When Repair Is Not Enough: Systematic Mitigation of Incomplete Knowledge Graphs

Masoud Ebrahimi[1,2], Kaj Hänninen[2], Kristina Lundqvist[2], and Marjan Sirjani[2]
[1] *Formal Trust AI, Sweden. E-mail: masoud.ebrahimi@formaltrust.ai*
[2] *Department of Computer Science & Engineering, Mälardalen University, Sweden.*

Safety analysis of knowledge graphs derived from system documentation is challenging because the documentation is often incomplete or ambiguous, and the analysis must support traceable risk-management actions. Existing validation methods that operate under the Open World Assumption miss hazards implied by missing information and do not provide a systematic path from hazard identification to minimal, traceable mitigation. We formalize the semantics of a knowledge graph (KG) using Kripke structures, enumerate feasible states, and classify the result of verification into four cases: satisfaction, repair, mitigation, and unrealizability. For repair and mitigation, we combine reachability analysis with delta debugging to compute minimal predicate-level changes and to synthesize a weakest safe completion that excludes unsafe terminal states while preserving maximal safe design freedom. We contribute a decision taxonomy for post-identification actions. We contribute an optimization-based synthesis of additions and flips for minimal mitigation. We contribute an unrealizability procedure that guides controlled relaxation of safety properties with explicit traceability to documentation and safety artifacts. For unrealizable specifications, the procedure escalates from model-level corrections to stakeholder-level actions, assigning follow-up decisions to appropriate stakeholders (e.g., the safety engineer, system designer, or user/operator) via safety-case traceability. In an automotive seatbelt case study, the framework identifies latent hazards, synthesizes minimal mitigation actions, and distinguishes system-actuated from occupant-actuated causes, thereby increasing the practical value of the analysis for assurance and certification.

Keywords: Safety-critical systems, knowledge graph mitigation, specification-level repair, delta debugging.

1. Introduction

KGs represent system documentation as directed, labeled graphs. Each node represents an entity (for example, a component, state, or concept), and each edge represents a relationship. Formally, a triple $\langle s, p, o \rangle$ encodes that subject s has predicate p with object o (Ehrlinger and Wöß, 2016; Ji et al., 2022). Ontology axioms and domain constraints govern which triples are valid (Gruber, 1993; Baader, 2003). However, incompleteness or ambiguity in the documentation propagates directly into the resulting KG (Ji et al., 2022; Shi and Weninger, 2017). Reasoners operating under the Open World Assumption (OWA) detect only explicit violations; this leaves latent hazards undiscovered (Baader, 2003; Shi and Weninger, 2017). Considering all possible subjects, predicates, and objects enables systematic exploration of every design that the documentation can realize. Kripke structures enable exhaustive enumeration under the Closed World Assumption (CWA), which we use to characterise feasible completions of incomplete documentation-derived knowledge graphs and to drive subsequent repair and mitigation synthesis (Ebrahimi et al., 2026). Hazard identification is only the first step in assurance because certification evidence requires risk-management actions that exclude hazards entirely.

This paper develops a mitigation pipeline based on Kripke semantics and asks the following question: given the catalogue of safe and unsafe states, which changes exclude the unsafe region while preserving valid design intent? We distinguish two types of changes:

System-level changes target a KG by modifying its triples, thereby altering the set of states that the KG realizes. System-level changes trace back to system documentation (requirements, design decisions, operational constraints).

Safety-level changes relax the safety property itself when it is unrealizable under the feasible

Proc. of the 36th European Safety and Reliability Conference
Edited by José C. Matos, Paulo B. Lourenco, Michael Beer, Daniel Oliveira and Edoardo Patelli
 Published by Research Publishing
ISBN: 981-973-0000-00-0 :: doi: 10.3850/981-973-0000-00-0_output

design space. These changes trace to safety-case arguments and hazard-analysis artifacts.

Section 2 covers preliminaries. Section 3 presents our contributions: (i) a taxonomy of violation patterns for systematic mitigation strategy, (ii) an adaptation of delta-debugging to compute minimal modifications from unsafe to safe states via system-level changes, and (iii) systematic unrealizability analysis via safety-level changes. Section 4 applies the framework to an automotive seatbelt case study. Sections 5 and 6 discuss related work and conclude.

2. Preliminaries

This paper brings together concepts from safety engineering, formal methods, and knowledge representation. To facilitate understanding across these disciplines, we first define key terms, then we introduce the formalism for knowledge graphs, Kripke structures, and state enumeration identifying latent hazards (Ebrahimi et al., 2026). Finally, we cover delta debugging as a method for isolating minimal modifications to specifications.

2.1. *Terminology*

System. The real-world entity being analyzed (e.g., an automotive seatbelt system). A system comprises interacting components whose behavior is documented in specifications.

Ontology. A structured vocabulary defining domain concepts, their relationships, and governing constraints.

Knowledge Graph (KG). A graph representation instantiating an ontology with specific facts extracted from documentation about the system, where nodes are entities and edges are relations.

Atomic Proposition. A Boolean fact that cannot be decomposed, e.g., $\mathrm{isBuckled}(\mathrm{seatbelt}_1)$.

State. A collection of atomic propositions with definite truth values. We use "state" throughout; when other literature uses "world," interpret it as "state".

Kripke structure. A graph in which nodes represent system states and edges represent the addition of facts; the structure represents the feasible search space of all possible system designs.

Terminal state. A maximally determined state in which adding any new fact would create a contradiction. Terminal states have self-loops.

Model. A state that satisfies constraints of a knowledge graph; that is, a collection of atomic propositions that conform to the KG while respecting ontological axioms.

Property. A logical assertion corresponding to required system behavior or safety criteria.

Specification. A description of required system behavior, constraints, and properties. They may be incomplete, contradictory, or ambiguous.

Validation. Establishing whether specifications correctly and accurately capture intended behavior and stakeholder requirements.

