Formal Foundations for Hierarchical Safety Cases

Ewen Denney and Ganesh Pai
SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA
{ewen.denney, ganesh.pai}@nasa.gov

Iain Whiteside
School of Computing Science
Newcastle University, UK
iain.whiteside@newcastle.ac.uk

***Abstract*—*Safety cases* are increasingly being required in many safety-critical domains to assure, using structured argumentation and evidence, that a system is acceptably safe. However, comprehensive system-wide safety arguments present appreciable challenges to develop, understand, evaluate, and manage, partly due to the volume of information that they aggregate, such as the results of hazard analysis, requirements analysis, testing, formal verification, and other engineering activities. Previously, we have proposed hierarchical safety cases, *hicases*, to aid the comprehension of safety case argument structures. In this paper, we build on a formal notion of safety case to formalise the use of hierarchy as a structuring technique, and show that hicases satisfy several desirable properties. Our aim is to provide a formal, theoretical foundation for safety cases. In particular, we believe that tools for high assurance systems should be granted similar assurance to the systems to which they are applied. To this end, we formally specify and prove the correctness of key operations for constructing and managing hicases, which gives the specification for implementing hicases in AdvoCATE, our toolset for safety case automation. We motivate and explain the theory with the help of a simple running example, extracted from a real safety case and developed using AdvoCATE.**

***Index Terms*—Abstraction, Hierarchy, Safety assurance, Safety cases, Tool support**

I. Introduction

The development and acceptance of a *safety case* is not only a key element of safety regulation in many safety-critical sectors [1], such as nuclear, defence, transportation, and oil & gas, but also increasingly becoming an accepted safety certification practice. Safety (or, more generally, *assurance*) cases are structured arguments supported by a body of evidence, providing a convincing and valid justification that a system meets its (safety) assurance requirements, for a given application in a given operating environment. Safety cases can be documented in a variety of ways, e.g., a combination of text descriptions and diagrams. Graphical notations, such as the Goal Structuring Notation (GSN) [2], have emerged over the past decade providing a graphical syntax to document the argument structure embodying a safety case[1]. However, these structures are semi-formal and, per se, not checked by any formal system. Rather, they are designed, in part, to provide a degree of structure to ameliorate safety case comprehension and inspection.

Currently, safety cases are largely manually created often using common argumentation *patterns*, similar to those found in software engineering. In general, understanding, developing, evaluating, and maintaining safety cases remains a challenge due to the volume and diversity of information that a typical system safety case must aggregate when best engineering practice is followed, e.g., the mandated work products which show compliance to the relevant regulations and standards, the results of (safety, system, and software) analyses, various inspections, audits, reviews, simulations, verification activities including various kinds of subsystem/system tests, formal verification, and, if applicable, also the evidence of safe performance from prior operations. As an anecdotal example, the size of the preliminary safety case for surveillance on airport surfaces [3] is about 200 pages, and is expected to grow as the interim and operational safety cases are created.

As a safety case evolves and lower-level details are added during refinement, the natural high-level structure that patterns offer can be obscured, making comprehension difficult. For example, to develop a claim of correctness of some safety-relevant function, say, we can iteratively argue correctness by formalizing the claim, through iterative property decomposition, and formal verification [4]. However, for such an argument, the relevance of a very low-level claim to the wider context of system safety may be difficult to gauge if the claim exists deep within the natural system hierarchy.

These observations, and our own prior experience [5], suggest a need for abstraction and structuring mechanisms in creating and managing a safety case. Additionally, since safety cases are intended to be a basis for various decisions (e.g., whether or not a system is acceptably safe, should additional evidence be required to accept a claim, whether or not an argument is fallacious, etc.) we (and others [6]) believe that there is also a need for a theoretical foundation: both for the basic concepts of a safety case, and for more advanced notions, such as hierarchy and its (automated) creation. The broad goal is to make safety cases amenable to formal analysis, thereby providing greater assurance.

Our work is further motivated by the need to provide tool support that also has a formal basis: our assurance case tool, AdvoCATE [7], can assist in and, to an extent automate, safety case construction from artefacts such as requirements tables [8], safety case patterns [9], and external verification tools [10]. Such generated arguments have inherent structure that can be exploited by hierarchisation. However, the formal specification against which it can be verified that AdvoCATE operations (including those relevant for hierarchy) are well-

[1]We will use *argument structure* and *safety case* interchangeably in this paper; however, a safety case is the argument structure *together* with all the documents to which it refers.

