Product Line Assurance Cases with Variability Points

Simon Diemert[1], Laure Millet[1], Torin Viger[1], Sara Magdalinski[1], Annie Chung[1], Jeffrey Joyce[1]

[1] Critical Systems Labs Inc., Vancouver, Canada

Abstract *A structured assurance case argument can be parameterized through the use of variability points that govern the conditional inclusion or exclusion of argument elements. An important use case is product line assurance, where a single core assurance argument is maintained across an entire product line. In contrast to template-based approaches, which instantiate and maintain a separate assurance case for each product variant, variability points enable product-specific differences to be represented within a unified argument structure. This approach reduces redundancy, improves maintainability, and avoids the inefficiencies associated with managing multiple instantiated assurance cases. In addition to describing the SIMPLE method for variability point modelling in GSN-style arguments, this paper illustrates the practical application of this approach for an automated emergency braking system that can be adapted for different system configurations.*

1 Introduction

Preparing and maintaining an assurance case (AC) is a widely used method for establishing trust in critical systems such as autonomous vehicles, automated trains, power plants, and medical devices. The goal-oriented approach to assurance offered by ACs is especially important for applications that depend on artificial intelligence (AI) or machine learning (ML) to realize safety-related functions. For instance, road vehicle manufacturers developing an automated emergency braking (AEB) advanced driver-assistance system (ADAS) might have different sensor configurations that each require unique ML models to evaluate the risk of a collision with a forward object, such as a pedestrian or another vehicle. For compliance with ISO 26262 and ISO/PAS 8800, an AC should be developed for such a system, arguing that the system, including ML-enabled functions, will achieve its safety goals.

However, automakers, and more generally technology providers in rail, energy, health, aerospace, and defence, rarely produce "one-off" systems. Instead, each product or service they offer is a variant from a "product line". In these situations, variation in products might arise from a range of sources, including offering different functionality (e.g., "premium" v. "normal" variants), different hardware configurations, changes to supply chains, differences among target operating domains, changes to engineering processes, updates to regulatory or compliance

© Critical Systems Labs Inc. 2027.
Published by the Safety-Critical Systems Club. All Rights Reserved

requirements, technology updates (e.g., "over-the-air" updates), and improved system designs in response to learnings from deployment. Considering all these factors, and the possibility that they interact to produce a combinatorial explosion of variants, technology providers can end up managing many product variants. For example, if each vehicle in the fleet can have three optional features enabled or disabled and may be deployed in one of three geographic regions, there are up to $2^3 \cdot 3 = 24$ unique vehicle variants that need to be assured. Developing, maintaining, and assessing a bespoke AC for each variant becomes an untenable proposition.

1.1 Existing Methods for Assurance Case Variability Management

Building on results from product-line engineering in the field of software engineering, product-line ACs allow technology developers to manage ACs across product variants (van Gurp et al., 2001). Several modelling frameworks and methods have been proposed towards this objective: Nešić et al. leverage AC patterns using contract-based design to demonstrate that all system variants in a product line satisfy a set of properties (Nešić et al., 2021), Habli used a model-based approach to represent and reason over product lines (Habli, 2009), and Murphy et al. introduce the PLACIDUS methodology, which integrates formal methods into AC development to create provably correct ACs across a product line (Murphy et al., 2026). While these models have advanced knowledge of product-line management, they have not been implemented as part of widely adopted tools for AC management and are often dependent on complex models of variability, limiting their accessibility by AC practitioners with limited time or resources and non-expert interest holders.

1.2 Summary of Contribution

To address this gap, this paper makes three contributions: 1) a simple model of variability based on Boolean expressions that is propagated through the argument such that child nodes inherit variability information from their parents; 2) the SIMPLE method for applying our variability model to product lines that integrates with core elements of AC methodology, such as dialectic reasoning, performance indicators, and assurance claim points; and 3) an implementation of our model and method in *Socrates*[1], a commercial AC management tool (Millet et al., 2023). To demonstrate our variability model, method, and tool, we apply them to a representative automotive AEB system, showing that they can manage variability arising from commonly occurring sources of variability for automotive systems.

The examples in this paper are presented using the Eliminative Argumentation (EA) notation for expressing arguments. However, our results are directly applicable to other notations, such as the Goal Structuring Notation (GSN) (Assurance Case Working Group, 2021) or Claims-Argument-Evidence (CAE) notation (Bishop & Bloomfield, 1998). In fact, our implementation of variability in *Socrates* supports all three of these notations.

[1] https://criticalsystemslabs.com/socrates-assurance/

2 Foundations

This section establishes foundations related to structured assurance argumentation, including introducing EA notation, and then provides background on variability modelling for software product lines, upon which our contributions build.

2.1 Structured Argumentation for Assurance

An AC is a structured argument, supported by a body of evidence, that a system is acceptably safe (or secure, reliable, etc.) for a given application in a defined operating context (Assurance Case Working Group, 2021). Structured arguments make the reasoning that connects evidence to top-level claims explicit, so that it can be easily understood and maintained. Graphical notations such as GSN, CAE, and Eliminative Argumentation (EA) (Goodenough et al., 2015) provide a vocabulary of typed nodes and relations for expressing assurance arguments. This paper uses EA because we find the semantics of the node types to be the easiest to understand, but our results are readily generalizable to GSN and CAE.

EA is based on the principle that confidence in a claim increases as reasons for doubting it are identified and eliminated (Goodenough et al., 2015). Alongside the claims, strategies, context, and evidence familiar from GSN, EA introduces defeaters, which represent doubt in the argument. A rebutting defeater challenges the truth of a claim; an undermining defeater challenges the validity of evidence; and an undercutting defeater challenges an inference, identifying circumstances in which the premises hold but the conclusion does not. Defeaters may be resolved ("eliminated") by further argumentation and evidence. Alternatively, if a defeater cannot be resolved, then it must be accepted as contributing to the "residual doubt" in the AC. If the totality of residual doubts is unacceptable to interest holders, then a system cannot be granted authority to operate. EA has been used to prepare ACs in several sectors including automotive, rail, and high-energy physics (Diemert & Joyce, 2020; Hobbs et al., 2024; Viger et al., 2025).

Fig. 1 shows a small argument prepared with EA about an electric kettle. The top-level claim (C1), "The kettle is acceptably safe", is interpreted in the context (X2) of boiling water for tea in the kitchen. The strategy (S3) argues over hazards, decomposing C1 into claims that electrical shock (C4) and burn (C5) hazards are mitigated. The inference rule (IR6) describes how the claims combine to support their parent. An undercutting defeater (D62) challenges the inference: there might be other hazards; it is not resolved and is marked as a residual. A rebutting defeater (D42) challenges the claim that the electronics are adequately insulated and is resolved by the claim that they are rated IP67 (C421), supported by test results (E4211). Similarly, an undermining defeater (D521) questions the thermal model used for the heat-transfer analysis (E52) and is resolved by experiments validating that model (C5211).