Verification. Establishing whether a system satisfies specified properties through formal proof or exhaustive analysis.

Hazard. A state with potential to cause harm, damage, or loss to people, property, or the environment. Hazards may arise from unsafe configurations, design flaws, or operational conditions.

Risk. The combination of harm likelihood and severity, quantifying exposure to hazardous configurations reachable during system operation.

Assurance. The systematic process of providing justified confidence that a system operates safely under specified conditions, supported by evidence linking properties to requirements.

Repair. Local modifications to a KG to eliminate unsafe models when the KG covers both safe and unsafe terminal states. Repair operates within the existing KG structure.

Mitigation. Specification-level changes required when a KG violates safety and no repair is possible. Mitigation transforms unsafe terminal states into safe states by revising system specifications to reach configurations outside the current KG.

Refinement. The process of making specifications more precise and complete to eliminate ambiguity and unsafe configurations while preserving intended system behavior.

Delta Debugging. A technique for isolating minimal sets of facts whose modification transforms an unsafe terminal state into a safe one, identifying problematic specification fragments.

Weakest state (q_K). The state with minimal labeling that satisfies all facts in the knowledge graph K (that is, it fixes only those atomic propositions forced by the documentation and leaves remaining predicates undecided).

Open-World Assumption. The principle that unknown facts are not assumed false, allowing incomplete information without contradiction.

Closed-World Assumption. The principle that unknown facts are assumed false, necessitating assigning truth values to all relevant facts, enabling exhaustive verification.

2.2. *Systematic Risk Identification*

Our goal is to have a self-contained presentation without requiring the reader to refer to prior art.

Notation. $\mathbb{S}$, $\mathbb{P}$, and $\mathbb{O}$ denote finite sets of subjects, predicates, and objects that we extract from documentation. $\mathcal{C}$ is the set of ontological classes governing subjects and objects. $\mathcal{R}$ is the set of ontological relations governing predicates. We use $\mathcal{A}$ for the immutable ontology axioms governing the domain. Together these form an ontology $\mathcal{O} = \langle \mathcal{C}, \mathcal{R}, \mathcal{A} \rangle$. A triple $\langle s, p, o \rangle$ encodes a documented fact with subject $s \in \mathbb{S}$, predicate $p \in \mathbb{P}$, and object $o \in \mathbb{O}$. Each triple $\langle s, p, o \rangle$ has propositional notation $p(s, o)$ with Boolean valuations. $\mathbb{AP}$ is the set of atomic propositions constructed from such propositional notations. We use ϕ for propositional safety properties defined over $\mathbb{AP}$.

Knowledge Graph is a triple $K = \langle N, E, \ell \rangle$ where N is a set of nodes representing entities, $E \subseteq N \times N$ is a set of edges representing relations between entities, and $\ell : E \to \mathbb{P}$ labels each edge. We represent K as a propositional formula $\varphi(K)$ by taking the conjunction of all edge predicates:

$$\varphi(K) = \bigwedge \{ p(s, o) : \langle s, p, o \rangle \in E \}.$$

Kripke Structure is a finite-state model $M = \langle Q, I, R, L \rangle$, where Q is a finite set of feasible states, $I \subseteq Q$ is the set of initial states, $R \subseteq Q \times Q$ is a transition relation modeling state evolution, and $L : Q \to 2^{\mathbb{L}}$ labels each state with a set of literals. Here $\mathbb{L} = \mathbb{AP} \cup \{ \neg p \mid p \in \mathbb{AP} \}$ and $\mathbb{AP}$ denotes the set of atomic propositions introduced above. Finally, we write $L(q) \models \phi$ to mean the conjunction of literal set $L(q)$ logically ϕ; i.e., $L(q) \models \phi \iff \bigwedge L(q) \models \phi$.

Running Example 1. For a medical infusion pump, we extract the following from its documents.
Ontology. The ontology $\mathcal{O}$ defines classes $\mathcal{C}$, relations $\mathcal{R}$, and axioms $\mathcal{A}$ as follows:

$$\mathcal{C} = \{\text{Pump}, \text{IVLine}, \text{Battery}, \text{Alarm}\}$$
$$\mathcal{R} = \{\text{poweredOn}, \text{isConnected}, \text{lowBattery}, \text{alarmOn}, \text{isInfusing}\}$$
$$\mathcal{A} = \{\text{isInfusing}(x) \to \text{poweredOn}(x)\}$$

A1. *infusing requires power.*

Alphabets. Subjects, predicates, and objects are:

$$\mathbb{S} = \{\text{pump}_1, \text{ivLine}_1, \text{battery}_1, \text{alarm}_1\}$$
$$\mathbb{P} = \{\text{poweredOn}, \text{isConnected}, \text{alarmOn}, \text{lowBattery}, \text{isInfusing}\}$$
$$\mathbb{O} = \{\text{pump}_1, \text{ivLine}_1, \mathsf{true}, \mathsf{false}\}$$

RDF Triples. The triple $\langle\, \text{pump}_1,\ \text{isConnected},\ \text{ivLine}_1 \rangle$ asserts that entity pump_1 is connected to entity ivLine_1. The subject $\text{pump}_1 \in \mathbb{S}$, predicate $\text{isConnected} \in \mathbb{P}$, and object $\text{ivLine}_1 \in \mathbb{O}$ form a factual statement about the system configuration.
Triple-to-Predicate. We convert triples as follows:

- $\langle \text{pump}_1, \text{type}, \text{Pump} \rangle \mapsto \text{type}(\text{pump}_1, \text{Pump})$,
- $\langle \text{pump}_1, \text{poweredOn}, \mathsf{true} \rangle \mapsto \text{poweredOn}(\text{pump}_1)$,
- $\langle \text{pump}_1, \text{isConnected}, \text{ivLine}_1 \rangle \mapsto \text{isConnected}(\text{pump}_1, \text{ivLine}_1)$.

