Hierarchical Safety Cases

Ewen Denney[1], Ganesh Pai[1], and Iain Whiteside[2]

[1] SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA
{ewen.denney,ganesh.pai}@nasa.gov
[2] School of Informatics, University of Edinburgh
Edinburgh, EH8 9AB, Scotland
i.whiteside@sms.ed.ac.uk

Abstract. The development of a safety case has become common practice for the certification of systems in many safety-critical domains, but large safety cases still remain difficult to develop, evaluate and maintain. We propose hierarchical safety cases (*hicases*) as a technique to overcome some of the difficulties that arise in manipulating industrial-size safety arguments. This paper introduces and motivates hicases, lays their formal foundations and relates them to other safety case concepts. Our approach extends the existing Goal Structuring Notation (GSN) with abstraction mechanisms that allow viewing the safety case at different levels of detail.

Keywords: Abstraction, Automation, Formal methods, Hierarchy, Safety assurance, Safety cases

1 Introduction

A *safety case*, or more generally an assurance case, is a structured argument supported by a body of evidence, which provides a convincing and valid justification that a system meets its (safety) assurance requirements, for a given application in a given operating environment. The development of a safety case is increasingly becoming an accepted practice for the certification of safety-critical systems in the nuclear, defense, oil and gas, and transportation domains. Indeed, the development and acceptance of a safety case is a key element of safety regulation in many safety-critical sectors [1].

At present, safety cases are manually constructed often using patterns; they also have some natural higher-level structure, but this can become obscured by lower-level details during their evolution. Furthermore, due to the volume of information aggregated, safety cases remain difficult to develop, evaluate (or understand), and maintain. As an anecdotal example, the size of the *preliminary* safety case for surveillance on airport surfaces with ADS-B [9] is about 200 pages, and is expected to grow as the operational safety case is created. Tools such as AdvoCATE [5] can assist in and, to an extent, automate the construction of assurance argument structures from external verification tools [6], and artifacts such as requirements tables [3]. Often, these have inherent structure that can be exploited to help comprehension.

These observations, and our own prior experience [2], suggest a need for abstraction and structuring mechanisms in creating, and when communicating, a safety argument.

The motivation for our work is the ongoing construction of a safety case [4] for the Swift unmanned aircraft system (UAS), being developed at NASA Ames. We have used the goal structuring notation (GSN) [10] to document the Swift UAS safety case. In brief, GSN is an effective graphical notation for representing the structure of an argument from its premises to its conclusions. Using GSN (e.g., as illustrated in Fig. 1), we can express the *goals* or claims made (rectangle), the *strategies* (parallelogram) to develop goals, *solutions* (circle) that justify the claims, together with the appropriate associated *context* (rounded rectangle), *assumptions*, and/or *justifications* (ovals). GSN also provides a graphical annotation ('$\diamond$') to indicate *undeveloped* elements. There are, additionally, two link types with which to connect the notational elements: *in-context-of* and *is-solved-by*.

In this paper, we extend GSN to include hierarchical structuring mechanisms, motivating and illustrating our ideas with a simple, but real, example argument structure fragment. The resulting structures, *hicases*, better clarify the structure of a safety case and, we believe, improve the quality and comprehensibility of the argument. Our specific contributions are a formalization of the notion of a *partial safety case (argument structure)*, its extension to include hierarchy, and relating the unfolding of a hicase to an (ordinary) safety case argument structure.

2 Types of Hierarchy and Their Restrictions

Fig. 1 shows part of a chain of claims, strategies and evidence, from the top level of the auto-generated fragment of the Swift UAS safety case (see [4] for details). The top-level claim AC1 concerns the correct computation of aileron control variable values, during descent, by the relevant PID control loop in the Swift UAS autopilot. The chain of argumentation shown represents a *direct* proof of a verification condition. Some of the details of the proof have been transformed into the safety case, such as the theorem prover used as context, the proof objects, i.e., verification conditions, as claims, etc. This is an instance of a sub-structure that we may abstract away in a hierarchical presentation.