Evidence and Artifacts. Evidence nodes in an argument are abstractions of the underlying evidence, which might include whole design reports, analyses, field studies, or verification results. We use *artifacts* to connect these nodes to the

concrete items that substantiate them, such as test reports, analysis records, data sets, or design documents. In Fig. 1, the artifacts linked to evidence nodes are denoted by a blue badge containing a document symbol connecting the evidence to the underlying technical drawings, test reports, and a thermal analysis report.

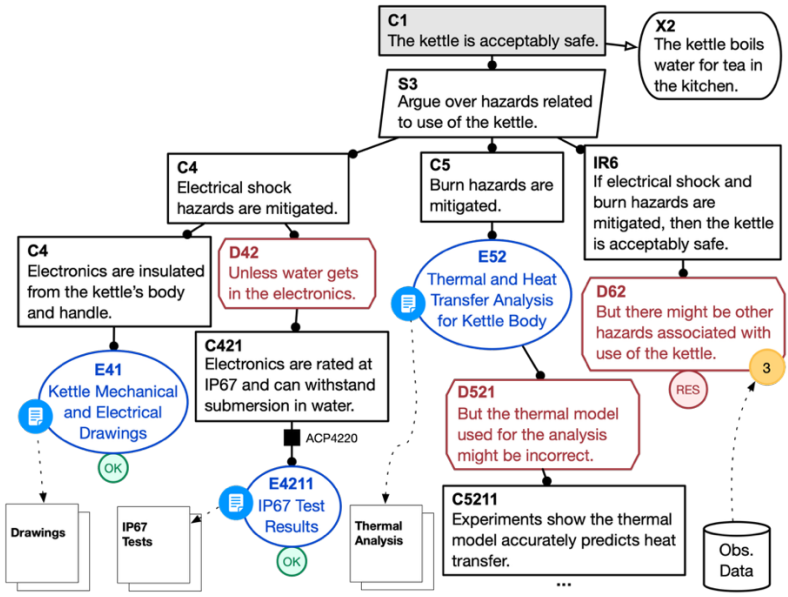


Fig. 1. Example of the EA notation.

Indicators. It is increasingly common to instrument AC arguments with *indicators* that are derived from live data collected from development activities or system operations (Koopman, 2022). These are sometimes referred to as Safety Performance Indicators (SPIs) or Key Performance Indicators (KPIs). Thresholds may be established that, once violated, indicate a claim has been invalidated or that a defeater should be reconsidered. For instance, the AC for an autonomous vehicle might include an indicator for the rate of "near-miss" collisions in field operations; if the rate goes above N events per day, then claims in the argument about collision avoidance and obstacle separation rules should be reviewed. Instrumenting an AC's argument with indicators is one way to make an otherwise static AC into a dynamic AC (Denney et al., 2015; Diemert, 2026). The argument in Fig. 1 contains an indicator with a violated threshold; "traffic light" colouring (red, amber, green) is used to indicate the status of the indicator. The indicator is derived from observations of kettle usage and monitors residual doubt (D62); so far, three hazardous events have been observed that are neither electrical shock nor a burn, so the indicator's threshold has been triggered and it is shaded amber in the argument.

Assurance Claim Points. An assurance claim point (ACP) annotates a specific relation in the primary argument, such as the link between a claim and its supporting evidence, and anchors a separate argument explaining why that relationship can be trusted (Hawkins et al., 2011). ACPs can be used to attach a "means" argument to the primary argument, that is, an argument about the engineering process by which

the evidence was produced and the argument was constructed (Motor Industry Software Reliability Association (MISRA), 2019). Using ACPs this way avoids cluttering the primary argument with process-related information, while still formally including them within the AC for review and assessment. In Fig. 1, ACP4220 appears on the edge between C421 and E4211 which is the root of a means argument justifying the adequacy of the procedure used to perform IP67 testing for the kettle.

2.2 Assurance Case Development Lifecycle

An AC should not be a static document produced once at the end of development. It should evolve alongside the system over time (Denney et al., 2015; Hobbs et al., 2024). Fig. 2 shows the AC development lifecycle. Our approach to variability management is embedded within this lifecycle. It consists of six steps, grouped into three phases: development, evaluation, and deployment. The AC is at the centre: each step contributes to, scrutinizes, or acts on it.

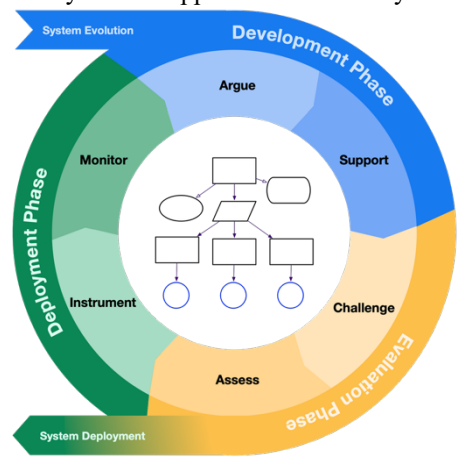


Fig. 2. Assurance Case Lifecycle.

During **development**, engineers argue, constructing the core structure of the argument's claims. Then they support the argument, connecting the argument to evidence, artifacts, and placeholders for indicators.

During **evaluation**, the argument is challenged by reviewers, assessors, and interest holders to identify new defeaters that expose weaknesses in the reasoning. The argument is then assessed to determine if there is sufficient confidence in the top-level claim for deployment.

Once the system is **deployed**, the system is monitored using performance indicators connected to claims and defeaters. System developers respond when indicators breach their thresholds, for instance, by introducing operational mitigations or by revising the system. Changes to the system trigger a return to development, so that the argument evolves alongside the underlying system.

2.3 Product-Line Management

Software product-line engineering aims to develop a family of related products from a shared set of core assets, rather than developing each product independently. Central to this approach is the explicit management of commonality and variability: a variation point identifies a location in an asset (e.g., a requirement, design element, or source file) where products may differ, and each variation point is resolved by selecting one or more variants (van Gurp et al., 2001). Variation points are often characterized by their binding time, i.e., when in the lifecycle (design, build, deployment, or runtime) a variant is selected, and dependencies between variants are

commonly captured using feature models. Habli extended these ideas to safety assurance, examining how safety arguments can be developed for product lines so that common reasoning and evidence are reused across products while product-specific differences remain explicit (Habli, 2009). Our variability model draws on this vocabulary, treating the AC itself as a core asset whose variation points are expressed as Boolean conditions over argument elements, including elements connected to the argument structure, such as artifacts, indicators and claim points.

