Proc. of the European Safety and Reliability Conference (ESREL 2026)
Edited by José C. Matos, Paulo B. Lourenço, Michael Beer, Daniel Oliveira,
Edoardo Patelli, Hélder S. Sousa, Rui A. Silva and Monica Santamaria
 Published by Research Publishing, Singapore.
doi: 10.3850/ESREL2026061419_esrel26-p27730-cd

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 (KGs) extracted from system documentation is challenging because documentation is often incomplete or ambiguous. Existing validation under open-world reasoning misses hazards implied by missing information and does not provide a systematic path from hazard identification to minimal, traceable mitigation. We use Kripke structures to formalize KG semantics by enumerating feasible states and classifying outcomes 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 synthesize a weakest safe completion that excludes unsafe terminal states while preserving maximal safe design freedom. The paper contributes a decision taxonomy for post-identification action, an optimization-based synthesis of additions and flips for minimal mitigation, and an unrealizability procedure that guides controlled safety-property relaxation with explicit traceability to documentation and safety artifacts. For unrealizable specifications, the procedure explicitly escalates from model-level correction to stakeholder-level action, assigning follow-up decisions to the appropriate actors (e.g., safety engineer, system designer, and user/operator) through safety-case traceability. In an automotive seatbelt case study, the framework identifies latent hazards, synthesizes minimal mitigation actions, and separates system-actuated from occupant-actuated causes, thereby improving the practical value of the analysis for assurance and certification.

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

1. Introduction

Knowledge Graphs (KGs) encode system documentation as directed labeled graphs. Each node represents an entity (a component, state, or concept), and each edge encodes a factual statement or relationship. Formally, a triple $\langle s, p, o \rangle$ asserts that subject s has predicate p with value or target o (Ehrlinger and Wöß, 2016; Ji et al., 2022). Ontology axioms and domain constraints govern which triples are valid (Gruber, 1993; Baader, 2003). Yet incompleteness or ambiguity in documentation propagates directly into the encoded KG (Ji et al., 2022; Shi and Weninger, 2017). Reasoners operating under the Open World Assumption (OWA) flag only explicit violations. This leaves latent hazards undiscovered (Baader, 2003; Shi and Weninger, 2017). Meanwhile, knowing all possible subjects, predicates, and objects enables systematic exploration of every design that the system documentation can realize. Prior work uses Kripke structures for this enumeration under Closed World Assumption (CWA), exposing unsafe states implicit in documentation (Ebrahimi et al., 2026). Identification is only a first step for assurance, because certification evidence eventually demands risk management actions that rule out the hazards entirely.

This paper develops a mitigation pipeline that begins with that Kripke semantics but pursues the question: *given the catalog of safe and unsafe states, which changes prevent the entire unsafe region while preserving valid design intent?* We distinguish two types of changes:

System-level changes targeting a KG by modifying its triples, thereby altering which state models the KG realizes. System-level changes trace to system documentation (requirements, design decisions, operational constraints).

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

olation 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

Here, we cover the formalism for understanding the workflow from (Ebrahimi et al., 2026). 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 $\mathbb{AP}$.

2.1. *Knowledge Graphs*

This 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:

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

Semantics. A *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 $\mathbb{L} = \mathbb{AP} \cup \{\neg p \mid p \in \mathbb{AP}\}$. An atomic proposition $ap \in \mathbb{AP}$ is true in state q if $ap \in L(q)$, false if $\neg ap \in L(q)$, and undecided if neither appears in $L(q)$. Finally, we write $L(q) \models \phi$ to mean the conjunction of literal set $L(q)$ logically ϕ; i.e., $L(q) \models \phi \iff \bigwedge L(q) \models \phi$.

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 $L(q_0) = \emptyset$ (all propositions undecided).

(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 all terminal states consistent with the KG's formula $\varphi(K)$. It answers the question: given the system documentation, what are all the ways the system could be fully specified? 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 all terminal states q such that every fact asserted in K is true in q:

$$T_K = \{q \in Q : \langle q, q \rangle \in R \wedge L(q) \models \varphi(K)\}$$

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 KG K is safely realizable wrt. a safety property ϕ if its cover T_K only contains safe terminals.

2.2. *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.

3. Repair, Mitigation, & Unrealizability

The previous section provides a complete identification of all unsafe completions of the documentation. The next step is mitigation: systematically updating a KG or an unrealizable safety property to eliminate unsafe outcomes while maintaining traceability and behavioral validity.

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

Satisfaction. If all terminal states $q \in T_K$ satisfy safety property ϕ, the documentation is complete and safe. No action is needed; axioms $\mathcal{A}$, safety property ϕ, and KG K remain unchanged.

