Author Pre-Print Copy. International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2016).
The definitive version appears on SpringerLink.

Composition of Safety Argument Patterns

Ewen Denney and Ganesh Pai

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

Abstract. Argument structure patterns can be used to represent classes of safety arguments. Such patterns can become quite complex, making use of loops and choices, posing a potential challenge for comprehension and evaluation, offsetting the likely gains that might follow from creating arguments using them. We show how complex patterns can be constructed by *composition* of simpler patterns. We provide a formal basis for pattern composition and show that this notion satisfies certain desirable properties. Furthermore, we show that it is always possible to construct complex patterns by composition in this way. We motivate this work with example patterns extracted from real aviation safety cases, and illustrate the application of the theory on the same.

Keywords: Argumentation, Composition, Patterns, Safety cases, Unmanned aircraft systems

1 Introduction

Over the past few years, we have been involved in engineering a number of real safety cases for unmanned aircraft system (UAS) operations: initially, those concerning NASA Earth science missions [1] and, more recently, increasingly complex aeronautics research missions[1]. Our previous safety cases have successfully undergone review and approval by the Federal Aviation Administration (FAA), the US civil aviation regulator, while the more recent ones are either undergoing FAA review, or are in development.

The current set of guidelines governing UAS operational approval [2] does not explicitly require the use of argumentation in a safety case. However, the guidelines do require that an explanation be supplied for how the hazard mitigation measures specified in the safety case are expected to reduce risk. Indeed, we have found argumentation to be largely useful for that purpose and, using our methodology for developing assurance arguments [3], we have slowly begun including structured arguments in the safety case (reports) to organize and document the reasons why the intended operations can be expected to be acceptably safe.

Based on our previous, and ongoing effort, and the experience gained, a number of observations follow to motivate the work in this paper. Firstly, many of the UAS operations have been the first of their kind conducted in civil airspace.[2] Individually, they have unique mission-specific constraints and safety requirements; so, much of the

[1] As part of NASA's UAS traffic management (UTM) effort: http://utm.arc.nasa.gov/
[2] To our knowledge, at least in the US, and within a non-military context.

associated safety reasoning is also tailored to the mission. Taking the various operations together, we have been able to identify similarities amongst the associated hazard control mechanisms and safety systems, e.g., ground-based surveillance, safe separation measures, a suite of avoidance maneuvers, emergency procedures for off-nominal situations, etc. That, in turn, has allowed us to develop both domain-independent and domain-specific patterns of safety reasoning—clarifying how the identified Safety measures contribute to risk reduction—which we have specified as argument structure patterns in the Goal Structuring Notation (GSN) using our tool, AdvoCATE [4].

Next, going forward we want to design the required safety systems for future UAS missions by carefully leveraging as many reusable safety assets that have a successful operational history, as possible. In conjunction, we want to apply our argument development methodology, and construct the corresponding safety case(s) from a combination of the relevant safety reasoning patterns, and tailored arguments, as appropriate. Intuitively, there is a need for exploring how patterns (and/or arguments) can be combined.

Third, as mission complexity grows, the associated safety cases can also be expected to become larger and more complex. In fact, that has indeed been our own experience. The way in which argument patterns are composed, and the results of such composition, can be thought of as providing a view of the overall *architecture* of the safety case and, thereby, an insight into the 'big picture' of how the safety measures contribute to managing risk. Moreover, by using argument patterns and their composition to the extent possible, we expect to be able to generate large parts of the arguments through automatic pattern instantiation [5]. We further anticipate that this will allow us to better manage the complexity of the safety cases we create while also amortizing the effort expended in their development.

As such, the main goal (and contribution) of this paper is a (preliminary) formulation of the formal foundations for composing GSN argument patterns. First, we give a running example to illustrate the intuition underlying the theory (Section 2). Specifically, we give some simple patterns which we extracted from the UAS safety cases we authored. Then, on the basis of this example, we formalize the notion of composition (Section 3), after which we illustrate how we have applied composition in practice (Section 4). We conclude this paper contrasting our work with related research, and identifying avenues for future research (Section 5).

2 Illustrative Example

For what follows, we assume familiarity with GSN for specifying arguments/patterns, and refer interested readers to [5] and [6] for details on GSN syntax and semantics. We have extracted a number of simple patterns of safety reasoning from the initial UAS safety cases we created. Fig. 1 shows a selection of those patterns, given using GSN pattern syntax[3] and representing, respectively: hazard enumeration (Fig. 1a); mitigat-

[3] Due to space constraints, and for figure legibility, we omit the contextual nodes (i.e., assumptions, justifications, and context) that provide additional clarification of the associated reasoning, from the patterns in Fig. 1. Also note that, in some cases, the strategies in these patterns include the safety measures used to achieve a goal in addition to the standard GSN strategies that provide inference explanations.

