Traversing Data in Symbolic Systems

Typed-Relation Traversal as a First-Class Retrieval Primitive

Perslis Research

2026-09-15

Abstract

In a vector-store system you *retrieve* by similarity: return the top-k items whose embeddings are nearest a query. In a symbolic system you *traverse*: you walk named, typed, directed edges between concepts, and the walk itself is the answer's justification. This paper treats **typed-relation traversal** as a first-class retrieval primitive and gives it a formal account. We model a typed knowledge graph — concepts joined by directed edges that carry a relation type (with attributes: *inverse*, *transitive*, *functional*) and a confidence — and define traversal as bounded, confidence-decayed composition along those edges. We prove four properties. **Theorem 1 (Conservativity):** traversal never returns a fact outside the transitive closure of the asserted edges — it cannot fabricate. **Theorem 2 (Termination):** with a per-hop decay factor $\gamma < 1$ and a confidence floor θ, every path has length at most $\lfloor \log_\gamma(\theta/c_0) \rfloor$, so traversal halts. **Theorem 3 (Path-as-proof):** every returned fact carries a path of asserted edges that re-verifies it in time linear in the path length — traversal yields checkable explanations; similarity retrieval does not. **Theorem 4 (Functional uniqueness):** along a functional relation, traversal yields at most one successor or detects a contradiction. We validate all four in a seed-reproducible simulation (0 fabrications in 44,853 derived facts; max path length equals the proved bound; 100% of facts re-verified; 50/50 functional contradictions caught). We connect traversal to a companion account of memory in which relevance is not embedding proximity but *reachability*, and locate it as the answer primitive beneath a deployed symbolic knowledge graph of tens of thousands of typed relations over hundreds of thousands of edges.

1 Introduction

Two systems are asked the same question, and they do two categorically different things to answer it.

A vector store **retrieves**: it embeds the query, computes similarity against stored embeddings, and returns the top-k nearest items. The operation is a nearest-neighbor lookup; the result is an unordered set; there is no direction, no composition, and no account of *why* an item was returned beyond "it was close."

A symbolic system **traverses**: it starts at a concept and walks *named, typed, directed* edges to other concepts — *this causes that*, *this is part of that*, *this means that* — composing edges into

paths. The result is not a bag of nearby items but a set of reached facts, each accompanied by the path that reached it. The path is not metadata; it is the *derivation*, and it is checkable.

This paper is about that second operation. Traversal is often treated as an implementation detail — "we use a graph database" — but it is a retrieval primitive with its own semantics and its own guarantees, and those guarantees are exactly the ones a similarity index cannot offer: it cannot fabricate a fact outside what was asserted (§4); it terminates by construction (§5); every answer carries a proof (§6); and it respects the logical constraints of the relations it walks (§7). We make each precise and prove it, then validate it numerically, then ground it in the scale of a real deployment.

2 The typed knowledge graph

We model knowledge as a directed graph $G = (V, E)$.

- V is a set of **concepts**.
- E is a set of **typed directed edges** (s, r, o, c): a subject concept s, a **relation type** r, an object concept o, and a **confidence** $c \in (0, 1]$. The asserted edges are $E_0 \subseteq E$.
- Each relation type r carries attributes: an **inverse** r^{-1} (so (s, r, o) entails (o, r^{-1}, s)); a **transitive** flag (if set, (a, r, b) and (b, r, c) entail (a, r, c)); and a **functional** flag (if set, a subject may bear r to at most one object — the OWL 2 `FunctionalProperty` sense [1]).

Typing is what distinguishes this from a plain graph: an edge does not merely connect two concepts, it *names the relationship*, and that name carries logical force (direction, composability, cardinality). A traversal that follows *causes* edges is answering a different question from one that follows *part-of* edges, and the system knows the difference.

3 Traversal semantics

A **traversal query** from a seed concept s_0 produces the facts reachable from s_0 under a chosen relation, subject to three bounds that keep it sound and finite:

1. **Composition** — along a *transitive* relation, edges compose: a path $s_0 \to a \to \cdots \to o$ yields the derived fact (s_0, r, o).
2. **Confidence decay with a floor** — a derived fact's confidence is the product of its edge confidences, and traversal abandons a path once confidence falls below a floor θ. With a per-hop decay factor $\gamma \in (0, 1)$ and initial confidence c_0, confidence after k hops is $c_0\gamma^k$.
3. **Known endpoints** — traversal only visits concepts in V; it cannot invent a concept that was never asserted.

Two further rules govern soundness at the relation level: a step along a *functional* relation yields at most one successor (§7), and a step that would assert a value contradicting an existing functional value raises a **contradiction** rather than proceeding.

The result of a traversal is a set of **derived facts**, each paired with the **path** — the sequence of asserted edges — that produced it.

4 Theorem 1 — Conservativity (no fabrication)

Let $\mathrm{Cl}(E_0)$ denote the transitive closure of the asserted edges under the declared transitive relations (the set of all facts entailed by composing asserted edges).