3 The SIMPLE Method for Product-Line Assurance Cases

An AC should be developed early and revised as a system's design progresses (Hobbs et al., 2024). The whole assurance argument might not be known at the outset of a project; for example, a design might change, or unexpected evidence or doubts might arise. In keeping with this perspective, our method takes the position that variability modelling should also be an activity that spans the AC and system-development lifecycle. The alternative, front-loading variability modelling, might be challenging if the dimensions of variability for a system are not entirely understood at the outset. Grafting a variability model onto an established AC, while possible, might result in a significant amount of hidden effort redesigning or restructuring the argument at the end of a project. The SIMPLE method connects common sources of variability that impact ACs with an existing, iterative, and dynamic method for AC development. The method steps are as follows.

1. **Survey sources:** Identify sources of variability that arise from outside the AC. Common sources of variability include product features, system architecture (components), development methodologies, regulatory or compliance activities, human/organizational aspects (e.g., permissions to view the argument), system versions, and operating environments. Our method does not prescribe a complete list of variability sources, as these will differ between systems.
2. **Identify impact:** For each source of variability, identify the potential impact on the AC. Phases of development and potential impacts of variability sources are described below.
3. **Model variability:** In the relevant phase of AC development, model variability by i) representing sources of variability as *variables*, and ii) annotating the argument structure and related elements (artifacts, indicators, etc.) with *variability points*. Adopting the terminology introduced by Murphy et al., variability modelling produces a *lifted argument* that raises the level of abstraction of a single AC to a product line (Murphy, 2026).
4. **Project variants:** Assign values to the variables to produce projections corresponding to each product instance or variant derived from the product line. Projections are themselves whole ACs. A procedure for projecting a variant-specific AC is given below.

5. **eva<u>L</u>uate:** There are two modes of evaluation applicable to product-line ACs. The developer of an AC can evaluate the lifted argument in its abstract form; e.g., checking that conditions are syntactically well-formed and are satisfiable. Alternatively, the projection for a specific product variant can be assessed; e.g., confirming that all residual doubts are acceptable.
6. **<u>E</u>volve:** As an AC evolves alongside its system, it is necessary to also maintain the variability model. This might require, for instance, adding new variables and variability points to take account of new architectural changes, or creating new projections that correspond to new product variants.

An earlier paper describes an example that illustrates a basic use of variability across a product line (Magdalinski et al., 2026). In this example, there are exactly four variables corresponding to the four different variants of a product line for a perception system. While even the very basic use of variability illustrated in this earlier paper achieves a great deal, the Survey step of SIMPLE should search for sources of variability at a finer level of granularity. Rather than one variable for each variant of the product line, the Survey step should consider "pushing down" the variability model to take account of various design-level choices, such as the choice of sensors in the case of a perception system. In addition to functional differences between variants of a product line, there are often other significant sources of variability. For example, variability might arise in the "means" layer of an AC where a particular product such as a real-time operating system is developed for use in different sectors with different certification criteria, e.g., ISO 26262 for the use of this product in automotive applications, RTCA DO-178C in aerospace, CENELEC EN 50126 for rail systems. Another often significant source of variability concerns differences in access to details of an AC. For example, variability might be used to control what details are made visible by a supplier to different customers.

Table 1 shows the relationship between the SIMPLE and AC lifecycle steps. A solid dot ● means the steps should be performed together, while a clear dot ○ means the AC method step informs the SIMPLE step.

Table 1. Mapping of SIMPLE steps to AC lifecycle.

SIMPLE Step	Argue	Support	Challenge	Assess	Monitor	Respond
<u>S</u>urvey sources	○	○	○	○	○	○
<u>I</u>dentify impacts	●	●	●			
<u>M</u>odel variability	●	●	●			
<u>P</u>roject variants	○	○	○	○	●	○
eva**<u>L</u>**uate variants				●	●	○
<u>E</u>volve	●	●	●	○	●	○

4 Formalizing Variability in Product Lines

This section formalizes our model of AC variability. After establishing preliminary definitions, we describe how variability is connected to the AC through *variability points*. Then we describe how specific instances of a product line can be extracted through *projections*. As a preliminary step, we formalize an AC as a tuple:

$$A = (N, E, P, L_{PE}, R, L_{RN}, I, L_{IN})$$

N is a set of typed graph nodes containing claims, strategies, defeaters, evidence, and so on, in keeping with the user's chosen notation. $E \subseteq (N \times N)$ is the set of typed and directed graph edges connecting nodes in N to form a directed acyclic graph, with edges defining a relation between nodes.

P is the set of assurance claim point nodes, which are themselves the roots of sub-trees consisting of nodes in N. The labelling relation $L_{PE} \subseteq (P \times E)$ connects claim point elements in P to edges of the argument. Every claim point must map to at least one edge, but multiple claim points may map to the same edge.

R is a set of external artifacts referenced by the argument, usually containing concrete evidence or additional data to support the argument. The labelling relation $L_{RN} \subseteq (R \times N)$ connects artifacts to nodes in the argument, e.g., connecting artifacts to evidence nodes. An artifact may connect to multiple nodes in the argument.

I is a set of safety performance indicators whose data is collected from development activities or system operations at runtime. Indicators also have threshold rules that trigger alerts (e.g., "the failure rate of the sensor exceeds our tolerable threshold"). The labelling relation $L_{IN} \subseteq (I \times N)$ connects indicators to nodes in the argument enabling continuous validation of claims or defeaters in the AC. An indicator may connect to multiple nodes.

4.1 Modelling Variability

Like other models of variability, our simple model of variability has two essential building blocks: *variables* and *variability points*.

First, we define a set of Boolean *variables*, $X = \{x_1, x_2, \ldots, x_k\}$ that will be used to model variability. The semantics of these variables are chosen by the user. For instance, variable $x_7 = True$ might correspond to a situation where a specific feature is enabled on a vehicle or where a component is used as part of the design.

Next, we define a variability point, v_i, as a simple logical expression that is the conjunction (logical AND) of one or more variables in X. More formally, a variability point is a Boolean function $v_i: \mathcal{P}(X) \rightarrow \mathbb{B}$ that returns true if an assignment of variables satisfies the point's expression, and false otherwise. The set of all variability points defined for the argument is $V = \{v_1, v_2, \ldots, v_k\}$.

Variability points may be assigned to nodes of the argument through the labelling function $L_{NV}: N \rightarrow V \cup \{\perp\}$ with $\perp$ representing a "null" assignment where a node does not have a variability point assigned. Similarly, variability points may be connected to claim points, artifacts, and indicators, with the functions $L_{PV}: P \rightarrow V \cup \{\perp\}$, $L_{RV}: R \rightarrow V \cup \{\perp\}$ and $L_{IV}: I \rightarrow V \cup \{\perp\}$, respectively.

