CertWare.** Includes decomposition rules, checks for missing node properties, and raises flags if evidence nodes are empty.
- **D-Case Editor.** Supports meta-model validation, validates node properties, and checks for circular argumentation. The user has the option to toggle any of these checks on or off.
- **NOR-STA.** Is able to check and identify missing node properties and validate the correctness of an AC's decomposition.
- **SafeEd.** Can validate an AC's decomposition, check for dangling nodes, circular reasoning, and missing node properties.

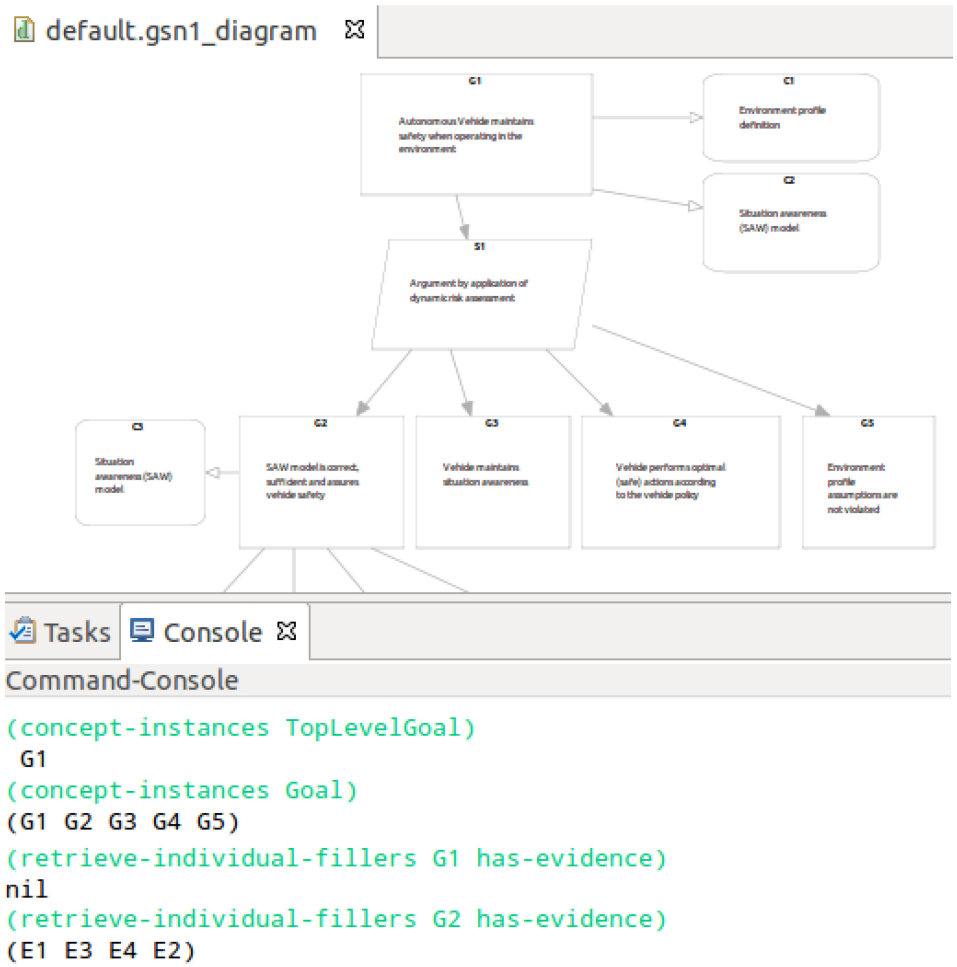


Fig. 9. An example of structural queries conducted in the tool SafeEd.

6.3 Structural Queries

Tool-implemented structural queries allow users/assessors to conduct searches and ask specific questions about an assurance case diagram. These questions can be concerned with the validity of the structure (e.g., a query to display all undeveloped goals) or they can be purely informative (e.g., get all context nodes related to a specific goal). The benefit of asking focused questions about an argument structure, as opposed to only receiving full structural feedback, is particularly pronounced when assessors are dealing with large and complex AC diagrams. This is because assessors can specifically address their questions or concerns without being overloaded with unnecessary information not related to their task. Two tools from our list, Astah GSN and SafeEd, include support for structural queries. Astah GSN supports searches for unconnected nodes and nodes containing expired evidence. SafeEd, however, allows queries for retrieving information about: (a) the top level goal of the diagram, (b) undeveloped goals, (c) the supporting goals of a given node, (d) the supporting evidence of a given goal, and (e) all contexts of a specific goal. Figure 9 presents an example of four queries executed in the SafeEd tool through the console. In the

Table 4. The List of AC Tools Supporting Content Assessment and a Summary of How They Support the Defined Assessment Categories

Tool name	Argument assessment	Evidence assessment	Assessment tracking	Assessment reporting	Assessment interaction
ACedit	None	None	None	None	None
AGSN	Via assignment of quality/confidence values	None	Via node coloring and node properties	Via generated reports	None
ASCE	Via assignment of confidence/strength values	Via change tracking of evidence files	Via node review metadata plugin	None	None
Astah GSN	None	None	None	None	None
AutoFOCUS 3	Via integration with system models	None	None	None	None
CertWare	Via assignment of probabilities	None	None	None	Via assessment UI
D-Case Editor	Via type checking	None	None	None	None
NOR-STA	Via assignment of quality/confidence values	Via presence tracking of evidence	Via node coloring and information logs	Via generated reports	Via assessment UI
Resolute	Via analysis on system models	None	None	None	None
SafeEd	None	None	None	None	None

example, the user retrieves: the top-level goal of the AC, all goals in the AC, the evidence nodes of the G1 goal, and the evidence nodes of the G2 goal, respectively.

7 ASSESSMENTS OF THE ASSURANCE CASE CONTENT

In this section, we discuss the AC content assessment features of each tool supporting this type of assessment. Seven out of the 10 tools in our study meet this requirement (see Table 4 for the list of tools). The remaining three tools, ACedit, Astah GSN, and SafeEd, are limited to assessing the AC structure and are not further discussed in this section.

Five recurring categories of assessment support were established after concluding the hands-on testing of the tools: (1) argument assessment, (2) evidence assessment, (3) assessment tracking, (4) assessment reporting, and (5) assessment interaction. These categories capture whether and how the tools: assess the assurance case argument, assess externally attached evidence supporting the argument, track the assurance case assessment process, report the assessment results, and provide a dedicated UI for assessment interaction. Table 4 presents each tool discussed in this section and a high-level summary of how it supports each of the defined assessment categories. For example, the CertWare tool includes support for assessing an assurance case's argument through the use of probabilities and offers a dedicated UI for easier assessment. We discuss each tool and its supported methods in detail below.

7.1 AGSN

The Assessable GSN (AGSN) tool primarily facilitates assurance case assessment, focusing on GSN-based ACs and supporting the importing of assurance cases developed using the D-Case Editor tool. The tool also provides support for argument assessment, assessment tracking, and assessment reporting, which are discussed below.

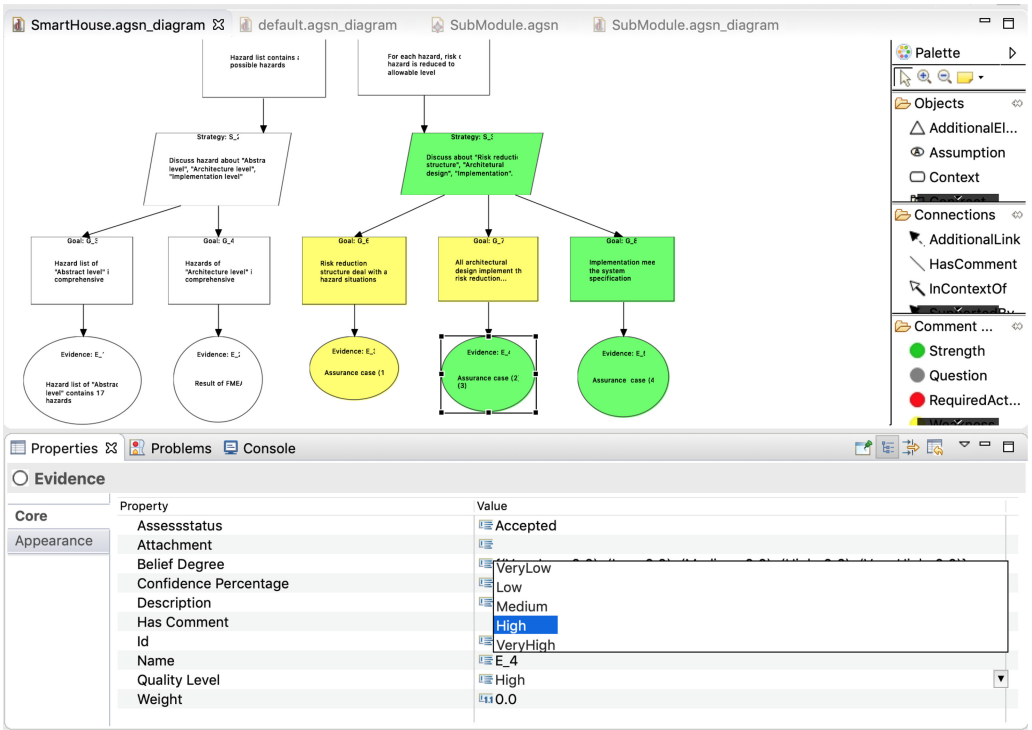


Fig. 10. Assigning quality, confidence, and weight scores using AGSN.

7.1.1 Argument Assessment. AGSN allows assessors to quantitatively evaluate an assurance case argument by setting quality, confidence, and weight values to each node in the assurance case. This is done through the use of node properties. Quality-level scores range from 1 (Very Low) to 5 (Very High), following the Likert scale. Confidence-level values range from 0% to 100% and are used to denote how certain the assessor is in his/her assigned quality-level score. The third property, weight, is used to specify how important a sub-goal is to the satisfaction of its parent goal, relative to the other sub-goals supporting that parent goal. Figure 10 illustrates the node properties window through which assessors may assign quality, confidence, and weight values.