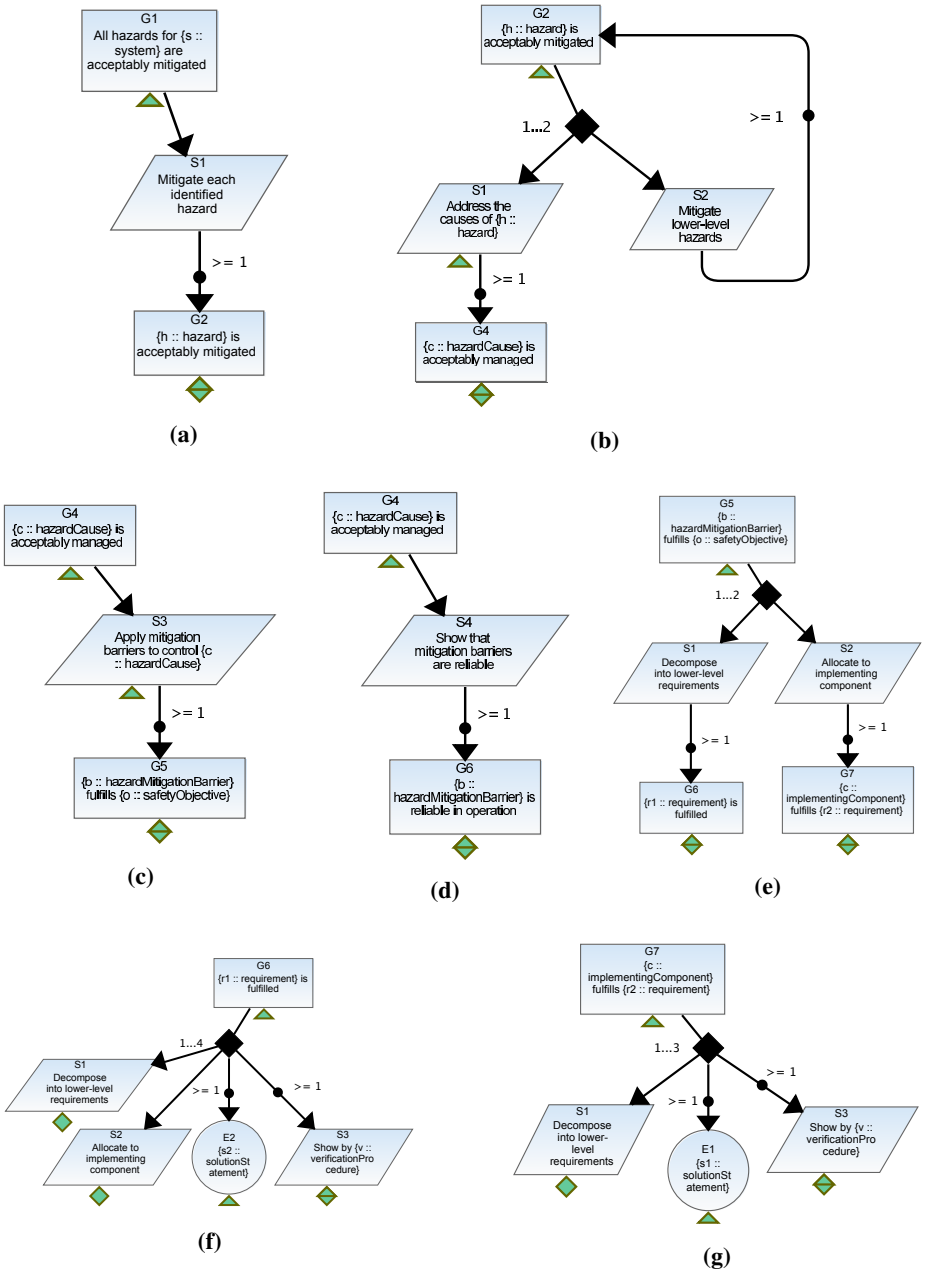

Fig. 1. A selection of simple, domain-independent GSN argument patterns extracted from real UAS safety cases, representing part of the reasoning underlying *i*) risk reduction with the use of mitigation barriers and *ii*) how mitigation barriers satisfy their applicable safety objectives. We will subsequently compose these patterns (see Fig. 2, which composes the latter three, and Fig. 4, which composes the former four).

ing a specific hazard by enumerating its causes or by hierarchical decomposition over its constituent lower-level hazards (Fig. 1b); managing a hazard cause by invoking multiple hazard mitigation barriers, each of which meets a particular safety objective that, in turn, specifies the requirement to be fulfilled to manage that hazard cause (Fig. 1c); managing a hazard cause by also showing that the applicable barriers are reliable in operation (Fig. 1d); hierarchical decomposition of a safety objective into lower-level requirements, or its allocation to specific components of a mitigation barrier (Fig. 1e); and supporting a safety objective by applying a verification procedure, via direct evidential support, reapplying hierarchical decomposition, or by reallocation to lower-level components (Figs. 1f and 1g, respectively). Note that these patterns do not encode a comprehensive collection of risk reduction measures, but reflect part of the approach that we have used successfully in the safety cases we authored. Moreover, the individual structures are variations on well-known safety argument patterns, such as *hazard-directed breakdown*, and *requirements breakdown* [5].