Using these definitions, we amend (shown as bold font) our formalization of an AC and denote a lifted, product-line, argument as A^0:

$$A^0 = (N, E, P, L_{PE}, R, L_{RN}, I, L_{IN}, \boldsymbol{X}, \boldsymbol{V}, \boldsymbol{L_{NV}}, \boldsymbol{L_{IV}}, \boldsymbol{L_{PV}}, \boldsymbol{L_{RV}})$$

Fig. 3 to the right shows a product-line AC for a toy argument that has six nodes, $N^0 = \{c_1, c_2, c_3, d_4, e_{21}, e_{31}\}$ with E as shown in the figure. Three artifacts are shown, $R^0 = \{r_1, r_2, r_3\}$ linked by $L^0_{RN} = \{r_1 \rightarrow e_{21}, r_2 \rightarrow e_{31}, r_3 \rightarrow e_{31}\}$. Two indicators are connected to d_4 so that $I^0 = \{i_1, i_2\}$ and $L^0_{IN} = \{i_1 \rightarrow d_4, i_2 \rightarrow d_4\}$ Finally, there is a single claim point, $P^0 = \{p_{32}\}$ with $L^0_{PE} = \{p_{32} \rightarrow (c_3, e_{31})\}$. Suppose this product-line AC has three variables corresponding to sources of variability determined during step one of the SIMPLE method, so $X = \{x_1, x_2, x_3\}$. These variables are used to construct six variability points, $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ with $L_{NV} = \{c_2 \rightarrow v_1, c_3 \rightarrow v_2\}$, $L_{RV} = \{r_2 \rightarrow v_3, r_3 \rightarrow v_4\}$, and $L_{IV} = \{i_1 \rightarrow v_5, i_2 \rightarrow v_6\}$. The claim point, p_{32}, does not have a variability point associated with it, so $L_{PV} = \{p_{32} \rightarrow \perp\}$; we omit the null mappings for brevity. The variability points have the following active conditions over the variables in X: $v_1 = x_1$, $v_2 = x_2$, $v_3 = x_2 \wedge x_3$, $v_4 = x_2 \wedge \neg x_3$, $v_5 = x_3$, and $v_6 = \neg x_3$. For instance, v_4 is active when x_2 is assigned True and x_3 is assigned False. If a variable is not used in a variability point's function, it has no effect when evaluating the active conditions of variability points.

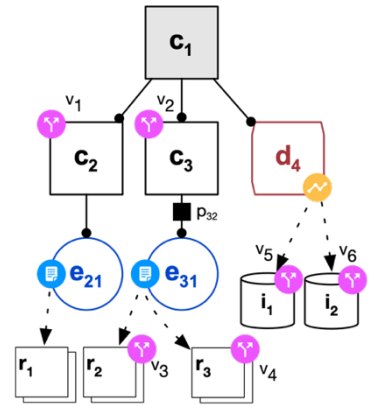


Fig. 3. A product line AC.

4.2 Projecting Variants of Assurance Cases

Given an AC with variables and variability points as described above, A^0, we create *projections* of the argument by assigning values to the variables in X to produce product instance ACs, one per projection, $A^1, A^2, \ldots, A^m$. Formally, a projection is a function that maps from variables to their assigned values, $\Theta^i : X \rightarrow (\mathbb{B} \cup \{\perp\})$, where it is possible that a mapping produces a null value ($\perp$) if the user has not specified the value of that variable for the projection.

To project the argument structure, we perform a depth-first traversal. For each node encountered in the traversal, we retrieve the variability point, if it exists, and then evaluate the point's active condition under the variable assignment. Formally, a node $n \in N$ with $v_n = L_{NV}(n) \neq \perp$ is excluded from the traversal if $v_n(\{x \in X : \Theta^i(x) \neq False\}) = False$. Children are excluded if they are never traversed because none of their parents are included in the projection. This formulation includes nodes by default, unless they are explicitly excluded by a variability point or the assignment. First, if a node does not have a variability point mapped through L_{NV}, then it is included in the projection; this means that a whole sub-tree in the argument can be included by assigning a single variability point to the root of that sub-tree,

which simplifies the variability modelling task. Second, if a variable is not mapped by the projection's variable assignment, Θ^i, then that variable has no effect on the nodes included in the projection. Because our variability model does not include variability points on edges in the argument, the set of edges in the projection, E^i, is computed by creating the sub-graph from the set of included nodes.

Artifacts, indicators, and claim points are all projected using the same procedure. For instance, the artifacts included in a projection are determined by evaluating variability points associated with each artifact in the context of the projection's variable assignment. Formally, this is computed as:

$$R^i = \left\{ r \in R, v_r = L_{RV}(r) \ : \ v_r \neq \perp \Longrightarrow \left(v_r(\{x \in X : \Theta^i(x) \neq False\}) \right) \right\}$$

Links from nodes to artifacts are only included in a projection if the artifact and node are both included in a projection, otherwise the pair is omitted from the projection's artifact labelling relation, L_{RN}.

Continuing the example from Fig. 3 above, consider two projections A^1 and A^2 that are induced by the variable assignments $\Theta^1 = \{x_1 \rightarrow True, x_2 \rightarrow False, x_3 \rightarrow False\}$ and $\Theta^2 = \{x_1 \rightarrow False, x_2 \rightarrow True, x_3 \rightarrow True\}$, respectively. The resulting projections are shown below in Fig. 4.

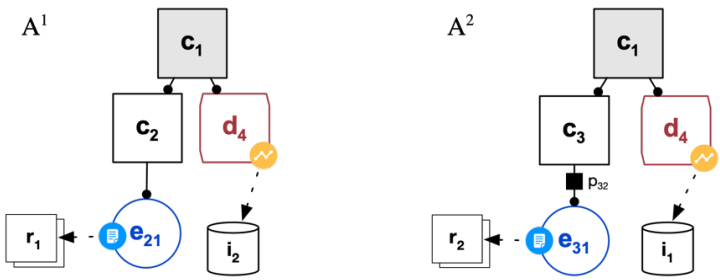


Fig. 4. Two projections derived from the product-line AC example.

In projection A^1, the sub-tree rooted in c_2 is included because v_1 evaluates to $True$ under the assignment Θ^1. Since a graph traversal is used to compute projections, and nothing under c_2 is labelled by a variability point, the whole sub-tree is included. On the other hand, c_3 is excluded from A^1 because the variability point v_2 evaluates to $False$ resulting in the entire sub-tree below being excluded during traversal. A^1 also includes an indicator, i_1, annotating the d_4, and connecting it to a specific data source that is relevant to this variant. For projection A^2, the sub-tree rooted in c_3 is included along with artifact r_2, since both variability points v_2 and v_3 evaluate to $True$ under Θ^2. We can also see that the indicator has changed from A^1 to A^2, connecting a different data source, i_2, to d_4 for this projection.