Atomic Propositions. The set of extracted atomic propositions for the pump system is:

$$\mathbb{AP} = \{\text{isConnected}(\text{pump}_1, \text{ivLine}_1), \text{poweredOn}(\text{pump}_1), \text{alarmOn}(\text{pump}_1), \text{lowBattery}(\text{pump}_1), \text{isInfusing}(\text{pump}_1)\}$$

Running Example 2. Given the documentation of the infusion pump system we extract a knowledge graph K:

$$K = \{\langle \text{pump}_1, \text{type}, \text{Pump} \rangle, \langle \text{ivLine}_1, \text{type}, \text{IVLine} \rangle, \langle \text{pump}_1, \text{poweredOn}, \mathsf{true} \rangle, \langle \text{pump}_1, \text{isConnected}, \text{ivLine}_1 \rangle\},$$

discarding type triples, formula representation is:

$$\varphi(K) = \text{poweredOn}(\text{pump}_1) \wedge \text{isConnected}(\text{pump}_1, \text{ivLine}_1)$$

State-Model Enumeration. Given an ontology $\mathcal{O}$ and atomic propositions $\mathbb{AP}$, we systematically enumerate feasible states consistent with ontological axioms $\mathcal{A}$ in a Kripke structure M defined over $\mathbb{AP}$. The construction follows:

(i) Initial states: I is a set of initial states q_0 with $L(q_0) = T_{\text{type}}$, where T_{type} is the set of fixed context predicates (for example, type predicates). Only mode-dependent propositions are undecided initially.

(ii) Systematic branching: in each state q, for every undecided predicate $p \in \mathbb{AP}$, create two successors: q^+ with $L(q^+) = L(q) \cup \{p\}$ and q^- with $L(q^-) = L(q) \cup \{\neg p\}$, and connect them via transition relation R.

(iii) Pruning and termination: check each successor against ontology axioms $\mathcal{A}$. Discard states whose literal set contradicts $\mathcal{A}$; retain consistent states. When no predicate can be added without violating $\mathcal{A}$, all propositions are decided, the state receives a self-loop in R forming a *terminal state*.

Terminal states. Because alphabets are finite and branching prunes violations of $\mathcal{A}$, M enumerates all feasible design choices consistent with the documentation and ontology culminating in a finite set of terminal states. We denote the set of all terminal states by $T = \{q \in Q : \langle q, q\rangle \in R\}$.

Terminal cover. The terminal cover represents the terminal states reachable from the weakest state q_K of the knowledge graph K. It answers the question: given the documentation, in which ways can the system be fully specified without contradicting the axioms? Each terminal state in this cover is a feasible design that respects every documented fact. Formally, the terminal cover T_K is the set of terminal states reachable from q_K:

$$T_K = \{q \in Q \mid \langle q, q\rangle \in R \wedge q_K \xrightarrow{*} q\}$$

When q_K encodes $\varphi(K)$ on the chosen alphabet, this reachability characterization coincides with the satisfiability characterization $T_K = \{q \in T : L(q) \models \varphi(K)\}$.

Running Example 3. Kripke State. A state q_1 in the Kripke structure corresponding to the infusion pump system might have labeling:

$$L(q_1) = \{\text{isConnected}(\text{pump}_1, \text{ivLine}_1), \text{poweredOn}(\text{pump}_1), \neg\text{alarmOn}(\text{pump}_1)\},$$

representing *"the pump is powered and connected to an IV line, but the alarm system is disabled"*.

Axiom-Driven Pruning. Denote terminal states in K by $T = \{q \in Q : \langle q, q\rangle \in R\}$. With 5 boolean predicates, we have $|T| = 2^5 = 32$. However, axiom (A1) prunes 8 invalid states where $\text{isInfusing}(\text{pump}_1) \wedge \neg\text{poweredOn}(\text{pump}_1)$, yielding $|T| = 24$ consistent terminals.

Terminal Cover. The terminal cover of knowledge graph K captures all terminal states in M reachable from q_K. In this example, that set is equivalent to the states satisfying $\varphi(K)$. Of note, $\varphi(K)$ only constrains $\text{poweredOn}(\text{pump}_1)$ and $\text{isConnected}(\text{pump}_1, \text{ivLine}_1)$ to be true, leaving the remaining three propositions in $\mathbb{AP}$ undecided, yielding $|T_K| = 8$ feasible completions.

If $|T_K| = 1$, the document fully determines a single state for the chosen alphabets. If the set is empty, K contains facts that contradict the immutable ontology $\mathcal{A}$. That is, K must be repaired to restore consistency before assurance proceeds.

Safety classification. Finally, each terminal state $q \in T_K$ is evaluated against every safety property ϕ to classify it as *safe* if $L(q) \models \phi$ or *unsafe* if $L(q) \not\models \phi$. A documentation is *safely realizable* if its Kripke M contains at least one safe terminal. A knowledge graph K is safely realizable wrt. a safety property ϕ if its cover T_K only contains safe terminals.

2.3. *Delta Debugging*

Delta debugging is a method for isolating minimal failure-inducing inputs (Zeller, 1999). Given a failing input that triggers a bug, delta debugging systematically minimizes the input while preserving the failure, ultimately yielding a minimal test case that still reproduces the issue. It operates by recursively partitioning the input into smaller subsets and testing each subset to see if it still triggers the failure. This process continues until no further reduction is possible without losing the failure-

inducing property. The method is particularly useful for isolating minimal changes needed to fix a problem, making it well-suited for our mitigation framework where we seek minimal modifications to the KG that eliminate unsafe terminals.