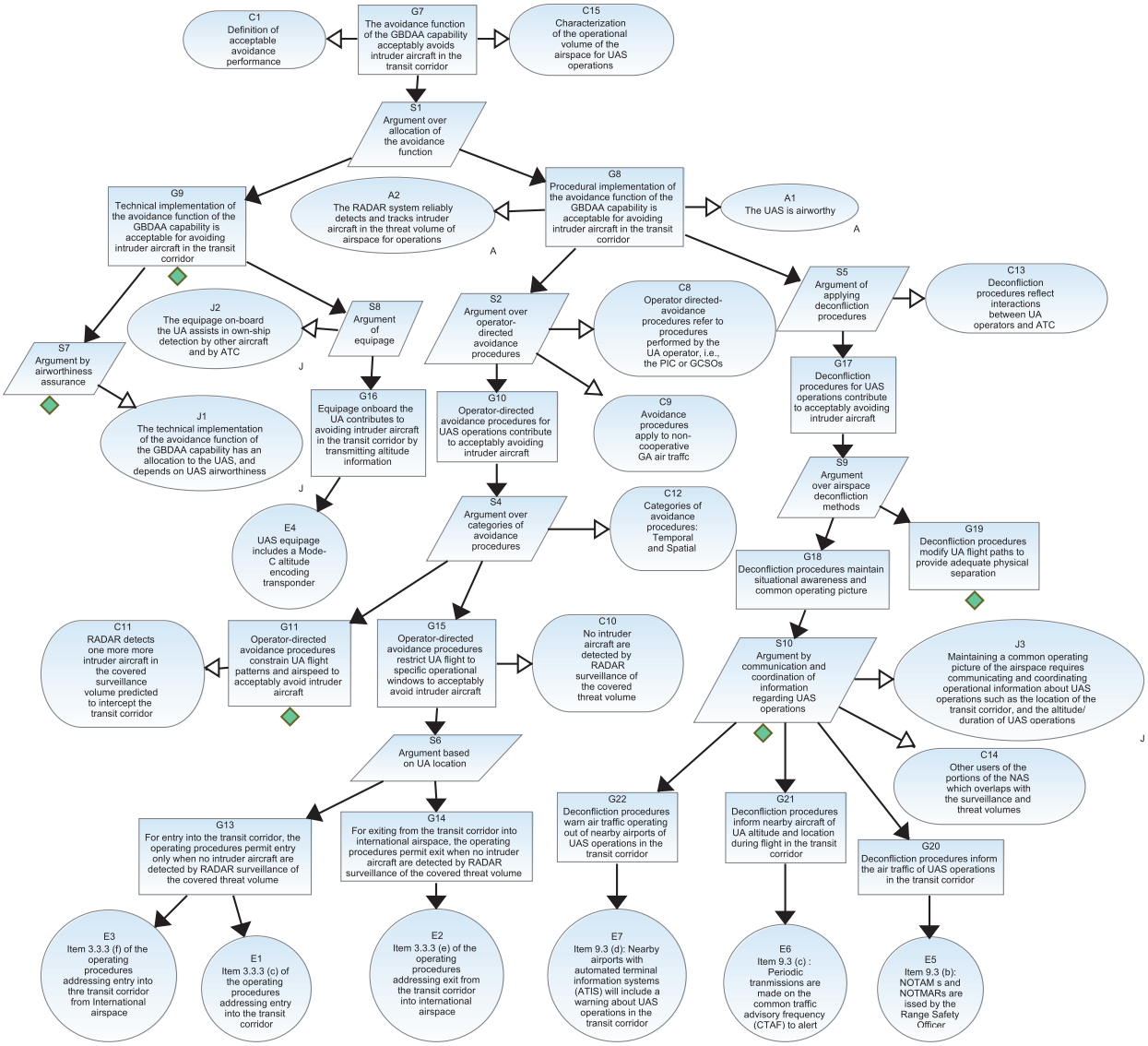

Fig. 1. *Flat* safety argument fragment, extracted from the safety case for a ground-based detect and avoid capability for UAS operations [14].

behaved, has lagged implementation, thus presenting a source of assurance deficits insofar as the arguments developed using tool-based automation are concerned. The work in this paper attempts to close that gap specifically with respect to hierarchical safety cases. The inspiration for our work comes from ideas from formal proof, where hierarchy can be added to proof trees [11]. We believe that a formal foundation will provide the requisite framework for independently and rigorously assuring that the hicases created using tool-supported automation are well-founded.

Previously [12], [13] we have proposed hierarchical safety cases, *hicases*, to aid comprehension of safety case argument structures through structured abstraction. In this paper, we use a running example, i.e., a fragment of an argument structure extracted from a real aviation system safety case [14], to show how hierarchical structuring can be introduced for abstraction (Section II, and Fig. 1). We use these modifications to motivate our formal definition of hicases, give the properties for well-formedness, and formalise the operations relevant for our implementation (Section III). In particular, our paper makes the following contributions: (i) a clear and abstract definition of the core concepts of the GSN standard [2] and a formal definition of the extension to hierarchical safety cases; (ii) an investigation of hicase properties: that hicases unfold to a non-hierarchical *skeleton*, that the hierarchical node types are mutually exclusive, and that (flat) safety cases can be embedded in a hicase; and, (iii) formal definitions and correctness proofs of key operations of hicases: a definition of views through the hierarchy, and the syntactic criteria governing when a fragment of a safety case can be hierarchically abstracted. These definitions, along with the formalization hicases, provide the specification for our implementation of hicases in AdvoCATE.

Finally, we illustrate our implementation of hicases in AdvoCATE (Section IV) by hierarchising the running example.

II. Running Example

A. Basic Goal Structuring Notation

Fig. 1 shows part of an argument structure extracted from a much larger safety case for the assurance of flight transit operations involving a *Ground-based Detect and Avoid (GBDAA)* capability [14]. The main claim (G7) for the fragment shown concerns the acceptability of the GBDAA function to safely avoid intruder air traffic when an Unmanned Aircraft System (UAS) is transiting through its operational airspace. In brief, to provide assurance that this claim holds, the argument presents a chain of reasoning (along with evidence) regarding the procedural and technical implementations of the avoidance function. The former further includes reasoning over deconfliction and operator-specific procedures, whereas the latter considers airworthiness and equipage issues. For a more detailed description, refer to [14].

We have presented the safety case fragment of Fig. 1 using the Goal Structuring Notation (GSN) [2], a graphical notation for representing the structure of an argument from its premises to its conclusions. Note that this structure has no hierarchical abstraction and we consider it to be a *flat* safety argument. As shown, the core elements of GSN each represent particular types of information that can be conveyed through a safety case. Specifically, we can state safety claims using *goals* and *sub-goals* (rectangle), e.g., nodes G7 and G8; the *strategies* to develop goals (parallelogram), e.g., nodes S1 and S5; the *evidence* used to substantiate the claims (circle), e.g., nodes E1 and E2; together with the appropriate associated *context* (rounded rectangle) e.g., nodes C1 and C15; *assumptions*, and *justifications* (ovals annotated with A and J respectively), e.g., nodes A1, and J2. GSN also provides a graphical '$\diamond$' annotation to indicate *undeveloped* (i.e., incomplete) elements, e.g., node G11. There are two link types with which to connect these notational elements: *in context of* ($\multimap$) and *is supported by* ($\rightarrow$). In addition to these core constructs, GSN provides infrastructure for modular abstraction via the notions of *modules*, *contracts*, and so-called *away* nodes, which we do not consider in scope for this paper. Moreover, GSN provides notational extensions for specifying argument structure *patterns* that, likewise, are out of scope.

In general, a GSN argument structure is a tree, with a *goal* as root, stating the safety objective to be demonstrated. It can be useful to relax this condition, and consider *partial* safety cases, i.e., argument structure fragments with a goal as a *local root* (as is the case for Fig. 1), each of which will be connected, eventually, to a unique *global root* goal. It is worth noting that the GSN standard [2] is very flexible, i.e., GSN elements are not subject to many syntactic restrictions and have no (formal) semantics. We believe that well-designed safety cases need such restrictions, and in this paper we will utilise our definitions to restrict the types of safety cases that can be constructed to those that are sensible.