Repair. If terminal cover T_K contains both safe and unsafe states, then KG K is underspecified. Repair adds missing predicates to K to prune its cover of unsafe terminal states while retaining at least one safe terminal state. The goal is a KG K' that adds the fewest literals while covering the maximal number of safe terminals and excluding unsafe ones, preserving future flexibility.

Mitigation. Repair cannot succeed if all states $q \in T_K$ violate safety property ϕ, yet there is a state $q' \in T$ satisfying ϕ. We apply mitigation to modify K to obtain a minimal modification K'.

Unrealizability. Safety property ϕ is unrealizable and system-level changes cannot help if no terminal state exists in M that satisfies ϕ; i.e., $\forall q \in T : L(q) \not\models \phi$. The solution is to relax or revise safety property ϕ so that at least one feasible state becomes safe. This must be documented in safety case arguments 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 fewest changes to K (the minimal modification K') excluding unsafe states $\mathcal{U}$ while preserving maximal safe coverage. We distinguish two action types: (i) *additions* Δ_R add literals shared by safe terminal states currently covered by K to narrow the terminal cover (repair), and (ii) *flips* Δ_M change existing assertions in K to redirect it toward safe regions of the design space (mitigation). Additions preserve valid design choices (cost α); flips alter K (cost $\beta \gg \alpha$).

We define the modified KG using $K' = (K \cup A) \oplus F$, where $A \subseteq \Delta_R$ are chosen additions and $F \subseteq \Delta_M$ are chosen flips. The flip operator $\oplus$ updates predicates: for each literal $\ell \in F$, it removes ℓ's negation from $(K \cup A)$ and adds ℓ:

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

Optimization formulation. Assigning weights $0 < \alpha \ll \beta$ to encode preference for additions over flips, we solve:

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

The constraint ensures K' avoids unsafe terminal states in $\mathcal{U}$ while retaining at least one safe state.

Repair actions Δ_R. Algorithm 1 intersects safe terminal states covered by K to find literals shared

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

amongst them (line 3). Each shared literal p not already in K that excludes at least one unsafe terminal state in T_K becomes a candidate addition (lines 4–8). Line 9 removes redundant actions.

Mitigation actions $\boldsymbol{\Delta_M}$. Algorithm 2 computes the symmetric difference $q_h \triangle q_s$ for unsafe state $q_h \in \mathcal{U}_K$ and safe witness $q_s \in \mathcal{S}$; i.e., the set of literal disagreements between the two states. Delta debugging minimization DDMIN finds a smallest subset of these literals whose flips are sufficient to restore safety. VALIDATEONT$(F, \mathcal{A})$ removes flips from F violating the axioms $\mathcal{A}$. PROJECTTOKG(F, K) restricts flips to predicates that appear in K, ensuring the result is a valid modification of K. Finally, we merge flip sets with identical coverage (line 10).

Mitigation synthesis. Algorithm 3 constructs a coverage matrix where rows are unsafe terminal states $\mathcal{U}_K = T_K \setminus \mathcal{S}$, and columns are actions in $\Delta_R \cup \Delta_M$. COVERAGEMATRIX$(\Delta_R, \Delta_M, \mathcal{U}_K)$ builds this matrix with entry (i, j) indicating whether action j eliminates unsafe state i. WEIGHTEDSETCOVER(C, α, β) solves a weighted

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: $F \leftarrow$ 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'
1: $\mathcal{S} \leftarrow \{q \in T : L(q) \models \phi\}$
2: $\mathcal{U} \leftarrow \{q \in T : L(q) \not\models \phi\}$
3: $\mathcal{U}_K \leftarrow \{q \in T_K : q \notin \mathcal{S}\}$
4: **if** $\mathcal{U}_K = \emptyset$ **then**
5: **return** $(\emptyset, \emptyset, K)$
6: **if** $\mathcal{S} = \emptyset$ **then**
7: **return** UNREALIZABLE
8: $\Delta_R \leftarrow$ COMPUTEADDITIONS$(\mathcal{S}, K)$
9: $\Delta_M \leftarrow$ COMPUTEFLIPS$(\mathcal{S}, K, \mathcal{A})$
10: $C \leftarrow$ COVERAGEMATRIX$(\Delta_R, \Delta_M, \mathcal{U}_K)$
11: $(A, F) \leftarrow$ WEIGHTEDSETCOVER(C, α, β)
12: $K' \leftarrow (K \cup A) \oplus F$
13: **if** $T_{K'} \cap \mathcal{U} \neq \emptyset$ **then**
14: **return** FAIL
15: **return** (A, F, K')

set cover problem (Vazirani, 2001) over matrix C with weights (α, β) to find minimal-cost actions eliminating all unsafe states. The algorithm then checks that the modified KG K' has no unsafe completions in the full design space. It returns (A, F, K') on success, otherwise FAIL; UNREALIZABLE is returned earlier when $\mathcal{S} = \emptyset$.