Running Example 4. Consider the safety property *"if infusing, alarm must be enabled and IV line connected"*, formally:

$$\phi_{\text{safe}} = \text{isInfusing}(\text{pump}_1) \rightarrow (\text{alarmOn}(\text{pump}_1) \wedge \text{isConnected}(\text{pump}_1, \text{ivLine}_1)),$$

A state $q \in T_K$ with labeling

$$L(q) = \{\text{isConnected}(\text{pump}_1, \text{ivLine}_1), \text{poweredOn}(\text{pump}_1), \text{isInfusing}(\text{pump}_1), \text{alarmOn}(\text{pump}_1), \ldots\}$$

satisfies this property because $L(q) \models \phi_{\text{safe}}$, while

$$L(q') = \{\text{isConnected}(\text{pump}_1, \text{ivLine}_1), \text{poweredOn}(\text{pump}_1), \text{isInfusing}(\text{pump}_1), \neg\text{alarmOn}(\text{pump}_1), \ldots\}$$

violates it because $L(q') \not\models \phi_{\text{safe}}$. Of note, q and q' differ only in the value of $\text{alarmOn}(\text{pump}_1)$.

3. Repair, Mitigation, & Unrealizability

The identification section provides a complete enumeration of unsafe terminal states in the terminal cover T_K of the documentation. The next step is mitigation: systematic updates of a KG or updates of an unrealizable safety property to exclude unsafe terminal states while maintaining traceability to documentation artifacts.

To address these above, we perform terminal diagnostics based on the following taxonomy:

Satisfaction. If all terminal states $q \in T_K$ satisfy the safety property ϕ, the documentation is complete with respect to ϕ and no action is required. The axioms $\mathcal{A}$, the safety property ϕ, and the knowledge graph K remain unchanged.

Repair. If the terminal cover T_K contains both safe and unsafe states, then the knowledge graph K is underspecified. Repair adds predicates to K to prune the terminal cover T_K by excluding unsafe terminal states while retaining at least one safe terminal state. The goal is a knowledge graph K' that adds the fewest literals while maximizing coverage of safe terminals and excluding unsafe ones, thereby preserving future flexibility.

Mitigation. Repair is impossible when every $q \in T_K$ violates the safety property ϕ and some $q' \in T$ satisfies ϕ. In such cases we apply mitigation to modify K and obtain a minimal modification K' that admits a safe terminal.

Unrealizability. The safety property ϕ is unrealizable if no terminal state in M satisfies ϕ (i.e., $\forall q \in T : L(q) \not\models \phi$). In this case, the property must be relaxed or revised so that at least one feasible state is safe; this change must be documented in the safety case and hazard-analysis logs.

Terminal diagnostics divide the problem into cases requiring fixing the KG and the safety property. However, following challenges arise:

Challenge 1: Minimality. Among many possible fixes, which is minimal (fewest changes)?

Challenge 2: Semantic validity. Not all fixes are valid, some violate axioms with cascade effects.

Challenge 3: Traceability. Mitigations should map back to documentation update clauses.

3.1. *Automated Repair and Mitigation*

Given a safety property ϕ, we partition the entire design space into safe and unsafe terminal states. Safe states are those satisfying the safety property; unsafe states are those violating it:

$$\mathcal{S} = \{q \in T : L(q) \models \phi\},$$
$$\mathcal{U} = \{q \in T : L(q) \not\models \phi\}.$$

We seek the fewest changes to K (the minimal modification K') that exclude unsafe states $\mathcal{U}$ while preserving maximal safe coverage. Two action types are considered: additions Δ_R, which introduce literals typically shared by safe terminal states in T_K to narrow the terminal cover (repair); and flips Δ_M, which change existing assertions in K to steer the KG toward safe regions of the design space (mitigation). Additions preserve existing design choices and incur cost α, while flips modify K and incur cost β with $\beta \gg \alpha$.

We write the modified KG as $K' = (K \cup A) \oplus F$, where $A \subseteq \Delta_R$ are chosen additions and $F \subseteq \Delta_M$ are chosen flips. The flip operator $\oplus$ updates predicates by removing negated counterparts and

Algorithm 1 COMPUTEADDITIONS$(\mathcal{S}, K)$

Ensure: Candidate additions Δ_R with coverage data
1: $\mathcal{S}_K \leftarrow \{q \in \mathcal{S} : L(q) \models \varphi(K)\}$
2: $\mathcal{U}_K \leftarrow \{q \in T_K : q \notin \mathcal{S}\}$
3: $core \leftarrow \bigcap_{q_s \in \mathcal{S}_K} L(q_s)$
4: $\Delta_R \leftarrow \emptyset$
5: **for** atom $p \in core$ **do**
6: **if** $p \notin K$ **and** $\exists q_h \in \mathcal{U}_K : p \notin L(q_h)$ **then**
7: $cover(p) \leftarrow \{q_h \in \mathcal{U}_K : p \notin L(q_h)\}$
8: $\Delta_R \leftarrow \Delta_R \cup \{(p, cover(p))\}$
9: $\Delta_R \leftarrow \{(p, c) \in \Delta_R : \nexists (p', c') \in \Delta_R, c' \subset c\}$
10: **return** Δ_R

adding the positive literals:

$$(K \cup A) \oplus F = ((K \cup A) \setminus \{\neg \ell : \ell \in F\}) \cup F\,.$$

Optimization formulation. With weights $0 < \alpha \ll \beta$ encoding preference for additions, the synthesis problem is

$$\min_{A \subseteq \Delta_R,\ F \subseteq \Delta_M} \alpha|A| + \beta|F| \quad \text{s.t.} \quad \emptyset \neq T_{K'} \subseteq \mathcal{S}.$$

The constraint requires that the resulting terminal cover $T_{K'}$ be non-empty and contained in the safe set $\mathcal{S}$.

Repair actions $\boldsymbol{\Delta_R}$. Algorithm 1 intersects safe terminal states in T_K to compute literals common to those terminals (line 3). Each shared literal p not already asserted in K that eliminates at least one unsafe terminal in T_K becomes a candidate addition (lines 4–8). Line 9 performs dominance filtering to remove redundant actions.