5 Variability Modelling in Socrates

We have successfully implemented our SIMPLE method and formalism for variability management in *Socrates*, a specialized commercial tool for developing and managing (dynamic) ACs. Our goal in implementing variability management in *Socrates* was to make variability management accessible to AC practitioners working in commercial and industrial settings. As a result, we were careful to design the implementation with usability in mind, and to ensure it is compatible with other workflows and feature areas in the tool. For instance, prior to undertaking this project, *Socrates* had features for artifact, indicator, and claim point management, all of which are compatible with our variability management concept.

5.1 Visualizing Variability in Argument Structures

A screenshot with one view from the implementation appears in Fig. 5 showing the lifted/product-line view of an argument. In this view, the user views the variability points overlaid on the nodes in the argument. For instance, claim C0319 is connected with a variability point, as depicted by the pink icon with the split/fork symbol. Selecting the claim reveals more information in the right-hand side panel.

Beyond linking nodes, there are two additional items of interest in this view. First, in the right-hand side panel, the details of the variability point are visible; see red annotations marked by “A”. The active condition for this variability point depends on a single variable called `Sensor_FwdLongRangeRadar`, which must be true for this node and its children to appear in a projection. Second, variability points *inherited* from nodes higher in the argument tree are also displayed in the right-hand side panel; see the purple annotations marked with “B”. For instance, because the claim C0256 is an ancestor (not shown in Fig. 5), C0319 will only appear in a projection if its assigned active condition is satisfied and the active condition for the variability point on C0256 is satisfied. Nodes with only inherited variability points appear with dashed pink badges, as shown by the “B” labels. Displaying inherited variability in this way makes the effect of projection visible to the user while they are editing the lifted argument.

5.2 Additional Views for Variability

Variability management also coexists with other features of the tool. The evidence nodes shown in Fig. 5 have blue artifact badges, and artifacts may have their own variability points assigned in another view. Similarly for indicators and claim points, which per our formulation, may also have variability points.

Our implementation also has views for managing variables and creating projections by assigning values to variables. When the user needs to view the argument for an instance of the product line, they open the corresponding projection and the procedure described above is automatically applied by the tool. The projection is rendered as a normal argument in *Socrates*.

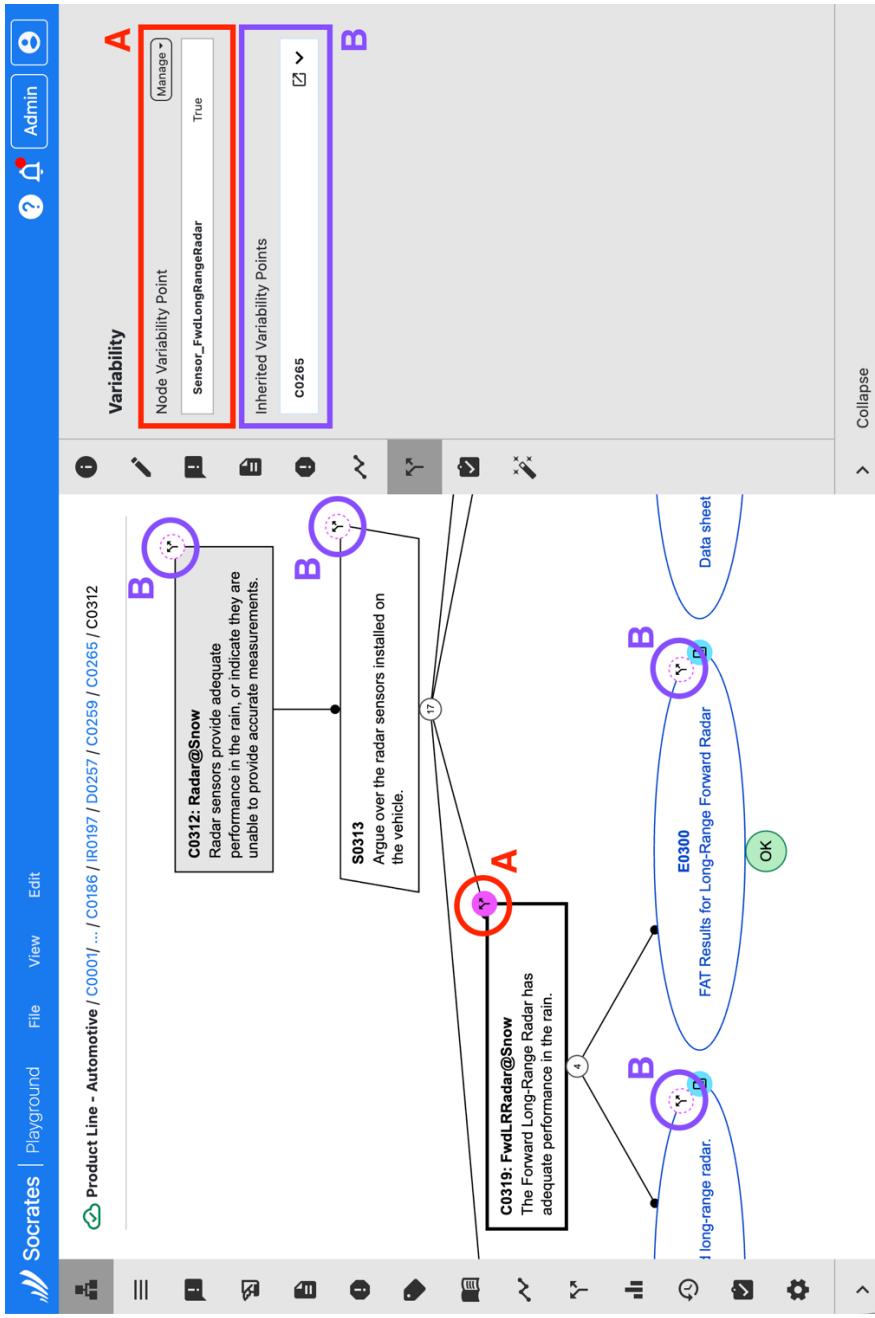


Fig. 5. Implementation of variability management in Socrates.

6 Case Study: Automated Emergency Braking System

As a means of further validation, we applied the SIMPLE variability management method to a representative Automated Emergency Braking (AEB) feature that uses sensors on the front of the vehicle to monitor for objects (pedestrians, cyclists, other vehicles, static objects, etc.) in the ego vehicle's trajectory and requests emergency braking if a collision is predicted. AEB systems are increasingly common in modern vehicles due to their potential to reduce the severity of forward collisions or to avoid them entirely. While this case study is fictitious, the system concept and architecture are broadly similar to ADAS and autonomous driving systems that we have contributed to. We used *Socrates* to develop the AC for this system and apply the SIMPLE method.