By examining Fig. 1, we can see that there is an intuitive notion of *sequential composition* where patterns are joined in a *top-down* way so that a leaf node of one pattern is the root of another. Similarly, there is also a notion of *parallel composition*, where patterns can be thought of as being placed alongside one another, and joined to reconcile common nodes and links. Using these patterns as the running example, we subsequently describe (Section 4) how we have composed patterns to supply safety rationale in a recently authored UAS safety case. The instance arguments of those patterns explain how the barriers of ground-based surveillance, and avoidance meet their safety objectives for managing the collision hazard posed by air proximity events.

3 Pattern Composition

We now formalize what it means to compose patterns. The goal is to develop a principled approach to composing arbitrary patterns, generalizing the intuition (as above) underlying the composition of the simple patterns of Fig. 1, to arbitrary (and larger) patterns. There are several subtleties that must be addressed, e.g., reconciling overlapping fragments, and determining when a composition will be well-formed. Moreover, we manually created the composition when we applied it in practice; however, we want to automate the functionality in our tool, AdvoCATE. We build on our previous work, using the following (slightly modified) definition of argument patterns from [5], and omit the conditions described there for brevity.

Definition 1 (Argument Pattern). *An* argument pattern *(or* pattern*, for short), P, is a tuple $\langle N, l, p, \mathfrak{m}, \mathfrak{c}, \rightarrow\rangle$, where $\langle N, \rightarrow\rangle$ is a directed hypergraph*[4] *in which each hyperedge has a single source and possibly multiple targets, and comprising a set of nodes, N, a family of labeling functions, l_X, where $X \in \{t, d, m, s\}$, giving the node fields* type*,* description*,* metadata*, and* status*; and $\rightarrow$ is the connector relation between nodes.*

Let $\{\mathcal{G}, \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{J}, \mathcal{C}\}$ be the node types goal, strategy, evidence, assumption, justification*, and* context *respectively. Then, $l_t : N \rightarrow \{\mathcal{G}, \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{J}, \mathcal{C}\}$ gives node types,*

[4] A graph where edges connect multiple vertices.

$l_d : N \to string$ gives node descriptions, $l_m : N \to A^*$ gives node instance attributes, and $l_s : N \to \mathcal{P}(\{tbd, tbi\})$ gives node development status.

There are additional (partial) labeling functions: p is a parameter label on nodes, $p : N \rightharpoonup Id \times T$, giving the parameter identifier and type; $\mathfrak{m} : N^2 \rightharpoonup \mathbb{N}^2$ gives the multiplicity range on a link between two nodes, with $\langle L, H \rangle$ representing the range from L to H; $\mathfrak{c} : N \times \mathcal{P}(N) \rightharpoonup \mathbb{N}^2$, gives the range on the choice attached to a given node, where $\mathfrak{c}(x, \mathbf{y})$ is the choice between child legs $\mathbf{y}$ with parent node x. Here, n is simply the number of legs in the choice, and so can be omitted.

The links of the hypergraph, $a \to \mathbf{b}$, where a is a single node and $\mathbf{b}$ is a set of nodes, represent choices. We write $a \to b$ when $a \to \mathbf{b}$ and $b \in \mathbf{b}$, and $a \to \{b, c\}$ when a is the parent of a choice between b and c. A pattern node n is a *data node*, if it has a parameter, i.e., $n \in dom(p)$. Otherwise, a node is *boilerplate*.

3.1 Composition

There are various alternative ways in which composition can be defined. The simplest definition, however, which works for our driving examples, is to take the union of all links in the respective patterns, using shared identifiers as the points at which to join. This is a *conjunctive* interpretation of composition, where we require fragments in *both* patterns to be satisfied. We will require that data be equivalent on corresponding nodes, and call such patterns *conflict-free*. For multiplicities on corresponding links and choices, however, it is not possible to reconcile distinct ranges without either losing information[5] or making ad hoc combinations. We thus adopt the simple solution of also assuming that there are no conflicts between corresponding multiplicities.

Definition 2 (Conflict-free Patterns). *The two patterns $P_1 = \langle N_1, l_1, p_1, \mathfrak{m}_1, \mathfrak{c}_1, \to_1 \rangle$ and $P_2 = \langle N_2, l_2, p_2, \mathfrak{m}_2, \mathfrak{c}_2, \to_2 \rangle$ are* conflict-free *whenever $l_1|_{N_1 \cap N_2} = l_2|_{N_1 \cap N_2}$ and $p_1|_{N_1 \cap N_2} = p_2|_{N_1 \cap N_2}$. If $x, y \in N_1 \cap N_2$ and $x \to_i y$ ($i = 1, 2$) then $\mathfrak{m}_1(x, y) = \mathfrak{m}_2(x, y)$, and if $x \in N_1 \cap N_2$, $\mathbf{y} \subseteq N_1 \cap N_2$, and $x \to_i \mathbf{y}$ ($i = 1, 2$) then $\mathfrak{c}_1(x, \mathbf{y}) = \mathfrak{c}_2(x, \mathbf{y})$.*

Henceforth, we will use P_1 and P_2 as metavariables for patterns representing the above tuples.