Mitigation actions $\boldsymbol{\Delta_M}$. Algorithm 2 computes the symmetric difference $q_h \triangle q_s$ for an unsafe state $q_h \in \mathcal{U}_K$ and a safe witness $q_s \in \mathcal{S}$; i.e., the set of literal disagreements between the two states. Delta-debugging minimization DDMIN searches for a smallest subset of candidate literals whose flips suffice to restore safety. VALIDATEONT$(F, \mathcal{A})$ discards flips that violate axioms $\mathcal{A}$, and PROJECTTOKG(F, K) restricts flips to predicates present in K, ensuring the modification is realizable as a KG update. Finally, flip sets with identical elimination coverage are merged (line 10).

Running Example 5. For safety property ϕ_{safe} the full design space pruned by axiom (A1) captured in M partitions as:

$$\mathcal{S} = \{q \in T : L(q) \models \phi_{\text{safe}}\} = 18 \text{ states}$$
$$\mathcal{U} = \{q \in T : L(q) \not\models \phi_{\text{safe}}\} = 6 \text{ states}$$

With $|T| = 24$ total consistent states (after axiom pruning), $6/24 = 25\%$ are unsafe.

Recall the K from the preliminaries; this KG yields $T_K \subset T$ with $|T_K| = 8$ completions. Two of these completions violate safety property ϕ_{safe}, those with $\text{isInfusing}(\text{pump}_1)$ and $\neg\text{alarmOn}(\text{pump}_1)$ in their labeling. This produces the same coverage gap $2/8 = 25\%$. Running Algorithms 1–2 on this KG and safety property results in:

$$K' = K \cup A = K \cup \{\text{alarmOn}(\text{pump}_1)\}\,,$$

deciding that alarm is enabled when the pump is powered and connected. With only lowBattery and isInfusing remaining undecided, the terminal cover shrinks from $|T_K| = 8$ to $|T_{K'}| = 4$ completions, all of which satisfy ϕ_{safe}.

Mitigation synthesis. Algorithm 3 constructs a coverage
matrix with rows corresponding to unsafe terminal states $\mathcal{U}_K = T_K \setminus \mathcal{S}$ and columns corresponding to candidate actions in $\Delta_R \cup \Delta_M$. The subroutine COVERAGEMATRIX$(\Delta_R, \Delta_M, \mathcal{U}_K)$ builds this matrix, where entry (i, j) indicates whether action j eliminates unsafe state i. The subroutine WEIGHTEDSETCOVER(C, α, β) solves a weighted set-cover problem (Vazirani, 2001) on C to find a minimal-cost set of actions that eliminates all unsafe states. The algorithm then verifies that the resulting knowledge graph K' has no

Algorithm 2 COMPUTEFLIPS$(\mathcal{S}, K, \mathcal{A})$

Ensure: Candidate flip sets Δ_M with coverage data
1: $\mathcal{U}_K \leftarrow \{q \in T_K : q \notin \mathcal{S}\}$
2: $\Delta_M \leftarrow \emptyset$
3: **for** unsafe terminal state $q_h \in \mathcal{U}_K$ **do**
4: **for** safe witness $q_s \in \mathcal{S}$ **do**
5: $F \leftarrow \text{DDMIN}(q_h \triangle q_s)$
6: $F \leftarrow$ VALIDATEONT$(F, \mathcal{A})$
7: $F \leftarrow$ PROJECTTOKG(F, K)
8: **if** $F \neq \emptyset$ **then**
9: $\Delta_M \leftarrow \Delta_M \cup \{(F, \{q_h\})\}$
10: $\Delta_M \leftarrow$ MERGECOVERAGE(Δ_M)
11: **return** Δ_M

Algorithm 3 Mitigation Synthesis

Require: M, K, ϕ, weights α, β
Ensure: Additions A, flips F, safe K'

$\mathcal{S} \leftarrow \{q \in T : L(q) \models \phi\}$
$\mathcal{U} \leftarrow \{q \in T : L(q) \not\models \phi\}$
$\mathcal{U}_K \leftarrow \{q \in T_K : q \notin \mathcal{S}\}$
if $\mathcal{U}_K = \emptyset$ **then**
  return $(\emptyset, \emptyset, K)$
if $\mathcal{S} = \emptyset$ **then**
  return UNREALIZABLE
$\Delta_R \leftarrow$ COMPUTEADDITIONS$(\mathcal{S}, K)$
$\Delta_M \leftarrow$ COMPUTEFLIPS$(\mathcal{S}, K, \mathcal{A})$
$C \leftarrow$ COVERAGEMATRIX$(\Delta_R, \Delta_M, \mathcal{U}_K)$
$(A, F) \leftarrow$ WEIGHTEDSETCOVER(C, α, β)
$K' \leftarrow (K \cup A) \oplus F$
if $T_{K'} \cap \mathcal{U} \neq \emptyset$ **then**
  return FAIL
return (A, F, K')

unsafe completions in the full design space; it returns (A, F, K') on success and FAIL otherwise. If $\mathcal{S} = \emptyset$ (no safe states exist in T), the algorithm returns UNREALIZABLE earlier.

Traceability. Each action corresponds to a predicate in the KG and requires documentation justification. We record: (i) the unsafe states $\mathcal{U}_K$ in T_K before mitigation; (ii) the selected actions A and F; (iii) the verification that all terminal states of K' are safe (i.e., $T_{K'} \subseteq \mathcal{S}$); and (iv) updated documentation clauses (requirements for additions and design decisions for flips). If no valid mitigation exists, the case escalates to unrealizability analysis via Algorithm 4.

3.2. *Automated Unrealizability*

When Algorithm 3 returns UNREALIZABLE, the property itself must be relaxed. We search for the weakest relaxation ϕ^* (i.e., the least restrictive formula implied by ϕ) such that $\phi \Rightarrow \phi^*$ and at least one state satisfies ϕ^*.