Theorem 1. *Every fact returned by traversal is in* $\mathrm{Cl}(E_0)$*. Traversal never returns a fact that is not entailed by the asserted edges and the declared transitivity.*

Proof. By induction on the length k of the path witnessing a returned fact. *Base* ($k = 1$): the fact is an asserted edge, $(s, r, o) \in E_0 \subseteq \mathrm{Cl}(E_0)$. *Step*: a returned fact of path length $k+1$ is formed by composing a returned fact of path length k — in $\mathrm{Cl}(E_0)$ by hypothesis — with one further asserted edge, along a relation declared transitive. By the definition of transitive closure, the composition of a closure fact with an asserted edge under a transitive relation is again in $\mathrm{Cl}(E_0)$. Traversal introduces facts only by these two moves (§3), so every returned fact lies in $\mathrm{Cl}(E_0)$. □

Conservativity is the traversal analogue of *no hallucination*: the operation can surface only what the asserted graph already entails. A similarity index has no such property — it returns whatever is near, and nearness is not entailment.

5 Theorem 2 — Termination (depth bound)

Theorem 2. *With per-hop decay* $\gamma \in (0, 1)$*, floor* θ*, and initial confidence* c_0*, every traversal path has length at most* $\lfloor \log_\gamma(\theta/c_0) \rfloor$*. On a finite graph, traversal halts and visits finitely many nodes.*

Proof. A path of length k has confidence $c_0\gamma^k$, and traversal continues only while $c_0\gamma^k \geq \theta$. Taking $\log_\gamma$ (and noting $\log_\gamma$ reverses inequalities because $\gamma < 1$) gives $k \leq \log_\gamma(\theta/c_0)$; as k is an integer, $k \leq \lfloor \log_\gamma(\theta/c_0) \rfloor$. The number of paths of bounded length in a finite graph is finite, so traversal terminates. □

The confidence floor is therefore not only an epistemic device (stop trusting a long, decayed chain) but a *termination guarantee*: it converts "how far should I search?" into a closed-form depth bound. (Illustratively, $\gamma = 0.75$, $\theta = 0.4$, $c_0 = 1$ give a bound of 3 hops; the deployed constants are part of the internal specification.)

6 Theorem 3 — Path-as-proof (verifiability)

Theorem 3. *Every fact returned by traversal carries a path* $p = (e_1, \ldots, e_k)$ *of asserted edges, and the fact re-verifies in* $O(k)$ *time by checking (i) each* $e_i \in E_0$ *and (ii) that consecutive edges chain under the transitive relation. Hence traversal answers are independently checkable.*

Proof. The path is exactly the derivation the traversal followed. Re-checking each e_i against E_0 confirms every hop is asserted; checking e_i.object $= e_{i+1}$.subject confirms the composition is well-formed; together these witness membership in $\mathrm{Cl}(E_0)$ (Theorem 1). There are k hops, each checked in constant time, so verification is $O(k)$. □

This is the property that most sharply separates traversal from similarity retrieval. A cosine-nearest result answers *"here is something close"* and offers nothing to check. A traversal result answers *"here is a fact, and here is the chain of asserted edges that entails it"* — the answer arrives with its own proof, at a verification cost linear in the proof's length. For a system whose downstream consumers must justify what they act on, this is not a convenience; it is the difference between an assertion and an argument.

7 Theorem 4 — Functional uniqueness and contradiction

Theorem 4. *Let r be a functional relation. In a consistent store, any traversal step along r from a subject s yields at most one successor. If the asserted edges contain two distinct objects $o_1 \neq o_2$ with $(s, r, o_1), (s, r, o_2) \in E_0$, traversal detects a contradiction rather than branching.*

Proof. A functional relation constrains $|\{o : (s, r, o) \in E_0\}| \leq 1$. If the store satisfies the constraint, the successor set of s under r has at most one element, so the step yields at most one successor. If it contains two distinct objects, the functional constraint is violated by definition, which is precisely the contradiction condition; traversal reports it. □

Functional relations thus give traversal two things at once: a **branching bound** (functional chains never fan out) and a **consistency check** (a functional relation is a place where the graph can *catch itself* in a contradiction). Both are properties of the *typing*, not of the data volume — they hold at any scale.

8 Relevance is reachability, not proximity

Traversal reframes the question a companion account of memory raises — *which facts are relevant to the task despite poor embedding similarity?* [2] — and answers it structurally.

Proposition 1. *Define the* traversal-relevance *of a fact to a query concept as its bounded, typed reachability from that concept. Traversal-relevance is orthogonal to embedding proximity: a fact can be highly traversal-relevant yet embedding-distant (reached in one* causes*-hop but sharing few tokens with the query), and embedding-near yet traversal-irrelevant (lexically similar but unreachable by any typed path).*

The proposition is established by construction: the two orderings are computed from different objects — one from graph structure, one from vector geometry — and neither refines the other. Its consequence is the point: for causal, procedural, and part-whole questions, the relevant fact is the one you can *reach along the right kind of edge*, and traversal finds it exactly where similarity, ranking by proximity, discards it. Direction matters, too: *causes* and *caused-by* are different edges, and a system that only measures proximity cannot tell a cause from its effect.