When the above node scores are assigned, AGSN can calculate the overall score of an assurance case by aggregating the individual node scores and propagating them to the root goal (claim) of the assurance case. The aggregation of the scores is achieved through the use of the Evidential Reasoning (ER) [40] algorithm, which utilizes the evidence combination rule of Dempster-Shafer theory discussed in Section 5. For AGSN to successfully calculate the score of an AC, the relevant AC nodes need to have their quality- and confidence-level properties set. If no weight values are assigned, all sub-goals supporting a goal are assumed to be equally important for its satisfaction. Only the goal, strategy, and evidence nodes are considered by AGSN when calculating the overall score of an assurance case.

7.1.2 Evidence Assessment. None.

7.1.3 Assessment Tracking. AGSN tracks assessment progress through the assignment of an assessment status node property. This property can take one of five states, namely, "Not Reviewed," "Accepted," "Incorrect," "Weak," and "Review Later," indicating the different possible node

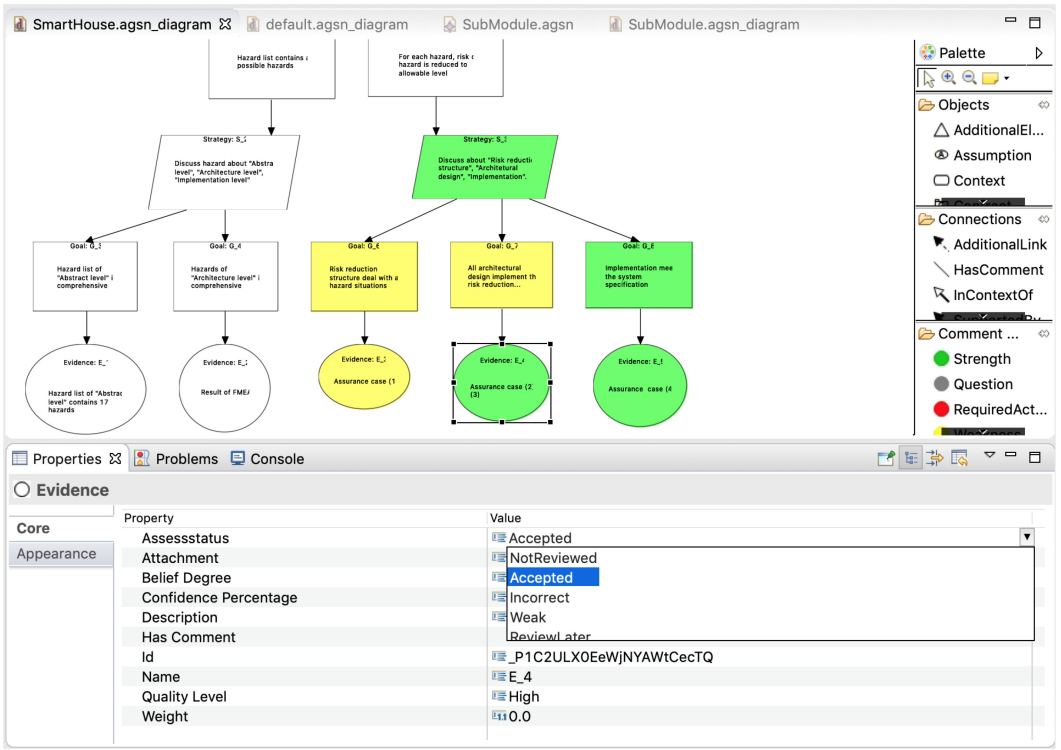


Fig. 11. AGSN assessment status property.

assessment states supported by the tool. Figure 11 presents the properties window through which an assessor may select assessment status values. Upon creation of the assurance case diagram, each node has its assessment status set to "Not Reviewed" by default. Once the assessment process is underway, the assessors can choose appropriate labels for each node based on their judgment.

Visual cues for tracking assessment status are also offered by the tool. Nodes in the assurance case diagram are colored based on their set assessment status property. Nodes marked as "Not Reviewed," "Accepted," "Incorrect," "Weak," and "Review Later" are colored white, green, red, yellow, and grey, respectively. This allows assessors to easily distinguish and understand the review status of nodes.

7.1.4 Assessment Reporting. AGSN can generate two assessment reports, a "Quality Distribution" and an "Assessment Status Overview" report. The Assessment Status Report contains a statistical bar chart overview of the overall assurance case assessment status (see Figure 12), as well as individual reports of the assessment status for the goal, strategy, and evidence nodes. Similarly, the Quality Distribution report offers a statistical report of the overall assurance case quality (see Figure 13), as well as individual quality reports for the goal, strategy, and evidence nodes.

Assessors may also add comments to the assurance case in the form of four comment node elements: "Strength," "Required Corrective Action," "Weakness," and "Question." Comment nodes are dragged and dropped into the canvas of the tool and can be linked to the assurance case via a "HasComment" relationship. These comments can be used to ask questions, provide assessment rationale and judgment, as well as to give overall feedback to the assurance case developers.

7.1.5 Assessment Interaction. None.

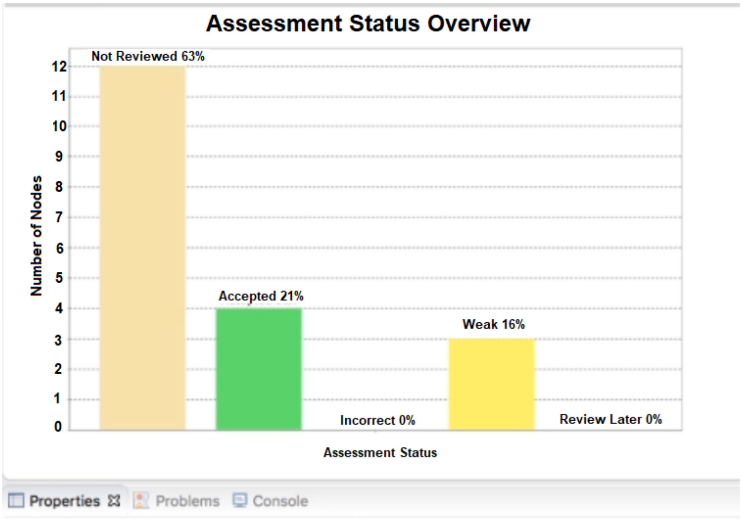


Fig. 12. An example assessment status report in the tool AGSN.

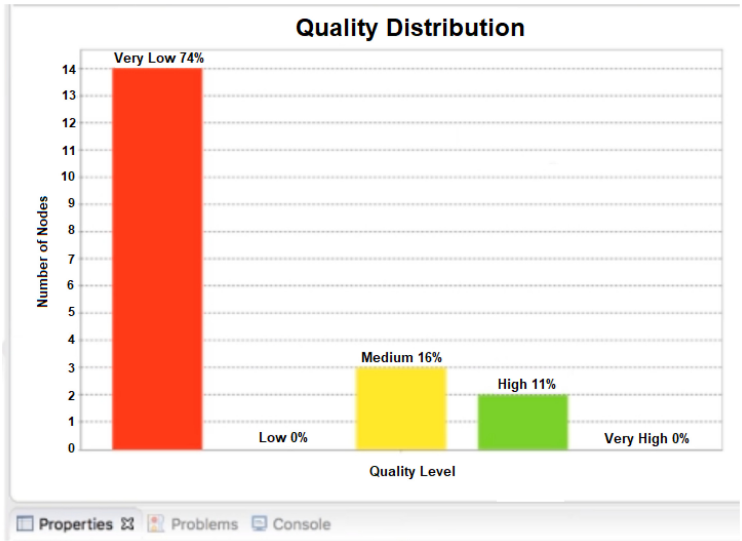


Fig. 13. An example quality distribution report in the tool AGSN.

7.2 ASCE

ASCE is a commercial software tool developed by the company Adelard LLP. Its primary focus is support for creating and maintaining assurance cases, and it is one of the most widely used assurance case tools in industry. The supported assessment features of ASCE are discussed below.

7.2.1 Argument Assessment. ASCE allows users to set confidence and strength values on the various components of the assurance case. Nodes can have their confidence values set to low, medium, or high. These values can then be used in conjunction with different display rules. An example of one such rule is the "traffic light" rule that colors the background of nodes depending on their set confidence value. The colors are red, amber, and green, corresponding to the confidence

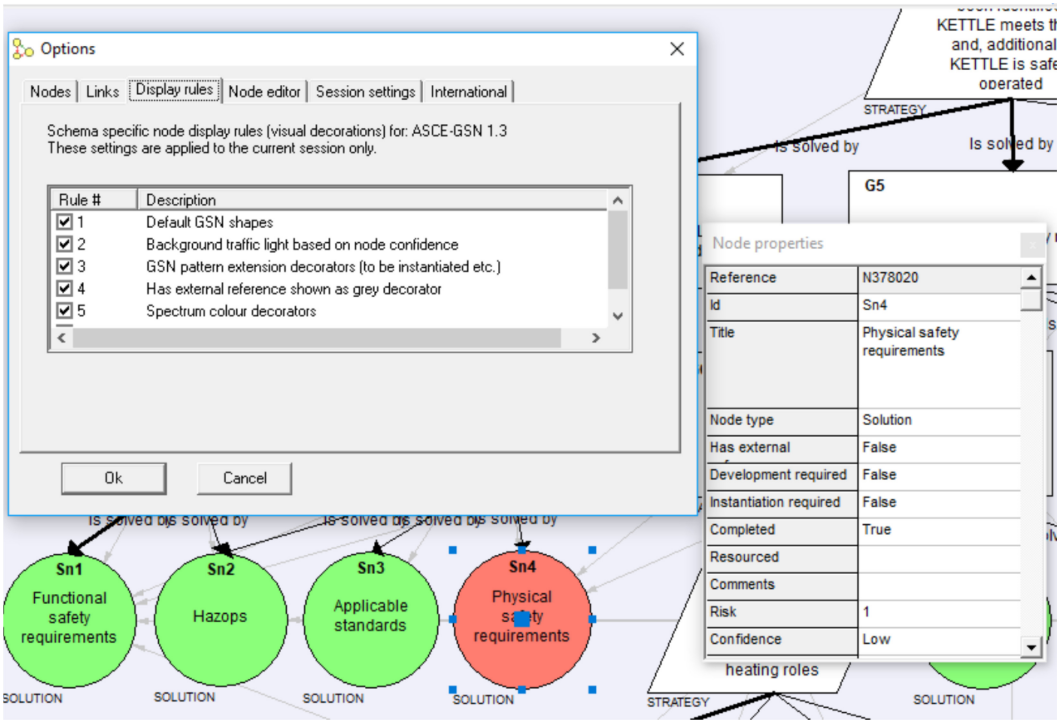


Fig. 14. Available node display rules in the tool ASCE.

values of low, medium, or high, respectively. Figure 14 shows a list of the available display rules featured in the tool (left), and the node properties window through which an assessor may select node confidence values (right).