Definition 3 (Pattern Composition). *Let P_1 and P_2 be conflict-free patterns. Then, $P_1 \,||\, P_2 = \langle N_1 \cup N_2, l'', p'', \mathfrak{m}'', \mathfrak{c}'', \to'' \rangle$ where i) $l'' = l_1 \cup l_2$; ii) $x \to'' \mathbf{y}$ iff $x \to_1 \mathbf{y}$ or $x \to_2 \mathbf{y}$; iii) $\mathfrak{m}'' = \mathfrak{m}_1 \cup \mathfrak{m}_2$; and iv) $\mathfrak{c}'' = \mathfrak{c}_1 \cup \mathfrak{c}_2$.*

It can be seen that this is a well-formed pattern. The definition is simple but subtle, since the merging of the links can introduce recursion. Note also that when composing a choice $A \to \{B, C\}$ with $A \to B$ we retain both links, rather than merging them. Also, choices can be interwoven in, for example, $A \to \{B, C\} \,||\, A \to \{C, D\}$. However, duplicates are removed in $A \to B \,||\, (A \to B, A \to C)$. Now, clearly $||$ is commutative and associative modulo renaming of the node identifiers, and so composition can be defined over sets of patterns.

[5] There is no single range that corresponds to the union of possibilities represented by two distinct ranges. This could be addressed, however, by generalizing annotations from ranges to logical constraints that can express dependencies between nodes.

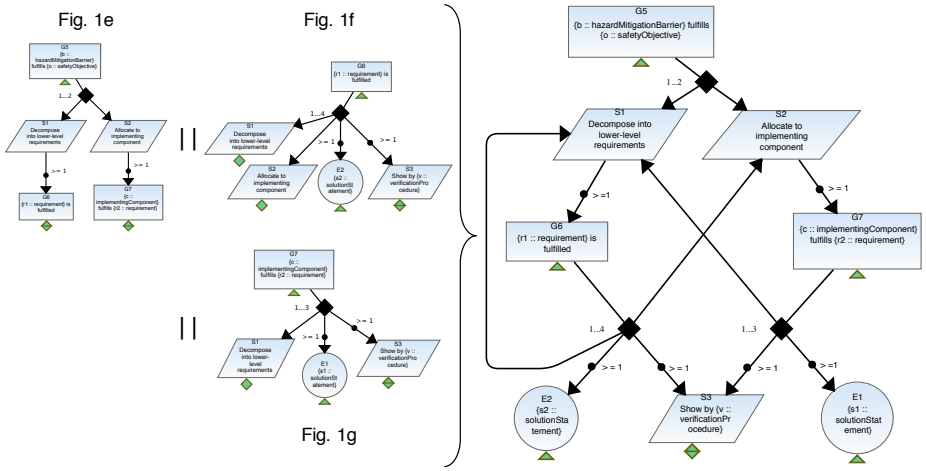

Fig. 2. Parallel composition (||) of the *elementary* patterns of Figs. 1e, 1f, and 1g (repeated here, above left) giving a *compound* pattern (above right).

Fig. 2 shows a *compound* pattern—a variation on the *requirements breakdown pattern* [5]—the result of the (parallel) composition of its reasoning elements, which are themselves the *elementary* patterns in Figs. 1e, 1f, and 1g. The elementary pattern in Fig. 1e describes how the claim that a hazard mitigation barrier fulfills a specific safety objective (goal node G5) is supported by decomposition into lower-level requirements, or by allocation to an implementing component of the technical system embodying the barrier (strategy nodes S1 and S2 respectively). The patterns in Figs. 1f and 1g, respectively, show how the resulting leaf claims (of Fig. 1e)—that a particular requirement is fulfilled (goal node G6), or that the allocated component fulfills a corresponding requirement (goal node G7)—are each either supported directly by relevant evidence items (solution nodes E1 and E2, respectively), or developed using an appropriate verification procedure (strategy node S3). Additionally, each of those claims can be further supported, again, by hierarchical decomposition (strategy S1).

Upon composing these elementary patterns, if there are repeated nodes (or fragments) we retain one copy and discard other copies, after which we resolve the relations between all the pattern nodes (as specified in Definition 3). Note, in Fig. 2, that the abstraction for iteration (i.e., the loop link from the choice following goal node G6, to strategy node S1) follows as a natural consequence of composition.