B. Abstracting Structure with Hierarchy

Now we describe how hierarchical structuring can be introduced into the flat argument structure of Fig. 1. Specifically, Fig. 2 shows the hierarchisation of one subtree of Fig. 1, while in Fig. 3 we show how Fig. 1 can be completely hierarchised. As shown in these figures, there are several ways in which a safety case can be hierarchically structured, by introducing hierarchical nodes (hinodes) of different types. Additionally, each hierarchical node can have two possible views: it can be *open* and the argument structure that it contains is visible thereby allowing detailed inspection; or, it can be *closed* and be viewed abstractly as a safety case node of the given type.

In general, we permit three types of hinodes:

(1) *Hierarchical goals* are an abstraction to hide a chain of goals and one of their main purposes is to provide a high-level view of an argument structure. Thus, in Fig. 1, we can hierarchically abstract the entire subtree (with root at G8) into the higoal HG2 (See Fig. 2 and Fig. 3).

(2) *Hierarchical strategies* either (i) abstract a goal refinement sequence that is considered as supplemental to the main argument of interest, thus reducing the degree of branching in the argument structure; or, (ii) aggregate a meaningful chain of (one or more) related strategy applications.

For instance, in Fig. 1 the fragment of the argument between, and including, strategy nodes S2 and S4 is a chain of strategies that we can consider together as a higher-level strategy. Thus, we can abstract that fragment as a hierarchical strategy. Fig. 2 presents this hierarchical strategy in its *open* view, where both the hinode and its content are visible (hinode HS1). Fig. 3 shows HS1 in the *closed* view, acting as a normal strategy node and reducing the size of the safety case. Effectively, the closed view allows parts of a safety case to be viewed at a more abstract level.

Of course, to guarantee assurance, the safety case must still be reviewed in its entirety, but we believe hierarchical abstraction enables both abstract content and concrete details to be viewed as required.

(3) *Hierarchical evidence* abstracts a *fully developed* chain of related strategy applications. In Fig. 1, since the fragment starting from the strategy node S6 is downwards complete, i.e., it has no undeveloped elements, we can construct a hierarchical entity that abstracts and encapsulates it. In other words, the subtree (with root at strategy node S6) that justifies goal G15 can be packaged as a hierarchical evidence node (Hinode HE2 in Fig. 2, and Fig. 3).

C. Restrictions on Hierarchical Structuring

There are restrictions on that which can be abstracted inside a hinode: firstly, to preserve well-formedness, we require that input and output node types are consistent. Thus, a hierarchical strategy would have a goal as an incoming node and goals as outgoing nodes, in the same way as a normal strategy. Next, we cannot abstract disconnected fragments as there would be no path from the input goal to all the outputs. A design decision was made to place any context, justification, and assumption nodes inside a hinode; thus, we may not link a hinode to any other node using a $\multimap$ link type. Finally, we permit containment of hinodes by other hinodes, which allows us to nest hierarchies (see Fig. 2, where the higoal

HG2 contains the hinodes HS1 and HE2) as a way to manage argument structure size. The combination of strict control over entities that can be abstracted, and the flexibility offered by a partial order of hierarchy gives rise to intricate definitions of the key operations on hicases. To mitigate against potential flaws in *hierarchisation* and *views*, we have proved that these operations are correct. Note that the value addition that hinodes provide, is mainly to the structure of the argument. Thus, whereas a hicase communicates the same essential argument as its flat counterpart, we believe that the former may be, intuitively, easier to understand during argument review. Furthermore, the relevance of the formalization (described next) to the higher assurance of a safety case is the confidence that can be placed in the well-formedness of a hicase constructed from an automatic hierarchisation.

III. Formalisation

A. Preliminaries

First, we give a mathematical account of (flat) safety cases by representing them formally as a labelled tree, where the labelling function distinguishes the types of nodes subject to some intuitive well-formedness conditions. Let $\{s, g, e, a, j, c\}$ be the set of node types: strategy, goal, evidence, assumption, justification, and context respectively. The subset $\{s, g, e\}$ are *core* node types; and $\{a, j, c\}$ are *contextual* nodes.

Definition 1 (*Partial Safety Case*)**.** A *partial safety case (argument structure)* is a triple $\langle N, l, \rightarrow \rangle$, comprising nodes N, the labelling function $l : N \rightarrow \{s, g, e, a, j, c\}$, and the connector relation, $\rightarrow$: $\langle N, N \rangle$, which is defined on nodes. We define the reflexive, transitive closure, $\rightarrow^*$: $\langle N, N \rangle$, in the usual way. We require the connector relation to form a *finite forest* with the operation $isroot_{\rightarrow}(r)$ checking if the node r is a root in some tree. Furthermore, the following conditions must be met:

(1) Each part of the partial safety case has a root goal: $isroot_{\rightarrow}(v) \Rightarrow l(v) = g$
(2) Connectors only leave strategies or goals: $v \rightarrow w \Rightarrow l(v) \in \{s, g\}$
(3) Goals cannot connect to other goals: $(v \rightarrow w) \wedge (l(v) = g) \Rightarrow l(w) \in \{s, e, a, j, c\}$
(4) Strategies cannot connect to other strategies or evidence: $(v \rightarrow w) \wedge (l(v) = s) \Rightarrow l(w) \in \{g, a, j, c\}$

By virtue of forming a tree, we ensure that nodes cannot connect to themselves, that there are no cycles and, finally, that two nodes cannot connect to the same child node[2].

For uniformity, Definition 1 does not make a distinction between *is supported by* edges that connect core nodes, and *in context of* edges that connect contextual nodes. We make this distinction with a notational convention: We write $v_1 \rightarrow v_2$, if $v_1 \rightarrow v_2$ and $l(v_2) \in \{a, j, c\}$. Contextual nodes play little part in what follows, so we will write $v_1 \rightarrow v_2$ to mean that $l(v_2) \in \{s, g, e\}$. We say that an argument structure is a *total* safety case when it has a unique root. We will also refer to partial safety cases as *flat* when we want to emphasize their non-hierarchical nature. An important notion that can be defined

[2]Arguments in some domains, such as security assurance, commonly exhibit structures wherein multiple claims may be supported by the same evidence. Though our tool does allow this, *evidence assertions* (i.e., claims that can be made given the evidence provided) are unlikely to be exactly identical in a strictly well-formed argument, when used in support of higher-level claims. The theory described here uses this strict definition.