In general, we define three types of hierarchical abstractions, i.e., *hinodes*:

(1) *Hierarchical evidence* abstracts a *fully developed* chain of related strategy applications, e.g., in Fig. 1, since the argument structure starting from the strategy AC10 downwards is complete (has no undeveloped elements), we can construct a hierarchical evidence node, H1: Proof using Vampire–0.6 Prover, that abstracts and encapsulates it. As there are many such verification conditions (in the auto-generated safety case fragment of the Swift UAS), we have many instances of this structure. We can iterate this procedure up the proof tree which offers opportunities for nesting hierarchies. Thus, iterated abstraction can greatly reduce the size of the argument structure when viewed.

(2) *Hierarchical goals*, or *higoals*, are an abstraction to hide a chain of goals; one of their main purposes is to provide a high-level view of an argument structure. In Fig. 1, we can abstract the argument structure starting from (but not including) the strategy AC2 downwards, into the higoal H2: Decomposed correctness properties hold.

(3) *Hierarchical strategies* aggregate a meaningful chain of (one or more) related strategy applications, e.g., in Fig. 1, we can abstract the strategy AC30, along with its sub-goals (AC76 and AC86), and its context elements (AC32 and AC34), into a single

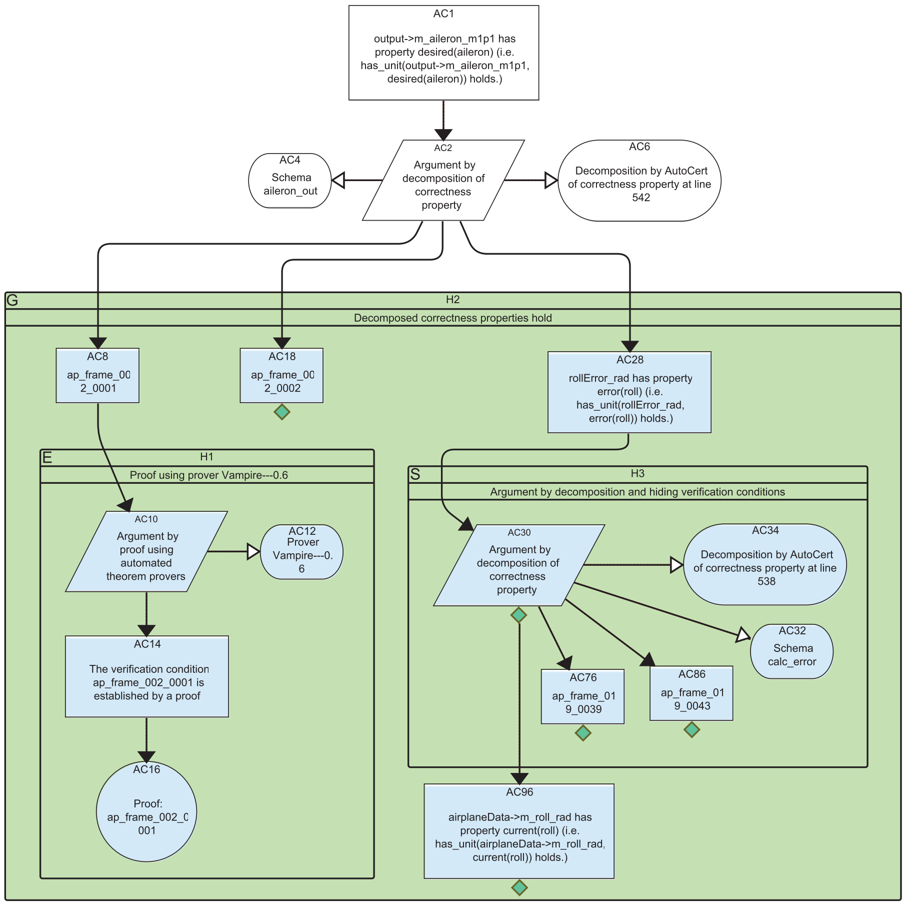
AC1
output->m_aileron_m1p1 has property desired(aileron) (i.e. has_unit(output->m_aileron_m1p1, desired(aileron)) holds.)
AC2
Argument by decomposition of correctness property
AC4
Schema aileron_out
AC6
Decomposition by AutoCert of correctness property at line 542
G
H2
Decomposed correctness properties hold
AC8
ap_frame_002_0001
AC18
ap_frame_002_0002
AC28
rollError_rad has property error(roll) (i.e. has_unit(rollError_rad, error(roll)) holds.)
E
H1
Proof using prover Vampire---0.6
AC10
Argument by proof using automated theorem provers
AC12
Prover Vampire---0.6
AC14
The verification condition ap_frame_002_0001 is established by a proof
AC16
Proof: ap_frame_002_0001
S
H3
Argument by decomposition and hiding verification conditions
AC30
Argument by decomposition of correctness property
AC34
Decomposition by AutoCert of correctness property at line 538
AC32
Schema calc_error
AC76
ap_frame_019_0039
AC86
ap_frame_019_0043
AC96
airplaneData->m_roll_rad has property current(roll) (i.e. has_unit(airplaneData->m_roll_rad, current(roll)) holds.)