Relaxation proceeds by atom dropping: removing atomic propositions from ϕ until realizability is achieved. Let $\mathcal{AP}(\phi)$ denote the set of atoms appearing in ϕ after conversion to negation-normal form. Define DROPATOM(ϕ, a) to remove atom a by substituting positive occurrences with $\top$ and negative occurrences with $\bot$, then simplifying. This operation weakens the formula: $\phi \Rightarrow$ DROPATOM(ϕ, a) because every model of ϕ remains a model after dropping constraints on a.

Algorithm 4 RELAXPROPERTY(M, ϕ)

Require: M, ϕ
Ensure: Realizable relaxation ϕ^* or FAIL

$Q \leftarrow [\langle \phi, \emptyset \rangle]$
$seen \leftarrow \{\mathrm{CNF}(\phi)\}$
while $Q \neq \emptyset$ **do**
  $\langle \phi^*, R \rangle \leftarrow$ DEQUEUE(Q)
  if $\exists q \in T : L(q) \models \phi^*$ **then**
    return ϕ^*
  for $a \in \mathcal{AP}(\phi^*) \setminus R$ **do**
    $\psi \leftarrow$ DROPATOM(ϕ^*, a)
    if $\mathrm{CNF}(\psi) \notin seen$ **then**
      $seen \leftarrow seen \cup \{\mathrm{CNF}(\psi)\}$
      $Q \leftarrow Q \cup \langle \psi, R \cup \{a\} \rangle$
return FAIL

Algorithm 4 explores relaxations with a breadth-first search over atom-dropping operations. Starting from ϕ, each unrealizable candidate spawns children by dropping a single additional atom. The set R records atoms already dropped to avoid redundant exploration, and CNF-based duplicate detection prevents cycles. BFS guarantees that the first realizable formula found minimizes the number of dropped atoms.

Traceability. We log: (i) original safety property ϕ, (ii) dropped atoms yielding ϕ^*, (iii) witness terminal satisfying ϕ^*, (iv) updated safety case documentation justifying the relaxation. If Algorithm 4 returns FAIL, no relaxation via atom dropping achieves realizability; the system alphabets or mission-level requirements must be revised.

4. Automotive Seatbelt Warning System

To demonstrate how identified hazards are mitigated in practice, we continue a case study on a simplified seatbelt-warning system from (Ebrahimi et al., 2026), which exposed latent hazards via Kripke enumeration.

4.1. *System overview and ontology*

The seatbelt-warning system model used in our experiments contains the following elements.
Entities: Vehicle (ignition, seat, alarm); Seatbelt (buckle mechanism); Person (driver, occupant).
Structural (time-invariant) relations: Relations

that do not vary across operational modes, e.g., entity containment and attachment (has).

Mode-dependent relations: Relations that vary with operational modes. We focus on seven predicates:

1. isOn: Engine running
2. isOccupied: weight ≥ 3 kg detected.
3. isFastened: chest and lap restrained.
4. isBuckled: belt latched (sensor reading).
5. isActive: warning active (auditory and visual).
6. speedExceeds: speed > 20 km/h.
7. speedLimit: speed limiter engaged.

The documentation for the system specifies:

> *The alarm activates if and only if the ignition is on, the seat is occupied, and the seatbelt is not buckled. When these conditions hold, the speed limiter must engage.*

The formula of the knowledge graph K is:

$$\begin{aligned}\varphi(K) =&[\text{isActive} \iff (\text{isOn} \wedge \\ &\quad \text{isOccupied} \wedge \neg\text{isBuckled})] \\ &\wedge[(\text{isOn} \wedge \text{isOccupied} \wedge \neg\text{isBuckled}) \\ &\quad \implies \text{speedLimit}]\,.\end{aligned}$$

For more details on K, see (Ebrahimi et al., 2026).

Fastened Seatbelt. The system includes a seatbelt with a buckle. The buckle can be buckled (latched) but not fastened (securely restraining the occupant). This distinction is critical for safety: a buckled but unfastened seatbelt may fail to restrain the occupant during a crash, while a fastened seatbelt provides proper restraint. However, documentation often conflates these states, leading to latent hazards where the system may incorrectly assume safety when the seatbelt is only buckled but not fastened.

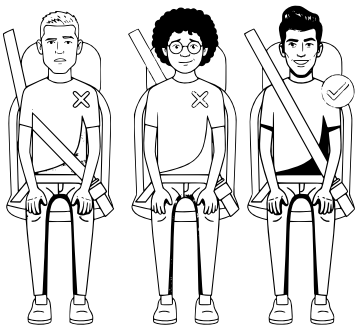

Fig. 1. A fastened seatbelt restrains the upper body.

Ontological Axioms. Four axioms constrain the state space. The core axiom reflects reality:

A1. *If a person is properly wearing the seatbelt for restraint, then the buckle must be latched.*
A2. *Belt cannot be fastened with no occupant.*
A3. *Buckle cannot be latched with no occupant.*
A4. *Speed cannot exceed 20 km/h if ignition off.*

With all four axioms, the state space of the Kripke structure M for knowledge graph K reduces from 2^7 terminals to 48 (Ebrahimi et al., 2026).

Hazard Patterns. Automotive safety standards (FMVSS 208, Euro NCAP protocols) mandate occupant restraint systems to prevent unrestrained vehicle operation. The requirement has two complementary aspects: *detection* (warning system activates precisely when hazard conditions exist) and *prevention* (speed limiting engages to mitigate unrestrained high-speed operation). Ebrahimi et al. (2026) specified the following two hazard patterns to expose latent hazards in the system:

$$\phi_{\text{fasten}} = \text{isBuckled} \wedge \neg\text{isFastened}$$
$$\phi_{\text{speed}} = \text{speedLimit} \wedge \text{speedExceeds}$$

These are *hazard patterns* (undesirable states), not safety properties. We define the safety property as

$$\phi_{\text{safe}} = \neg\phi_{\text{fasten}} \wedge \neg\phi_{\text{speed}}.$$

Safe terminal states are those that satisfy ϕ_{safe}.

