CatDB: A Categorical Semantic Model for Executable Database Schemas

Matthew Long
The YonedaAI Collaboration
YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com · https://yonedaai.com

22 July 2026

Abstract

Database integration often stores meaning outside the database, in migration scripts, extract, transform, and load jobs, application adapters, and local conventions. Those artifacts are hard to compose and easy to misread after a schema changes. We study a narrower question than whether category theory can replace them: which categorical structures can serve as explicit, checkable objects in an executable schema layer?

We define a finite CatDB profile with entities, total functional arrows, typed attributes, finite paths, parallel path equations, finite total instances, and total schema mappings. The presentation that an author writes is not the small category that it presents, nor is it the normalized runtime record proposed for execution. We keep those three objects separate. Under stated assumptions, we give paper-level proofs for typed path laws, closure and category laws for lawful mappings, and the validity and contravariant composition laws of pullback migration. A running example with customers, orders, and accounts shows a nontrivial equation, well-typed and ill-typed paths, a failed mapping obligation, and a composable mapping chain.

CatDB's candidate contribution is operational, not foundational. It consists of immutable semantic records, stable diagnostics, obligations classified by both meaning and discharge site, and explicit preservation and loss at backend boundaries. These remain ENGINEERING HYPOTHESIS. Schemas as categories, instances as functors, functorial data migration, algebraic integration, and executable categorical query languages are SUPPORTED BY PRIOR LITERATURE. The semantic-core boundary now has limited executable support. Independent Rust and Haskell evaluators agree on all 16 admitted schema, path, and mapping fixtures; eight later-slice envelopes remain structural only. A Lean 4 library machine-checks nine declarations at the exact `PATH-02`–`PATH-04` and `MAP-03`–`MAP-05` boundaries. Nothing here validates finite instances, runs a data migration or backend, or reports an experiment or benchmark. We do not claim that the profile subsumes relational, entity-relationship, property-graph, RDF/OWL, document, vector, or multimodel semantics.

Contents

1 Introduction

1.1 Plain-language problem

Suppose a commerce team stores customers and orders in one system, billing accounts in another, and analytical summaries in a third. The systems use names for roughly the same concepts, but transformation code carries the relationships among them. One job joins an

order to its customer. Another selects the billing account. The analytical model recreates both decisions. After a schema change, checking that a column moved is the easy part. The team must also determine whether a relationship changed meaning, whether its constraint still holds, and which downstream results remain justified.

CatDB starts from a simple design preference: represent that meaning as an artifact that people and programs can type, compose, version, and inspect. In the profile studied here, an order-to-customer relationship is not an untyped configuration string. It has a declared source and target. A longer relationship is an ordered traversal, so every intermediate endpoint must match. If two traversals should agree, the schema says so. A mapping to another schema assigns every declared entity, relationship, and attribute, then accounts for every source law.

Physical data may still need to move, and some transformations lose information. Many database constraints are not path equations. Federation still has latency, availability, and consistency costs. Graph, ontology, document, and vector systems keep their native semantics when a categorical view is added. CatDB therefore does not promise to eliminate integration work. Its narrower claim is that typed, composable, versioned schema mappings can carry some meaning that bespoke pipelines carry today, while federation and materialization remain execution choices.

1.2 Research question

The research question for this paper is:

> *Which categorical structures have useful, checkable operational meaning for executable database schemas?*

The word *checkable* needs care. Finite path typing is a different problem from deciding arbitrary equality in a finitely presented category. Checking a mapping assignment is total is different from proving that it preserves every equation. Evaluating a law on one finite database snapshot is different from proving it for every future state. Compiling a law into a foreign key or validation query is different from proving a categorical theorem. This paper treats those distinctions as part of the model rather than as implementation detail.

1.3 Contributions and evidence status

The paper makes four kinds of statements, with explicit labels.

1. SUPPORTED BY PRIOR LITERATURE identifies results or system capabilities already established by primary literature or normative standards.
2. PROVED IN THIS PAPER identifies a complete mathematical argument in this paper under listed assumptions. It is not a machine-checked claim.
3. ENGINEERING HYPOTHESIS identifies a proposed operational design that requires implementation and evaluation.
4. OPEN CONJECTURE identifies a mathematical or engineering question for which this paper supplies no complete answer.

Within that discipline, the paper contributes:

- a bounded profile that explicitly separates the category presented by a schema, the finite presentation that a user authors, a validated normalized representation, and a physical realization;
- formal typing judgments for paths, composition, equations, mappings, instances, and pullback migration;
- paper-level proofs of the elementary path, mapping, and pullback laws;
- bounded Rust/Haskell computational conformance for the admitted schema, path, and mapping operations, including deterministic positive and negative fixtures;
- Lean machine checking for the exact path and lawful-mapping laws named in the published coverage table;
- a proposed two-axis obligation model separating what must hold from where evidence may be produced;
- a feature-level comparison that lists preservation and residuals rather than claiming universal subsumption; and
- a falsifiable reproducibility contract for the future Rust, Haskell, Lean, and shared-fixture artifacts.

The first three contributions are definitions and paper mathematics. The fourth is bounded *computational* evidence, not proof. The fifth is machine-checked only at its named theorem boundaries. The obligation, comparison, and broader operational claims remain ENGINEERING HYPOTHESIS. No proposition, fixture result, property test, or Lean declaration in this paper establishes backend equivalence or a performance result.

1.4 Paper organization

Section 2 locates the work in categorical database theory and adjacent data models. Section 3 defines the finite profile. Section 4 develops the running commerce example. Section 5 specifies the proposed normalized semantic representation, and Section 6 classifies obligations and enforcement sites. Section 7 treats mapping validation and composition. Section 8 gives the mathematical results. Section 9 provides the feature-level comparison. Section 10 states the executable evidence contract. Sections 11 to 14 discuss implications, limits, open questions, and present status.

2 Established categorical database foundation

2.1 Categories, ologs, and functorial migration

Status. SUPPORTED BY PRIOR LITERATURE.

Ologs use types as objects, functional aspects as arrows, and commuting diagrams as facts; functors connect ologs [20]. Functorial data migration models a schema as a small category,

an instance as a functor to **Set**, and a schema mapping as a functor that induces migration operators customarily denoted Σ, Δ, and Π [19]. Finite graph-and-equation presentations and relational realizations of a functorial fragment are also established [21].

Algebraic data integration and algebraic databases extend this foundation with multisorted equational theories, concrete data, richer mappings, and integration constructions [18, 16]. Algebraic model management explicitly connects schemas, mappings, and model-management operations [17]. Executable case studies in FQL and CQL show that this family of ideas is not merely abstract [26, 4]. The maintained CQL documentation describes executable typesides, schemas, instances, mappings, equations, and migration workflows [6].

It follows that CatDB cannot claim novelty for:

- categories or algebraic theories as schemas;
- paths and equations as schema structure;
- functors or algebraic mappings between schemas;
- functor-like or term-model instances;
- mapping-induced data migration;
- relational execution of selected categorical operations; or
- the existence of an executable categorical query language.

The candidate research delta is instead a particular operational boundary: the finite profile, normalized cross-language records, explicit evidence labels, stable diagnostics, and preservation/loss reports proposed below. Whether that boundary produces practical value is an empirical question.

2.2 Relational and conceptual baselines

Status. SUPPORTED BY PRIOR LITERATURE.

Codd's relational model separates a logical relation-based view from physical access paths and develops keys, normalization, and relational operations [9]. Chen's entity–relationship model raises entities, relationships, and attributes to a conceptual design layer [7]. Therefore neither logical/physical separation nor semantic conceptual modeling begins with CatDB.

The profile in this paper is narrower than both the full relational and the full ER design spaces. A total functional arrow can represent a selected foreign-key-like relationship, but arbitrary n-ary relations, bags, nulls, relationship attributes, participation conditions, and cardinality constraints require additional constructs or explicit reification. A translation that forgets those features must report the residual rather than describe the source model as a special case without qualification.

Classical data exchange. Data exchange already gives source-to-target schema mappings a precise relational semantics. Its established theory includes dependencies, solutions and universal solutions, chase-based construction, and certain answers to target queries [11]. These results are an essential baseline for any CatDB claim about mappings, migrations,

or integration. The finite categorical profile developed here does not subsume that theory: it neither supplies a general dependency language nor proves chase termination, universal-solution results, or certain-answer guarantees. Its candidate delta is instead the typed path-and-equation representation and the cross-language validation boundary evaluated in this paper.

2.3 Graph, ontology, and heterogeneous-system baselines

Status. SUPPORTED BY PRIOR LITERATURE.

RDF defines graphs and datasets built from IRIs, blank nodes, literals, and triples [24]; OWL adds ontology constructs and distinct semantic profiles [22]; SPARQL defines graph-pattern query semantics [25]; and R2RML defines relational-to-RDF mapping documents [23]. Ontop demonstrates mapping-based virtual ontology access and SPARQL-to-SQL rewriting over relational sources [5]. These systems already make semantic mappings and virtual access operational, though under semantics different from those selected here.