Fig. 1. Fragment of an auto-generated part of the Swift UAS safety case [4], showing *hinodes* annotated as G (goal), S (strategy) and E (evidence).

hierarchical strategy H3: Argument by decomposition and hiding verification conditions. Thus, hierarchical strategies can hide side-conditions (or trivial subgoals) by fully enclosing particular paths of the safety case argument structure. This gives us the flexibility to concentrate an inspection on specific *important* paths through the safety argument, e.g., those paths addressing claims having 'high-risk'.

There are restrictions on what can be abstracted inside a hinode: firstly, to preserve well-formedness, input and output node types should be consistent, e.g., a hierarchical strategy would have a goal as an incoming node and goals as outgoing nodes, in the same way as an ordinary strategy. Next, we cannot abstract disconnected fragments as there would be no path from the input goal to all the outputs. It is important to note that this restriction does not force each hinode to have only one input. Rather, the restriction applies to the input, so multiple connections can enter a hinode. A design decision was to place any context, justification, and assumption nodes inside a hinode; thus, we may not link two (or more) hinodes, using a link of type *in-context-of*. Finally, we permit encapsulation of both hierarchical evidence and strategies by higoals (e.g., as shown in Fig. 1), or of both hierarchical goals and evidence by hierarchical strategies. This gives us a notion of *nesting* of hierarchies as a way to manage the size of an argument structure.

3 Formalization

To formalize standard safety case argument structures and hierarchical argument structures, we represent them as a labeled tree. The labeling function distinguishes the types of nodes subject to some intuitive well-formedness conditions.

Definition 1. *Let* $\{s, g, e, a, j, c\}$ *be the node types* strategy, goal, evidence, assumption, justification*, and* context *respectively. A* partial safety case (argument structure) *is a triple* $\langle N, l, \rightarrow\rangle$*, comprising nodes* N*, the labeling function* $l : N \rightarrow \{s, g, e, a, j, c\}$ *that gives the node type, and the connector relation,* $\rightarrow$: $\langle N, N\rangle$*, which is defined on nodes. We define the transitive closure,* $\rightarrow^*$: $\langle N, N\rangle$*, in the usual way. We require the connector relation to form a* finite forest *with the operation* $isroot_N(r)$ *checking if the node* r *is a root in some tree*[3]*. Furthermore, the following conditions must be met:*
(1) Each part of the partial safety case has a root goal: $isroot_N(r) \Rightarrow l(r) = g$
(2) Connectors only leave strategies or goals: $n \rightarrow m \Rightarrow l(n) \in \{s, g\}$
(3) Goals cannot connect to other goals: $(n \rightarrow m) \wedge [l(n) = g] \Rightarrow l(m) \in \{s, e, a, j, c\}$
(4) Strategies cannot connect to other strategies or evidence:
$(n \rightarrow m) \wedge [l(n) = s] \Rightarrow l(m) \in \{g, a, j, c\}$

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. Additionally, we see that the two link types (*is-solved-by* and *in-context-of*) have no semantic content, but rather provide an informational role.

Now, we extend Definition 1 with an additional partial order relation $\leq$ representing hierarchical structure, where $n < n'$ means that the node n is encapsulated in n'. We define a partial hierarchical safety case, i.e., hicase, such that we can always *unfold* all the hierarchy to regain an ordinary safety case argument structure.

