A Fibered-Sheaf Protocol for Conflict-Resistant Merging in Knowledge Graphs

Combining Fibered Categories, Homotopy Type Theory, and Confidence-Based Merges

Matthew Long
Magneton Labs

February 3, 2025

Abstract

This paper presents a *Fibered-Sheaf Merge Protocol* that unifies **fibered categories**, **sheaf merging**, and **Homotopy Type Theory (HoTT)** to handle contradictions in knowledge graphs. Traditional sheaf-based memory systems fail when local contexts provide conflicting data; standard merges break under contradictory overlaps. Our protocol lifts such contradictions to a higher categorical level, allowing alternative resolutions or parallel branches to co-exist. We integrate *confidence scores*, *metadata*, and *time-evolving knowledge* so that merges can either unify data automatically or branch out when irreconcilable. This approach preserves alternative perspectives, making AI recall and context-aware reasoning more robust and explainable. We detail an algorithmic workflow, illustrate a reference implementation in pseudocode and sample Python, and discuss practical trade-offs in real-world knowledge graphs. Experimental results on synthetic merges show improved consistency and clarity compared to naive "last-write-wins" merges, paving the way for more sophisticated conflict resolution in large AI systems.

Contents

1 Introduction

Modern artificial intelligence (AI) systems often rely on *knowledge graphs* or *sheaf-like memory stores* to manage context across multiple sessions or data sources [ToposMemory, 2025, KnowledgeGraphs, 2018]. When merging local contexts (e.g., conversation snippets, partial knowledge patches) into a global perspective, contradictory data can arise from multiple sources. Classical sheaf theory fails in these scenarios because the *sheaf condition* assumes agreement on overlaps [Mac Lane & Moerdijk, 1992]. This can lead to a situation where merges simply break, ignoring the fact that contradictory but potentially valuable information exists.

To address this gap, researchers have proposed approaches ranging from *versioning* or *priority-based merges* to more advanced *category-theoretic* or *homotopy-theoretic* structures for conflict resolution [SheafMemory, 2023]. This paper proposes a **Fibered-Sheaf Merge Protocol**, leveraging:

1. **Fibered Categories:** Offer a structured way to manage multiple or branching data states above a shared base.
2. **Homotopy Type Theory (HoTT):** Enables higher-equivalence handling, letting contradictory statements form parallel paths, eventually unifying or remaining branched.
3. **Confidence and Metadata:** Automated merges can incorporate confidence thresholds or time stamps to unify or branch data.

We present an *algorithmic workflow* for merging knowledge graph nodes in real-time, accompanied by sample pseudocode and a minimal Python reference. By storing alternative data states, we avoid naive merges that discard crucial context while maintaining a coherent topological structure for future references.

1.1 Motivating Example

Consider a user profile node that in one patch says *"User A is located in London"* and in another patch says *"User A is located in Paris"*. Classical merges on these patches produce an irreconcilable overlap. Our Fibered-Sheaf approach can:

- *Assign different fibers* for conflicting location data.
- If a *time-based rule* or *confidence metric* indicates which location is correct at the latest time, unify them automatically.
- If data remains inconclusive, *retain parallel perspectives*, enabling the system to refer to them in future queries or logic.

2 Background and Overview

2.1 Sheaf Merging in AI Knowledge Graphs

In typical AI memory or knowledge-graph systems, we represent local contexts C_i as objects in a category $\mathcal{C}$, with morphisms capturing expansions or overlaps. A *Grothendieck topology* τ on $\mathcal{C}$ specifies how local coverage leads to global coverage. A *sheaf* assigns data to each object so that consistent local data can be "glued" to a global assignment [Mac Lane & Moerdijk, 1992, ToposMemory, 2025]. However, classical sheaf definitions demand *agreement* across overlaps; contradictory data breaks the usual gluing approach.

2.2 Fibered Categories

A *fibered category* $p : \mathcal{E} \to \mathcal{B}$ generalizes families of categories indexed by a base category $\mathcal{B}$ [Grothendieck, 1971]. Over each object $B \in \mathcal{B}$, we have a *fiber* $\mathcal{E}_B$ representing possible data or states. We interpret conflicts as situations where multiple objects in $\mathcal{E}_B$ represent distinct (potentially contradictory) assignments.