ISO/IEC 39075:2024 standardizes GQL for property graphs [12]. Property graphs support labeled nodes and edges, properties, multiplicity, edge identity, and graph-pattern matching [10]; PG-Schema provides formal property-graph schema and key machinery [2]. AsterixDB supplies a broad typed semistructured model with open and closed records and nested collections [1]. HNSW establishes an approximate vector-retrieval method, not a logical identity or schema theory [13].

Finally, mechanized database semantics predates the proposed Lean component of CatDB. HoTTSQL provides a machine-checkable SQL semantics and verified rewrites in Coq [8]. A future CatDB theorem may be specific to its own definitions, but "database semantics have been mechanized" cannot be its novelty claim.

3 Mathematical framework

This section fixes the terminology used in all later formal statements. Unless a block is explicitly labeled otherwise, definitions in this section are ENGINEERING HYPOTHESIS profile choices rather than claims about an implemented system.

3.1 Typesides and the profile boundary

Definition 3.1 (Typeside)**.** *Status.* SUPPORTED BY PRIOR LITERATURE.

A *typeside* is a many-sorted equational theory

$$\mathbb{T} = (\mathrm{Sort}_{\mathbb{T}}, \mathrm{Op}_{\mathbb{T}}, \mathrm{Eq}_{\mathbb{T}}).$$

A typeside algebra D assigns a carrier D_τ to every scalar sort τ, a total function to every operation, and satisfies every typeside equation.

Definition 3.2 (Profile-v1 typeside boundary)**.** *Status.* ENGINEERING HYPOTHESIS.

Profile v1 admits six scalar tags:

$$\{\texttt{string}, \texttt{integer}, \texttt{boolean}, \texttt{decimal}, \texttt{date}, \texttt{uuid}\}.$$

Integers are signed 64-bit values. Decimal values use canonical decimal strings without exponent notation, leading plus, or negative zero. Dates use `YYYY-MM-DD` lexical form and require a separate calendar-validity check. UUID values use lowercase canonical RFC-variant spelling and versions one through eight. All profile-v1 attributes are required and total.

The serialized profile does not admit arbitrary typeside operations, typeside equations, host-language callbacks, or observation equations. Formal results below assume one fixed typeside algebra D. This strict boundary avoids claiming that a host parser or user-defined function is part of the portable semantic kernel.

3.2 Small categories, presentations, and runtime records

Definition 3.3 (Small category)**.** *Status.* SUPPORTED BY PRIOR LITERATURE.

A category $\mathcal{C}$ is *small* when $\mathrm{Ob}(\mathcal{C})$ and $\mathrm{Mor}(\mathcal{C})$ are sets. Smallness implies neither finiteness nor a finite presentation.

Definition 3.4 (Finite schema presentation)**.** *Status.* ENGINEERING HYPOTHESIS.

Fix a typeside $\mathbb{T}$. A profile-v1 schema presentation is a tuple

$$S = (s, E_S, G_S, A_S, R_S)$$

where s is a stable schema identifier; E_S is a finite set of entities; G_S is a finite set of generating arrows with sources and targets in E_S; A_S is a finite set of attributes $a : A \to \tau$, with $A \in E_S$ and τ a typeside sort; and R_S is a finite set of equations between parallel typed entity paths.

The entity-path category presented by S, written $\mathcal{C}_S$, is the free category on the entity-and-generator graph quotiented by the congruence generated by R_S. Attributes are typed observations into the fixed typeside and are not entity arrows of $\mathcal{C}_S$.

Remark 3.5 (Finite syntax is not a finite category). One entity A, one unconstrained loop $u : A \to A$, and no equations form a finite presentation but yield the syntactically distinct family

$$\mathrm{id}_A, u, u^2, u^3, \ldots.$$

Thus no result below relies on the presented category being finite.

Definition 3.6 (Compiled semantic record)**.** *Status.* ENGINEERING HYPOTHESIS.

A *compiled semantic record* is a proposed finite, language-independent value containing stable declaration identifiers, normalized entity/arrow/attribute/equation arrays, paths with declared starts and ordered arrow lists, mapping assignments, finite-instance assignments when present, and stable validation or unsupported outcomes. A *validated record* pairs such a value with evidence that the selected profile checks were discharged. It represents a presentation and metadata; it is not the category $\mathcal{C}_S$ itself.

Object	Finiteness claim	Equality relevant here	Role
Small category $\mathcal{C}_S$	Objects and morphisms form sets; either may be infinite	Equality in the quotient category	Mathematical semantics
Finite presentation S	Finitely many declarations and relations	Stable identifiers plus declared relations	Authorable syntax
Compiled semantic record	Finite serialized value	Canonical structural equality for a declared format	Validation and interchange boundary
Physical realization	Backend-specific finite metadata plus potentially large data	Native equality, transaction, and snapshot semantics	Execution and enforcement

Table 1: Four layers that the profile does not conflate.

3.3 Typed paths and equations

Definition 3.7 (Typed generated path)**.** *Status.* ENGINEERING HYPOTHESIS.

A typed path $p : A \rightsquigarrow B$ is a finite sequence $[f_1, \ldots, f_n]$ of generating arrows satisfying

$$\mathrm{src}(f_1) = A, \qquad \mathrm{tgt}(f_i) = \mathrm{src}(f_{i+1}), \qquad \mathrm{tgt}(f_n) = B.$$

For $n = 0$, the path is the empty sequence $\epsilon_A : A \rightsquigarrow A$, also written id_A.

The typing rules are:

$$\frac{A \in E_S}{S \vdash \epsilon_A : A \rightsquigarrow A} \qquad \frac{f : A \to B \in G_S}{S \vdash [f] : A \rightsquigarrow B}$$

and

$$\frac{S \vdash p : A \rightsquigarrow B \qquad S \vdash q : B \rightsquigarrow C}{S \vdash q \circ p : A \rightsquigarrow C}.$$

Composition uses traversal order:

$$\mathrm{arrows}(q \circ p) = \mathrm{arrows}(p) +\!\!+ \mathrm{arrows}(q).$$

The mathematical expression $q \circ p$ therefore serializes the arrows of p first.

Definition 3.8 (Parallel path equation)**.** *Status.* ENGINEERING HYPOTHESIS.

A path equation $p = q$ is well formed only when both sides have the same declared source and target:

$$S \vdash p : A \rightsquigarrow B \quad \text{and} \quad S \vdash q : A \rightsquigarrow B.$$

Let $\equiv_S$ be the smallest equivalence relation on typed paths that contains the declared equations and is a congruence under typed left and right composition.

Literal equality of arrow lists is *syntactic equality*. Equality under $\equiv_S$ is *presented equality*. The relation is a mathematical definition, not an assertion that the implementation has a complete decision procedure.

Remark 3.9 (Equality-decision boundary). *Status.* OPEN CONJECTURE.

Profile v1 has not selected a complete, useful decision procedure for presented equality. A validator may recognize syntactic equality or provide a sound derivation in a stated fragment. Failure to find such a derivation cannot be reported as a universal proof of inequality.

3.4 Finite total instances

Definition 3.10 (Finite total instance)**.** *Status.* ENGINEERING HYPOTHESIS.

Fix S and a typeside algebra D. A profile-v1 finite instance I assigns:

1. a finite set $I(A)$ to every entity $A \in E_S$;
2. a total function $I(f) : I(A) \to I(B)$ to every $f : A \to B \in G_S$;
3. a total function $I(a) : I(A) \to D_\tau$ to every $a : A \to \tau \in A_S$; and
4. pointwise satisfaction of every declared path equation.

Path interpretation is

$$I(\epsilon_A) = \mathrm{id}_{I(A)}, \qquad I([f_1, \ldots, f_n]) = I(f_n) \circ \cdots \circ I(f_1).$$

For every declared $p = q : A \rightsquigarrow B$, validity requires

$$\forall x \in I(A), \qquad I(p)(x) = I(q)(x).$$

Because the declared equations hold and function equality is a congruence, the entity-and-arrow part factors through $\mathcal{C}_S \to \mathbf{FinSet}$. The totality assumption is material. If an arrow is missing at one source element, path interpretation may be undefined and the results in Section 8 no longer apply.

3.5 Mappings and attribute expressions

Definition 3.11 (Profile-v1 attribute expression)**.** *Status.* ENGINEERING HYPOTHESIS.

An attribute expression of sort τ from entity A has the form

$$p; a$$

where $p : A \rightsquigarrow B$ is an entity path and $a : B \to \tau$ is an attribute. In an instance J,

$$[\![p; a]\!]_J = J(a) \circ J(p).$$

No arbitrary scalar operation is admitted.

Definition 3.12 (Total typed mapping)**.** *Status.* ENGINEERING HYPOTHESIS.

For schemas S and T over one fixed typeside, a total typed mapping $F : S \to T$ assigns:

1. every $A \in E_S$ an entity $F(A) \in E_T$;
2. every generator $f : A \to B$ a target path $F(f) : F(A) \rightsquigarrow F(B)$; and
3. every attribute $a : A \to \tau$ a target attribute expression $F(a) : F(A) \to \tau$.

Exactly one assignment is required for every source declaration.