3.2 Correctness

We now discuss in what sense the pattern composition is *correct*.[6] Intuitively, a pattern represents a set of traces, or paths, and the composition should, in some sense, be a conservative combination of the paths in the component patterns. Since we have defined the composition as the union of (single-step) links, this is trivially true so, instead, we

[6] Proofs of the theorems in the rest of this paper have been omitted due to space constraints.

ask whether interesting properties are preserved. In [5], we defined various properties of patterns. It can be shown that composition preserves some of those properties, while for others, we need additional conditions. We will discuss two such properties now:

i) We say that a pattern is *unambiguous* when for all paths $s_1, s_2 : A \to B^*$ such that every internal node is boilerplate, we have $s_1 = s_2$, and that a pattern is *complete* when every leaf node is a data node.

ii) We say that $a \to^{must} \mathbf{b}$, when every loop-free path from a that is sufficiently long must eventually pass through some $b \in \mathbf{b}$. Then, an argument pattern is *well-founded* when, for all pattern nodes a, and sets of nodes $\mathbf{b}$, such that $a \notin \mathbf{b}$, if $a \to^{must} \mathbf{b}$ then it is not the case that for all $b \in \mathbf{b}$, $b \to^{must} a$.

Theorem 1 (Property Preservation). *Let P_1 and P_2 be patterns.*

i) If P_1 and P_2 are complete and unambiguous, then if there are not distinct paths of boilerplate nodes such that $A \to^* B$ in both patterns, the composition is complete and unambiguous.

ii) If P_1 and P_2 are well-founded and, in addition, if whenever $A \to^* B$ in P_1 and $B \to^* A$ in P_2, then $\exists C \,.\, B \to C$ in either P_1 or P_2, and $C \not\to^* A$ in either P_1 or P_2, then the composition is well-founded.

The preservation theorem thus tells us that (with some additional 'compatibility' conditions) composition of 'good' patterns gives us a good pattern. We would now like to formulate a dual theorem, that any pattern can be constructed from elementary patterns.

Definition 4 (Elementary Pattern). *A pattern is* elementary *(or loop-free) if for all nodes A, B, if $A \to^* B$ then $B \not\to^* A$.*

Prima facie, however, it is a trivial observation that it is always possible to construct a pattern by composition of elementary patterns, since we can simply compose fragments consisting of all the separate links (and hyperlinks). Instead, we need to show that a pattern can be factorized into a collection of elementary patterns which are maximal in some sense. We make two observations:

i) 'tight' loops between a node and its child can only be composed from non-pattern fragments. Thus we either allow such loops in the factors or, as we do here, simply exclude them from the statement of the theorem;

ii) the factors need not actually be unique. Even if we limit ourselves to maximal factors, it is still possible to move branches between factors, so any characterization of uniqueness needs to be modulo an equivalence under such rearrangements.

Hence we define an equivalence relation on pairs of patterns, $p_1, p_2 \sim p_3, p_4$ when we can rearrange a branch in p_1, p_2 to get p_3, p_4 and then extend this in the obvious way to arbitrary sets of patterns. In other words, pruning a branch b from p_1 gives p_3, and grafting it on p_2 gives p_4.

Theorem 2 (Pattern Factorization). *All patterns with no tight loops can be expressed as a maximal composition of elementary patterns. That is, if p is a pattern with no tight loops, then $\exists p_1 \cdots p_n \,.\, p_i$ elementary and $p = p_1 \,||\, \cdots \,||\, p_n$, such that $\forall q_1 \cdots q_m \,.\, p = q_1 \,||\, \cdots \,||\, q_m \Rightarrow$ there exists a partition $I_{i \cdots n}$ of $\{1, \ldots, m\}$ with for each $I_i = \{x_1, \ldots, x_{n_i}\}$, $r_i = q_{x_i} \,||\, \cdots q_{x_{n_i}}$, such that we have $r_1, \ldots r_n \sim p_1, \ldots, p_n$.*

That is, any factorization $\{q_i\}$ of p can be partitioned so that each subset of the partition corresponds to a single factor p_i, modulo rearranging.

3.3 General Composition

Rather than use overlapping node identifiers to determine composition points, we want to be able to compose *arbitrary* patterns, placing no assumptions on identifiers. We thus generalize the above definition so that nodes of P_1 and P_2 may or may not overlap. Without loss of generality, however, we will typically assume that they are disjoint.

Since the overlap between two patterns need not, itself, be a pattern, we need to generalize to *pre-patterns*. A pre-pattern has the same type of data (i.e., nodes, links, labels, etc.) as a pattern but need not respect the well-formedness rules. We define embeddings as mapping between pre-patterns that preserve structure. To express that embeddings do not introduce loops, we first define $a \leq b$ if for all paths from the root $s : r \rightarrow^* b$, we have $a \in s$, and $a < b$ when $a \leq b$ and $a \neq b$.