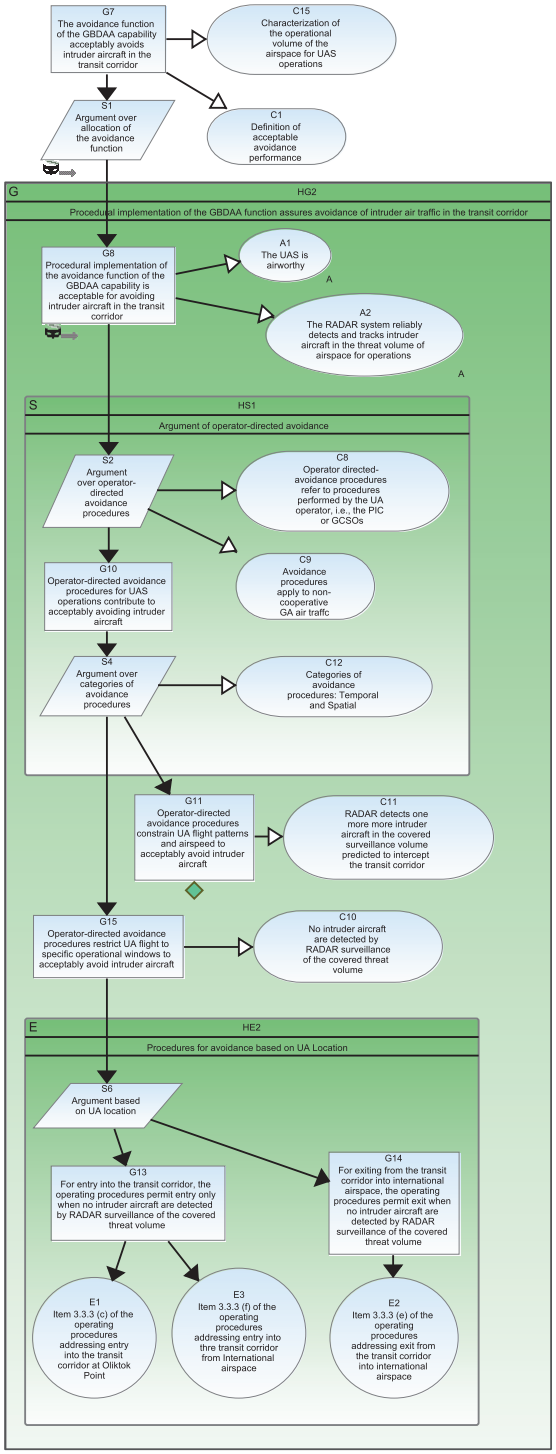

Fig. 2. Hierarchisation of a subtree of the argument structure of Fig. 1, with all hinodes open. The remaining branches of the argument have been hidden.

on a safety case is that of being *fully developed*, as one way to syntactically characterize its *internal completeness*.

Definition 2 (*Fully Developed Safety Case*)**.** A total safety case $\langle N, l, \rightarrow \rangle$, is *fully developed* if for every goal $g \in N$, $\exists v \in N$, such that $g \rightarrow^* v$ with $l(v) = e$. That is, all goals lead to evidence.

In fact, it is easily seen that this can be equivalently rephrased to state that all paths (connecting core nodes) lead to evidence. In other words, irrespective of node content, we say that a total safety case is internally complete if it is fully developed. Although there are many potential definitions of completeness, Definition 2 is the one we will use subsequently, in defining hierarchy.

Note, also, that in Definition 1 we have excluded the concept of an undeveloped node, and capture it by means of *attributes*. Our tool AdvoCATE [7], for example, permits the following attributes: (a) *developed*: whether the node is developed or still *to be developed*; (b) *identifier*: of the goal, strategy, etc.; (c) *description*: of the goal, strategy, etc. In general, we allow safety case nodes to contain arbitrary attributes, or *metadata*; in turn, this allows us to define complex rules, e.g., for node colouring. Attributes for nodes are accessed via projection functions tbd, id, etc., e.g., the attribute value associated with the undeveloped goal G9 in Fig. 1 is $tbd(\mathsf{G9}) = true$.

Now, we extend Definition 1 to *hierarchical safety cases* (or *hicases* for short). Hicases extend the definition of safety cases with an additional relation to represent hierarchical structure. We use the partial order symbol $\leq$ where $n < n'$ means that the node n *is inside* n'. We wish to define hicases in such a way that we can always *unfold* all the hierarchy to regain a flat safety case.

Definition 3 (*Partial Hierarchical Safety Cases*)**.** A partial *hierarchical safety case* is a tuple $\langle N, l, \rightarrow, \leq \rangle$. The set of nodes N, labelling function l, and connector relation $\rightarrow$ are as given in Definition 1. The forest $\langle N, \rightarrow \rangle$ is subject to all conditions except condition (1) of Definition 1.[3] The hierarchical relation is a (necessarily finite) forest *qua poset* $\langle N, \leq \rangle$. Generalising condition (1), global roots must be goals: $isroot_{\rightarrow,\leq}(v) \Rightarrow l(v) = g$. Finally, we impose conditions on the interaction between the two relations $\rightarrow$ and $\leq$:

(1) The connectors will target the outer nodes: $(v \rightarrow w_1) \wedge (w_1 < w_2) \Rightarrow v < w_2$.
(2) Connectors come from inner nodes: $(v \rightarrow w_2) \wedge (w_1 \leq v) \Rightarrow v = w_1$.
(3) Enclosure and connectors are mutually exclusive: $(v \leq w) \wedge (v \rightarrow^* w) \Rightarrow v = w$.
(4) Two nodes which are both at the top level (or immediately included in some node) means at most one has no incoming $\rightarrow$ edge. That is: $siblings_{<}(v_1, v_2) \wedge isroot_{\rightarrow}(v_1) \wedge isroot_{\rightarrow}(v_2) \Rightarrow v_1 = v_2$ where $siblings_{\leq}$ refers to *siblingness* in the forest.
(5) If v is a local root (using $\rightarrow$) of a higher-level node w (i.e., $v < w$), then
   - $l(w) = s$, if $l(v) = s \wedge [\forall v'\ v''.(v' < w \wedge v' \rightarrow v'' \wedge v'' \not< w) \Rightarrow l(v'') = g \vee$ subtree rooted at v is not fully developed], or
   - $l(w) = e$, if $l(v) = s \vee l(v) = e \wedge [\nexists v'\ v''.\ (v' < w \wedge v'' \not< w \wedge v' \rightarrow v'') \wedge$ subtree rooted at v is fully developed, or
   - $l(w) = g$, if $l(v) = g \wedge \forall v'\ v''.\ (v' < w \wedge v' \rightarrow v'' \wedge v'' \not< w) \Rightarrow l(v'') = s \vee l(v'') = e$

Conditions (1) – (4) of Definition 3 are designed to produce a mapping from a hierarchical safety case to its (flat) safety case unfolding, i.e., its *skeleton*, and they are identical to the conditions on hierarchical proofs [11]. We show subsequently (Section III-B), that safety cases can be viewed as (trivial) hicases and that the skeleton operation unfolds into a flat safety case. The final condition ensures that:

– A hierarchical strategy must have a strategy as root, and either (a) any node immediately outside the hierarchical strategy is a goal, or (b) the subtree with root v inside, is not fully developed. The latter case catches the possibility that there are no outgoing goals, but the node is not evidence.
– A hierarchical evidence node is the special case of a hierarchical strategy with no outgoing goals, but the subtree with root v is fully developed. Degenerate hierarchical evidence nodes are also allowed with the $l(v) = e$ condition.
– A higoal must have a goal as root, and any nodes immediately outside must be strategy or evidence nodes.
– All other node types must be leaves of $<$.

Hicases also have attributes in the same manner as their *flat* counterparts; however, some of the attributes for hinodes must be consistent with those of the nodes they enclose. For example, a hinode is considered undeveloped if any of its contained nodes is undeveloped.

B. Hicase Properties

In this section, we present and prove[4] the hicase properties that demonstrate that hicases are *fit for purpose*. In particular, we show (a) the mutual exclusivity of hinodes, demonstrating that the choice of hierarchical node type is uniquely determined by the contents; (b) the skeleton operation that removes hierarchical structure, leaving a well-formed flat safety case; (c) the precise conditions under which a hicase fragment can be hierarchised, and the correctness of hierarchisation in the technical sense that adding hierarchy preserves skeletons; and, (d) a notion of *view* through the hierarchy, that formalises the notion that a hinode can be either seen as a *black-box* node, or have its contents exposed. Firstly, we establish that there is no ambiguity about the type of a hierarchical node.

Theorem 1 (*Mutual Exclusivity*)**.** Any hierarchical node—not a leaf in the $\leq$ forest—can only satisfy the criteria for one of the hierarchical strategy, evidence, or goal types.

Next, we relate safety cases and hicases: our main result is that the hierarchy can be unfolded to produce a *flat* safety

[3] Since the root of a hinode will match the type of the node itself.

[4] Due to space constraints, we omit the proofs of mutual exclusivity and correctness of the skeleton operation.

case. Conversely, we note that a safety case $\langle N, l, \rightarrow \rangle$ can be mapped to a hicase $\langle N, l, \rightarrow, id_N \rangle$ where id_N is the trivial partial order with only reflexive pairs. This ordering trivially satisfies all the well-formedness properties of a hicase.

To retrieve the original safety case from a hicase, we define a *skeleton* operation, (sk), mapping hicases into flat safety cases and state that the tuple it constructs is indeed well-formed with respect to the safety case conditions of Definition 1.

Definition 4 (*Skeleton Operation*)**.** The *skeleton* operation, sk, is defined to map a hierarchical safety case $\langle N, l, \rightarrow, \leq \rangle$ to a safety case $\langle N', l', \rightarrow' \rangle$, where N' is the set of leaves of $\leq$, l' is the restriction of the labelling function l, and $v_1 \rightarrow' v_2$ iff: (1) $v_1 \rightarrow v_2$, or (2) $\exists\ w.\ v_1 \rightarrow w$ and v_2 is a local root of w.

Theorem 2 (*Skeleton*)**.** If h is a hicase, then $\mathrm{sk}(h)$ is a well-formed safety case. That is, it satisfies the properties in Definition 1.

Now, we characterise the fragments of a hicase in which it is possible to create a hinode. Thereafter, we describe the hierarchisation operation that encloses the fragment in a hicase with a new hinode. Finally, we state the correctness of this operation, i.e., it preserves skeletons. As a first step towards this, we define a safety case fragment.

Definition 5 (*Safety Case Fragment*)**.** Given a safety case $\langle N, l, \rightarrow \rangle$ and a set $F \subseteq N$. A safety case fragment is a tuple $\langle F, l_{\restriction_F}, \rightarrow_{\restriction_F} \rangle$ such that $\langle F, \rightarrow_{\restriction_F} \rangle$ is closed under $\twoheadrightarrow$ and $\rightarrow \_ \leftarrow$, with root v where $l_{\restriction_F}(v) \in \{s, g, e\}$.

That is to say, a *fragment* is a connected subset of a safety case. We write $\rightarrow \_ \leftarrow$ to mean that if $v_1 \rightarrow v_2 \rightarrow v_3$ and $v_1, v_3 \in F$ then $v_2 \in F$. A fragment can simply be one node (a strategy, goal, or evidence node); or, it may be the whole safety case. A safety case fragment will also satisfy properties (2) – (4) of Definition 1. If the root is a goal, it will be a safety case, i.e., also satisfy property (1).