Extend F to paths using

$$F(\epsilon_A) = \epsilon_{F(A)}$$

and traversal-order substitution:

$$F([f_1, \ldots, f_n]) = F(f_1) +\!\!+ \cdots +\!\!+ F(f_n).$$

Definition 3.13 (Lawful mapping)**.** *Status.* ENGINEERING HYPOTHESIS.

A total typed mapping $F : S \to T$ is *lawful* when every declared source equation is preserved:

$$p \equiv_S q \quad \Longrightarrow \quad F(p) \equiv_T F(q)$$

for each generating equation in R_S.

Definition 3.14 (Extensional mapping equality)**.** *Status.* ENGINEERING HYPOTHESIS.

Mappings F and G are extensionally equal, written $F \approx G$, when their source and target schema identifiers and their canonical object, arrow-path, and attribute-expression assignments agree. Only the caller-prescribed mapping artifact identifier is ignored.

Total and typed do not mean injective, surjective, invertible, lossless, writable, inexpensive, or lawful. Lawfulness is a distinct obligation.

3.6 Pullback migration

Definition 3.15 (Finite pullback migration)**.** *Status.* SUPPORTED BY PRIOR LITERATURE for the construction; ENGINEERING HYPOTHESIS for the profile boundary.

Let $F : S \to T$ be lawful and total, and let J be a valid finite T-instance. Define $\Delta_F J$ on S by

$$(\Delta_F J)(A) = J(F(A)), \qquad (\Delta_F J)(f) = J(F(f)), \qquad (\Delta_F J)(a) = [\![F(a)]\!]_J .$$

Profile v1 selects Δ as its only in-profile migration operation. A bounded Σ requires quotient and canonical-representative semantics that are not fixed. General Π is explicitly outside the profile. Neither operation may be silently approximated and reported as supported.

4 Running customer, order, and account example

4.1 The schema law

Status. ENGINEERING HYPOTHESIS.

Let the schema identifier be `CommerceV1`. It has entities

$$\texttt{Customer}, \qquad \texttt{Order}, \qquad \texttt{Account},$$

generating arrows

$$\begin{aligned} \texttt{placedBy} &: \texttt{Order} \to \texttt{Customer}, \\ \texttt{accountOf} &: \texttt{Customer} \to \texttt{Account}, \\ \texttt{billedTo} &: \texttt{Order} \to \texttt{Account}, \end{aligned}$$

and attributes

$$\begin{aligned} \texttt{customerName} &: \texttt{Customer} \to \texttt{string}, \\ \texttt{orderTotal} &: \texttt{Order} \to \texttt{decimal}, \\ \texttt{accountCode} &: \texttt{Account} \to \texttt{string}. \end{aligned}$$

The declared law is

$$[\texttt{billedTo}] \equiv [\texttt{placedBy}, \texttt{accountOf}] : \texttt{Order} \rightsquigarrow \texttt{Account}. \tag{CO-1}$$

Equivalently,

$$\texttt{billedTo} = \texttt{accountOf} \circ \texttt{placedBy}.$$

We chose a deliberately strong rule because it makes the formal machinery visible. A business that permits third-party billing should not declare this equation. The schema language can expose the rule for inspection, but it cannot decide whether the rule matches the business.

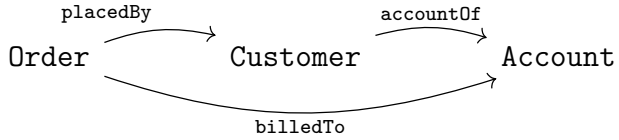


Figure 1: The running schema. The direct lower traversal and the two-step upper traversal are declared equal by (CO-1).

4.2 Typed and ill-typed traversals

The identity path at `Order` is represented by a declared start and an empty arrow list. The path to the placing customer is [`placedBy`]. The path to that customer's account is [`placedBy`, `accountOf`]. The list [`accountOf`, `placedBy`] starting at `Order` is not a path: its first arrow requires source `Customer`.

This failure is small enough to miss in a shape-only validator. Every list item is a known, well-formed identifier, yet the traversal is meaningless. The schema checker must resolve the declared start, follow the arrows in order, and return either one endpoint or a typed failure.

Meaning	Start	Serialized arrows	Endpoint
Identity on order	`Order`	`[]`	`Order`
Order's customer	`Order`	`[placedBy]`	`Customer`
Customer's account	`Customer`	`[accountOf]`	`Account`
Order's account through customer	`Order`	`[placedBy, accountOf]`	`Account`
Direct billed account	`Order`	`[billedTo]`	`Account`
Ill-typed list	`Order`	`[accountOf, placedBy]`	rejected at first arrow

Table 2: Traversal-order path representation for the running example.

4.3 A finite lawful instance and a witness of violation

Let

$$I(\texttt{Customer}) = \{c_1, c_2\},$$
$$I(\texttt{Order}) = \{o_1, o_2\},$$
$$I(\texttt{Account}) = \{a_1, a_2\}.$$

Interpret arrows by

	first assignment	second assignment
$I(\texttt{placedBy})$	$o_1 \mapsto c_1$	$o_2 \mapsto c_2$
$I(\texttt{accountOf})$	$c_1 \mapsto a_1$	$c_2 \mapsto a_2$
$I(\texttt{billedTo})$	$o_1 \mapsto a_1$	$o_2 \mapsto a_2$

and use total typed attribute values such as

$$I(\texttt{customerName})(c_1) = \text{Ada}, \qquad I(\texttt{customerName})(c_2) = \text{Grace},$$
$$I(\texttt{orderTotal})(o_1) = \texttt{19.95}, \qquad I(\texttt{orderTotal})(o_2) = \texttt{8.50},$$
$$I(\texttt{accountCode})(a_1) = \texttt{A-1}, \qquad I(\texttt{accountCode})(a_2) = \texttt{A-2}.$$

Both sides of (CO-1) send o_i to a_i. The instance is valid.

Now change only $I(\texttt{billedTo})(o_2)$ to a_1. The assignment remains finite, total, and correctly typed, but it is no longer a valid `CommerceV1` instance. The two sides of (CO-1) disagree at o_2. Totality, typing, and equation satisfaction are therefore separate validation phases.

4.4 A small relational realization, without an equivalence claim

One possible relational realization uses tables `Customer(id, name, account_id)`, `Account(id, code)`, and `Order(id, customer_id, billed_account_id, total)`. Non-null foreign keys can enforce that each represented arrow is total and targets an existing row. They do not by themselves enforce

$$\texttt{Order.billed\_account\_id} = \texttt{Customer.account\_id}$$

after joining through `Order.customer_id`. PostgreSQL, for example, documents that a `CHECK` constraint is not a general cross-row or cross-table assertion mechanism [14].

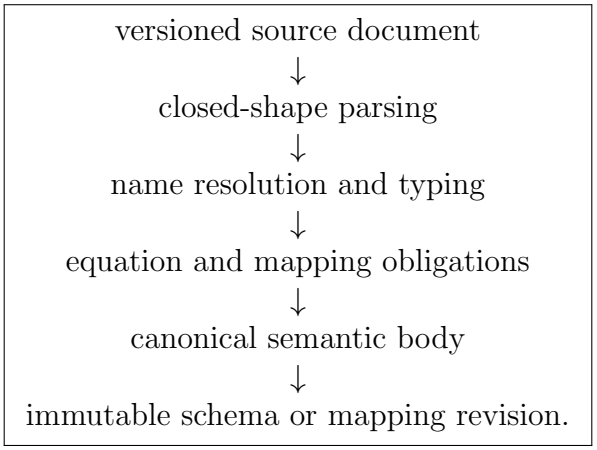
This is an illustrative realization, not generated SQL. We do not claim that transactions maintain (CO-1) or that the relational encoding preserves every source feature. A later backend profile must identify the obligations that lower to native constraints, those that require runtime validation, and those that remain unsupported.

5 From presentation to normalized semantic representation

5.1 Compilation boundary

Status. ENGINEERING HYPOTHESIS.

The proposed compilation sequence is:



Compilation ends before target-specific row execution. It may produce an immutable description of work, but it does not establish that the work is cheap or that category-level reasoning is absent from every runtime path. Those are later systems questions.

The semantic outcome has three states:

`accepted` Every required static check in the selected fragment succeeded, and every residual obligation has an explicit disposition. Acceptance says nothing about unseen data or unsupported backends.

`rejected` An in-profile structural or semantic defect was found and a canonical primary diagnostic is returned.

`unsupported` The request is outside the selected profile. Unsupported is neither malformed input nor permission to approximate.

5.2 Identity distinctions

Four forms of identity must remain separate.

Table 3: Proposed identity model.