Links connecting nodes in the assurance case can similarly have link strength values that can be assigned. These values range from one to five and are used to represent the strength of the argument being made between two nodes. After the application of these values, the user can query different assurance case views based on a variety of criteria with the use of a view filter. An example would be to filter and display only certain types of nodes of the assurance case, or only parts that have links with a certain range of link strengths, i.e., the user can specify only nodes with a link strength of two or higher to be visible. The tool's graphical view filter is presented in Figure 15.

7.2.2 Evidence Assessment. Attached files are tracked by the tool, allowing the user to perform evidence validation checks to uncover changed or no longer valid evidence attachments. Upon changes to evidence files, the tool has the ability to display the old and new file versions, along with information about the change date and size difference. Figure 16 presents the results of a performed validation check that returns two broken links.

ASCE also comes with a variety of plugins and allows users to create and integrate their own extensions. These can significantly extend the capabilities of the tool. Plugins are split into two categories, "Dynamic Narrative Regions" (DNRs) and "Macro plugins." DNRs support the mapping of content from external documents, such as spreadsheets, databases, Word files, and so on, which can be integrated into nodes of the diagram or used to generate assurance case reports from a variety of materials held and maintained in different locations. This external content is traced to its source files and can be checked for validity or consistency.

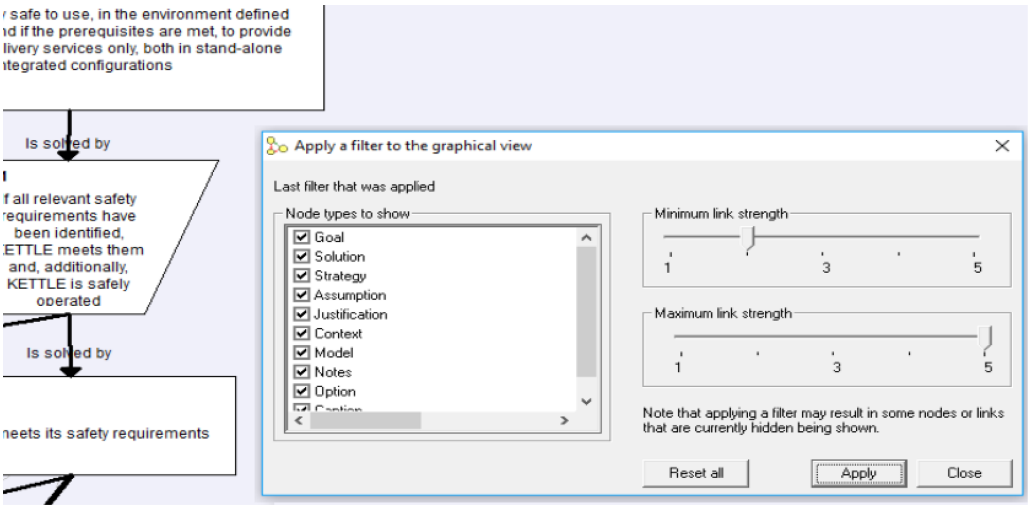


Fig. 15. The graphical view filter in the ASCE tool.

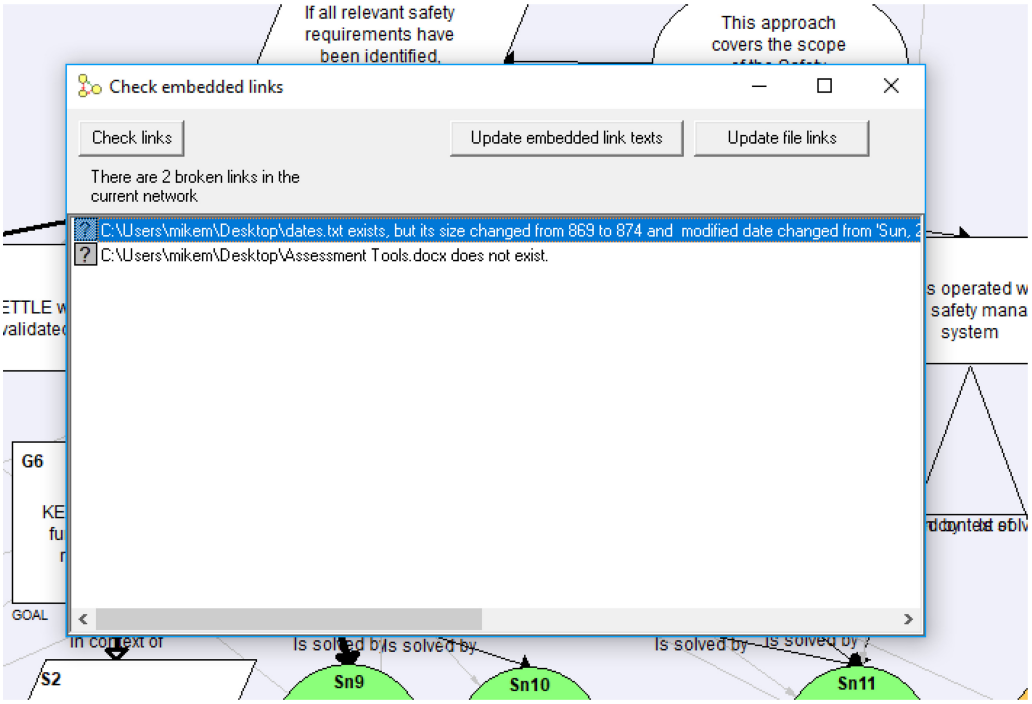


Fig. 16. ASCE change tracking in evidence files.

7.2.3 Assessment Tracking. Another DNR plugin implemented in ASCE is the "Node review metadata manager." It supports the assignment of node review metadata and allows nodes to contain information about their review history, last review date, next review date, and other information via user notes.

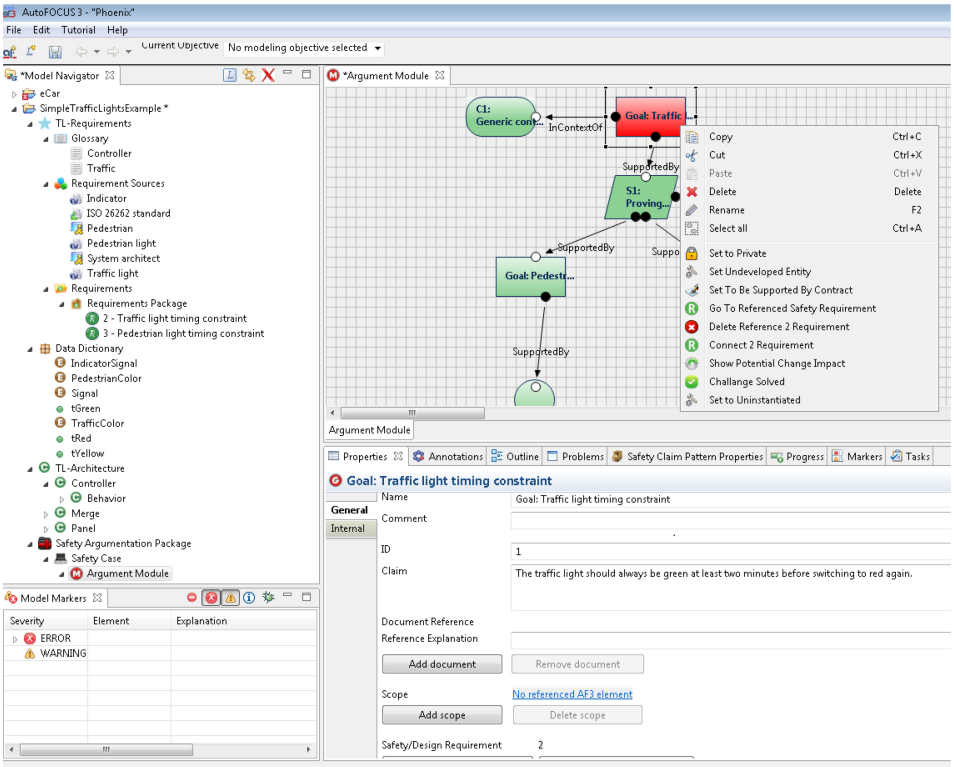


Fig. 17. An example of the Goal node actions supported by the tool AutoFOCUS 3.

7.2.4 Assessment Reporting. None.

7.2.5 Assessment Interaction. None.

7.3 AutoFOCUS 3

AutoFOCUS 3 (AF3) was originally a model-based development platform for developing embedded systems that has been extended to support the construction of GSN-based assurance cases. It was created by the Technical University of Munich and is currently being maintained by its spinoff company Fortiss. AF3 supports a wide collection of models, such as design and hardware architectural models, requirement models, logical architecture models, and others.