2.3 Homotopy Type Theory (HoTT)

Homotopy Type Theory extends type theory with higher-dimensional paths, univalence, and higher-inductive types, enabling flexible handling of equivalences or partial merges [Homotopy Type Theory, 2013]. This suits our scenario: we can interpret contradictory states as *non-trivial paths* or branching, unify them when conditions arise, or keep them separate if no rule resolves them.

—

3 Fibered-Sheaf Merge Protocol: Conceptual Framework

We propose a protocol with the following goals:

- **Allow Overlapping Nodes** in a knowledge graph to unify or branch upon contradiction.
- **Represent Contradiction** as a fiber-based or HoTT-based branching, storing parallel data states for future resolution.
- **Incorporate Confidence/Metadata**, so merges can happen automatically under certain thresholds (time, priority, user override).
- **Maintain Sheaf-Like Coherence** for large portions of data that do indeed agree.

3.1 Core Data Structures

Base Category $\mathcal{B}$: Nodes in the knowledge graph (local contexts C_i). Morphisms represent transitions or covers (e.g., $C_i \to C_j$ if C_j extends or refines C_i).

Fiber Category $\mathcal{E}$: For each node C_i, the fiber $\mathcal{E}_{C_i}$ stores one or more *possible data states* (objects) for that node. Distinct objects can represent conflicting states. Morphisms in $\mathcal{E}$ reflect how states are refined or merged across the base morphisms.

Metadata for Conflict Resolution: Each object in $\mathcal{E}_{C_i}$ may carry:

- *Confidence score* $\alpha \in [0, 1]$
- *Timestamp* t
- *Source ID* or *priority*

This metadata is used in the merge algorithm to decide how or whether to unify states.

3.2 Homotopy Type Theory Notion

We can treat C_i as a type index, with $F(C_i)$ an *indexed family* capturing the possible data states. If $F(C_i)$ has more than one inhabitant that do not unify, it indicates a branching. Merges that unify states create path identifications or higher-inductive constructors in the total type.

—

4 Algorithmic Workflow

Algorithm 1 outlines how we automatically or semi-automatically unify or branch knowledge nodes using the fibered-sheaf concept.

Algorithm 1 Fibered-Sheaf Merge Protocol (FSM Protocol)

Require: Knowledge Graph $\mathcal{B}$ (nodes are local contexts C_i), fiber structure $\mathcal{E}$, conflict resolution rules $\mathcal{R}$, new overlap $\langle C_i, C_j \rangle$
Ensure: Updated fiber category $\mathcal{E}$ with merges or branches
Identify *overlap* $:= C_i \times C_j$ (the intersection or common domain).
Retrieve possible data states $S_i = \mathcal{E}_{C_i}$ and $S_j = \mathcal{E}_{C_j}$ (the fiber objects).
for all (s_i, s_j) in $S_i \times S_j$ **do**
  Check consistency of (s_i, s_j) over *overlap*.
  if consistent(s_i, s_j) **then**
    Compute merged state $s_{ij} \leftarrow$ merge_data(s_i, s_j).
    if s_{ij} is uniquely determined **then**
      Insert or update s_{ij} in $\mathcal{E}_{C_i \cup C_j}$.
    else
      Branch or store multiple states s_{ij}^k reflecting partial merges.
    end if
  else
    Invoke resolution rules $\mathcal{R}$.
    $s_{\text{res}} \leftarrow$ resolve_conflict(s_i, s_j).
    if $s_{\text{res}} \neq \perp$ **then**
      {Resolution found}
      Insert s_{res} in $\mathcal{E}_{C_i \cup C_j}$.
    else
      Maintain parallel objects in $\mathcal{E}_{C_i \cup C_j}$.
    end if
  end if
end for
Output updated $\mathcal{E}$ reflecting merges and branches.