Definition 5 (Pre-pattern Mappings & Embeddings). *Let A and B be (pre-)patterns. We say that $e : A \rightarrow B$ is a* (pre-)pattern mapping *if it maps nodes to nodes and whenever $A \rightarrow B$ then $e(A) < e(B)$, i.e., all paths to $e(B)$ must pass through $e(A)$, and $e(A) \neq e(B)$. A* pre-pattern embedding *is a pre-pattern mapping that preserves data, that is, 1) $l^B_x(e(a)) = l^A_x(a)$ for $x \in \{t, d, m, s\}$; 2) If $\mathfrak{m}^A(x, y) = m$ then for some link $x' \rightarrow y'$ in $e(x) \rightarrow^* e(y)$ we have $\mathfrak{m}^B(x', y') = m$. Similarly for $\mathfrak{c}^A(x, \mathbf{y}) = c$. If e is an embedding from A to B we write this as $e : A \hookrightarrow B$.*

To define compositions more generally we make use of some simple category theory[7] and, in particular, the notion of *pushout*. A pushout encodes the minimal (and thus unique) object which combines two objects in a specific way. We define this within the *category of pre-patterns*, $\mathcal{P}re\mathcal{P}at$, which has pre-patterns for objects and pre-pattern embeddings for morphisms. We are now in a position to define general compositions.

Definition 6 (General Composition). *Let C be a pre-pattern, and $e_1 : C \hookrightarrow P_1$, $e_2 : C \hookrightarrow P_2$ (a so-called* span*) be pre-pattern embeddings. Then the pushout of e_1 and e_2, which we write as $P_1 \,||_{e_1,e_2}\, P_2$, gives us the general composition of P_1 and P_2.*

Note that the notion of context-freedom is now generalized by e_1 and e_2 being embeddings. Next, since $\mathcal{P}re\mathcal{P}at$ is not co-complete (as co-equalizers do not exist, in general), we rely on an explicit construction to show that pushouts exist.

Theorem 3 (Well-definedness of General Composition). *The general composition of P_1 and P_2 is well-defined. That is, pushouts exist in $\mathcal{P}re\mathcal{P}at$ and, moreover, if P_1 and P_2 are patterns, then $P_1 \,||_{e_1,e_2}\, P_2$ is also a pattern.*

We define the pushout $\langle N, l, p, \mathfrak{m}, \mathfrak{c}, \rightarrow \rangle$ as follows. Let P_1 and P_2 be pre-patterns, and $e_1 : C \hookrightarrow P_1$, $e_2 : C \hookrightarrow P_2$ the common embeddings. We sketch the construction of the pushout (omitting the definitions of l, $\mathfrak{m}$, and $\mathfrak{c}$ to save space): $N = N_c \oplus$

[7] For basic concepts of category theory, we refer the reader to an introductory textbook, such as [7].

$N_1 \backslash ran(e_1) \oplus N_2 \backslash ran(e_2)$, i.e., disjoint union of the node sets minus ranges of the embeddings. Also,

$$x \to y \Leftrightarrow \begin{cases} x = x_i, y = y_i \in N_i, \nexists z \in C \,.\, e_i(z) \in \{x, y\} \text{ and } x_i \to_i y_i \\ x = x_i \in N_i, y \in C, \text{ and } x_i \to_i e_i(y) \\ y = y_i \in N_i, x \in C, \text{ and } e_i(x) \to_i y_i. \end{cases}$$

Finally, we observe that the general definition is equivalent to Definition 3 in the following sense.

Corollary 1 (Equivalence of Composition). *Let P_1 and P_2 be patterns. There exists a span giving a general composition of P_1 and P_2 which is isomorphic to $P_1 \,||\, P_2$. Define the span $e_1 : C \hookrightarrow P_1$, $e_2 : C \hookrightarrow P_2$ as: i) $N_c = N_1 \cap N_2$; ii) $l_c, p_c, \mathfrak{m}_c, \mathfrak{c}_c, \to_c$ are the obvious restrictions to N_c; and iii) $e_1(n) = e_2(n) = n$. Then, $P_1 \,||_{\,e_1,e_2} P_2 \cong P_1 \,||\, P_2$.*

4 Application

We have used the elementary patterns identified in Fig. 1 (and others) along with their combinations to explain the required safety rationale—by creating *instance arguments* of the combined patterns—in a more recent UAS safety case to provide assurance of safe operations. We are also applying them to other safety cases currently in development. In brief, our approach is as follows.

First, we select the patterns that we can meaningfully compose into larger patterns to address specific concerns, e.g., how a hazard is managed by the combination of different mitigation barriers, how a specific barrier meets its safety objectives, etc. Then we examine the composed pattern to determine the extent to which it is applicable, e.g., whether it is (internally) complete or whether additional reasoning content is required in the pattern. Here, there may be a need to define additional domain- or application-specific patterns. Thereafter, we instantiate the patterns and examine the instance arguments to determine the extent to which the instance provides the assurance required. Again, there may be a need to define additional argument elements or bespoke arguments to *complete* the overall reasoning. The result comprises argument structures that supply the required safety rationale, e.g., how specific mitigation barriers meet their safety objectives and contribute to reducing risk.