7.3.1 Argument Assessment. The main feature of the tool is its substantial support for hyperlinking model-based artifacts and embedding them into the assurance cases. This integrated environment enables assurance case assessors and developers to have quick access to the wide variety of safety-relevant information contained in the model-based components of the system. The AF3 environment supports artifacts ranging from various stages of the safety assurance process, such as requirement and development models, as well as implementation and verification artifacts, such as generated code, formal verification, and testing results. These artifacts can be hyperlinked to specific words in the text of a node or to an entire GSN node.

Each node type in the assurance case (excluding strategy and justification nodes) has a specific subset of artifact types that it can reference to. Goal nodes are primarily connected to safety requirement documents living in the "Requirements" section of the project. Figure 17 presents the

tool's interface and the available goal node actions/properties. Once a goal node is connected to a system requirement artifact, the former will mirror the name and claim defined in the latter. Context nodes can be referenced to the test suites of certain logical components in the project and may only connect to test suites belonging to the system defined in their associated goal node. Solution nodes can be referenced to test suite results, generated code, test coverage files, and more. Similarly, only test coverage files from the test suite defined in the context node to which the solution node is associated with are available for selection. Last, assumption nodes can reference documents describing a specific state (state diagrams) or mode of a system.

Once system artifacts have been linked to the assurance case, a dependency between them is created. This means that changes in the underlying system models can be reflected in the assurance case in the form of error and warning messages. These messages notify the user/assessor when nodes are referencing to outdated system properties that no longer hold.

7.3.2 Evidence Assessment. None.

7.3.3 Assessment Tracking. None.

7.3.4 Assessment Reporting. None.

7.3.5 Assessment Interaction. None.

7.4 CertWare

The CertWare workbench is an open-source NASA-sponsored tool developed by Kestrel Technology LLC. It is described as a tool for developing, maintaining, and analyzing assurance cases. The tool itself offers little documentation and no longer seems to be supported, but is advertised to provide argument assessment through the use of Bayesian reasoning as well an assessor's view in the form of an assurance case assessment interface. Unfortunately, the Bayesian reasoning functionality of the tool did not work and could not be used and tested. All of the information regarding it was acquired from the CertWare website [26]. The features of the tool are discussed below.

7.4.1 Argument Assessment. The CertWare workbench supports Bayesian reasoning through the construction of Bayesian networks (refer to Section 5). For its reasoning platform, CertWare uses the publicly available sensitivity analysis and inference engine called SamIam [9], developed by the Automated Reasoning Group at the University of California. Safety engineers/assessors using the tool can construct Bayesian networks that model the causal or conditional relationships among the nodes in an assurance case. Once the network is constructed, leaf nodes (particularly their supporting evidence) are manually assessed and assigned prior probability values. These values are then used to compute a probability of belief for the various goals in the argument, particularly the top-level goal. This approach is similar to the one discussed in the tool AGSN (and later for the tool NOR-STA), where scores are manually assigned to lower-level nodes of an assurance case and propagated to the top claim using Dempster-Shafer theory.

Assessor can run queries on the network model through the provided "Hugin Network Analysis" view featured in Figure 18. This view displays the network file as a structured tree, where the network variables are the branch nodes and the variable states are leaves on each node (only mutually exclusive states are supported). The assessor may then insert evidence by selecting different variable states via the provided checkboxes.

7.4.2 Evidence Assessment. None.

7.4.3 Assessment Tracking. None.

7.4.4 Assessment Reporting. None.

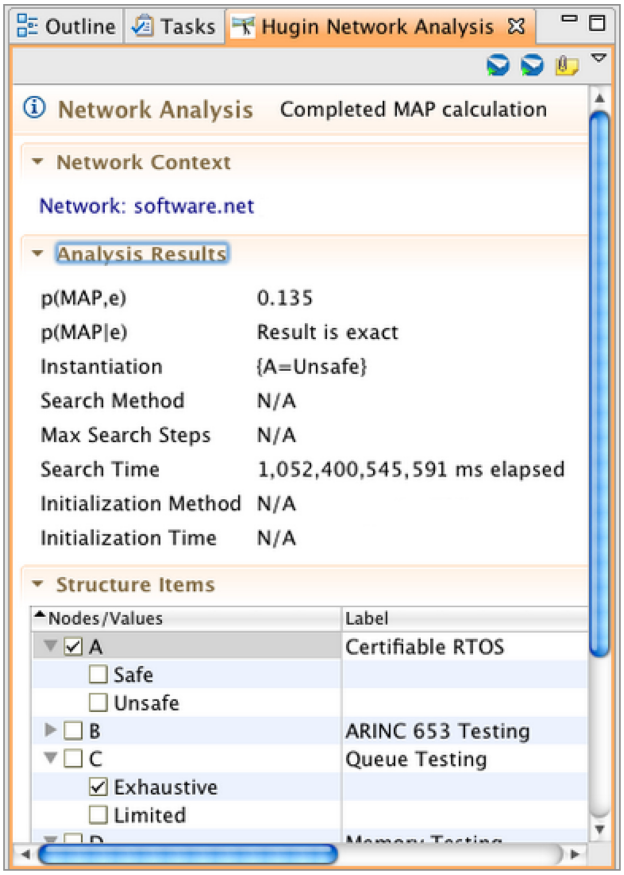


Fig. 18. CertWare's Hugin Network Analysis view used to perform Bayesian reasoning. Adapted from Reference [26].

7.4.5 Assessment Interaction. CertWare includes an assessment interface for semi-formal proofs, which are defined in the tool website [26] as "proof structures in which the overall argument of a proof is a formal data structure, while individual propositions within the structure are expressed either formally or informally." This is motivated by the need to make the proof easier to review by a human assessor while preserving some automated capabilities. This assessment interface can be used as an easy way to represent and evaluate assurance case claims by representing them as statements. Figure 19 presents an example of an assurance case represented as statements in a semi-formal proof file. Once a semi-formal proof file has been created with the necessary statements, hypotheses, entailments, and so on, an assessor may run a review wizard that allows him to assess the proof file in a step-by-step manner, prompting him to make decisions on whether the inferences made are valid or invalid, as shown in Figure 20. The semi-formal proof editor also supports a variety of automated verification rules that prompt error notifications if broken. The verification rules contained in the editor are the following:

- Each justification must be non-empty.
- An entailment statement cannot include itself in the head or tail.
- A proof cannot contain cycles.

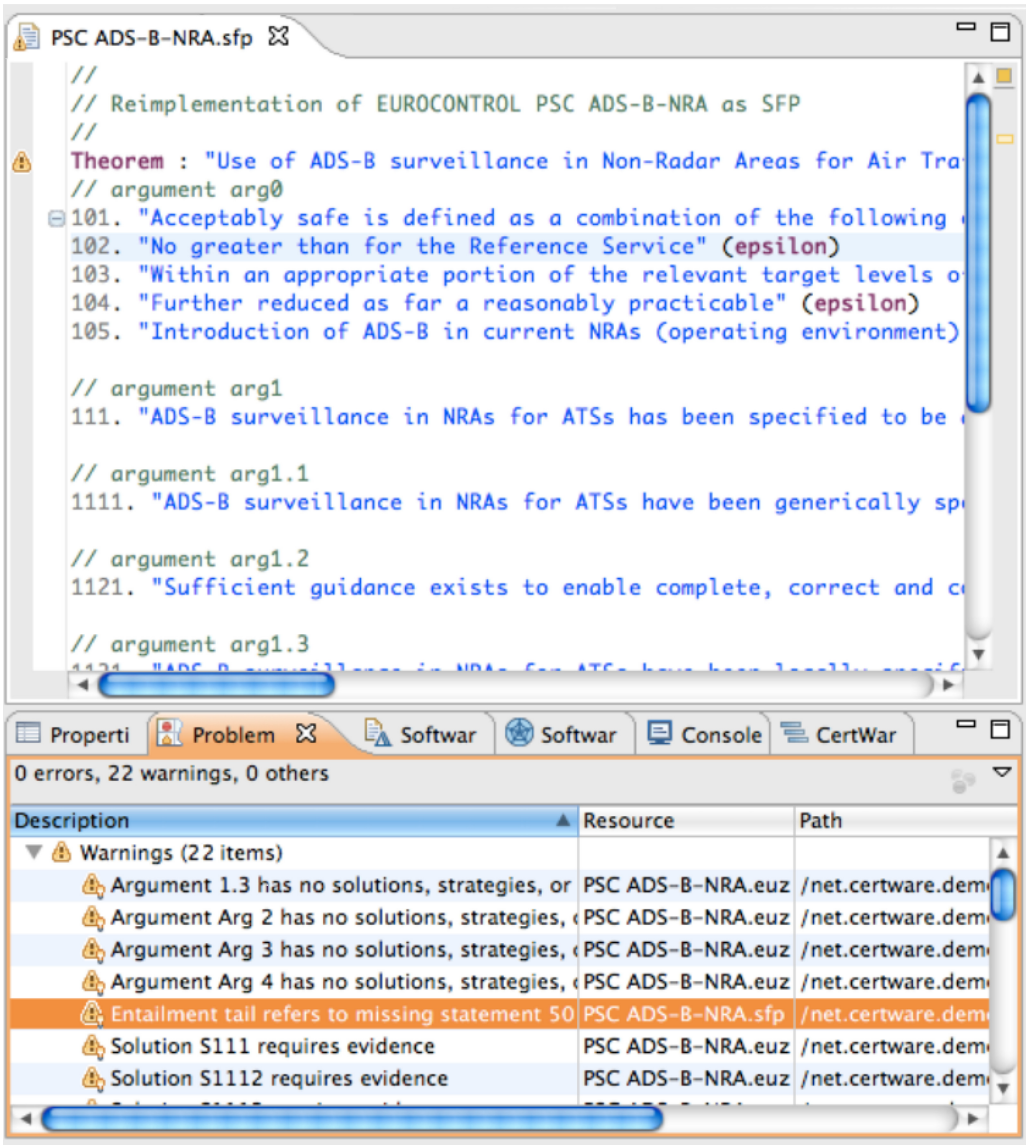


Fig. 19. A safety claim represented as a semi-formal proof in CertWare. Adapted from Reference [26].