Identity	Purpose	Stability rule	Does not imply
Logical artifact ID	Governance continuity across revisions	Caller-assigned under a version policy	Byte equality, compatibility, or category equivalence
Declaration ID	Continuity of one entity, arrow, or attribute	Preserved only when semantic identity is intentionally preserved	Same definition across revisions
Immutable revision ID	Exact canonical semantic body	Algorithm-tagged digest over a versioned preimage	Business identity, isomorphism, or authenticity
Runtime slot	Compact index inside one compiled revision	Deterministically assigned from canonical order	Cross-revision identity

A content digest can identify one exact canonical record. It is not a proof that two categories are equivalent, that a migration is compatible, or that two customer records denote the same person. Profile v1 therefore proposes the unambiguous declaration reference

$$(\text{schema revision}, \text{declaration kind}, \text{declaration ID}).$$

5.3 Canonical paths and equations

A path value contains a start and an ordered arrow list:

```
{
  "start": "Order",
  "arrows": ["placedBy", "accountOf"]
}
```

The list is semantically ordered and must not be sorted. A proposed display and sort key is

$$\text{pathKey}(p) = \text{start}(p) \text{“/”-prefixed arrow identifiers.}$$

Thus the identity at `Order` has key `Order`, while the two-arrow path has key `Order/placedBy/accountOf`. The v1 identifier grammar excludes slash; a future grammar that admits slash must version the encoding.

Equation orientation is not semantically relevant in this profile. Given path keys k_1, k_2, the canonical equation subject is

$$\texttt{equation:}\ \min(k_1, k_2) = \max(k_1, k_2).$$

This ordering cannot repair ill-typed or nonparallel sides. Typing precedes canonicalization.

5.4 Immutable schema and mapping records

The proposed schema revision includes:

- a closed artifact header with format and semantic-profile versions;
- the exact typeside profile;
- canonically sorted entities, arrows, and attributes;
- interned typed paths referenced by equations and mappings;
- canonically oriented equations;
- an obligation ledger; and
- a sidecar diagnostic source map.

The proposed mapping revision includes exact source and target schema revisions, one object assignment per source entity, one path assignment per source arrow, one attribute expression per source attribute, equation preservation obligations, and conservative loss findings. It cannot abbreviate "no syntactic witness found" to "lossless."

Canonical JSON under RFC 8785 is one candidate for a cross-language hash preimage [15]; JSON itself is an interchange grammar, not a semantic constraint system [3]. The adoption of RFC 8785 remains open until every admitted scalar round-trips across the intended Rust and Haskell decoders. A hash must carry an algorithm and format tag, and source line locations should remain outside the semantic preimage.

5.5 Deterministic diagnostics

The acceptance surface for an in-profile failure is proposed as:

```
{
  "code": "PATH_SOURCE_MISMATCH",
  "phase": "type_check",
  "json_pointer": "/input/path/arrows/0",
  "related": [
    "arrow:accountOf",
    "entity:Customer",
    "entity:Order"
  ]
}
```

Free-form exception text, stack traces, vendor messages, and host-language type names are not comparison keys. If several defects exist, the primary diagnostic follows a fixed phase order: parse, shape, duplicates, unresolved references, path/equation typing, mapping completeness/typing, mapping preservation, instance totality/typing, instance equations, and migration preconditions. Within a phase, the smallest JSON Pointer and then diagnostic code wins.

This rule is intended to make negative fixtures reproducible under different map iteration orders. The accepted Rust and Haskell artifacts reproduce the declared diagnostics for the

single-fault negative fixtures in the admitted Slice-1 profile. The full within-phase tie-break for arbitrary multi-fault inputs is not implemented and remains ENGINEERING HYPOTHESIS.

6 Obligations and enforcement sites

6.1 Two axes

Status. ENGINEERING HYPOTHESIS.

Calling every requirement a "constraint" hides two independent questions:

1. What kind of semantic statement must hold?
2. At which site can evidence for that statement be produced?

The first axis includes representation well-formedness, typing judgments, presentation laws, mapping preservation, finite-instance constraints, migration preconditions, conservative analyses, algebraic invariants, and physical-realization conditions. The second axis includes static semantic checking, native database constraints, compiled runtime checks, higher invariants, and explicit unsupported outcomes.

Table 4: Semantic kinds and representative evidence.

Semantic kind	Question	Representative evidence	Boundary
Representation well-formedness	Are closed shapes and identifiers valid?	Deterministic decoding and duplicate detection	Does not establish references or typing
Typing judgment	Do path endpoints and assignment types align?	Finite declaration fold with stable result	Does not establish equation preservation
Presentation law	Is an equation declared between parallel paths?	Validated presentation record	Does not establish physical enforcement
Mapping preservation	Does a mapping send source laws to justified target equalities?	Derivation in a stated target fragment	Failure to derive need not be falsity
Instance constraint	Does one finite interpretation satisfy totality and equations?	Exhaustive check on a named snapshot	Does not cover future states
Migration precondition	Are mapping and instance valid for an admitted migration?	Validity evidence tied to exact revisions	Does not establish performance
Conservative analysis	Is there a named syntactic loss witness?	Auditable source/target references	No witness is not invertibility

Semantic kind	Question	Representative evidence	Boundary
Algebraic invariant	Do identity, associativity, or pullback laws hold?	Paper proof or exact proof-assistant theorem	Does not verify runtime code
Physical realization condition	Does a backend preserve one selected condition?	Capability-specific DDL, introspection, and violation tests	Does not establish portability

6.2 Four concrete classifications

Static semantic check. Typing [`placedBy`, `accountOf`] needs only the schema declarations. The checker starts at `Order`, advances to `Customer`, then to `Account`. No instance data or backend is involved.

Native database constraint. A relational realization of total `placedBy` may use a non-null foreign key. Evidence requires exact DDL, backend/version capability, introspection, a valid insertion, and rejected null/dangling insertions. Even that evidence establishes only the selected realization.

Compiled runtime check. Equation (CO-1) can be checked pointwise on a finite snapshot. A useful failure result names the equation, source element, left target, right target, and snapshot. Zero counterexamples discharges the obligation only for that state.

Higher invariant. Associativity of typed path composition is established by a mathematical proof in Theorem 8.1. The exact indexed-path statement `CatDB.Path.comp_assoc` is also machine-checked by Lean under the assumptions in `lean/COVERAGE.md`. The corresponding Haskell property test remains computational evidence, not proof.

6.3 Obligation record

A proposed record is:

```
{
  "obligation_id": "obl.eq.order-billing",
  "semantic_kind": "instance_constraint",
  "discharge_site": "compiled_runtime_check",
  "subjects": [
    "path:Order/billedTo",
    "path:Order/placedBy/accountOf"
  ],
  "invariant_ids": ["EQ-02"],
  "assumptions": [
    "finite carrier",
    "total arrow interpretation",
    "one coherent source snapshot"
```

```
  ],
  "procedure_profile": "pointwise-finite-v1",
  "status": "required",
  "evidence_refs": []
}
```

An accepted schema may contain undischargeable-at-compile-time instance obligations, because it has no instance. An instance cannot be called valid until its totality, typing, and equation obligations are discharged for the named data and snapshot. The empty `evidence_refs` array above is an explicit statement that evidence does not yet exist.

7 Mapping validation and composition

7.1 A nontrivial mapping chain

Define a target schema `CommerceCanonical` with entities `Person`, `Sale`, `LedgerAccount`, arrows

$$\texttt{buyer} : \texttt{Sale} \to \texttt{Person}, \qquad \texttt{primaryAccount} : \texttt{Person} \to \texttt{LedgerAccount},$$

and attributes

$$\begin{aligned} \texttt{displayName} &: \texttt{Person} \to \texttt{string}, \\ \texttt{amount} &: \texttt{Sale} \to \texttt{decimal}, \\ \texttt{ledgerCode} &: \texttt{LedgerAccount} \to \texttt{string}. \end{aligned}$$

No direct billed-account generator is needed. Define $F : \texttt{CommerceV1} \to \texttt{CommerceCanonical}$ by

$$\begin{aligned} \texttt{Customer} &\mapsto \texttt{Person}, \\ \texttt{Order} &\mapsto \texttt{Sale}, \\ \texttt{Account} &\mapsto \texttt{LedgerAccount}, \\ \texttt{placedBy} &\mapsto [\texttt{buyer}], \\ \texttt{accountOf} &\mapsto [\texttt{primaryAccount}], \\ \texttt{billedTo} &\mapsto [\texttt{buyer}, \texttt{primaryAccount}]. \end{aligned}$$

Map the three attributes to the corresponding target attributes through empty paths.

The source law (CO-1) maps on both sides to the literal list $[\texttt{buyer}, \texttt{primaryAccount}]$. Thus this particular obligation is discharged by syntactic equality; no general target equality decision procedure is required.

Now define `AnalyticsV1` with entities `Actor`, `Purchase`, `AccountDim`, arrows

$$\texttt{actor} : \texttt{Purchase} \to \texttt{Actor}, \qquad \texttt{account} : \texttt{Actor} \to \texttt{AccountDim},$$

and analogous attributes. Mapping $G : \texttt{CommerceCanonical} \to \texttt{AnalyticsV1}$ sends `buyer` to $[\texttt{actor}]$ and `primaryAccount` to $[\texttt{account}]$. The composite sends `billedTo` to $[\texttt{actor}, \texttt{account}]$.

This example is nontrivial because a source generator maps to a two-arrow target path and remains composable. It is still only a paper example. It does not establish a parser, normalizer, compiler, or data movement job.