For this case study, we have selected six use cases for variability modelling that are supported by our method and tooling: 1) variability for an argument's claims; 2) variability in the dialectic perspective (i.e., defeaters); 3) variability in engineering process (with an ACP); 4) variability in evidence and artifacts; 5) variability in safety performance indicators; and 6) conditional display. After introducing the AEB system, we describe our application of the SIMPLE method in terms of these six use cases. At the time of writing, our case study is under development, with sub-arguments for several sub-systems (Sensing and Perception, Braking, and Communication) nearing completion. Nevertheless, it is complex enough to explore different use cases for variability modelling in AC management in this paper.

6.1 Overview of the AEB System

We imagine a situation where an OEM is providing the AEB feature in several vehicles with different actuators, sensors, and communication busses provided by multiple suppliers. The AEB feature will be deployed in different markets with variable operating environments; e.g., vehicles intended for the North American, Northern European, and U.K. markets accommodate a wider range of precipitation types (snow, rain, etc.), requiring different ML models for object detection and additional cold weather testing. Further, we imagine that some safety assessment activities are carried out by independent assessment firms using different assessment methodologies. Finally, the AEB feature is part of the OEM's "fleet safety monitoring" program that gathers data on system performance which is provided back to the OEM and can be used to derive safety performance indicators for the feature. The system architecture for the AEB system is shown in Fig. 6.

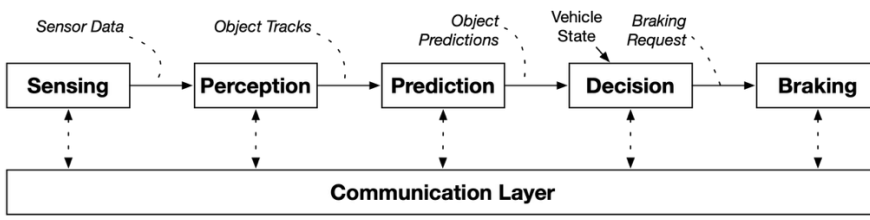


Fig. 6. High-level architecture of the AEB system.

The AEB system is organized as a pipeline of five functions. *Sensing* acquires raw sensor data from camera and radar units. Multiple sensors are available: forward camera, rear camera, forward long-range radar, forward short-range radar, and rear radar. The *Perception* sub-system cleans and fuses the raw sensor data, identifies objects, and tracks them over time to produce object tracks that include the object's type/class and position, speed, and heading relative to the ego vehicle. The *Prediction* sub-system uses the object tracks as a basis for predicting future object behaviour, producing object predictions. The Decision sub-system combines the object predictions with information about the vehicle's current state (speed, acceleration, heading, etc.) and requests emergency braking when necessary to avoid, or reduce the severity of, a collision. Requests are sent to the *Braking* sub-system, which includes a brake control unit that controls brake actuation. All five sub-systems exchange data over a shared communication layer that consists of a pair of redundant (1oo2) communication networks.

We imagine the AEB system has two variants that are compatible with vehicles sold by the OEM, they are called: the *Tamarack* and the *Arbutus* variants. The *Tamarack* variant is suitable for use in urban settings during daytime hours in fair weather, and the *Arbutus* variant is intended for highway operations at either day or nighttime and can be used in the rain or light snow. Technical differences between these system configurations appear in Table 2.

Table 2. Differences between AEB system variants.

	AEB System Variant	
Sub-System	**Tamarack**	**Arbutus**
Sensing	Forward and Rear Camera	Forward (LR, SR) and Rear Radar
Perception	ML models tuned for camera-based object perception in urban settings during daytime hours.	ML models tuned for radar-based object perception on highways in diverse weather.
Prediction	Tuned for urban setting.	Tuned for highway setting.
Decision	Tuned for low to moderate speeds.	Tuned for high speeds.
Braking	Uses the BBZ888 brake controller.	Uses the ABX123 brake controller.
Communication	Redundant CAN XL and Ethernet networks.	Redundant CAN FD and Ethernet networks.

6.2 Assurance Argument for the AEB System

Fig. 7 shows the top-level structure of the assurance argument for the AEB system. The top-level claim (`C0001`) is decomposed by strategy `S0002` over the safety goals for AEB which are intended to mitigate two system-level hazards: H1) Inadequate Emergency Braking, and H2) Spurious Emergency Braking. Of these hazards, spurious braking is higher criticality because it is continuously exposed, there is little control the vehicle operator has over the situation, and can lead to severe collisions from a rear vehicle, especially at high speeds. The two safety goals are represented by the sub-claims `C0003` and `C0004`, respectively. We focus on the more critical goal and its argument under `C0004`, which is decomposed by `S0007` over the

architecture of the AEB system processing pipeline. Each branch addresses one or more sub-systems. For instance, the branch rooted in claim `C1000` is focused on the Sensing and Perception functions in the pipeline.

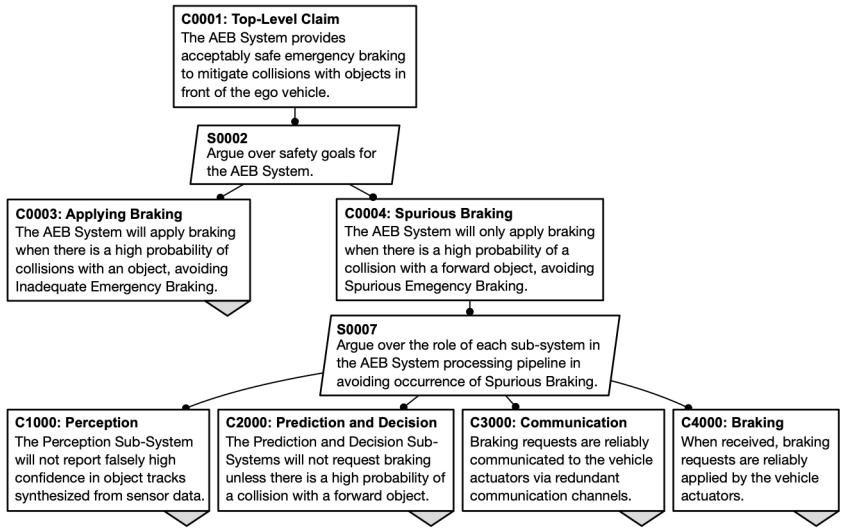


Fig. 7. Top-level assurance argument for the AEB system.

In keeping with the notation from the GSN standard, each of these claims is annotated with an inverted triangle, indicating that there is further argumentation provided in other figures. For brevity, we do not provide the full argument; instead, we offer some summary statistics for the *lifted* (product-line) argument in Table 3. At the time of writing, the argument contains 259 nodes. The argument also references 46 artifacts and 3 indicators, which are bound to variability points as described in the next section. Finally, it contains two assurance claim points.