Definition 2. *A partial* hierarchical safety case *is a tuple* $\langle N, l, \rightarrow, \leq\rangle$*. The set of nodes* N *and labeling function* l *are as in Definition 1. The forest* $\langle N, \rightarrow\rangle$ *is subject to the same conditions as in Definition 1. The hierarchical relation* $\leq$ *fulfils the axioms of a partial order and can thus also be viewed alongside* N *as a forest. Finally, we impose the following conditions on the interaction between the two relations* $\rightarrow$ *and* $\leq$*:*
(1) If v *is a local root (using* $\rightarrow$*) of a higher-level node* w *(i.e.* $v < w$*), then* $l(w) =$

$$\begin{cases} g, \text{ if } l(v) = g \wedge \forall v'\, v''.\, (v' < w \wedge v' \rightarrow v'' \wedge v'' \nless w) \Rightarrow l(v'') = s \\ s, \text{ if } l(v) = s \wedge [\forall v'\, v''.\, (v' < w \wedge v' \rightarrow v'' \wedge v'' \nless w) \Rightarrow l(v'') = g \\ \qquad\qquad \vee \text{ subtree rooted at } v \text{ is not fully developed}] \\ e, \text{ if } l(v) = s \wedge [\nexists v'\, v''.\, (v' < w \wedge v'' \nless w \wedge v' \rightarrow v'') \\ \qquad\qquad \wedge \text{ subtree rooted at } v \text{ is fully developed}] \end{cases}$$

[3] A safety case argument structure has a single root.

(2) Connectors will target the outer nodes: $(v \to w_1) \wedge (w_1 < w_2) \Rightarrow v < w_2$
(3) Connectors come from inner nodes: $(v \to w_1) \wedge (w_1 \leq w_2) \Rightarrow v = w_1$
(4) Hierarchy and connection are mutually exclusive: $(v \leq w) \wedge (v \to^* w) \Rightarrow v = w$
(5) Two nodes which are both at the top level, or immediately included in some node, means that at most one node has no incoming $\to$ *edge:*
$\mathit{siblings}_i(v_1, v_2) \wedge \mathit{isroot}_s(v_1) \wedge \mathit{isroot}_s(v_2) \Rightarrow v_1 = v_2$

Condition (1) formalizes our intuition that (a) a higoal must have a goal as root and any nodes immediately outside the higoal must be strategy nodes, (b) a hierarchical strategy must have a strategy as root, and either any nodes immediately outside the hierarchical strategy must be goals, or the subtree rooted at v inside is not fully developed. The latter accounts for the possibility that there are no outgoing goals, but the node is not evidence; and (c) a hierarchical evidence node is the special case of a hierarchical strategy with no outgoing goals, but where the subtree with root at v is fully developed. That is, we can view hierarchical evidence as a hierarchical strategy without outgoing goals just as evidence is an axiomatic strategy. Conditions (2) through (5) are designed to produce a mapping from a hierarchical argument structure to its ordinary argument structure unfolding, i.e., its *skeleton*.

We note that a safety case argument structure $\langle N, l, \to \rangle$ can be mapped to a hicase $\langle N, l, \to, id_V \rangle$ where id_V is the trivial partial order with only reflexive pairs. This ordering trivially satisfies all the well-formedness properties of a hicase. Conversely, we define a *skeleton* operation (sk), which maps hicases to ordinary safety case argument structures, such that the tuple it constructs is well-formed with respect to the safety case argument structure conditions (of Definition 1).

Theorem 1. *The* skeleton *operation (sk) which maps a hicase* $\langle N, l, \to, \leq \rangle$ *to a safety case argument structure* $\langle N', l', \to' \rangle$*, where* N' *is the set of leaves of* $\leq$*,* l' *is the restriction of the labeling function l, and* $v_1 \to' v_2$ *iff* $\exists w \in N \mid v_2 \leq w$ *and* $v_1 \to w$ *maps a well-formed hicase to a safety case argument structure.*

Proof sketch. The relationship between *hiproofs* [7] and hicases (as well as the corresponding relationship between safety cases and proofs) allows us to claim that the mapping constructs the appropriate forest structure on $\langle N', \to' \rangle$. We simply need to show the well-formedness conditions (2) through (4) of Definition 1. For instance, condition (2), i.e., $(v_1 \to v_2) \Rightarrow l(v_1) \in \{s, g\}$, comes for free since if $v_1 \to w$ then it already has this property for $v_1 \to' v_2$.