- A proof must have at least one proof step.
- A justification identifier must exist.
- A statement justification cannot refer to itself.
- An entailment tail identifier must exist.
- Each entailment head identifier must exist.

7.5 D-Case Editor

The D-Case Editor is a free and open-source Eclipse plug-in developed as part of Japan's DEOS (Dependability Engineering for Open Systems) project. The main advertised features of the D-Case

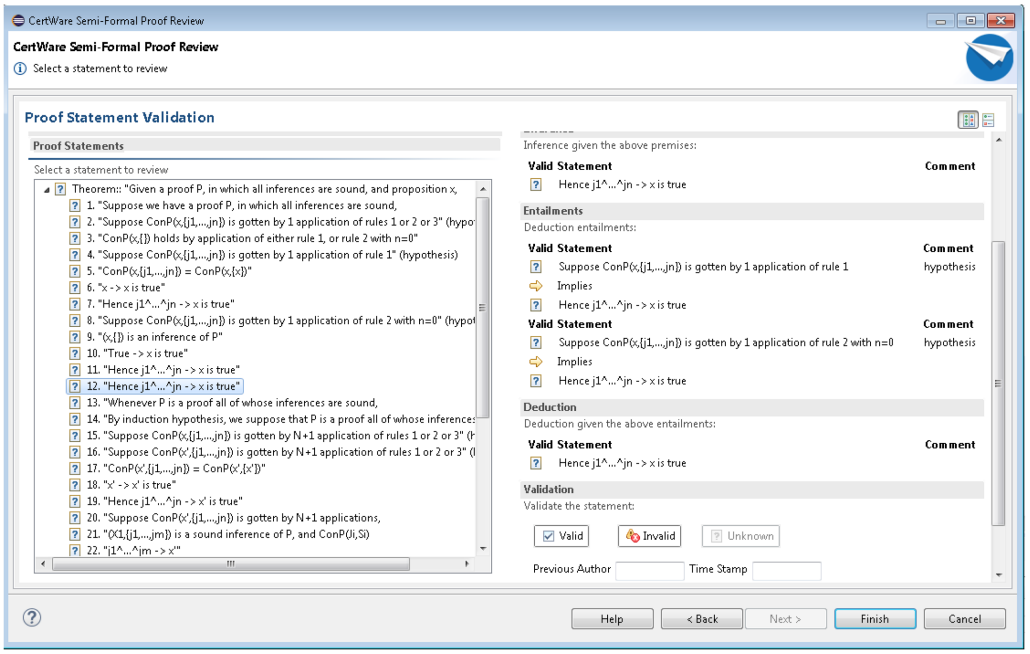


Fig. 20. The semi-formal proof assessment interface contained in the CertWare workbench.

Editor are support for constructing GSN-based assurance cases, a GSN pattern library that uses prototype type-checking functions, and a consistency-checking mechanism.

7.5.1 Argument Assessment. The D-Case Editor implements the concept of type checking (discussed in Section 5) into assurance case arguments. This programming language concept was implemented in the tool through the work of Matsuno and Taguchi [31, 32], with the goal of minimizing and avoiding the incorrect use of GSN patterns [22]. It allows the D-Case Editor to create GSN patterns that include parameterized expressions with types, scoping rules for those types, and a type-checking mechanism for validating the correctness of their use. Basic types (e.g., integers, strings, and floats) are currently supported. The declaration of types and the instantiation of variables is done in the context nodes of the assurance case. The scope of variables defined in a context node begin with its associated goal node and cover all subsequent nodes in the sub-trees leading from that initial goal node. Figure 21 illustrates this with a simple example of an AC that has two declared variables: "Automobile" and "Automobile2."

The D-Case Editor also features an extension called D-Case/Agda that is created by Makoto Takeyama [38]. It is a tool used for constructing D-Case arguments and verifying their consistency using the dependently typed functional programming language and proof assistant Agda [1]. D-Case/Agda allows users to create GSN diagrams both in the D-Case Editor and in Agda itself, through the use of formal semantics. Figure 22 illustrates an example AC created in the D-Case Editor (left) and Agda (right). The user/assessor can switch between the two programs and validate the assurance case diagram in Agda, where it is represented as a proof tree and can be checked for type consistency and structural soundness. The diagram is analyzed as a tree, and the scoping rules for variables follow the branches (sub-trees) that they are on. Unfortunately, D-Case/Agda seems to no longer be supported and could not be downloaded for actual use at the time of this writing.

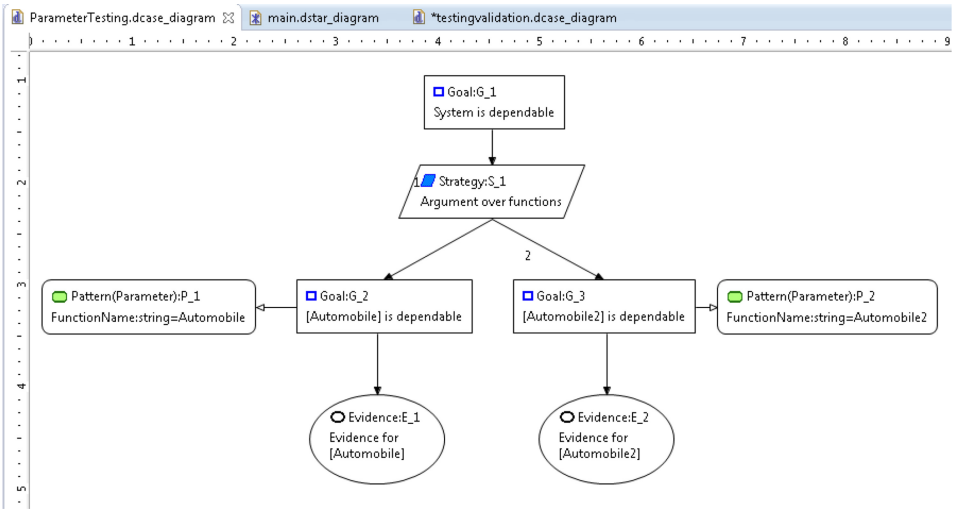
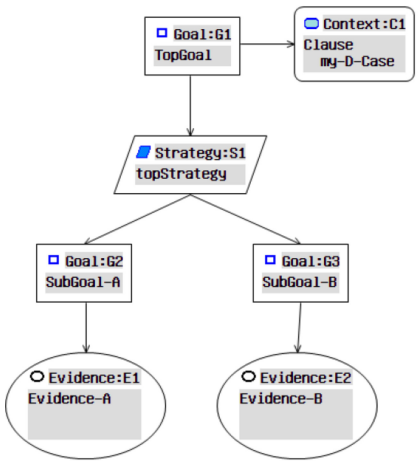


Fig. 21. A simple example of an instantiated parameterized pattern in the D-Case Editor.




```
module Example0 where
-- import the functions used for writing D-Cases
open import DCaseSpecShallow

-- Preparatory declations
postulate
  TopGoal SubGoal-A SubGoal-B : Set
  topStrategy : SubGoal-A → SubGoal-B → TopGoal
  Evidence-A  : SubGoal-A
  Evidence-B  : SubGoal-B

-- The example D-Case written as an Agda term
my-D-Case =
      TopGoal ∋
        topStrategy
        • (SubGoal-A ∋ Evidence-A)
        • (SubGoal-B ∋ Evidence-B)
-- The pragma that designates my-D-Case as
-- the root of a D-Case tree
{-# DCASE my-D-Case root #-}
```

Fig. 22. An example of a simple GSN diagram represented using the D-Case Editor (left) and Agda (right). Adapted from Reference [38].

The two type/consistency checking mechanisms implemented in the tool help avoid the misplacement of variables at creation time. D-Case/Agda may also analyze a complete assurance case and check the correctness of an argument with regards to its type consistency. Although currently limited to that of basic types, the type consistency validation may be particularly useful when assessing large assurance cases.

7.5.2 Evidence Assessment. None.

7.5.3 Assessment Tracking. None.

7.5.4 Assessment Reporting. None.

7.5.5 Assessment Interaction. None.

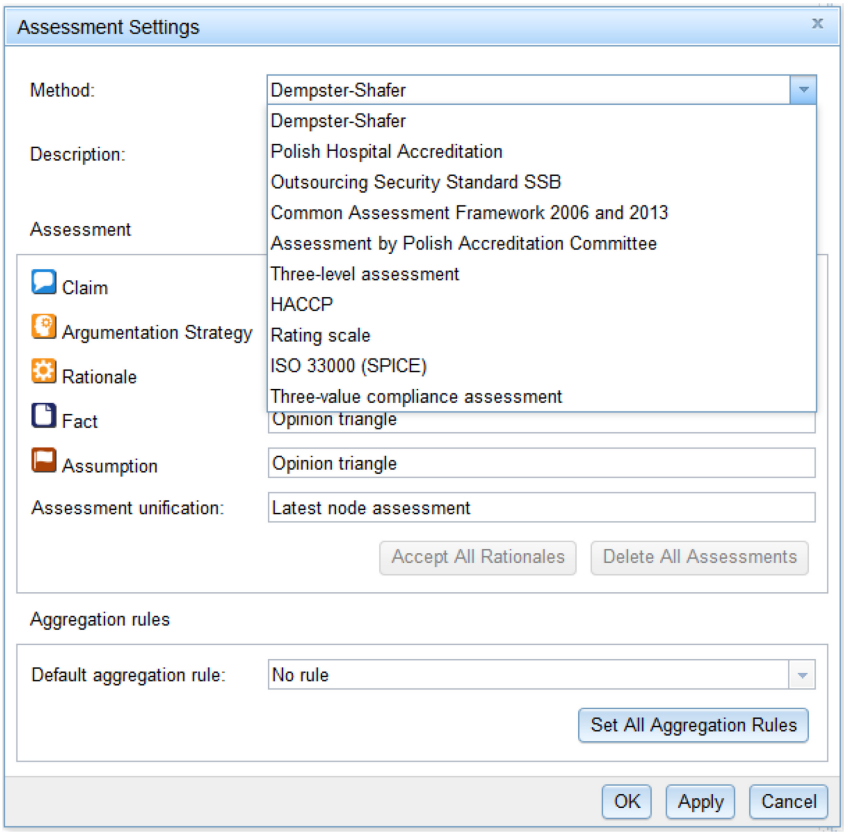


Fig. 23. A list of the assessment methods offered in NOR-STA.