Fig. 2 shows an example of a safety case fragment enclosed by a hierarchical evidence node (HE2). Now, we extend the definition to the more general situation of hicases, by ensuring all nodes inside a chosen hinode are also part of the fragment:

Definition 6 (*Hicase Fragment*)**.** Given a hicase $\langle N, l, \rightarrow, \leq \rangle$, a hicase fragment is a tuple $\langle F, l_{\restriction_F}, \rightarrow_{\restriction_F}, \leq_{\restriction_F} \rangle$ such that the following conditions hold:

(1) $F \subseteq N$ and is closed under $\twoheadrightarrow$ and $\rightarrow \_ \leftarrow$ as before.
(2) F is also closed under $\leq$ i.e., if $w \in F$ and $v \leq w$ then $v \in F$.
(3) For $v, v' \in F$, if $v \rightarrow w$ and $v' \leq w$ then $w \in F$, i.e., it is closed under entering hierarchy.

If a hicase fragment satisfies an extra structural condition, then it can be *hierarchised* and we say that the fragment is *hierarchisable*. That is, it corresponds to the set of nodes enclosed by a hinode.

Definition 7 (*Hierarchisable Hicase Fragment*)**.** A hicase fragment defined by the set F with root v is hierarchisable iff:

$$l_{\restriction_F}(v) = \begin{cases} s & \wedge \text{ all leaves } w \text{ are strategies or evidence:} \\ & l_{\restriction_F}(w) \in \{s, e\}, \text{ or} \\ g & \wedge \text{ all leaves } w \text{ are goals or evidence:} \\ & l_{\restriction_F}(w) \in \{g, e\}, \text{ or} \\ e & \end{cases}$$

That is, if the hicase fragment has a strategy as root, for example, then the leaves (in that fragment) must be either strategies (producing the output goals of the hierarchical strategy) or evidence (no outputs).

Definition 8 (*Hierarchisation*)**.** Given a hierarchisable fragment $\langle F, l_F, \rightarrow_F, \leq_F \rangle$ of a hierarchical safety case $\langle N, l, \rightarrow, \leq \rangle$, let $f \in F$ be the global root of the fragment; i.e., a root of $\rightarrow$ and of $\leq$. Furthermore, let f_p be the parent of f with respect to $\leq$ (if one exists). We define the *hierarchisation* of the fragment by a new node h as $\langle N_h, l_h, \rightarrow_h, \leq_h \rangle$ where the individual elements are defined as follows:

(1) $N' = N \cup \{h\}$
(2) $\leq$ is extended to $\leq_h$ where $h < f_p$ and $v < h$, for all $v \in F$.
(3) For defining $\rightarrow_h$ there are two possibilities:
  (a) if there exists $v \in N$ such that $v \rightarrow f$ then $\rightarrow_f$ is the modification to $\rightarrow$ such that $v \rightarrow h$ instead;
  (b) otherwise, $\rightarrow_h = \rightarrow$.
(4) Finally, we extend l by defining $l_h(h)$:
  (a) if $l(f) \in \{g, e\}$ then $l_h(h) = l(f)$;
  (b) if $l(f) = s$ and there is a leaf of $\rightarrow$ in the fragment that is a strategy, then $l(h) = s$.
  (c) if $l(f) = s$ and all leaves are evidence nodes. Then we need to check that there are no nodes connected to f that are not in the fragment, i.e., if $\exists v' \in N$ such that $f \rightarrow v'$ but $v' \notin F$ then $l(h) = s$; otherwise, $l(h) = e$.

Theorem 3 (*Correctness of Hierarchisation*)**.** If $\langle F, l_F, \rightarrow_F, \leq_F \rangle$ is a fragment of a hicase $h_1 = \langle N, l, \rightarrow, \leq \rangle$, and $h_2 = \langle N_h, l_h, \rightarrow_h, \leq_h \rangle$ is the *hierarchisation* of a new hinode h onto h_1, then h_2 is a well-formed hicase, and the hierarchisation is skeleton-preserving, i.e., $sk(h_1) = sk(h_2)$.

Proof. The proof requires showing each of the cases of the definition hold; we just sketch some of the details, e.g., $\rightarrow_h$ forms a forest: if $\rightarrow_h = \rightarrow$ this is trivial; if not, we note that the modification $f_p \rightarrow h$ does not break the partial order axioms. In particular, it cannot cause cycles since h did not previously exist. To show condition (5) of Definition 3, we need, in part, a case analysis on $l(h)$. Then, the conditions on a hierarchisable fragment allow us to show that this property holds. To show that the skeletons match, note that the modification (1) in defining $\rightarrow_h$ is exactly the opposite of the skeleton transformation (2) from Definition 4. □

To understand the open/closed representation of hinodes in a hicase, we introduce the notion of a *view*. A view can be seen as a particular slice through the hierarchy: either choose the hinode or its contents. We define a hicase view as follows:

Definition 9 (*Hicase View*)**.** Given a hicase $\langle N, l, \rightarrow, \leq \rangle$. Let $N' \subset N$ such that N' is closed under '<' incomparability, i.e., (1) $v_1, v_2 \in N' \Rightarrow v_1 \not< v_2$ and $v_2 \not< v_1$.