7.2 Attribute substitution

Suppose $F(a) = p; b$, where $p : F(A) \rightsquigarrow B$ and $b : B \to \tau$. Suppose $G(b) = q; c$, where $q : G(B) \rightsquigarrow C$ and $c : C \to \tau$. Then the proposed composite attribute assignment is

$$G_*(p; b) = (q \circ G(p)); c. \tag{ATTR-SUB}$$

The serialized prefix contains the arrows of $G(p)$ followed by the arrows of q. The scalar sort remains τ. This clause is necessary: composing only the entity-arrow map would leave attribute assignments underspecified.

7.3 A mapping that fails lawfulness

Let source schema S have parallel generators $u, v : A \to B$ and declare $u = v$. Let target schema T have parallel generators $x, y : X \to Y$ with no equation justifying $x \equiv_T y$. The assignments

$$A \mapsto X, \quad B \mapsto Y, \quad u \mapsto [x], \quad v \mapsto [y]$$

are total and typed but not lawful. This counterexample refutes any validator that treats mapping completeness as equation preservation.

An operational checker must also distinguish three outcomes:

1. preservation established by syntactic equality;
2. preservation established by a derivation in the selected target fragment; and
3. not established in that fragment.

The third outcome is not automatically a counterexample. Profile v1 should fail closed for an executable mapping, while retaining the unresolved obligation as evidence about why acceptance failed.

7.4 Information loss and writeability are separate

Object identification, arrow collapse, and attribute omission can be syntactic witnesses of information loss. Their absence is not a proof of invertibility. A lawful mapping can identify two source entities, omit distinctions, or be unsuitable for writes. Conversely, a read-only mapping can be meaningful and lawful. Later papers treat conservative loss analysis and bidirectional update semantics; P1 only requires that neither property be smuggled into the word "mapping."

8 Results

This section contains the claims labeled PROVED IN THIS PAPER. Each proof is a human-auditable mathematical argument over the definitions above. None is machine-checked, and none establishes code behavior.

Proposition 8.1 (Typed paths present a small category)**.** *Status.* PROVED IN THIS PAPER.

Let S have a set of entities, a set of typed generating arrows, and a set of equations between parallel finite paths. Then:

1. *a declared start and typed arrow list determine at most one endpoint;*
2. *typed finite paths form a small free category under traversal-order concatenation;*
3. *empty paths are left and right identities;*
4. *composition is associative; and*
5. *quotienting by the congruence generated by the equations yields a well-defined small category $\mathcal{C}_S$.*

Proof. For entities A, B, let $\mathrm{Path}_S(A, B)$ be the finite generator lists that begin at A, end at B, and satisfy the endpoint check at each adjacent pair. Finite lists over a set form a set, so their disjoint union over entity pairs is a set.

Endpoint uniqueness follows by induction on list length. The empty list ends at its declared start. A nonempty list begins with a generator having one declared target. After its source has been checked, each later generator has one declared target and must have source equal to the current endpoint. Therefore a successfully typed start/list pair cannot receive two endpoints.

If $p : A \rightsquigarrow B$ and $q : B \rightsquigarrow C$, concatenating their traversal lists yields $q \circ p : A \rightsquigarrow C$. The empty list ϵ_A is typed $A \rightsquigarrow A$. The list identities

$$[] +\!\!+\, p = p, \qquad p +\!\!+\, [] = p$$

give the left and right category identities. Associativity of list concatenation gives

$$r \circ (q \circ p) = (r \circ q) \circ p$$

whenever the endpoints align.

Finally, $\equiv_S$ is defined as a congruence under typed composition. Source, target, identity, and composition therefore do not depend on a chosen path representative and descend to equivalence classes. A quotient of the set of typed paths is a set, so $\mathcal{C}_S$ is small. □

Lemma 8.2 (Lawful assignments respect presented equality)**.** *Status.* PROVED IN THIS PAPER.

Let $F : S \to T$ be total and typed and preserve every declared equation of S. If $p \equiv_S q$, then $F(p) \equiv_T F(q)$.

Proof. Presented equality is generated from declared equations by reflexivity, symmetry, transitivity, and typed left and right composition. The mapping sends each generating equation to a target equality by hypothesis. Target equality has the same equivalence and congruence rules. Path substitution preserves identities by the empty-list definition and composition by concatenation. Induction over the derivation of $p \equiv_S q$ therefore produces a derivation of $F(p) \equiv_T F(q)$. □

Proposition 8.3 (Identity and composition of lawful mappings)**.** *Status.* PROVED IN THIS PAPER.

Assume all schemas use one fixed typeside; mappings are total on entities, arrows, and attributes; target paths and attribute expressions are well typed; lawfulness means preservation of declared path equations; and mapping equality is extensional as in Theorem 3.14. Then:

1. *the identity mapping $1_S : S \to S$ is lawful;*
2. *if $F : S \to T$ and $G : T \to U$ are lawful, their composite formed by path substitution and (ATTR-SUB) is total, typed, and lawful;*
3. *identity mappings are left and right units; and*
4. *mapping composition is associative.*

Proof. Define $1_S(A) = A$, $1_S(f) = [f]$, and $1_S(a) = \epsilon_A; a$. Every source equation maps to itself, and every assignment is total and typed, so 1_S is lawful.

For composition define $(G \circ F)(A) = G(F(A))$. Substitute each target generator in $F(f)$ by its G-image and concatenate in traversal order. For an attribute use (ATTR-SUB). Endpoint alignment in each mapping shows that the resulting arrow paths and attribute expressions are typed. Totality follows because both inputs assign every source declaration.

If $p \equiv_S q$, lawfulness of F gives $F(p) \equiv_T F(q)$. Applying Theorem 8.2 to G gives

$$G(F(p)) \equiv_U G(F(q)),$$

which is precisely preservation by $G \circ F$.

Substitution by singleton generator paths and empty attribute prefixes leaves each assignment unchanged, yielding the unit laws extensionally. For associativity, both parenthesizations of path substitution flatten to the same traversal list. The same argument applies to each attribute prefix, and the terminal attribute is unchanged. Object assignments compose associatively. Thus the resulting mappings are extensionally equal. □

Proposition 8.4 (Validity and contravariant composition of pullback)**.** *Status.* PROVED IN THIS PAPER.

Assume one fixed typeside algebra D; lawful total mappings $F : S \to T$ and $G : T \to U$; valid finite total instances; extensional instance equality that ignores only the instance artifact identifier; and compositional interpretation of attribute expressions. Then:

1. *if J is a valid finite T-instance, $\Delta_F J$ is a valid finite S-instance;*
2. *$\Delta_{1_S} I \approx I$; and*
3. *for every valid finite U-instance K,*

$$\Delta_{G \circ F} K \approx \Delta_F(\Delta_G K).$$

Proof. Each carrier $(\Delta_F J)(A) = J(F(A))$ is finite. Every mapped arrow path denotes a composite of total functions, and every mapped attribute expression denotes a total typed scalar function. For a source equation $p \equiv_S q$, lawfulness gives $F(p) \equiv_T F(q)$. Validity of J therefore gives

$$(\Delta_F J)(p) = J(F(p)) = J(F(q)) = (\Delta_F J)(q).$$

Thus the pullback assignment is valid.

For identity, evaluate each component:

$$(\Delta_{1_S} I)(A) = I(A), \quad (\Delta_{1_S} I)(f) = I([f]) = I(f), \quad (\Delta_{1_S} I)(a) = I(a) \circ I(\epsilon_A) = I(a).$$

The instances agree extensionally.

For composition, entity carriers agree because both sides assign $K(G(F(A)))$. Arrow interpretations agree by compositional path interpretation. For a source attribute a, write

$$F(a) = p; b, \qquad G(b) = q; c.$$

The left-hand interpretation is

$$[\![(q \circ G(p)); c]\!]_K = K(c) \circ K(q) \circ K(G(p)).$$

The right-hand side first interprets b in $\Delta_G K$ as $K(c) \circ K(q)$, then interprets p as $K(G(p))$, producing the same function. Every semantic component agrees; only a generated artifact identifier may differ. □

Corollary 8.5 (Categorical status of the selected structures)**.** *Status.* PROVED IN THIS PAPER.

Under the assumptions above, profile-v1 schema presentations and lawful mappings form a category under extensional mapping equality, and finite pullback migration is contravariantly functorial on valid finite instances.

Proof. The first statement is Theorem 8.3; the second is the identity and composition portion of Theorem 8.4. □

8.1 What the results do not prove

The proofs establish laws of the mathematical profile. They do not show:

- that a parser accepts exactly the intended records;
- that path typing or diagnostics are correctly implemented;
- that arbitrary presented equality is decidable;
- that the accepted Rust/Haskell agreement extends beyond the 16 admitted semantic fixtures;
- that Lean checks serialization, diagnostics, fixture translation, instance semantics, or the paper's pullback proposition;
- that Σ, Π, a chase, or a general Kan extension is supported;
- that a physical query or migration preserves denotation; or
- that the operational proposal improves performance or integration effort.