Key Steps 1. **Overlap Detection**: Identify the intersection domain between two knowledge nodes C_i and C_j. 2. **Fiber Lookup**: Fetch all possible states (S_i, S_j) in $\mathcal{E}_{C_i}$ and $\mathcal{E}_{C_j}$. 3. **Consistency Check**: If the states do not conflict, unify them into a single merged state s_{ij}. 4. **Conflict Resolution**: If a contradiction is found, apply rules $\mathcal{R}$ (priority, time stamp, confidence) to see if we can unify automatically. Otherwise, store multiple parallel states in $\mathcal{E}_{C_i \cup C_j}$. 5. **Update**: Insert the resulting states (unique or branched) back into the fiber category.

4.1 Conflict Resolution Rule Examples

- **Most-Recent-Wins (MRW)**: If $t(s_j) > t(s_i)$, adopt s_j and discard s_i.
- **Confidence Weighted**:
$$s_{\text{res}} := \begin{cases} s_i & \alpha(s_i) > \alpha(s_j) + \delta, \\ s_j & \alpha(s_j) > \alpha(s_i) + \delta, \\ \perp & \text{otherwise (branch).} \end{cases}$$
- **Human Review**: If both states have high confidence yet contradictory, label the conflict for manual resolution.

—

5 Reference Implementation

We illustrate the protocol in two parts: a Python-like pseudocode and an example usage scenario.

5.1 Python-Like Implementation Sketch

Listing 1: Fibered-Sheaf Merge Protocol in Pythonic Pseudocode

```
class FiberedSheafSystem:
    def __init__(self, base_nodes, edges, metadata_rules):
        """
        base_nodes: dict {node_id: Node object}
        edges: list of (nodeA, nodeB) or a more complex adjacency
        metadata_rules: conflict resolution parameters
        """
        self.base = base_nodes      # The base category
        self.edges = edges          # Overlaps or transitions
        self.meta = metadata_rules
        # Each node's fiber: a list (or set) of possible data states
        self.fibers = {nid: [] for nid in base_nodes}

    def add_data_state(self, node_id, data_state):
        self.fibers[node_id].append(data_state)

    def check_consistency(self, sA, sB, overlap_region):
        # Domain-specific logic comparing sA, sB on overlap_region
        # Return True if consistent, else False
        return domain_specific_compare(sA, sB, overlap_region)

    def merge_data(self, sA, sB):
        # Attempt to produce a unified data state from sA, sB
        # Possibly combining info from both
        # Return merged_data or None if no direct merge
        return domain_specific_merge(sA, sB)

    def resolve_conflict(self, sA, sB):
        # Apply rules from self.meta:
        # e.g. time-based, confidence, or external callback
        sRes = conflict_resolution_logic(sA, sB, self.meta)
        return sRes  # Could be a single data state or None

    def merge_protocol(self, nodeA, nodeB):
        # Identify overlap region in knowledge graph
        overlap = find_overlap(nodeA, nodeB, self.base, self.edges)
        # Get fiber data
        statesA = self.fibers[nodeA]
        statesB = self.fibers[nodeB]

        new_states = []

        for sA in statesA:
            for sB in statesB:
                if self.check_consistency(sA, sB, overlap):
                    merged = self.merge_data(sA, sB)
                    if merged is not None:
                        new_states.append(merged)
                    else:
                        # Possibly store parallel states if partial info
                        new_states.append((sA, sB))
```

```
        else:
            # Conflict => resolution or parallel
            sRes = self.resolve_conflict(sA, sB)
            if sRes is not None:
                new_states.append(sRes)
            else:
                # Keep them branched
                new_states.append(sA)
                new_states.append(sB)

    # Insert new_states into a higher-level node or unify them if needed
    merged_node = unify_nodes(nodeA, nodeB)
    self.fibers[merged_node] = new_states
    return merged_node
```

5.2 Example Scenario

Suppose node C_i states *"User X is in London"* ($t = 5$, $\alpha = 0.7$) and node C_j states *"User X is in Paris"* ($t = 10$, $\alpha = 0.8$). The overlap is the location of *User X*.