Table 3. Node type distribution in the AEB system product-line argument.

Node Type	Count (%)	Node Type	Count (%)
Claim	85 (33%)	**Inference**	9 (4%)
Strategy	19 (7%)	**Defeater**	23 (9%)
Evidence	45 (17%)	**Residual Doubts**	16 (6%)
Context / Assumption	0 (0%)	**Complete / Undeveloped**	62 (23%)
		TOTAL	**259**

6.3 Applying the SIMPLE Method to the AEB System

The SIMPLE variability management method was applied to the AEB system; each step's application is discussed below. Though we describe our findings linearly, development of the lifted AC was performed iteratively, first developing the argument, then considering variability with SIMPLE, and so on. Our experience

reaffirms our initial impression that AC development and variability management must be performed in parallel, as inter-dependent activities.

Step 1: Survey Sources. To begin, we studied the system description, architecture, operating environment, and development context. We identified several sources of variability that might impact the AC. Table 4 summarizes these sources along with their impact on the argument and our modelling approach, which is discussed further in Steps 2 and 3. For instance, we identified a source of variability arising from different choices of sensors and braking sub-systems. Of these, the sensor configuration has the most significant impact on the AC, since the choice of sensors will change the detailed design of the Perception sub-system: different ML models are required for camera- or radar-based object detection and object tracking over time requires different algorithms since these sensors provide different information.

Step 2: Identify Impact. For each source of variability, we analysed its potential impact on a *lifted* argument and categorized impact scope as one of the argument structure (claims, strategies, defeaters, evidence), artifacts (evidence linked to the argument), indicators, and claim points. Table 4 summarizes our findings. Unsurprisingly, the most impacted area was the argument structure, followed by supporting artifacts. We developed our product-line AC in parallel with variability modelling; however, if our goal had been to "lift" an existing argument to a generalized product-line argument, then we might have opted for a more fine-grained impact assessment. For example, we might have identified the impact on specific nodes.

Step 3: Model Variability. Table 4 contains a variability modelling approach for each impact scope. Broadly, the approach is two-fold: create variables corresponding to variability exhibited at the source, then assign variability points (VPs) to argument elements over these variables. For example, to address variable sensor configurations, we defined five variables (one per sensor) and then created sub-arguments assuring each sensor, conditioning their inclusion with variability points bound to each variable. In some places, where a single node was relevant to one sensor, we annotated with a variability point as well, as in Fig. 8. In total, our product-line argument contains 16 variables and 79 user-defined variability points (43 on nodes, 31 on artifacts, 3 on indicators, and 2 on claim points).

Step 4: Project Variants. Using the variables mentioned above, we defined two projections corresponding to the *Tamarack* and *Arbutus* vehicle models being sold by our fictitious OEM. Because our variant projection procedure only uses variables with user-provided assignments, it is necessary to explicitly define the value of each variable for each variant. For instance, since the Tamarack model uses a camera-only vision system, we set all camera-related variables to $True$ and all radar-related variables to *False.* After applying the projection procedure, the *Tamarack* variant AC had 130 nodes, and the *Arbutus* variant had 142 nodes.

Step 5: evaLuate Variants. Each variant of the AEB system is evaluated as a standalone AC to determine if that variant is acceptable for deployment. Using EA, the preferred form of evaluation is to inspect the residual doubts that have not been resolved. Since some of the defeaters are connected to variability points, the

residuals differ between the *Tamarack* and *Arbutus* variants of the AEB system. For example, the *Arbutus* variant, which uses radar-based object detection, contains a residual doubt "Unless the long- and short-range radar signals interfere with each other" (`D1026`), which is not rebutted by a counterargument. This residual doubt, among others, suggests that *the Arbutus* variant is not adequately assured.

Step 6: Evolve. Our case study has not progressed to the point where the argument must be evolved due to change in the AEB system. However, during development we made changes to the choice of variables which we consider to be a form of model evolution. For example, we initially had more variables to describe sensor configurations but then opted to collapse them to only five. The tool support provided by *Socrates* helped to find and remove occurrences of deprecated variables.

Table 4. Sources of variability and impacts on product-line AC.

Source	Source Description	Impact Scope	Variability Modelling Approach
Sensors	Vehicles have different sensor configurations (radar v. camera)	Argument Structure	Assign VPs to sub-arguments assuring each sensor; bind variables by sensor.
		Argument Structure	Assign VPs to sub-arguments assuring perception processing for each sensor.
		Artifacts	Assign VPs to artifacts with evidence about each sensor.
Operating Environment	System designed for different operating environments: urban and low-speed daytime v. highway and high-speed in variable weather.	Argument Structure	Assign VPs to sub-arguments assuring system performance in each operating condition; bind variables by condition.
		Artifacts	Assign VPs to artifacts with verification results for each environmental condition.
		Indicators	Assign VPs to indicators for different operating conditions.
Brake Actuators	Different braking actuators appear on different vehicles provided by different suppliers, e.g., the BBZ888 v. ABX123.	Argument Structure	Assign VPs for each braking system and bind variables to each.
		Artifacts	Assign VPs to artifacts with certification results.
Communications	Different vehicles have different redundant communication configurations, e.g., Ethernet and either CAN XL or CAN FD.	Argument Structure	Assign VPs to sub-arguments assuring each network type and bind variables for each choice.
		Artifacts	Assign VPs to each artifact with evidence for each network type.
Development Process	Different contractors apply different functional safety assessment methodologies.	Claim Points	Assign VPs to claim points arguing about assessment adequacy; bind variables for each firm.
		Artifacts	Assign VPs to artifacts for each firm.

6.4 Use Cases for Variability Management

This case study highlighted several use cases for variability management as part of the larger AC development lifecycle. The first, variability in the argument structure, is described at length above. The remaining five use cases are as follows.

Variability in Defeaters and Residual Doubt. We used variability points to conditionally apply defeaters based on the projection's variables. For example, we identified doubts related to the field of view of the sensors, such as a missing camera or forward radar. Fig. 8 shows part of the product-line argument with residual doubts for the field of view of the camera and radar which are conditionally included in the argument depending on the configured sensors. During the evaLuate step of the SIMPLE method, such a residual doubt would be reviewed, catching an error in either variable assignment for the projection, or more concerningly, in the underlying system configuration.

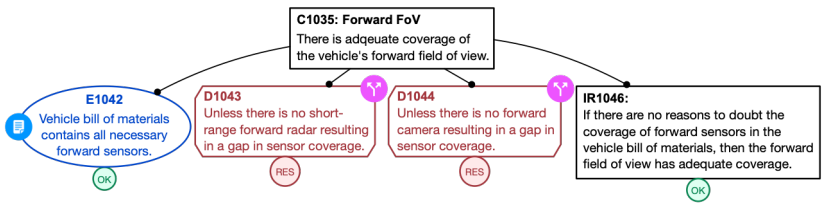


Fig. 8. Connecting variability points to residual doubts.