9 Feature-level comparison

Table 5: Feature-level comparison. "Profile bridge" means a proposed representable subset, never universal subsumption.

Approach	Native semantic unit	Native relationship or query model	Profile-v1 bridge	Non-preserved or residual feature	Evidence boundary
Relational	Relations, tuples, domains, keys; SQL systems add bags, nulls, constraints, views, and transactions	n-ary relations and joins	Selected normalized tables as entities; required foreign-key-like columns as total arrows; scalar columns as attributes	Bags, nulls, arbitrary dependencies, recursion, view/update semantics, collation, and physical design	Logical/physical independence and relational semantics are prior art [9]; no backend equivalence is claimed
Entity–relationship	Entity sets, relationship sets, roles, attributes, keys, participation	Conceptual relationships, possibly n-ary, with cardinality constraints	Entities and selected functional relationships elaborate into objects and arrows	Weak entities, n-ary relationships, relationship attributes, role and participation semantics	Conceptual semantic modeling is prior art [7]
Categorical/ CQL	Typeside, schema presentation, instance, mapping, transform, query	Typed paths, equations, functorial/algebraic migration	Closest semantic baseline; profile v1 selects a finite total subset with normalized records	General typeside terms, observation equations, richer instances, general Σ/Π, queries, theorem proving	Core mathematics and executable precedent are prior art [19, 16, 6]
RDF	IRI, blank node, literal, triple, graph, dataset	Binary predicates and SPARQL graph patterns	Typed closed-world projection with declared identifiers and selected predicate roles	Blank-node scope, arbitrary predicates, named graphs, entailment, SPARQL solution semantics	RDF semantics are normative prior art [24, 25]
OWL/OBDA	Ontology axioms, classes, individuals, properties, profiles, entailment regimes	Description-logic and RDF-based semantics; mapping-based virtual access	At most a reviewed translation of a selected subset into obligations	Open-world reasoning, existentials, inconsistency, class subsumption, profile-specific entailment	Path equations are not OWL inference; OBDA is prior systems work [22, 5]

Approach	Native semantic unit	Native relationship or query model	Profile-v1 bridge	Non-preserved or residual feature	Evidence boundary
Property graph/GQL	Nodes, edges, labels/types, properties, graph identity	Multivalued edges, edge identity, graph patterns, path matches, mutation	Selected node types as entities; exactly-one edge projections as arrows; reified edges when explicitly modeled	Parallel/multiple edges, edge properties, labels, inheritance, variable paths, GQL update and result semantics	GQL and property-graph schema work are established [12, 10, 2]
Document/ semistructured	Open or closed records, nested values, arrays, collections	Field navigation, nesting, unnesting, heterogeneous records	Closed required leaf fields and explicit reference paths	Missing versus null, arrays, order, multiplicity, arbitrary nesting, schema-on-read evolution	Typed semistructured systems are prior art [1]
Vector retrieval	Vector, metric, index, scored neighbor result	Approximate nearest-neighbor search	Versioned derived attribute plus typed retrieval operator in later work	Identity, authority, equation satisfaction, mapping lawfulness, logical equivalence	HNSW is retrieval prior art; similarity is not semantic equality [13]
Multimodel/ polystore	Model-specific objects governed by several engines	Model-native operators plus cross-model conversion and data movement	One semantic revision with separate realization descriptors and explicit residuals	Universal constraints, transactions, identity, zero-cost conversion, and native expressiveness	P1 proposes only a contract; no integrated runtime is reported

9.1 Preservation and loss questions

The comparison suggests a minimum discipline for every bridge:

1. Name the source feature, not merely the source product.
2. State the admitted fragment.
3. Give the target representation.
4. State which equality, multiplicity, null, identity, and update semantics are preserved.
5. Record residual checks and known losses.
6. Tie every physical enforcement claim to a backend/version capability and a data-snapshot contract.

For example, representing a property-graph edge as a profile-v1 arrow is sound only when exactly one target exists for every source under the selected snapshot semantics. Choosing an arbitrary edge when two exist would conceal loss. Treating an absent RDF triple under open-world semantics as a closed world violation would conflate reasoning regimes. Treating a high vector similarity score as customer identity would conflate retrieval evidence with an authority decision.

10 Executable evidence and reproducibility contract

10.1 Present evidence

The present artifact contains:

- source-checked primary-literature positioning;
- the formal profile and assumptions;
- a running example and counterexamples;
- complete paper proofs for Theorems 8.1 to 8.4;
- a 24-case accepted fixture snapshot with 11 positive, 11 negative, and two explicitly unsupported envelopes;
- Rust computation of 16 admitted semantic outcomes and closed structural handling of eight later-slice envelopes;
- an independent Haskell decoder and evaluator that computes the same 16 semantic outcomes, classifies the same eight envelopes as structural only, and disagrees with Rust on zero admitted cases;
- seven deterministic Haskell properties totaling 1,550 generated cases with no counterexample;
- nine Lean declarations with an empty kernel axiom inventory covering exactly `PATH-02`–`PATH-04` and `MAP-03`–`MAP-05`; and

- explicit implementation and experimental nonclaims.

The computational evidence is bound to implementation commit `abd655f4db31f1e0a9e78de8bbe83c71805f46c7` and terminal Claude review digest `a9f8debc972623accb4ae9cc8edcf5e353752e9bbdc438093890969192a77c95`. The Lean evidence is bound to its pinned toolchain and coverage inventory. None of these artifacts validates finite instances, evaluates Δ, Σ, Π, lowers a backend constraint, or measures a runtime.

10.2 Required shared fixtures

Status. ENGINEERING HYPOTHESIS.

The versioned fixture contract requires eight operation categories:

1. schema parsing and normalization;
2. path typing;
3. path composition;
4. identity mapping construction;
5. mapping composition;
6. migration validation, initially finite Δ;
7. conservative information-loss detection; and
8. stable-identifier schema diff.

Each case carries one deterministic expected result: `ok`, `error`, or `unsupported`. An expected error is a passing case only when code, phase, JSON Pointer, and related references match. An unsupported Σ or Π request must not become a skipped test or partial result.

The initial P1 gate covers:

`PATH-01`...`PATH-04`, `MAP-01`...`MAP-05`.

The fixture set must include empty paths, valid multi-arrow paths, unknown arrows, endpoint mismatches, identity mappings, successful composition, schema mismatch, equation-preserving and equation-failing mappings, and artifact-ID versus extensional-equality cases.

10.3 Cross-language roles

Agreement among Rust, Haskell, and fixtures is computational conformance evidence, not proof. In the accepted Slice-1 comparison, 16 independently computed semantic outcomes agree, eight cases remain structural only, and there are zero disagreements. Any future disagreement is resolved against the written definition and classified as a definition, fixture, implementation, reference-model, or theorem defect. Majority vote is not a semantic rule.

Artifact	Authority	Required evidence	Does not establish
Rust runtime	Shipping parser, normalized IR, diagnostics, and production behavior	Native tests over every applicable fixture; independent code review	General mathematical laws or backend equivalence
Haskell model	Independent executable reference for the finite semantics	Independent decoder/evaluator and property tests over the same fixtures	Proof merely by using Haskell
Lean library	Exact theorem statements accepted by the Lean kernel	Build without `sorry`/`admit`; theorem and axiom audit	Parser, connector, SQL, transaction, or performance behavior
Shared fixtures	Cross-language finite acceptance surface	Complete expected outcomes, canonical ordering, stable diagnostics	Completeness beyond the represented cases

Table 6: Accepted evidence roles and their nontransfer boundaries.

10.4 Falsification conditions

The operational hypothesis must be narrowed or rejected if:

- an ill-typed path or mapping is accepted;
- the written and serialized composition orders disagree;
- identity or extensional equality erases more than the specified artifact identifier;
- Rust and Haskell disagree on an accepted or rejected fixture;
- a machine-check claim names no building theorem, contains `sorry`/`admit`, or hides material axioms;
- the IR cannot represent the Customer–Order–Account example without a backend-specific callback;
- an obligation record does not distinguish semantic kind from discharge site;
- a backend translation silently drops a non-preserved feature; or
- the alleged research delta reduces to new syntax or record names without measurable validation, interchange, or repair benefit.

10.5 Accepted evidence commands

The accepted evidence can be reproduced with these native commands from the repository root, except that the standalone Cabal commands run from `haskell/catdb-reference`:

```
python3 scripts/validate-fixtures.py \
  --schema fixtures/schema-v1.schema.json --cases fixtures/cases
python3 scripts/audit-fixture-manifest.py fixtures/manifest-v1.json
cargo fmt --all -- --check
cargo clippy --workspace --all-targets -- -D warnings
cargo test --workspace
cargo run -p catdb-cli -- fixtures verify fixtures/cases \
  --implementation rust --capability-set slice-1
cargo run -p catdb-cli -- fixtures compare fixtures/cases \
  --left rust --right haskell --capability-set slice-1
cabal build all -fstrict --disable-documentation \
  --project-file=cabal.project --with-compiler=/opt/homebrew/bin/ghc
cabal test all -fstrict --disable-documentation \
  --project-file=cabal.project --with-compiler=/opt/homebrew/bin/ghc \
  --test-show-details=direct
lake build
python3 scripts/audit-lean.py lean
```

The release records preserve the exact tool versions, outputs, source hashes, and review digests. Paper build and paper-review commands are frozen separately when the manuscript digest is final.