7.6 NOR-STA

NOR-STA is a web-based assurance case tool developed by the Gdańsk University of Technology and is currently maintained by its spin-off company Argevide. It uses the TRUST-IT [18] argumentation model as the notation for constructing assurance cases but offers the option to export and visualize assurance cases in the GSN notation. The tool is described as a platform that supports argument construction, assessment, communication, and management. The assessment features available in the tool are discussed below.

7.6.1 Argument Assessment. NOR-STA provides assessors with 10 assessment techniques that are based on linguistic appraisal scales (see Figure 23 for the list of techniques). The aim of these techniques is to apply a mathematical formalism to quantitatively reason about arguments, as well as to capture uncertainty due to a lack of evidence, or argument fallacies.

All 10 assessment methods offered in NOR-STA follow the same underlying principle. The methods use simple scales to capture user/assessor information about the strength or validity of an argument and its supporting evidence. This information is then aggregated to obtain a conclusion about the overall assurance case (same concept as the one described for the AGSN tool). Out of the 10 available methods, the one that is most used for assessing assurance cases is the method utilizing Dempster-Shafer theory. This method features two linguistic scales, the decision scale and the confidence scale (see Figure 24).

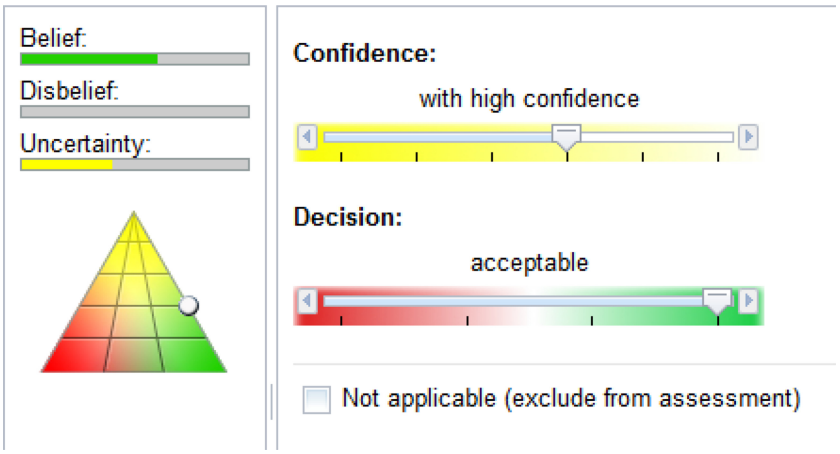


Fig. 24. The assessment scales under the Dempster-Shafer method of assessment in NOR-STA.

The decision scale is used to capture whether the assessor believes that the assessed element should be accepted or rejected. Four decision values, "acceptable," "tolerable" (a soft accept), "opposable" (a soft reject), and "rejectable" are offered and may be chosen from. The confidence scale is used to capture the confidence for the choice made in the decision scale, and offers six degrees of confidence: "for sure," "with very high confidence," "with high confidence," "with low confidence," "with very low confidence," and "lack of confidence." The combination of choices from these two scales are plotted on a two-dimensional graph and mapped onto the Josang's opinion triangle [21] for a more intuitive visualization, as seen in Figure 24. Assessors have the choice to select their assessment values from the traditional scales or the opinion triangle. Note that all of the "lack of confidence" points in each of the decision values featured in the plotted graph are mapped to a single point of uncertainty at the tip of the opinion triangle. This means that there is no distinction between the various accept/reject decisions if they are accompanied by high levels of uncertainty.

The Dempster-Shafer assessment method allows the assessor to choose from four different aggregation rules. These rules define how the assessment values for each individual node are aggregated to calculate the score for the overall assurance case. The supported aggregation rules as described in Reference [11] are:

- **A-argument rule.** "In A-arguments rule, confidence in assessments coming from different argumentation branches is reinforced if the assessments agree, or it is decreased if they contradict each other."
- **C-argument rule.** "C-argument relates to a situation where the premises provide complementary support for the conclusion. In such a case not only the assessments of the premises and the warrant but also the weights associated with each premises are taken into account."
- **SC-argument rule.** "In SC-arguments, acceptance of the premises leads to the acceptance of the conclusion, as for NSC-arguments. However, rejection of a single premise leads to the rejection of the whole inference, i.e., to complete lack of confidence."
- **NSC-argument rule.** "In NSC-arguments, negative assessments are strongly reinforced. In such arguments the acceptance of all premises leads to the acceptance of the conclusion, whereas rejection of a single premise leads to the rejection of the conclusion."

Once decision and confidence values have been set to the elements in the assurance case, NOR-STA provides visual cues of the assessment results in the form of argument colors. Figure 25 shows an example of colored arguments based on their set decision and confidence values.

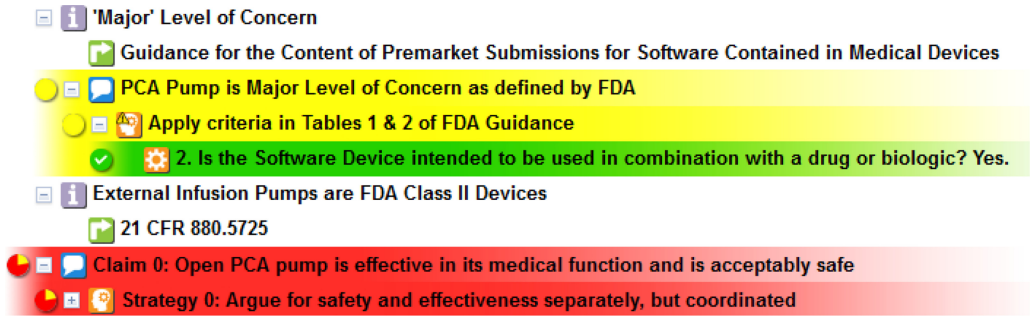


Fig. 25. An example of arguments in NOR-STA being colored based on their decision and confidence values.

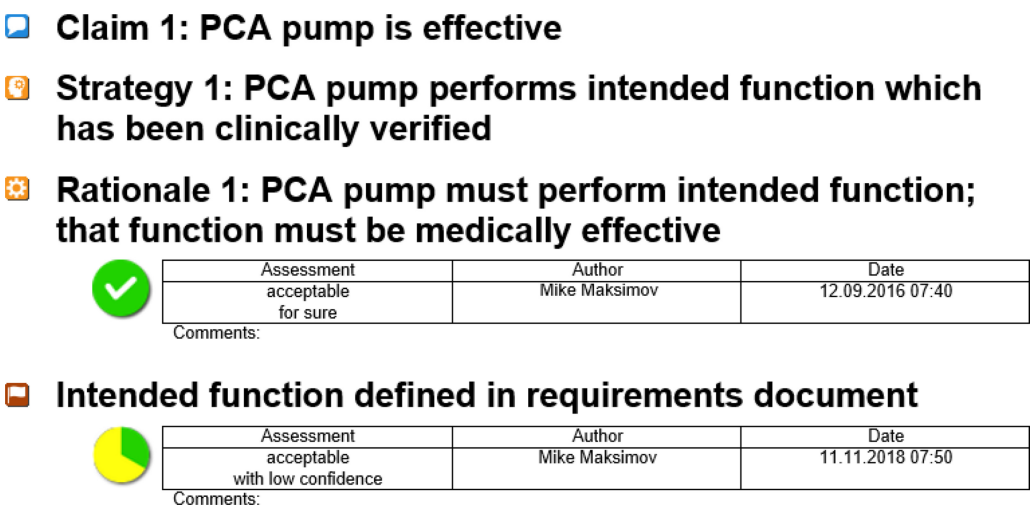


Fig. 26. A snippet from a NOR-STA-generated assurance case report.

7.6.2 Evidence Assessment. NOR-STA displays small warning signs on reference element nodes (the tool uses these nodes to store evidence for claims) that do not have any evidence contained in them, notifying users of which elements still need supporting evidence. Evidence files can also have expiration dates manually set to them, prompting a similar warning that notifies the user/assessor of which reference elements need to be revisited and rechecked due to the expired evidence files contained in them.

7.6.3 Assessment Tracking. Assessment status tracking is achieved by the previously mentioned visual cues. Each assessed element is colored to represent its assigned score. The tool can also track and display the history of the assessment process by logging user actions and account information.

7.6.4 Assessment Reporting. The tool can generate two reports, an "Assurance Case Report," and a "Project Metrics Report." The former represents the assurance case in a text-based document, featuring all the claims of the assurance case in a top down manner, an assessment breakdown of each claim with the corresponding assessor comments, along with the evidence artifacts and any other documents relevant to the assurance case (see Figure 26). The latter report contains an overview of the assurance case metrics, including structural information (e.g., number of elements used), quality information (e.g., size statistics, completeness statistics), and others.