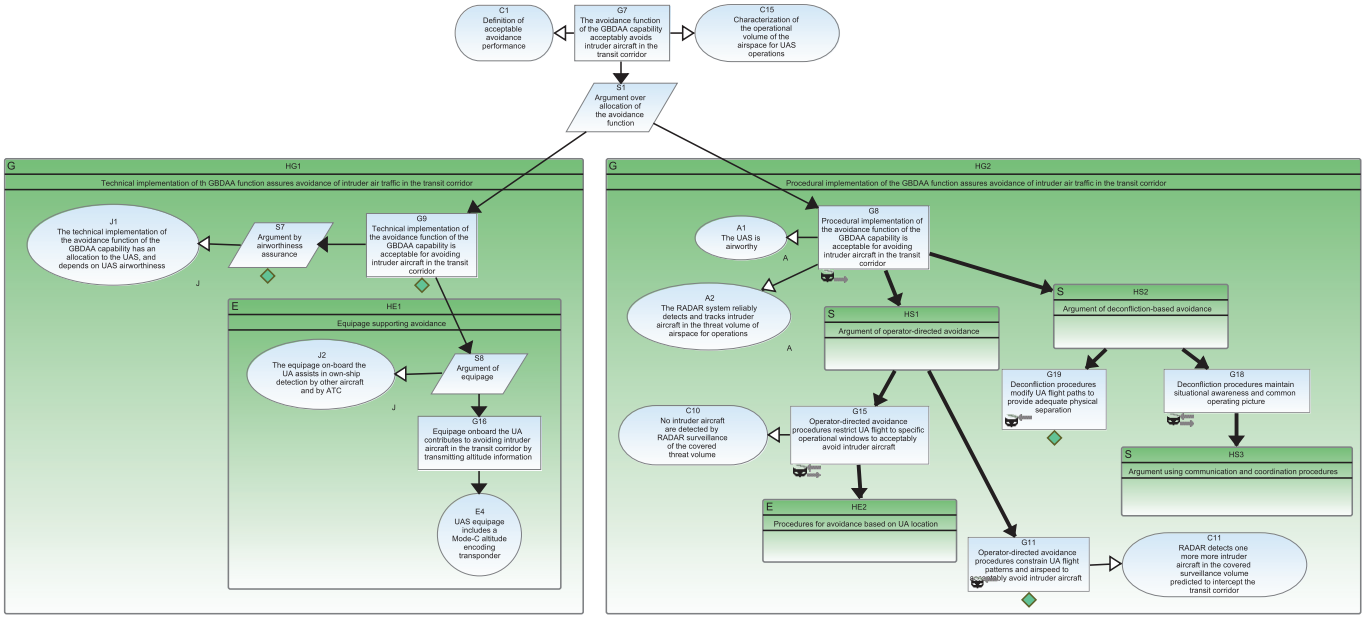

Fig. 3. A complete hierarchisation of the flat argument structure of Fig. 1, showing containment as well as both *open* and *closed* hinodes.

(2) $v_1 \in N'$ and $v_2 \not< v_1$ and $v_1 \not< v_2 \Rightarrow v_2 \in N'$
Then define $\rightarrow'$ as follows: For $v_1, v_2 \in N'$, $v_1 \rightarrow' v_2$ iff:
(1) $v_1 \rightarrow v_2$
(2) if $\exists\ w \in N.\ v_1 \rightarrow w$ and v_2 is a local root of w; or,
(3) if $\exists\ w \in N.\ w \rightarrow v_2$ and $w < v_1$.
The tuple $\langle N', l_{\restriction N'}, \rightarrow' \rangle$ is a *hicase view*.

It is easy to see that the skeleton is a special case of a hicase view.

Theorem 4 (*A Skeleton is a View*)**.** The skeleton $\langle N_s, l_s, \rightarrow_s \rangle$ is also a view of a hicase $\langle N, l, \rightarrow, \leq \rangle$.

Proof. By definition $N_s \subseteq N$ and the conditions on $\rightarrow_s$ match the first two conditions for a view. For the final condition, which states if $\exists\ w \in N.\ w \rightarrow v_2$ and $w < v_1$, note that $v_1 \in N_s$ is a leaf and therefore there is no such w. □

More generally, any view forms a safety case.

Theorem 5 (*A View is a Safety Case*)**.** A hicase view $\langle N', l_{\restriction N'}, \rightarrow', \rightarrowtriangle_{\restriction N'} \rangle$ of a hicase $\langle N, l, \rightarrow, \leq \rangle$ satisfies the well-formedness properties of a safety case.

Proof. Firstly, we need to show that $\rightarrow'$ forms a finite forest. In fact, it forms a tree. It is straightforward to show that it is rooted by a goal, since the underlying hicase must also be rooted by a goal (and the closure property ensures that it is chosen). To show that connectors only leave strategies or goals, we reason about the possible source of $v_1 \rightarrow' v_2$. Either $v_1 \rightarrow v_2$, in which case the property holds trivially; or, $v_1 \rightarrow q$ for some w which is again trivial; or, finally, $w \rightarrow v_2$ and $w < v_1$ for some w. In the final case, note that w is a hinode and therefore $l(w) \in \{s, g, e\}$. If it is a strategy or goal, we are done. If it is evidence, we derive a contradiction because by definition $w \nrightarrow v_2$ for any v_2. Similarly for other cases. □

IV. Application

Our implementation of hicases in AdvoCATE closely matches the formalization described earlier (Section III). Fig. 3 shows the hierarchisation of the flat argument structure of Fig. 1. As shown, the higoal HG2 contains additional hinodes (all in their closed view) besides those shown in Fig. 2. In particular, HG2 contains the additional hinodes HS2 and HS3, which were hidden in Fig. 2, besides HS1 and HE1. Whereas the fragment supporting goal node G15 has been abstracted as the hierarchical evidence node HE2, observe that the fragment supporting goal node G18 is, rather, a hierarchical strategy (HS3) owing to the '◇' annotation on the supporting strategy (node S10) indicating incompleteness (See Fig. 1). The higoal HG1 contains a hierarchical evidence node HE1, both of which have been presented in their open view.

The *open* and *closed* views in AdvoCATE, as shown in Fig. 3, can be seen as modelled by the *views* formalised in Definition 9. The only difference is that the *open* view in AdvoCATE keeps the contents of the 'opened' hinode visible, whereas the formal notion is designed so that a view is, itself, a flat safety argument, but with no notion of visibility. Additionally, the ability to create hinodes in AdvoCATE is governed by Definition 8. The benefit of hierarchical abstraction on the size of a safety case is immediately apparent in Fig. 3 where, after hierarchisation and progressively closing the hinodes (i.e., also closing the higoals HG1 and HG2), we can reduce the size of Fig. 1, from 45 nodes, to only 6 nodes.

V. Related Work

Hiproofs [11] are the immediate inspiration for hicases, and we follow the hiproof notation for the graphical representation of hicases. Hiproofs *could* be viewed as a more general model (for hierarchical trees) than safety cases, without the particular

node typing present in hicases. An alternative understanding would be to consider a hiproof as the strategy/evidence subset of our hicase representation (where the flow of goals is represented by the connections). However, hicases go beyond hiproofs by distinguishing multiple node types and requiring a more complex condition on hierarchy. Although there are several theorem provers and proof assistants which support hierarchical proof to varying degrees, the specific data structures they use have details which are not central to an understanding of hierarchy [11].