11 Discussion

11.1 What categorical structure contributes

The categorical portion contributes a disciplined account of composition. Paths have endpoints, identities, and associative composition. A schema mapping acts on generators and extends to paths. Its lawfulness is not only a collection of local type checks; it requires source equations to remain true in the target presentation. Pullback migration then composes contravariantly. These are established mathematical ideas, but they matter operationally because they suggest concrete validation boundaries.

The practical value does not come from displaying category notation beside a database. It comes, if at all, from turning the relevant structure into artifacts that can fail precisely. An ill-typed path should point to one arrow. A mapping should report a missing assignment separately from an unproved preservation obligation. A materialization should record the schema and mapping revisions that gave it meaning. A backend should report which obligations it enforces and which remain residual.

11.2 Why the category is not the wire format

A category may have infinitely many morphisms even when its presentation is finite. A runtime cannot materialize that collection merely to type a short path. The proposed IR therefore stores generators, declared equations, and the finite paths used by artifacts. It is a representation of syntax plus validation metadata, not a serialization of every morphism in $\mathcal{C}_S$.

This distinction also protects language independence. Rust can use compact numeric slots after validation, Haskell can use transparent algebraic data types, and Lean can use proof-indexed objects. Their shared boundary is a closed normalized value and agreed semantics, not a shared in-memory layout.

11.3 Meaning versus execution

Treating a mapping as semantic does not make physical execution optional. A federated query may ship predicates to sources and join locally. A materialization may copy rows. A migration may rewrite all data. A cache may retain results. The claim is only that those plans can refer to one mapping rather than re-encode its meaning independently.

The distinction is also relevant to performance. If a schema and mapping compile into a backend-native plan, the categorical layer may remain outside the row-processing hot path. That is a plausible architecture, not a result of this paper. Later experiments must compare native SQL with compiled SQL, measure compilation and cache reuse, and inspect the generated plans.

11.4 Versioning consequences

Immutable schema and mapping revisions allow a result to name the exact interpretation under which it was produced. They do not by themselves solve distributed agreement. A mutable catalog name such as "current schema" requires an authority and compare-and-swap or coordination protocol. Concurrent schema transformations may not commute. Content addressing gives stable references, not convergence, availability, or authorization.

11.5 Human and organizational meaning

No type checker determines whether two source fields mean the same business concept. A mapping can encode a reviewed decision and preserve its evidence, but the decision may require domain expertise, data profiling, contractual authority, or reconciliation. Likewise, a categorical identity path says that traversing no arrows leaves an existing element unchanged. It does not say that records from two systems identify the same customer.

12 Limitations and explicit nonclaims

12.1 Profile limitations

Profile v1 does not represent:

- null, missing, error, or partial values;
- optional attributes or partial arrows;
- keys, general functional dependencies, cardinality bounds, or participation constraints as first-class declarations;
- arbitrary n-ary relations without reification;

- multivalued graph edges, edge properties, or variable-length path patterns;
- arrays, record openness, heterogeneous nesting, or ordering;
- bags, aggregation, recursion, volatility, outer joins, or update semantics;
- open-world entailment, existential ontology semantics, or inconsistency handling;
- arbitrary typeside computation or observation equations;
- probabilistic identity, confidence, authority, or reconciliation;
- general Σ, Π, chase, or Kan-extension execution; or
- distributed consistency, conflict resolution, and transaction semantics.

12.2 Evidence limitations

The paper reports no:

- implemented CatDB query compiler, storage runtime, connector, or server;
- accepted finite-instance validator or data-migration evaluator;
- SQL, graph, document, RDF, or vector backend lowering;
- benchmark, performance comparison, or integration-complexity study;
- peer-review acceptance; or
- compatibility result against a pinned CQL release.

12.3 Explicit nonclaims

We do not claim that:

1. category theory or CatDB subsumes relational, ER, property-graph, RDF/OWL, document, vector, or multimodel semantics;
2. every database constraint is a path equation;
3. finite presentations yield finite categories or a decidable general equality problem;
4. a total lawful mapping is injective, surjective, invertible, information preserving, writable, cheap, or physically executable;
5. one finite-instance check proves a law for future states;
6. a database constraint is a categorical proof;
7. a digest proves semantic equivalence, business identity, authorization, or durability;
8. vector similarity proves identity or truth;
9. cross-language agreement is a theorem;

10. the paper-level proofs verify a program; or
11. a semantic layer removes ETL, migration cost, data movement, query optimization, or distributed-systems tradeoffs.

13 Open questions

1. **Equality fragment.** Which restricted path-equation language admits a sound, terminating, useful preservation checker, and for which fragment can it be complete?
2. **Residual mapping obligations.** Should an unproved preservation obligation reject every mapping artifact, or can a nonexecutable artifact remain useful for analysis?
3. **Richer typeside.** How should scalar operations, parsers, equations, and observation equations be serialized without embedding host-language functions?
4. **Partiality.** Which explicit semantics should replace total functions when null, missing, error, or partial data is admitted?
5. **Constraint taxonomy.** How should keys, dependencies, cardinalities, policies, and temporal conditions compose with path equations?
6. **Forward migrations.** What finite quotient and canonicalization boundary would justify a bounded Σ, and what resource limits must be part of the contract?
7. **Semantic equivalence.** Which equivalence or compatibility notions beyond exact canonical equality are useful and decidable enough for tooling?
8. **Backend preservation.** How should a capability manifest state null, collation, transaction, and constraint semantics precisely enough to justify lowering?
9. **Value of obligation metadata.** Does the two-axis obligation ledger reduce debugging time, review error, or repair surface compared with existing categorical and model-management tools?
10. **Authoring ergonomics.** Can users work through domain-friendly notation while the compiled profile remains explicit and auditable?

14 Implementation status

Status. ENGINEERING HYPOTHESIS and OPEN CONJECTURE.

At the 23 July 2026 evidence cut, the manuscript remains provisional, but its first semantic-core artifacts are accepted internally. The repository contains a closed shared fixture profile, a Rust production subset, an independent Haskell reference subset, and a Lean theorem subset. The exact status is:

- the mathematical propositions remain PROVED IN THIS PAPER under their stated assumptions;

- 16 schema/path/mapping fixture outcomes are computationally supported by independent Rust and Haskell implementations;
- eight later-slice fixture envelopes are structural only at the accepted evidence snapshot;
- the nine declarations in `lean/CatDB/Coverage.lean` are machine-checked with no reported axioms at the exact six invariant boundaries in `lean/COVERAGE.md`;
- normalized-IR and diagnostic claims are evidenced only for the bounded admitted fragment and single-fault fixtures;
- instance, migration, compiler, backend, and realization statements remain ENGINEERING HYPOTHESIS or OPEN CONJECTURE;
- the general equality-decision and forward-migration boundaries are OPEN CONJECTURE;
- there are no *experimentally supported* claims; and
- final paper acceptance is blocked on clean final build, human-readable recheck, AGY paper review, PDF, digest, and cover gates.

Any later implementation change that alters the semantic profile, composition order, equality boundary, or examples makes the manuscript stale. The paper must then be rebuilt, its evidence tables updated, and its review reopened.

15 Conclusion

Categorical database theory already supplies a compositional account of schemas, instances, mappings, and migration. The question for CatDB is not whether that theory exists. It is whether a deliberately bounded portion can be carried into ordinary database engineering as explicit, versioned, testable artifacts without erasing the semantics of the systems it connects.

This paper defines such a profile for finite entities, total functional arrows, typed attributes, paths, path equations, finite instances, lawful mappings, and pullback migration. It separates the presented category from the authored syntax, normalized record, and physical realization. Under stated assumptions, typed paths form a small category, lawful mappings compose as a category, and pullback migration respects identity and reverses mapping composition. These are paper-level mathematical results rooted in established theory.

The first semantic-core boundary is now executable. Rust and Haskell agree on the admitted schema, path, and mapping cases, while Lean checks selected categorical laws at its published boundary. The broader operational layer still needs finite-instance and migration evidence, obligations classified by meaning and discharge site, preservation/loss records at physical boundaries, backend conformance, and measurable engineering benefit. CatDB therefore remains a research program with a precise falsification contract, not a completed universal data platform.

A Notation and evidence-label index

Table 7: Notation used in the profile.