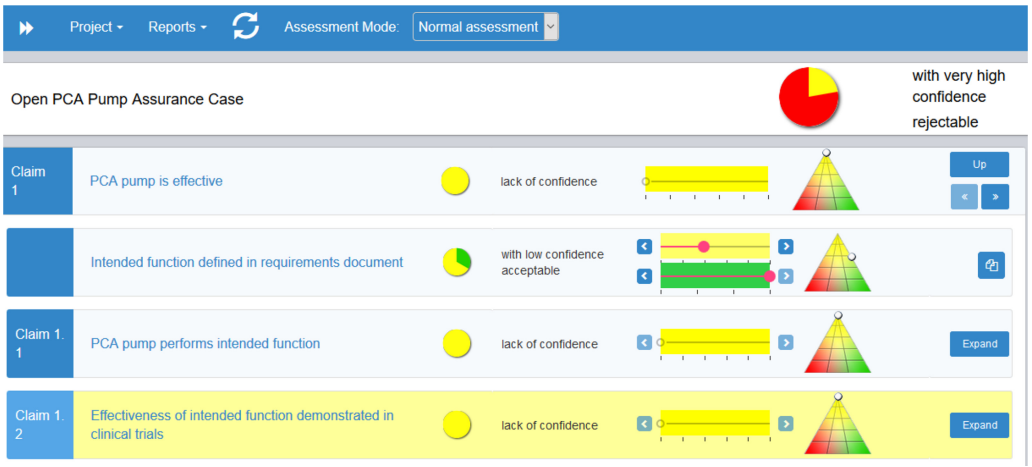


Fig. 27. NOR-STA's assessment view.

7.6.5 Assessment Interaction. NOR-STA provides a dedicated assessment view by allowing the assessor to review all the claims in the assurance case in a step-by-step manner (see Figure 27). While in this interface, the assessor has the ability to review the details of each claim and set decision/confidence values. A summary of the overall assessment status of the assurance case is also available at the top of the interface.

7.7 Resolute

Resolute is an assurance case tool built on top of the Eclipse-based Open Source AADL Tool Environment (OSATE) [37]. The resulting framework can automatically generate assurance cases based on system models specified in the AADL architectural design language and a set of user-defined logical rules used to run formal analyses on those models.

7.7.1 Argument Assessment. The functionality of Resolute is inspired by logic programming principles such as the ones discussed in Section 5. An assurance case created by the tool can be thought of as a proof tree in Resolute logic, where user-formulated claims and sub-claims represent the nodes of the tree. Claims in Resolute correspond to a first-order predicate. For example, a user might represent the claim "The component **x** only receives messages that pass Decrypt" using the predicate **only_receive_decrypt(x : component)**. Claims are also accompanied by user-defined logical rules. If a claim's rule is satisfied by Resolute's analysis, that claim is then considered valid. The tool's shared environment with system models allows both claims and rules to be parameterized by AADL variable types (e.g., threads, systems, memories, connections, etc.), along with basic types (e.g., integers, strings, Booleans, floats). Figure 28 presents an example of two claims and their accompanying rules.

Once the assurance claims and their corresponding rules have been defined and parameterized with system variables, Resolute can generate an assurance case by running an analysis on those claims. Figure 29 presents an example assurance case constructed in Resolute. From the image, we can see that Resolute represents every assurance claim on a separate line, with supporting sub-claims being indented underneath the parent claim. Resolute analyzes claims about a system against the system's properties defined in the models. If a claim has successfully been proven, it is indicated with a green check mark. Similarly, a red exclamation mark is displayed for claims that have failed.

```
annex Resolute {**
    MaximumWeight : real = 1.2kg

    SCSReq1(self : component) <=
    **  "R1: SCS shall be no heaver than " MaximumWeight%kg **
     SCSReq1VA1(self, MaximumWeight) or SCSReq1VA2(self, MaximumWeight)

    AddBudgets(self: component) : real =
        sum({WeightBudget(t) for (t: subcomponents(self))})
```

Fig. 28. An example of claims and their rules written in Resolute.

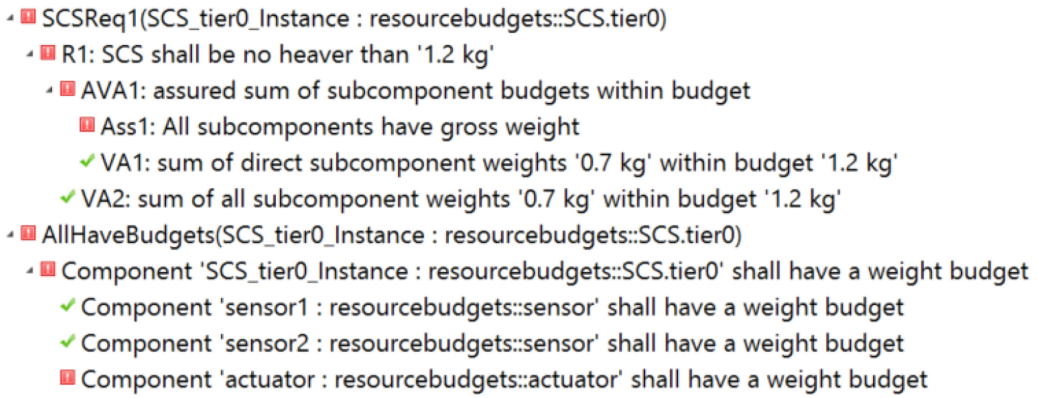


Fig. 29. A simple example of an assurance case constructed using Resolute.

Resolute's framework supports type checking and parameterized claims by definition, meaning that type errors and undefined references will be caught by the tool. The dependency created by Resolute's integrated environment also enables the tool to capture system design changes and reflect them on that system's assurance case. An important note is that assurance cases constructed in Resolute are only ensured to be compliant with respect to the claims and rules the user has defined. The validity of an assurance case created by Resolute ultimately depends on, and is subject to, expert review.

7.7.2 Evidence Assessment. The unique approach and tightly coupled environment utilized by Resolute allows evidence for the assurance case to be directly gathered from systems design artifacts, through the use of user-defined logical rules and claims. Resolute does not, however, offer assessment techniques to analyze the evidence itself.

7.7.3 Assessment Tracking. None.

7.7.4 Assessment Reporting. None.

7.7.5 Assessment Interaction. None.

8 RELATED WORK

We consider two categories of literature related to our work: assessment of assurance cases and assurance case tools. We describe connections between them and our work in order.

Literature on assessing assurance cases. Significant research has gone into developing and surveying methods for evaluating/assessing assurance cases. Rushby et al. [20] provided an introduction to assurance cases (with a focus on airborne systems), popular notations and tools, as well

as examined methods that may be used to evaluate whether an assurance case argument is sound and provides the necessary confidence. In their work, both human and technical factors were considered for the evaluation process, such as fallacies in human reasoning and the technical semantics of assurance cases. Similarly, Rinehart et al. [34] contributed a comprehensive introduction to assurance cases and provided examples from the energy, aviation, rail, automobile, and medical domains. Among other things, the work included a specific discussion about assurance case evaluation, evaluation standards, guidelines, stakeholders, and approaches (e.g., Baconian probabilities, Bayesian belief networks, argumentation theory). Duan et al. [15] have surveyed the literature to identify how researchers reason about confidence and uncertainty in assurance cases. The surveyed approaches were analyzed and categorized based on the type of uncertainty they deal with (epistemic or aleatory) and the type of approach (qualitative or quantitative). Finally, de la Vara et al. [13] presented a research preview on the assessment of the quality of safety cases. The insights provided allow researchers and practitioners to gain an understanding of why safety case quality requires further investigation, what aspects must be considered, and how quality assessment could be performed in practice. We consider the body of work discussed above as complimentary to ours, since we do not focus on general and theoretical approaches to assessing assurance cases, but rather survey which of those techniques are implemented and/or automated via tool support.

Literature on assurance case tools. For the purpose of creating the Assessable GSN (AGSN) tool, Luo et al. [28] have conducted a preliminary study featuring a number of assurance case tools. Their work only identifies the presence or absence of a set of predefined tool assessment features, namely, node coloring, annotations, and inspection status, and does not further discuss the tools, their features, or their implementations. J. Linnosmaa [25] evaluated five assurance case tools from the perspective of their abilities to develop assurance cases for the nuclear domain. The tools were evaluated based on their support for five categories: planning, structure, data, review, and management. The results of the industrial survey by de la Vara et al. [12] on safety evidence change impact analysis practice demonstrate that the evolution of safety cases should be better managed and the level of automation in safety evidence change impact analysis is low. Our work is closely related to this literature through the evaluation of assurance case tools and their features. To the best of our knowledge, however, we are the first to provide a detailed and comprehensive survey on tool support for assurance case assessment.

9 CONCLUSION

Assurance cases are an emerging way of arguing and relaying the safety, security, and dependability, as well as other properties, of safety-critical systems in an extensive manner. These documents are used by regulatory bodies to certify products, and their importance makes it necessary for them to be effectively assessed and reviewed. The significant size and complexity of these documents, however, make the process of evaluating and assessing their validity a non-trivial task. To this end, software tools supporting developers and assessors in the assurance assessment process have been created.

In this article, we surveyed the current state-of-the-art support for graphical-based assurance case assessment featured in 10 assurance case tools, all of which exhibiting notable assessment capabilities. More specifically, through examining the assessment techniques supported by the tools, we identified three categories of methods for assessing an assurance case's structure (structural constraints, correctness and completeness checks, user queries), and five categories of support for assessing an assurance case's content (argument assessment, evidence assessment, assessment tracking, assessment reporting, assessment interaction). We researched and tested each tool's abilities to support these defined categories of assessment, with the goal of classifying the assessment approaches currently existing in assurance case tools, and presented the results.