The GSN safety case notation has a concept of *module*. Modularity allows safety cases, as with other artifacts, to be decomposed into discrete modules so as to contain change impact, support distributed development, etc. Hierarchy, on the other hand, permits a system to contain sub-systems of the same kind. The concepts are thus distinct, though complementary; if modules can themselves contain modules, this results in hierarchical modularity [15]. However, there are important differences between modularity and hierarchy for safety cases.

Whereas *away* objects [2] are simply references to argument fragments in another module, a hinode is an additional node enclosing an already existing argument structure in the current argument. Moreover, hinodes and modules encompass different fragments. For example, a GSN module cannot correspond to a hierarchical strategy, i.e., a fragment beginning with a strategy, as an enclosure of an arbitrarily complex (unfinished) safety case fragment. Similarly, a module can contain multiple argument fragments (resulting in multiple roots), while a hinode always encloses a fragment with a single root. Modules also have informal *contracts* that they must fulfill to be well-formed, but hinodes do not enforce any such semantic constaints between nodes. Furthermore, while there is a formal basis for hierarchy (now), GSN modules do not (yet) have a formal basis. Thus, we can exactly distinguish between hierarchisable and non-hierarchisable fragments, whereas the constraints on modularizing safety cases are less clear. We plan to place GSN modules on a formal footing, and investigate more fully the relationship with hierarchy.

Safety case hierarchy using a notion of argument structure *depth* has been previously proposed [16], and represented as a basic decomposition. Although this approach can create the equivalent of hierarchical evidence, it cannot hierarchically abstract strategies, as in our approach, where we consider node combination for meaningful abstraction.

VI. Conclusions and Future Work

This paper advances our previous work [12], [13] on hierarchical structuring, by more precisely clarifying hierarchical safety cases and their properties. In particular, we have formally defined a *skeleton* operation as the unfolding of a hicase into a *flat* safety case, the notion of hicase *view*, the conditions for hierarchisation, and proved the correctness of these operations.

There are many interesting avenues for future development of hierarchy in safety cases. However, since our current theory only accounts for the core GSN, one key task is to extend the notion of hierarchy to also account for patterns and modules. Secondly, we would like learn potential hierarchical structure, e.g., using metadata, after which one might be able to abstract hierarchical patterns from existing safety cases.

We believe that both a formal basis and tool support are crucial for improving the credibility and wider acceptance of structured safety arguments during the certification of safety-critical products. Our goal has been to provide an abstract specification of hierarchy for safety cases, and a corresponding implementation. The theory provides a formal foundation to the implementation of hicases in our toolset, AdvoCATE [7], providing features for constructing, modifying, and viewing hinodes. To the best of our knowledge, this work describes the first implementation of hierarchy for safety cases.

Acknowledgement

This work was supported by the AFCS element of the SSAT project in the Aviation Safety Program of NASA ARMD.

References

[1] R. Bloomfield and P. Bishop, "Safety and Assurance Cases: Past, Present and Possible Future—an Adelard Perspective," in *18th Safety Critical Systems Symp.*, pp. 51–67, 2010.

[2] Goal Structuring Notation Working Group, "GSN Community Standard Version 1," Nov. 2011.

[3] EUROCONTROL, "Preliminary Safety Case for ADS-B Airport Surface Surveillance Application," Tech. Rep., Nov. 2011.

[4] E. Denney and G. Pai, "Evidence Arguments for Using Formal Methods in Software Certification," in *Proc. 2013 Intl. Symp. Soft. Reliability Eng. Workshops (ISSREW)*, Nov. 2013, pp. 375–380.

[5] E. Denney, I. Habli, and G. Pai, "Perspectives on Software Safety Case Development for Unmanned Aircraft," in *Proc. 42nd IEEE/IFIP Intl. Conf. Dependable Systems and Networks (DSN 2012)*, Jun. 2012.

[6] A. Wassyng, T. Maibaum, M. Lawford, and H. Bherer, "Software Certification: Is There a Case Against Safety Cases?" in *Foundations of Computer Software, Modeling, Development and Verification of Adaptive Systems*, LNCS no. 6662, pp. 206–227, 2011.

[7] E. Denney, G. Pai, and J. Pohl, "AdvoCATE: An Assurance Case Automation Toolset," in *Computer Safety, Reliability and Security (SAFECOMP) Workshops*, LNCS no. 7613, 2012.

[8] E. Denney and G. Pai, "A Lightweight Methodology for Safety Case Assembly," in *Computer Safety, Reliability and Security (SAFECOMP)*, LNCS no. 7612, pp. 1–12, 2012.

[9] E. Denney and G. Pai, "A Formal Basis for Safety Case Patterns," in *Computer Safety, Reliability and Security (SAFECOMP)*, LNCS no. 8153, pp. 21–32, 2013.

[10] E. Denney, G. Pai, and J. Pohl, "Heterogeneous Aviation Safety Cases: Integrating the Formal and the Non-formal," in *17th IEEE Intl. Conf. Engineering of Complex Computer Systems (ICECCS)*, Jul. 2012.

[11] E. Denney, J. Power, and K. Tourlas, "Hiproofs: A Hierarchical Notion of Proof Tree," *Electr. Notes Theor. Comput. Sci.*, vol. 155, pp. 341–359, 2006.

[12] E. Denney, G. Pai, and I. Whiteside, "Hierarchical Safety Cases," in *NASA Formal Methods Symp.*, LNCS no. 7871, pp. 478–483, 2013.

[13] E. Denney and I. Whiteside, "Hierarchical safety cases," NASA Ames Research Center, Tech. Rep. NASA/TM-2012-216481, Dec. 2011.

[14] R. Berthold, E. Denney, M. Fladeland, G. Pai, B. Storms, and M. Sumich, "Assuring Ground-based Detect and Avoid for UAS Operations," in *Proc. 33rd IEEE/AIAA Digital Avionics Systems Conf. (DASC)*. Oct. 2014.

[15] M. Blume and A. W. Appel, "Hierarchical Modularity," *ACM Trans. Prog. Lang. and Systems*, vol. 21, pp. 813–847, Jul. 1999.

[16] G. Stone, "On Arguing the Safety of Large Systems," in *10th Australian Workshop on Safety-Related Programmable Systems*, vol. 162, pp. 69–75, 2006.