9 Numerical validation

We validate the four theorems in a seed-reproducible simulation (`validate_traversal.py`, seed `20260915`): a typed graph of $N = 4{,}000$ concepts with 20,000 asserted edges of a transitive relation, plus a functional relation with 50 injected contradictions; illustrative $\gamma = 0.75$, $\theta = 0.4$, $c_0 = 1$ (depth bound 3); traversal run from 300 seeds.

Every derived fact was entailed, every path re-checked, the depth bound was tight, and every functional conflict was caught — the four guarantees hold exactly as proved.

10 Scale in a real deployment

The account above is not asymptotic hand-waving; the traversal primitive runs, today, over a symbolic knowledge graph at nontrivial scale. In the deployed system behind a companion paper on offline symbolic AI [3], the store holds on the order of **65,000 typed relation instances**

Property	Result	Status
T1 Conservativity	0 fabrications across **44,853** derived facts (all in the closure)	PASS
T2 Termination	max observed path length **3** = proved bound $\lfloor \log_{0.75}(0.4) \rfloor = 3$	PASS
T3 Path-as-proof	**44,853 / 44,853** facts re-verified by re-walking the path	PASS
T4 Functional uniqueness	**50 / 50** contradictions detected; **950 / 950** consistent subjects yielded a unique successor	PASS

across **35 relation types** (three of them functional) over roughly **427,000 graph edges** — and answers are produced by exactly the bounded, confidence-decayed, path-carrying traversal analyzed here, not by embedding retrieval. The theorems are what make that safe at scale: conservativity bounds what can be returned, termination bounds the work, and path-as-proof makes every answer auditable regardless of how large the graph grows.

11 Relation to the collection

Traversal is the **answer primitive** of the symbolic layer the rest of a companion line of work relies on. *Peel* [3] supplies the typed graph and the functional-relation vetoes; this paper is the formal account of how that graph is *walked* to produce an answer. *Retrieval Is Not Memory* [2] argues relevance is not proximity; §8 makes that concrete as reachability. *Verified Before Acting* [4] uses a symbolic oracle for bounded inference; traversal, with Theorems 1–2, is that bounded, sound inference. *The Orchestration Gap* [5] requires provenance on every fact that shapes behavior; Theorem 3 is provenance by construction — the path is the provenance. The through-line: in a symbolic system, to *find* something and to *justify* it are the same act, because the path you took to reach a fact is the reason the fact holds.

12 Limitations

- **Guarantees are relative to the asserted graph and declared relation attributes.** Conservativity means traversal returns nothing beyond the closure; it does not certify that the asserted edges are *true* (a false asserted edge yields a false-but-well-derived fact — source correctness is a separate obligation).
- **The transitive-closure model assumes the transitivity declaration is correct.** Declaring a non-transitive relation transitive would let composition over-reach; the guarantee is *conditional on* correct relation typing.
- **The simulation uses a DAG and illustrative constants.** Cyclic graphs are handled by visited-set traversal (still terminating by Theorem 2's confidence bound), but the reported closure statistics are for the acyclic case; the deployed confidence constants are withheld (moat scrub).
- **This is analysis plus a controlled validation, not a benchmark** against graph-database or SPARQL engines; the contribution is the guarantees, not a performance comparison.

13 Conclusion

A vector store answers by proximity and cannot tell you why. A symbolic system answers by traversal — walking typed, directed, composable edges — and the walk is the reason. We have shown that this primitive is conservative (it cannot fabricate), terminating (a confidence floor is a depth bound), self-justifying (every answer carries a linear-time-checkable proof), and consistency-aware (functional relations bound branching and catch contradictions), and that these hold at the scale of a real deployment. Retrieval finds what is near. Traversal finds what follows — and shows its work.

References

Verified against primary sources on 2026-09-15.

[1] W3C. "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)." W3C Recommendation. `owl:FunctionalProperty`, transitive and inverse properties. https://www.w3.org/TR/owl2-syntax/

[2] Perslis Research. "Retrieval Is Not Memory: Memory as a Governance Function over Experience." Preprint, 2026. https://research.perslis.com/memory.html

[3] Perslis Research. "Peel: Structural Hallucination Prevention for Offline AAC Through Symbolic Fact Authorship." Preprint, 2026. https://research.perslis.com/peel.html

[4] Perslis Research. "Verified Before Acting: A Pre-Action Adversarial Cognition Loop with Factored Authorization." Preprint, 2026. https://research.perslis.com/adversarial-loop.html

[5] Perslis Research. "The Orchestration Gap: Why Model-Level Alignment Cannot Survive Multi-Model Runtimes." Preprint, 2026. https://research.perslis.com/orchestration-gap.html

PREPRINT / DRAFT — Perslis Research, 2026-09-15. Moat-scrubbed public version; deployment mechanism internals withheld. `validate_traversal.py` *reproduces §9 from seed 20260915.*