Fig. 3 shows a fragment of one such argument structure resulting from this approach. In particular, the nodes *not* highlighted by the thick border in the figure are a fragment of the instance argument of the composed pattern in Fig. 2, instantiated with respect to the ground-based surveillance barrier. The argument is intended to show how the barrier meets its safety objective (root goal node G2, in Fig. 3). The highlighted (goal, context, and solution) nodes are additional argument elements/fragments that we subsequently introduced to complete the argument, and to address the concerns/essential information that the pattern did not include. Note that the instance argument also includes contextual nodes of the pattern that we had previously omitted (e.g., the context nodes C37 and C41). We similarly instantiated the pattern in Fig. 2) with respect to the avoidance barrier (not given here).

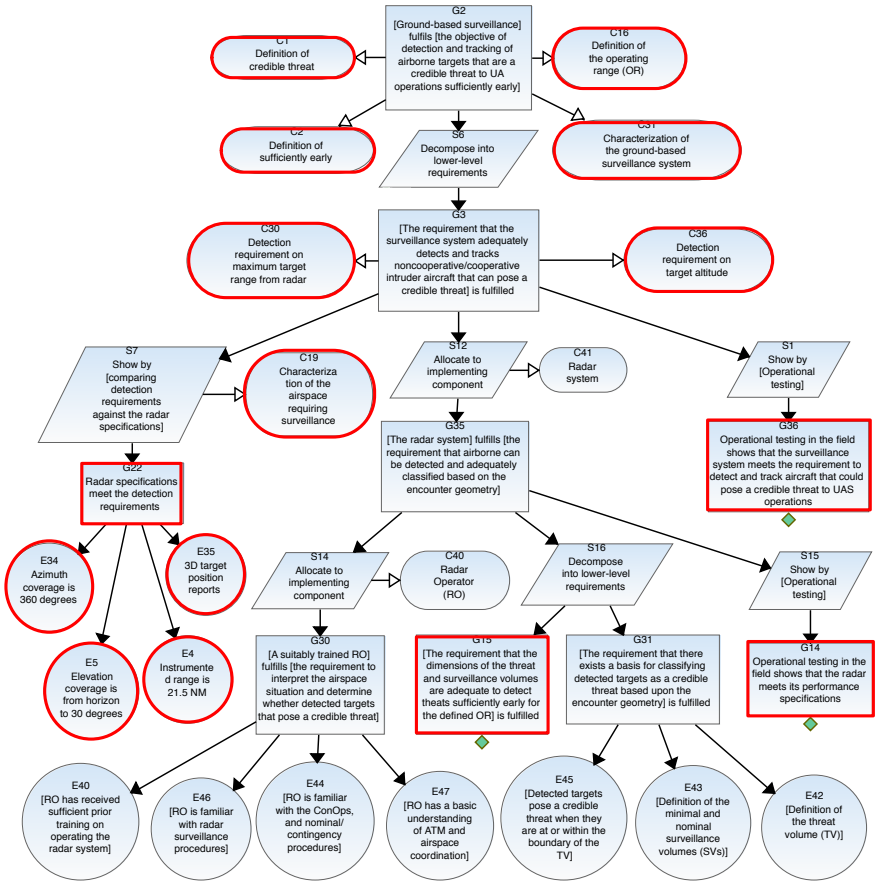

Fig. 3. Fragment of the instance argument of the compound pattern in Fig. 2, when instantiated for the surveillance barrier, and appended with tailored argument elements (shown by the goal, context, and solution nodes highlighted with a thick border).

Fig. 4 shows the (structure of the) compound pattern which explains the contribution of hazard mitigation barriers to managing hazard causes and, in turn, to mitigating the identified hazards. This pattern is produced from the general composition (see Section 3.3) of the patterns in Figs. 1a, 1b, 1c, and 1d. Intuitively, it can also be seen as the result of a sequence of simpler compositions, in particular the (sequential) composition of the patterns in Figs. 1a and 1b which, in turn, is (sequentially) composed with the parallel composition of the patterns in Figs. 1c and 1d. Similarly, the compound pattern of Fig. 2 can, in fact, be sequentially composed with the compound pattern of Fig. 4. The result, another compound pattern, is equivalent to the general composition of all the elementary patterns in Fig. 1. The instance argument for that pattern[8], which includes the argument fragment shown in Fig. 3, explains the role of all applicable mitigation

[8] Due to space constraints, neither this compound pattern nor its instance are given here.

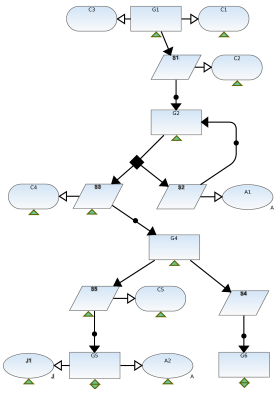

Fig. 4. Result of the composition of the elementary patterns in Figs. 1a–1d. Note that this figure primarily illustrates the compound pattern *structure*, also indicating the contextual nodes not shown earlier. For node/link content, see Figs. 1a–1d.

barriers in reducing the likelihood of the different identified hazards during the UAS mission, e.g., a near midair collision (NMAC), or air proximity event (AIRPROX).