4 Related Work and Conclusions

Hierarchy in safety cases has been proposed as a basic (hierarchical) decomposition represented as indentations in a spreadsheet-based argument structure [11]. This work creates the equivalent of hierarchical evidence, but cannot hierarchically abstract strategies, as in our approach. Our notion of hierarchy considers ways in which to combine nodes for meaningful abstraction, unlike the notion of argument structure *depth*. GSN supplies a concept for *modules* and references to *away* nodes [10] that are complementary to hicases, though neither modules nor hicases subsume each other's functionality.

Whereas *away* objects are simply references to a separate safety case fragment, *higoals* are an additional node enclosing an existing argument structure. GSN modules do not have an equivalent notion of a hierarchical strategy as an enclosure of (possibly) a complex (unfinished) safety case fragment. Modules can be seen as a large segment of a safety case, typically applied at a higher level, whereas we view hinodes as being viable at all scales. Modules also have informal *contracts* that they must fulfill to be well-formed, but hinodes do not enforce any semantic properties.

We have implemented hicases in our assurance case toolset, AdvoCATE [5], providing basic features for constructing, modifying, and viewing hinodes, e.g., we can modify existing argument structures to add hinodes with *open* (white-box) or *closed* (black-box) views. We can also generate a tree representation of a hicase and modify its contents [8]. Our current definition for safety cases and hicases only accounts for core GSN and potential meta-data extensions. In practice, most safety case argument structures make use of either (or all) of the GSN modular extensions and pattern mechanisms; we would like to give an account for each of these within our model, with careful thought about the module language to ensure that no inconsistencies are introduced.

We would also like to investigate the formal notions of *hicase view* (a slice through the hierarchy giving a safety case fragment), and *hicase refinement* (providing a mathematical meaning for well-formed changes to the hicase); although both exist informally in our tool implementation, we believe it is important to formalize these concepts.

Acknowledgement. This work has been funded by the AFCS element of the SSAT project in the Aviation Safety Program of the NASA Aeronautics Mission Directorate.

References

1. Bloomfield, R., Bishop, P.: Safety and Assurance Cases: Past, Present and Possible Future – An Adelard Perspective, In: Proc. 18th Safety-Critical Sys. Symp. (Feb 2010)
2. Denney, E., Habli, I., Pai, G.: Perspectives on Software Safety Case Development for Unmanned Aircraft. In: Proc. 42nd Intl. Conf. Dependable Sys. and Networks. (Jun 2012)
3. Denney, E., Pai, G.: A lightweight methodology for safety case assembly. In: Proc. 31st Intl. Conf. Comp. Safety, Reliability and Security (SafeComp). pp. 1–12. (Sep 2012)
4. Denney, E., Pai, G., Pohl, J.: Automating the generation of heterogeneous aviation safety cases. Tech. Rep. NASA/CR-2011-215983, NASA Ames Research Center (Aug 2011)
5. Denney, E., Pai, G., Pohl, J.: AdvoCATE: An Assurance Case Automation Toolset. In: 31st Intl. Conf. Comp. Safety, Reliability and Security Workshops. pp. 8–21. (Sep 2012)
6. Denney, E., Pai, G., Pohl, J.: Heterogeneous aviation safety cases: Integrating the formal and the non-formal. In: 17th IEEE Intl. Conf. Eng. of Complex Comp. Sys. (Jul 2012)
7. Denney, E., Power, J., Tourlas, K.: Hiproofs: A hierarchical notion of proof tree. In: Electr. Notes on Theoretical Comp. Sci. 155, pp. 341–359 (May 2006)
8. Denney, E., Whiteside, I.: Hierarchical safety cases. Tech. Rep. NASA/TM-2012-216481, NASA Ames Research Center (Dec 2012)
9. European Organisation for the Safety of Air Navigation: Preliminary safety case for ADS-B airport surface surveillance application. PSC ADS-B-APT. (Nov 2011)
10. Goal Structuring Notation Working Group: GSN Community Standard v.1 (Nov 2011) `http://www.goalstructuringnotation.info/`
11. Stone, G.: On arguing the safety of large systems. In: 10th Australian Workshop on Safety-Related Programmable Sys. ACM Intl. Conf. Proc. Series, vol. 162, pp. 69–75 (2006)