Latent Hazard Identification. Ebrahimi et al. (2026) identified latent hazards in the seatbelt system through exhaustive Kripke enumeration. With all four axioms (A1–A4), the design space contains 48 terminals; of which, 22 terminals satisfy the documented specification $\varphi(K)$ However, verification against hazard patterns reveals that documented-safe terminals harbor latent hazards:

Misuse Hazard: Five terminals satisfy $\varphi(K)$ but also satisfy the hazard pattern ϕ_{fasten}. These represent scenarios where the buckle sensor reports latched ($\text{isBuckled} = \texttt{true}$) but the person is

not properly restrained ($\mathrm{isFastened} = \mathsf{false}$); e.g., belt buckled behind back or under arm. The system believes it is safe (alarm off, speed unrestricted) but the occupant lacks crash protection. This hazard arises from conflating observable sensor state ($\mathrm{isBuckled}$) with unobservable safety state ($\mathrm{isFastened}$).

Enforcement Hazard: Three terminals satisfy $\varphi(K)$ but also satisfy the hazard pattern ϕ_{speed}. These represent dangerous transitions where the speed limiter is engaged ($\mathrm{speedLimit} = \mathsf{true}$) while the vehicle travels at high speed ($\mathrm{speedExceeds} = \mathsf{true}$). While low-speed engagement of speed limiter safely prevents acceleration, high-speed engagement causes abrupt deceleration leading to loss of control and multi-vehicle collisions. This hazard exposes the dangerous transition path toward compliance.

One terminal exhibits both hazards. Coverage analysis shows that 40.9% (9 of 22) of documented-safe terminals harbor latent hazards. This result demonstrates that relying solely on documentation, without verification, overlooks systematic risks.

4.2. *Mitigation Analysis*

To eliminate the nine latent hazards while preserving the 13 truly safe terminal states, we define safe and unsafe sets using the safety property ϕ_{safe}:

$$\mathcal{S} = \{q \in T : L(q) \models \phi_{\text{safe}}\}$$
$$\mathcal{U}_K = \{q \in T_K \setminus \mathcal{S}\}$$

Here $|\mathcal{S}| = 13$ (truly safe) and $|\mathcal{U}_K| = 9$ (latent hazards), of which 6 are misuse-hazards and 4 are enforcement-hazards (one terminal exhibits both). We apply Algorithms 1–3 to find minimal modifications to K that exclude $\mathcal{U}_K$ while retaining $\mathcal{S}$.

Additions. Intersecting the literals of truly-safe terminal states reveals no predicate holding universally; thus, $core = \emptyset$. Operational diversity (engine on/off, occupied/empty, various belt configurations) prevents any single predicate from characterizing all safe scenarios. Therefore, repair via additions alone is insufficient: $\Delta_R = \emptyset$.

Flips. Algorithm 2 computes minimal predicate flips redirecting each latent-hazard in $\mathcal{U}_K$ toward safe witnesses in $\mathcal{S}$. Delta debugging identifies:

- Flipping $\mathrm{isFastened}$ to true eliminates six misuse hazards (instances of ϕ_{fasten}).
- Flipping $\mathrm{speedExceeds}$ to false eliminates four enforcement hazards (instances of ϕ_{speed}).
- Flipping $\{\mathrm{isFastened}, \mathrm{speedExceeds}\}$ covers all nine latent hazards in $\mathcal{U}_K$.

Algorithm 3 with weights $\alpha = 1$, $\beta = 10$ selects this pair at cost $2\beta = 20$ as the minimal solution achieving complete coverage.

Causality. Examining predicate control reveals:

- System-actuated predicates are $\mathrm{speedLimit}$ and $\mathrm{isActive}$. These are directly controlled by the system design (speed limiter, alarm).
- Occupant-actuated predicates are $\mathrm{isBuckled}$, $\mathrm{isFastened}$, isOn, and $\mathrm{speedExceeds}$ that depend on occupant actions (buckle engagement, proper fastening, ignition, driving behavior).

The optimal solution requires flipping $\mathrm{isFastened}$ and $\mathrm{speedExceeds}$, both of which are occupant-actuated predicates, because:

- Although $\mathrm{isFastened}$ is unobservable without invasive sensors (e.g., chest-strap, pressure mat), it is directly controlled by the occupant.
- $\mathrm{speedExceeds}$ is a sensor reading and is not directly controlled by the system, rather it reflects occupant driving behavior.

From this observation, the expert infers that these latent hazards trace to user compliance rather than to system-design deficiencies. This shifts the role of the expert from inquisitive subjective judgment (*"How is this user error or system failure?"*) to objective analysis of algorithm outputs. Automating this distinction, by tagging predicates as controllable inputs versus observable outputs, is analogous to reactive-synthesis approaches for open systems and remains future work.

Alternative Flips. Beyond the optimal solution, delta debugging discovers additional flip actions. Table 1 summarizes all risk management actions, their coverage, and their applicability. The viable alternatives (flipping $\mathrm{isBuckled}$ or $\mathrm{speedLimit}$) cover four terminals (44%) occurring exclusively

Table 1. Synthesized risk management actions; we distinguished non-operational contexts occurring when $\neg\text{isOn} \vee \neg\text{isOccupied}$ from operational contexts when $\text{isOn} \wedge \text{isOccupied}$.

Action	Type	Coverage	Context	Realizable
FLIP $\{\text{isFastened}\}$	Occupant	6	Mixed	No
FLIP $\{\text{speedExceeds}\}$	Occupant	4	Mixed	No
FLIP $\{\text{isBuckled}\}$	Occupant	2	Non-operational	Yes
FLIP $\{\text{speedLimit}\}$	System	2	Non-operational	Yes
Viable combined	**Mixed**	**4 (44%)**	$\neg\text{isOn} \vee \neg\text{isOccupied}$	**Yes**
Optimal (unrealizable)	**Occupant**	**9 (100%)**	**All contexts**	**No**

in non-operational contexts (ignition off OR seat unoccupied). The five remaining terminals (56%) occur in operational scenarios ($\text{isOn} \wedge \text{isOccupied}$) and require user compliance.