Traceability. Each action corresponds to a predicate in the KG requiring documentation justification. We log: (i) unsafe states $\mathcal{U}_K$ in T_K before mitigation, (ii) selected actions A and F, (iii) verifying that all terminal states of K' are safe; i.e., $T_{K'} \subseteq \mathcal{S}$, (iv) updated documentation clauses (requirements for additions, 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 strongest (least permissive) relaxation 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 re-

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

mains a model after dropping constraints on a.

Algorithm 4 explores relaxations via breadth-first search. Starting from ϕ, each unrealizable candidate spawns children by dropping one additional atom. The set R tracks already-dropped atoms to avoid redundant exploration; CNF-based duplicate detection prevents cycles. BFS guarantees the first realizable formula found is minimal.

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 exemplify how exposed hazards are systematically mitigated by our mitigation framework, we continue a case study on a simplified seatbelt system from (Ebrahimi et al., 2026) that identified latent hazards via Kripke enumeration.

4.1. *System Overview and Ontology*

The seatbelt warning system comprises:
Entities: Vehicle (ignition, seat, alarm), Seatbelt (buckle mechanism), Person (driver/occupant)
Structural (time-invariant): Relations encoding entity containment and attachment; e.g., "has".
Mode-dependent: Relations varying across operational modes. We focus on seven such 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 (auditory & visual)
6. speedExceeds: Speed > 20 km/h
7. speedLimit: Speed limiter engaged

The system's documentation 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 KG K is:

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

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

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 Kripke M corresponding to KG 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:

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

These are *hazard patterns* (undesirable states), not safety properties; safe terminal states are those that do *not* satisfy these patterns.

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 ($\text{isBuckled} = \mathsf{true}$) but the person is not properly restrained ($\text{isFastened} = \mathsf{false}$); 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 (isBuckled) with unobservable safety state (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 ($\text{speedLimit} = \mathsf{true}$) while the vehicle travels at high speed ($\text{speedExceeds} = \mathsf{true}$). 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. The coverage analysis quantified that 40.9% (9 of 22) terminals documented-safe harbor latent hazards. This demonstrates that relying solely on documentation without verification disregards systematic risks.

4.2. *Mitigation Analysis*

To eliminate 9 latent-hazards while preserving the 13 truly-safe terminal states, we define safe and unsafe sets. Since ϕ_{fasten} and ϕ_{speed} are hazard patterns (not safety properties), safe states are those that do *not* satisfy these patterns:

$$\mathcal{S} = \{q \in T : L(q) \not\models \phi_{\text{fasten}} \wedge L(q) \not\models \phi_{\text{speed}}\}$$

$$\mathcal{U}_K = \{q \in T_K \setminus \mathcal{S}\}$$

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 isFastened to true eliminates 6 misuse-hazards (satisfying ϕ_{fasten})
- Flipping speedExceeds to false eliminates 4 enforcement-hazards (satisfying ϕ_{speed})
- Flipping $\{\text{isFastened}, \text{speedExceeds}\}$ covers all 9 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 speedLimit and isActive. These are directly controlled by the system design (speed limiter, alarm).
- Occupant-actuated predicates are isBuckled, isFastened, isOn, and speedExceeds that depend on occupant actions (buckle engagement, proper fastening, ignition, driving behavior).

The optimal solution requires flipping isFastened and speedExceeds, both of which are occupant-actuated predicates, because:

- Although isFastened is unobservable without invasive sensors (e.g., chest-strap, pressure mat), it is directly controlled by the occupant.
- 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 system design deficiencies. This shifts the expert's role 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 for future work.

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**

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 isBuckled or speedLimit) cover four terminals (44%) occurring exclusively 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 in feasible risk management actions. Only 2 of 9 latent hazards (22%) are addressable through true system-level actions: modifying when speedLimit engages in non-operational contexts. The remaining 7 hazards (78%) trace to occupant-actuated predicates. Even the 2 hazards addressable by flipping isBuckled in non-operational contexts still require occupant compliance, since the system cannot latch or unlatch the buckle. The 5 operational hazards similarly demand occupant actions (proper fastening, speed compliance) that lie beyond system control.

Converting occupant-actuated predicates to system-observable requires installing chest-strap monitors and pressure mats to detect proper restraint beyond buckle latching. Specification updates would enforce restraint verification before high-speed operation: *"Restraint detection shall use redundant sensors measuring both buckle engagement and proper chest and lap restraint,"* and *"Speed shall not exceed 20 km/h when occupant restraint is unconfirmed, enforced through throttle limiting preventing high-speed operation."* Where sensor investment proves infeasible, the safety case must explicitly assign occupant responsibilities for proper fastening and speed compliance, with residual risks mitigated through 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 extend the framework to include temporal behavior and corresponding safety properties class, compositional analysis for hierarchical system decomposition, continuous documentation evolution, and automated traceability link generation 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.