5 Related Work and Conclusions

Compositional approaches to safety case construction have been considered in [8], however the focus there is on composing modular arguments. A catalogue of GSN patterns for software safety assurance has been supplied in [9], along with the assertion that the patterns link together to form a single software safety argument upon instantiation. Thus, that work (implicitly) alludes to the capability and utility of pattern composition, although it stops short of describing what composition means, and providing examples for the same. Similarly, [10] gives generic patterns of reasoning empirically identified from real safety cases—-i.e., so called *building blocks*, given in the Claims-Argument-Evidence (CAE) notation—with the explicit intent to combine them into *composite blocks*—analogous to hierarchical (argument) nodes [11]—and *templates*, which are closer to the compound patterns presented here. This work also asserts the capability and utility of composition, but only gives examples of building blocks as opposed to the templates produced from their composition. Moreover, little has been said about what composition means, and what modifications, if any, result to template semantics, and their graphical structure, in relation to their constituent building blocks.

In this paper we have continued our ongoing line of work on developing formal foundations to support automation in safety case development, in which argument structures are a first class object of study. We are now using our preliminary theory of pattern composition to provide a formal basis for implementing a suite of features in our tool, AdvoCATE, including automated refactoring of patterns, identifying reusable pattern components, and composing them in an automated (or interactive) way.

Although we currently manually create patterns for instantiation, composition lets us incrementally construct larger patterns of safety reasoning by combining smaller

patterns (extracted from, say, legacy safety cases). When combined with automated pattern instantiation [5], we can increase the level of useful automation that can be brought to bear when creating larger, more complex safety cases. The value addition for creating arguments this way, we believe, is that patterns give the *type* of an instance argument, providing a richer abstraction than argumentation schemes [12], for example, and allowing us to determine whether larger arguments can be sensibly combined by examining abstract, and relatively smaller, structures. Moreover, though there are differences, similar techniques could be used for merging and refactoring of argument fragments themselves. An interesting avenue of inquiry for future work is to determine what a suitable representation of *argument architecture* should be. Modular structure has been proposed for this [13], but here we have suggested that patterns and their combination can serve as such an architecture. It might also be useful to represent 'glue' argumentation that connects patterns, or refinements between domain-independent and domain-specific patterns.

Acknowledgement. This work was funded by the SASO project under the Airspace Operations and Safety Program of NASA ARMD.

References

1. Berthold, R., Denney, E., Fladeland, M., Pai, G., Storms, B., Sumich, M.: Assuring ground-based detect and avoid for UAS operations. In: 33rd IEEE/AIAA Digital Avionics Systems Conference (DASC 2015). pp. 6A1–1–6A1–16. (Oct. 2014)
2. Federal Aviation Administration (FAA): Flight Standards Information Management System, Volume 16, Unmanned Aircraft Systems. Order 8900.1 (Jun. 2014)
3. Denney, E., Pai, G.: A methodology for the development of assurance arguments for unmanned aircraft systems. In: 33rd International System Safety Conference (ISSC 2015) (Aug. 2015)
4. Denney, E., Pai, G., Pohl, J.: AdvoCATE: An assurance case automation toolset. In: SAFECOMP 2012 Workshops. LNCS, vol. 7613, pp. 8–21. (2012)
5. Denney, E., Pai, G.: A Formal Basis for Safety Case Patterns. In: SAFECOMP 2013. LNCS, vol. 8153, pp. 21–32. (2013)
6. Goal Structuring Notation Working Group: GSN Community Standard Version 1 (Nov. 2011). `http://www.goalstructuringnotation.info/`
7. Pierce, B.C.: Basic Category Theory for Computer Scientists. MIT press (1991)
8. Kelly, T.: Concepts and Principles of Compositional Safety Case Construction. Technical Report COMSA/2001/1/1, University of York (2001)
9. Hawkins, R., Kelly, T.: A systematic approach for developing software safety arguments. In: 27th International System Safety Conference (ISSC 2009) (2009)
10. Bloomfield, R., Netkachova, K.: Building blocks for assurance cases. In: 2014 IEEE ISSRE Workshops. (ISSREW). pp. 186–191 (Nov 2014)
11. Denney, E., Pai, G., Whiteside, I.: Formal foundations for hierarchical safety cases. In: 16th IEEE Intl. Symp. High Assurance Sys. Eng. (HASE 2015). pp. 52–59. (Jan. 2015)
12. Walton, D., Reed, C.: Argumentation schemes and defeasible inferences. In: Workshop on Computational Models of Natural Argument, 15th European Conference on Artificial Intelligence. (2002) pp. 11–20
13. Industrial Avionics Working Group: Modular Software Safety Case Process GSN – MSSC 203 Issue 1 (Nov. 2012)