Design Implications. The coverage gap reveals fundamental limitations of feasible risk-management actions. Only two of nine latent hazards (22%) are addressable via system-level actions (modifying when speedLimit engages in non-operational contexts); the remaining seven (78%) trace to occupant-actuated predicates. Even the two addressable hazards still depend on occupant compliance, since the system cannot physically latch or unlatch the buckle. The five operational hazards likewise require occupant actions (proper fastening, speed compliance) outside system control.

Converting occupant-actuated predicates into system-observable signals requires installing additional sensors (for example, chest-strap monitors or pressure mats) to detect proper restraint beyond buckle latching. Corresponding specification updates could require restraint verification prior to high-speed operation, for example: *"Restraint detection shall use redundant sensors measuring both buckle engagement and proper chest and lap restraint."* If sensor investment is infeasible, the safety case must explicitly assign occupant responsibility for proper fastening and speed compliance, and residual risks should be mitigated via training, warnings, and user documentation.

Traceability. The analysis propagates through certification artifacts. The hazard log records 9 latent hazards (6 misuse, 4 enforcement, 1 shared) with root causes traced to occupant-actuated predicates through expert observation of delta debugging solutions. Coverage analysis documents that only 22% (2 terminals) are addressable via true system-level actions (speedLimit modifications in a non-operational context), while 78% (7 terminals) require occupant compliance. Design decisions acknowledge that specification refinement alone cannot eliminate these occupant-dependent hazards even with sensor upgrades to make isFastened observable. The safety case documents explicit accountability assignments: system responsibility for speed limiter control, occupant responsibility for proper belt fastening and speed compliance in operational scenarios, with residual risks accepted and mitigated via warnings and user documentation.

5. Related Work

This section positions our method within existing KG validation and formal-verification research, and clarifies the concrete value it adds to the present study: moving from hazard identification to minimal, auditable mitigation synthesis. Existing knowledge graph validation tools (Baader, 2003) operate under OWA, so they flag explicit inconsistencies but miss hazards implied by incompleteness (Ji et al., 2022; Shi and Weninger, 2017). Embedding-based completion methods (Ji et al., 2022) predict missing triples statistically, yet they do not provide safety guarantees or traceability to documentation updates. We complement these lines by enumerating all feasible states under CWA and by coupling validation to mitigation actions with explicit documentation traceability.

Building on Ebrahimi et al. (2026)'s systematic

enumeration of feasible states in a Kripke model, we extend the analysis from hazard detection to risk management actions. We formalize repairs as additions and mitigations as flips, derive minimal flip sets via delta debugging, and synthesize a weakest safe completion using a weighted hitting-set formulation. This bridges model exploration with specification-level change, which is not addressed in prior KG validation work.

Formal methods, e.g., model checking (Clarke et al., 2018), can verify properties against formal specifications, typically at the code or model level. Our framework targets documentation-derived KGs where incompleteness is the primary hazard source, and it preserves auditability by mapping each mitigation action to a concrete documentation clause. This aligns with safety certification practice where evidence must link hazards, mitigations, and design decisions. Accordingly, Section 3 provides the formal synthesis machinery and Section 4 demonstrates its practical effect on a realistic seatbelt scenario, establishing the logical role of related work in motivating and validating our contribution.

6. Conclusion

Incomplete documentation creates implicit hazards in safety-critical systems because OWA-based validators only detect explicit violations. We address this gap by enumerating all feasible states, classifying safe and unsafe terminal states, and synthesizing the weakest safe completion of the KG via a weighted hitting-set formulation over repair additions and mitigation flips. This yields actionable changes that remain traceable to documentation artifacts.

Our taxonomy (satisfaction, repair, mitigation, unrealizability) provides a decision framework for assurance distinguishing the scope of change; i.e., system-level repair/mitigation versus property-level relaxation with distinct traceability requirements for each regime. Delta debugging supplies minimal flip sets, ontology validation preserves domain constraints, and the global check ensures that the modified KG excludes unsafe completions in the design space.

The case study demonstrates practical feasibility under the full axiom set: 48 terminal states are enumerated (22 safe, 26 unsafe), repair additions are empty, and mitigation selects a minimal two-flip solution that removes all latent hazards while preserving design flexibility and traceability.

In future work, we will extend the framework to handle temporal behavior and classes of temporal safety properties, develop compositional analysis for hierarchical system decomposition, support continuous documentation evolution, and automate generation of traceability links from mitigation actions to documentation artifacts.

Acknowledgement

This work was supported by the ASSURE project (MDU multi-disciplinary research initiative) and in part by the ITEA4 24026 CLEAR (Comprehensive Learning for Enhanced AI Responsiveness) project.

References

Baader, F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press.

Clarke, E. M., O. Grumberg, D. Kroening, D. A. Peled, and H. Veith (2018). *Model Checking* (2nd ed.). Cambridge, MA, USA: MIT Press.

Ebrahimi, M., K. Hänninen, K. Lundqvist, and M. Sirjani (2026). What we know we don't know: Systematic risk identification from contradictory safety documentation. Submitted to ESREL 2026.

Ehrlinger, L. and W. Wöß (2016). Towards a definition of knowledge graphs. *SEMANTiCS 48*(1-4), 2.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*(2), 199–220.

Ji, S., S. Pan, E. Cambria, P. Marttinen, and P. S. Yu (2022). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems 33*(2), 494–514.

Shi, B. and T. Weninger (2017). Open-world knowledge graph completion.

Vazirani, V. V. (2001). *Approximation algorithms*, Volume 1. Springer.

Zeller, A. (1999). Yesterday, my program worked. today, it does not. why? In *Software Engineering — ESEC/FSE '99*, Volume 1687 of *Lecture Notes in Computer Science*, pp. 253–267. Springer.