Content checks. We observed that while assurance case tools have been developed over the past 20 years, the majority of their supported assessment techniques are still heavily reliant on manual effort. This is especially true when assessing the content and supporting evidence of an assurance case, much of which takes the form of natural language. For the most part, the tools in our survey appear to utilize very different approaches of assessment, offering interesting but incomparable analyses. There does not seem to be one agreed-upon notion of what a useful analysis is, meaning that the area of automated support for assurance case assessment has not yet matured. For example, if we refer to the column *argument assessment* in Table 4, four of the seven tools that offer support for assessing an AC's content predominantly focus on facilitating the ability of an assessor to convey his or her judgment in the form of quality and confidence values. In addition to the benefits previously covered in Section 7, setting quality and confidence scores to individual elements allows assessors to easily structure and communicate their judgment. Sources of low confidence contributing to the overall claim, as well as the rationale for the expert's decisions can easily be traced and investigated if necessary. Alternatively, the remaining three tools (AutoFOCUS 3, Resolute, and D-Case Editor) offer a degree of automated assessment through either integrating the assurance case with system models, allowing changes between the two to be immediately flagged, or by using type-checking principles for validating the consistency of the argument. We believe that each of these techniques covers important and different aspects of the assurance case assessment process, and that a combination of these approaches might be beneficial.

Structural checks. Structural checks, however, seem to be better supported by automated methods. Yet, the structural analyses observed in the surveyed tools are limited to model constraints and do not account for the actual composition or logical validity of the argument. When constructing an assurance case, various inference strategies are utilized to decompose claims into sub-claims, which are then easier to manage and prove. Proving the validity of these decompositions, however, is a non-trivial task. Currently, tools based on GSN and other popular notations are unable to assume any semantics that capture the logical decomposition of an argument, thereby significantly reducing their ability to make such analyses. In principle, however, structural checks can be expanded to identify logical fallacies or validate the soundness of an argument's structural decomposition. One example of this is the work by Rushby et al. [20], who have proposed the use of logic to evaluate deductive reasoning steps. Their methodology suggests using automated deduction (in the form of theorem provers) to check for logical errors, abstracting the assurance case argument to its logical form. To the best of our knowledge, however, this methodology has not yet been implemented in any assurance case tool.

In our opinion, analyses that address the aforementioned issues are essential in supporting the assurance case assessment process. Additionally, we believe that further work should be done to add lightweight semantics to assurance case notations. This could enable techniques to check for an argument's essential properties, such as its decomposition, completeness, and coherence, and would allow for significant progress in this area. In closing, we have observed that while many approaches address specific aspects of evaluating and assessing assurance cases, our experience shows that there is significant room for improvement in all of the discussed categories, and we hope that this survey will help identify such opportunities. The results of this survey may be used by safety engineers to assess relative effectiveness of the different techniques for validating assurance cases, thereby leading to guidelines of which technique to select for a particular problem or a particular domain. Also, while no regulatory bodies currently consider these assessment features in the software certification process, we hope that this work can lead to specific recommendations being put into certification standards or tool qualification processes.

REFERENCES

[1] [n.d.]. Agda wiki page. http://wiki.portal.chalmers.se/agda/pmwiki.php.

[2] 2008. AFI RVSM PRE-IMPLEMENTATION SAFETY CASE. Retrieved from: https://www.icao.int/ESAF/Documents/RVSM/AFI_RVSM_PISC_Core_Document_FEB2008.pdf.

[3] 2010. AFI RVSM Post-Implementation Safety Case. Retrieved from: goo.gl/wD64FT.

[4] 2011. Preliminary Safety Case for an ADS-B Airport Surface Surveillance Application. Retrieved from: goo.gl/2VMJzu.

[5] RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification (RTCA'11). Retrieved from: https://my.rtca.org/NC__Product?id=a1B36000001IcmqEAC.

[6] 2015. Generic Infusion Pump Research Project. Retrieved from: https://rtg.cis.upenn.edu/gip/.

[7] 2018. Goal Structuring Notation Working Group, "GSN Community Standard Version 2." Retrieved from: http://www.goalstructuringnotation.info/.

[8] 2018. Object Management Group: Structured Assurance Case Metamodel (SACM) version 2.1. Retrieved from: https://www.omg.org/spec/SACM/About-SACM/.

[9] University of California. Automated Reasoning Group. [n.d.]. Sensitivity Analysis and Inference engine: SamIam. Retrieved from: http://reasoning.cs.ucla.edu/samiam/.

[10] Robin Bloomfield and Peter Bishop. 2010. Safety and assurance cases: Past, present and possible future—An Adelard perspective. In *Proceedings of the Safety-Critical Systems Symposium (SSS'10)*. 51–67.

[11] Lukasz Cyra and Janusz Górski. 2011. Support for argument structures review and assessment. *Reliab. Eng. Syst. Saf.* 96, 1 (2011), 26–37. Special Issue on Safecomp 2008.

[12] Jose Luis de la Vara, Markus Borg, Krzysztof Wnuk, and Leon Moonen. 2016. An industrial survey of safety evidence change impact analysis practice. *IEEE Trans. Softw. Eng.* 42, 12 (2016), 1095–1117.

[13] Jose Luis de la Vara, Gabriel Jiménez, Roy Mendieta, and Eugenio Parra. 2019. Assessment of the quality of safety cases: A research preview. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 124–131.

[14] A. P. Dempster. 1967. Upper and lower probabilities induced by a multivalued mapping. *Ann. Math. Stat.* 38, 2 (1967), 325–339.

[15] Lian Duan, Sanjai Rayadurgam, Mats P. E. Heimdahl, Anaheed Ayoub, Oleg Sokolsky, and Insup Lee. 2017. Reasoning about confidence and uncertainty in assurance cases: A survey. In *Software Engineering in Health Care*. Springer International Publishing, 64–80.

[16] International Organization for Standardization. [n.d.]. ISO 26262: Road Vehicles—Functional Safety (2011), 1st version. Retrieved from https://www.iso.org/standard/43464.html.

[17] Goal Structuring Notation Working Group, "GSN Community Standard Version 2". 2018. Creative Commons Attribution 4.0 International License. Retrieved from: http://creativecommons.org/licenses/by/4.0/. http://www.goalstructuringnotation.info/.

[18] Janusz Górski, Aleksander Jarzebowicz, Jakub Miler, Michal Witkowicz, Jakub Czyznikiewicz, and Patryk Jar. 2012. Supporting assurance by evidence-based argument services. In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP'12) (LNCS)*, Vol. 7613. 417–426.

[19] Finn V. Jensen. 2001. *Bayesian Networks and Decision Graphs*. Springer-Verlag.

[20] Murali Rangarajan, John Rushby, Xidong Xu, and Thomas L. Weaver. 2015. *Understanding and Evaluating Assurance Cases*. Technical Report. NASA. https://ntrs.nasa.gov/search.jsp?R=20160000772.

[21] A. Josang and T. Grandison. 2003. Conditional inference in subjective logic. In *Proceedings of the 6th International Conference of Information Fusion*, Vol. 1. 471–478.

[22] T. Kelly and J. McDermid. 1998. Safety case patterns—Reusing successful arguments. In *Proceedings of the IEEE Colloquium on Understanding Patterns and Their Application to Systems Engineering (Digest No. 1998/308)*. 3/1–3/9.

[23] Z. Langari and T. Maibaum. 2013. Safety cases: A review of challenges. In *Proceedings of the 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE'13)*. 1–6.

[24] Robert Lewis. 2009. Safety case development as an information modelling problem. In *Proceedings of the Safety-Critical Systems Symposium (SSS'09)*. 183–193.

[25] Joonas Linnosmaa. 2016. *Structured Safety Case Tools for Nuclear Facility Automation*. Master's thesis. Tampere University of Technology, Finland.

[26] Kestrel Technology LLC. 2011. CertWare website. Retrieved from: https://nasa.github.io/CertWare/index.html.

[27] J. W. Lloyd. 1984. *Foundations of Logic Programming*. Springer-Verlag, Berlin.

[28] Yaping Luo, M. van den Brand, Zhuoao Li, and A. K. Saberi. 2017. A systematic approach and tool support for GSN-based safety case assessment. *J. Syst. Arch.* 76 (2017), 1–16.

[29] Mike Maksimov, Nick L. S. Fung, Sahar Kokaly, and Marsha Chechik. 2018. Two decades of assurance case tools: A survey. In *Computer Safety, Reliability, and Security*. Springer International Publishing, 49–59.

[30] Y. Matsuno. 2014. A design and implementation of an assurance case language. In *Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks*. 630–641.
[31] Yutaka Matsuno. 2014. Design and implementation of GSN patterns: A step toward assurance case language. *IPSJ Online Trans.* 7 (01 2014), 59–68.
[32] Y. Matsuno and K. Taguchi. 2011. Parameterised argument structure for GSN patterns. In *Proceedings of the 11th International Conference on Quality Software*. 96–101.
[33] Benjamin C. Pierce. 2002. *Types and Programming Languages* (1st ed.). The MIT Press.
[34] David J. Rinehart, John C. Knight, and Jonathan Rowanhill. 2015. *Current Practices in Constructing and Evaluating Assurance Cases with Applications to Aviation*. Technical Report. NASA. https://ntrs.nasa.gov/search.jsp?R=20150002819.
[35] Leonard J. Savage. 1954. *The Foundations of Statistics*. Wiley Publications in Statistics.
[36] Glenn Shafer. 1976. *A Mathematical Theory of Evidence*. Princeton University Press.
[37] Carnegie Mellon University. Software Engineering Institute. 2011. OSATE. Retrieved from: http://www.aadl.info/aadl/currentsite/tool/osate.html.
[38] Makato Takeyama. 2011. A brief introduction to D-Case/Agda *(draft ver. 0.2). Retrieved from: http://wiki.portal.chalmers.se/agda/uploads/D-Case-Agda.D-Case-Agda/D-Case-Agda-HomePage-Data.html.
[39] Stephen E. Toulmin. 1958. *The Uses of Argument*. Cambridge University Press.
[40] Jian-Bo Yang and Dong-Ling Xu. 2002. On the evidential reasoning algorithm for multiple attribute decision analysis under uncertainty. *Trans. Sys. Man Cyber. Part A* 32, 3 (May 2002), 289–304.
[41] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *J. Inform. Softw. Technol.* 53, 6 (2011), 625–637.

Received March 2019; revised June 2019; accepted June 2019