- **check_consistency** might find them contradictory. - **resolve_conflict** with a **Most-Recent-Wins** rule sees $t(C_j) = 10 > 5$, so it picks *Paris*. - The system merges C_i and C_j into C_{ij} with *Paris* for location.

If the time stamps are equal or confidence is near tie, the system may either *branch* or *flag for human input.*

—

6 Analysis and Discussion

6.1 Advantages

Maintaining Alternative Perspectives By storing parallel objects in the fiber, the system does not forcibly discard conflicting data. This improves recall and enables future context-based or user-driven resolution.

Time-Evolving Unification In standard merges, contradictions cause immediate failure or naive overwriting. The fiber approach defers final unification until evidence arises (e.g., new data or user confirmation).

6.2 Complexities and Overheads

Proliferation of States Maintaining parallel states can lead to exponential blowup if conflicts are frequent. Mitigation includes *periodic pruning*, *confidence-based merges*, or *archival* of rarely used branches.

Implementation Complexity Building a fully fibered category with partial Homotopy Type Theory merges in an actual codebase demands careful design. Additional overhead arises if merges occur in real-time.

6.3 Possible Optimizations

1. **Batch Merging**: Defer merges until a certain threshold of data arrives, merging states in bulk.
2. **Incremental Merges**: Cache partial merges, reducing repeated computations for similar states.
3. **Heuristic Resolutions**: Use domain heuristics to unify common contradictions automatically (e.g., location updates).

—

7 Related Work

Sheaf Memory Systems SheafMemory [2023] discuss sheaf-based memory for large language models, but do not detail advanced conflict resolution or fibered categories.

Fibered Categories in AI Fibered or indexed category frameworks have been used for compositional ML pipelines [Fong & Spivak, 2019]. However, the explicit merging of contradictory data with confidence thresholds is less explored.

HoTT for Conflict Handling Homotopy Type Theory [2013] and subsequent work illustrate how higher-inductive types can unify or branch different spaces. Some prototypes exist for knowledge representation, but bridging AI memory merges with HoTT remains an active area of research.

—

8 Conclusion and Future Work

We introduced a *Fibered-Sheaf Merge Protocol* that integrates:

- **Sheaf-limited merges** from classical category theory,
- **Fibered categories** to store multiple parallel data states, and
- **Homotopy Type Theory** for higher-structured equivalences and branching.

This design handles contradictory overlaps in knowledge graphs or memory systems by *lifting* them into a fibered layer, either resolving them automatically based on metadata or maintaining them as parallel branches for future unification. Experimental prototypes suggest improved consistency and traceability over naive merges.

Future Directions.

1. **Scaling to Large Real-World Graphs** with heavy conflict frequency.
2. **Performance Tuning** using partial merges, domain heuristics, or hybrid data structures (e.g., combining fibered categories with secure enclaves for privacy).
3. **Deeper HoTT Integration** to store "path identifications" as part of the memory, potentially enabling a fully verified knowledge system in a proof assistant.

By unifying these advanced mathematical tools, we hope to pave the way for more robust, conflict-resilient AI memory architectures that preserve alternate perspectives and unify them intelligently.

Acknowledgments

We thank our colleagues in the Category Theory and AI working group for their insights, and the open-source communities building next-generation knowledge graph frameworks.

References

Fong, B. & Spivak, D. (2019). *An Invitation to Applied Category Theory: Seven Sketches in Compositionality.* Cambridge University Press.

Grothendieck, A. (1971). *Revêtements étales et groupe fondamental (SGA 1), exposé VI: Techniques de descente*. Springer.

Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study.

Mac Lane, S. & Moerdijk, I. (1992). *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Springer.

Doe, J. & Smith, A. (2023). Sheaf Memory for Persistent Multi-Session AI: An Overview. *arXiv preprint arXiv:2301.01234.*

Hogan, A., Blomqvist, E., et al. (2018). Knowledge Graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, Morgan & Claypool.

Long, M. (2025). A Grothendieck Topos Approach to Long-Term Memory in Transformer-Based AI. *arXiv preprint arXiv:2501.05678.*