Variability in Artifacts. Our case study includes numerous variability points assigned to artifacts such that, depending on the system configuration, these artifacts appear in the variant-specific AC. For example, we have listed a suite of engineering and test artifacts for both the BBZ888 and ABX123 (compliance certificate, acceptance tests, integration testing, etc.) which appear in the AC depending on the configured Braking sub-system.

Variability in Indicators. Since our case study considers multiple operating domains, the safety performance of the AEB System might differ across operating domains. We have identified three safety performance indicators that track the rate of spurious braking events reported through the OEM's field monitoring program, each bound to different environmental conditions: fair weather, rain, and rain and snow. Depending on the configured operating environment, the safety goal for spurious braking (`C0004`) is instrumented with a different indicator.

Variability in Engineering Process. The OEM in our case study has engaged different firms to perform functional safety assessments for the Braking sub-systems. Each firm uses a different process for assessment: the adequacy of the assessment methodology is captured under an ACP for the functional safety certificate obtained by the OEM for the braking sub-systems. In highly distributed supply chains, where multiple firms contribute by providing products and engineering services (e.g., testing, analysis, software development) across a product line, establishing the variant-specific "means arguments" is very powerful.

Conditional Information Display for Diverse Audiences. Finally, we imagine a situation where the OEM would like to share their AC with different interest holders, but where the argument contains sensitive information (e.g., intellectual property) that cannot be revealed to some parties. Variability points can be created for elements containing sensitive information and then be excluded by toggling a variable, thus producing a variant of the AC that is suitable for different audiences.

7 Discussion and Future Work

This paper presented SIMPLE, a method for variability management in product-line ACs, together with a formalism for variability that has been implemented in a commercial AC management tool, *Socrates*. Using an automated emergency braking case study, we showed that SIMPLE, the formalism, and the tool support variability in argument structure, dialectic reasoning, evidential artifacts, safety performance indicators, engineering process (via ACPs), and information display for different audiences. Our approach is intentionally simple, making variability management accessible to time- and resource-constrained AC practitioners.

Our automotive case study is focused on product-level variability emerging from different feature configurations. However, another yet under-explored application is managing variability across *fleets of assets*. We have observed challenges managing heterogeneous fleets that have similar, but subtly dissimilar, assurance arguments in sectors with extremely high-value and long-lived assets, such as marine, rail, aerospace, defence, process control, and energy. For instance, a fleet of ships will share many aspects of assurance (similar technologies, operational procedures, naval architecture, etc.), but each ship might be specialized for different missions (arctic patrol v. search and rescue) and will have a unique maintenance history measured in decades. Through their lifetime, the assurance rationale for each member of a fleet often diverges, each requiring its own AC. We anticipate that a "*fleet-level AC*" will help to address this challenge by reducing duplication while still permitting variant-specific arguments that delve into the details for each ship.

Looking forward, we see opportunities to improve SIMPLE and its tooling. First, our formalism restricts variability point conditions to conjunctions of variables; though this sufficed for our case study, we would like to support full propositional logic, including disjunction and implication. Second, the AEB product-line AC is not yet complete; we intend to finalize and publish it. Third, procedures for rigorous evaluation of projects/variants in SIMPLE require further work, using both manual and automated mechanisms. A weakly evaluated variant from an otherwise robust product-line AC could result in false confidence in a specific projection because other similar projections were deemed acceptable. Finally, through partnerships with organizations developing product-line or fleet-level ACs, we would like to characterize common sources of variability and their impact on ACs.

References

Assurance Case Working Group. (2021). *Goal Structuring Notation Community Standard (Version 3)*. Safety Critical Systems Club.

Bishop, P., & Bloomfield, R. (1998). A Methodology for Safety Case Development. *Industrial Perspectives on System Safety*. Sixth Safety Critical Systems Symposium.

Denney, E., Pai, G., & Habli, I. (2015). Dynamic Safety Cases for Through-Life Safety Assurance. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, *2*, 587–590.

Diemert, S. (2026). *Confidence in Dynamic Assurance Cases* [Dissertation, University of Victoria].

Diemert, S., & Joyce, J. (2020). Eliminative Argumentation for Arguing System Safety - A Practitioner's Experience. *2020 IEEE International Systems Conference (SysCon)*, 1–7.

Goodenough, J. B., Weinstock, C. B., & Klein, A. Z. (2015). *Eliminative Argumentation: A Basis for Arguing Confidence in System Properties*. Carnegie Mellon University - Software Engineering Institute Pittsburgh United States.

Habli, I. (2009). *Model-Based Assurance of Safety-Critical Product Lines*. University of York.

Hawkins, R., Kelly, T., Knight, J., & Graydon, P. (2011). A New Approach to creating Clear Safety Arguments. In *Advances in Systems Safety* (pp. 3–23). Springer, London.

Hobbs, C., Diemert, S., & Joyce, J. (2024). Driving the Development Process from the Safety Case. *Safe AI Systems*. Safety Critical Systems Symposium.

Koopman, P. (2022). *How Safe Is Safe Enough?: Measuring and Predicting Autonomous Vehicle Safety*. Independently published.

Magdalinski, S., Chung, A., & Millet, L. (2026). *Assurance Cases Across the Product Line*. Critical Systems Labs Inc.

Millet, L., Diemert, S., Rees, C., Viger, T., Chechik, M., Menghi, C., & Joyce, J. (2023). Assurance Case Arguments in the Large: The CERN LHC Machine Protection System. In J. Guiochet, S. Tonetta, & F. Bitsch (Eds.), *Computer Safety, Reliability, and Security* (pp. 3–10). Springer Nature.

Motor Industry Software Reliability Association (MISRA). (2019). *Guidelines for Automotive Safety Arguments*.

Murphy, L., Viger, T., Di Sandro, A., Chechik, M., Kosmatov, N., & Kovács, L. (2026). PLACIDUS: Engineering Product Lines of Rigorous Assurance Cases. *Integrated Formal Methods*. Springer Nature.

Nešić, D., Nyberg, M., & Gallina, B. (2021). Product-line assurance cases from contract-based design. *Journal of Systems and Software*, 176, 110922.

van Gurp, J., Bosch, J., & Svahnberg, M. (2001). On the notion of variability in software product lines. *Proceedings Working IEEE/IFIP Conference on Software Architecture*, 45–54.

Viger, T., Joyce, J., Diemert, S., Menghi, C., Chechik, M., Uythoven, J., Zerlauth, M., & Felsberger, L. (2025). Assessing the Usefulness of Assurance Cases: Experience With the Large Hadron Collider. *Systems Engineering*, *29*, 99–113.