Notation	Meaning
$\mathbb{T}$	Fixed many-sorted typeside theory
D	One fixed algebra interpreting the typeside
S, T, U	Finite schema presentations
E_S, G_S, A_S, R_S	Entities, generating arrows, attributes, and path equations of S
$\mathcal{C}_S$	Small entity-path category presented by S
$p : A \rightsquigarrow B$	Typed finite entity path from A to B
$\epsilon_A, \mathrm{id}_A$	Empty identity path at A
$q \circ p$	Path that traverses p and then q
$\equiv_S$	Congruence on paths generated by R_S
I, J, K	Valid finite total instances
F, G	Total typed mappings; lawful when equations are preserved
$F \approx G$	Extensional equality ignoring only mapping artifact ID
$p; a$	Attribute expression: traverse p, then observe a
$\Delta_F J$	Pullback of target instance J along lawful F
SUPPORTED BY PRIOR LITERATURE	Checked result or capability from primary literature or a normative source
PROVED IN THIS PAPER	Complete human-auditable proof in this manuscript
ENGINEERING HYPOTHESIS	Proposed operational claim requiring implementation/evaluation
OPEN CONJECTURE	Unresolved mathematical or engineering claim

B Adversarial validation catalogue

Table 8: Cases required before the operational profile can be accepted.

ID	Input or environmental fault	Unsound behavior to prevent	Required outcome
AF-01	`Order/accountOf` references a known arrow with the wrong source	Accept because the ID exists	Reject with path-source mismatch at that arrow
AF-02	Two declarations share one identifier	Map insertion silently overwrites one	Preserve duplicate evidence and reject deterministically
AF-03	A future identifier contains slash under the v1 path-key grammar	Ambiguous path key	Reject under v1 or introduce a new key version

ID	Input or environmental fault	Unsound behavior to prevent	Required outcome
AF-04	Equation sides arrive in opposite order	Different revision for the same unoriented equation	Canonicalize only after typing; retain both semantic paths
AF-05	Set-like arrays arrive in random order	Host-map order changes bytes, slots, or diagnostics	Canonically sort set-like arrays; preserve path order
AF-06	Mapping hashes are compared to test identity laws	Artifact identity replaces extensional equality	Use the explicit comparator that erases only mapping ID
AF-07	Target prover cannot derive an equality	Treat non-derivation as falsity or success	Return an unproved-in-fragment obligation and fail closed
AF-08	Relational lowering uses a nullable foreign key for a total arrow	Null source rows pass	Require non-null plus referential integrity or residual validation
AF-09	Cross-row path equation is emitted as a portable SQL `CHECK`	Apparent constraint is not maintained soundly	Use supported maintained semantics or a snapshot-tied runtime check
AF-10	Property graph has zero or two outgoing edges for one projected arrow	Pick an arbitrary edge	Reject, validate exactly-one, or report unsupported/loss
AF-11	RDF graph lacks a required predicate	Open-world absence is treated as closed-world validity	Require an explicit closed-world validation profile
AF-12	Document embeds divergent copies of one account	Copies are treated as one stable element	Report identity/duplication loss or use explicit references
AF-13	ANN returns a close customer vector	Similarity becomes canonical identity	Record candidate evidence only; discharge no identity law
AF-14	Backend capabilities change after plan compilation	Stale enforcement disposition is reused	Include capability revision in realization identity and recompile
AF-15	Equation scan reads multiple database snapshots	Zero witnesses is reported for a state that never existed	Name one coherent snapshot/transaction contract or remain inconclusive
AF-16	Backend collation differs from exact profile string equality	Native equality is treated as semantic equality	Require exact codec/collation capability or residual check
AF-17	Request asks for Σ or Π	Approximate and return success	Return exact unsupported result
AF-18	Shared artifact embeds a Rust discriminant, pointer, or driver object	Cross-language decoding or hashing depends on process layout	Reject nonportable value; use closed serialized fields

ID	Input or environmental fault	Unsound behavior to prevent	Required outcome
AF-19	Source file is reformatted	Semantic revision changes with line locations	Keep source map as sidecar to canonical semantic body
AF-20	Two schemas are categorically isomorphic but syntactically distinct	Deduplicate without an isomorphism witness	Keep distinct revisions; equivalence is outside the identity contract

C Claim-status matrix

Table 9: Major claims at the provisional manuscript cut.

ID	Claim	Evidence status
P1-C01	Categories/algebraic presentations as schemas and functor-like instances/mappings are established theory	SUPPORTED BY PRIOR LITERATURE
P1-C02	Functorial migration and executable FQL/CQL precede CatDB	SUPPORTED BY PRIOR LITERATURE
P1-C03	Typed generated paths modulo declared congruence form a small category	PROVED IN THIS PAPER, Theorem 8.1
P1-C04	A lawful assignment respects all presented equality	PROVED IN THIS PAPER, Theorem 8.2
P1-C05	Lawful mappings have identities and associative composition under the stated extensional equality	PROVED IN THIS PAPER, Theorem 8.3
P1-C06	Finite pullback preserves validity and respects identity/composition under stated assumptions	PROVED IN THIS PAPER, Theorem 8.4
P1-C07	A four-layer distinction among category, presentation, normalized record, and physical realization is useful	ENGINEERING HYPOTHESIS
P1-C08	Stable diagnostics and canonical fixtures can support independent implementations	ENGINEERING HYPOTHESIS
P1-C09	Separating semantic kind from discharge site improves executable schema governance	ENGINEERING HYPOTHESIS
P1-C10	Preservation/loss records can prevent silent model-boundary errors	ENGINEERING HYPOTHESIS
P1-C11	A useful, sufficiently complete equality fragment can be selected	OPEN CONJECTURE
P1-C12	The proposed operational contract yields measurable integration benefit	OPEN CONJECTURE

ID	Claim	Evidence status
P1-C13	Rust and Haskell agree on the admitted P1 fixture profile	computational: 16 semantic agreements, 8 structural-only cases, 0 disagreements at reviewed commit `abd655f`
P1-C14	Selected path and mapping laws are machine-checked in Lean	machine-checked only for `PATH-02`–`PATH-04` and `MAP-03`–`MAP-05` under `lean/COVERAGE.md`
P1-C15	Compiled execution is competitive with native execution	no claim; P2/P14 question

D Paper-specific artifact ledger

Table 10: Artifacts required for eventual acceptance.

Artifact	Current status	Acceptance requirement
LaTeX manuscript and bibliography	acceptance candidate	clean build, no unresolved citations/references, at least 20 substantive pages
Related-work ledger	provisional	source-audited and reconciled with final bibliography
Novelty ledger	provisional	adversarial review against CQL and model-management baselines
Disputed-claim ledger	open	every dispute resolved, retained as open, or explicitly rejected
Shared fixture corpus	accepted bounded snapshot	24 closed cases: 11 positive, 11 negative, 2 unsupported; manifest and schema gates pass
Rust P1 implementation	internally accepted	16 semantic-operation passes, 8 structural-only; terminal Claude review pass
Haskell P1 model	internally accepted	independent computation, 1,550 property cases, terminal Claude review pass
Lean P1 representation	internally accepted	nine declarations for six invariant IDs; clean build and axiom/source audit; terminal Claude review pass

Artifact	Current status	Acceptance requirement
Cross-language report	internally accepted	exact reviewed revision; 16 semantic agreements, 8 structural-only, 0 disagreements
Human-readable edit	complete with paper-local audit	abstract, introduction, and changed explanatory prose rechecked without changing formal meaning
Paper peer review	external digest-bound artifact	canonical AGY review must name the exact current source digest and end in a publishable terminal verdict
Cover image	external generated artifact	regenerated at 300 DPI from the exact reviewed PDF

References

[1] Sattam Alsubaiee, Yasser Altowim, Hotham Altwaijry, Alexander Behm, Vinayak Borkar, Yingyi Bu, Michael Carey, Inci Cetindil, Madhusudan Cheelangi, Khurram Faraaz, Eugenia Gabrielova, Raman Grover, Zachary Heilbron, Young-Seok Kim, Chen Li, Guangqiang Li, Ji Mahn Ok, Nicola Onose, Pouria Pirzadeh, Vassilis Tsotras, Rares Vernica, Jian Wen, and Till Westmann. Asterixdb: A scalable, open source bdms. *Proceedings of the VLDB Endowment*, 7(14):1905–1916, 2014.

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. Pg-schema: Schemas for property graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–25, 2023.

[3] Tim Bray. The javascript object notation (json) data interchange format. RFC 8259, Internet Engineering Task Force, 2017.

[4] Kristopher S. Brown, David I. Spivak, and Ryan Wisnesky. Categorical data integration for computational science. *Computational Materials Science*, 164:127–132, 2019.

[5] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

[6] CategoricalData. Categorical query language (cql). Official project documentation. Accessed 2026-07-22.

[7] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[8] Shumo Chu, Konstantin Weitz, Alvin Cheung, and Dan Suciu. Hottsql: Proving query rewrites with univalent sql semantics. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 510–524, 2017.

[9] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[10] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Fred Zemke. Graph pattern matching in gql and sql/pgq. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2246–2258, 2022.

[11] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[12] ISO/IEC JTC 1/SC 32. Information technology—database languages—gql. Technical Report ISO/IEC 39075:2024, International Organization for Standardization, April 2024.

[13] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[14] PostgreSQL Global Development Group. Postgresql 18 documentation: Constraints. Official documentation, 2025. Accessed 2026-07-22.

[15] Anders Rundgren, Bradley Jordan, and Samuel Erdtman. Json canonicalization scheme (jcs). RFC 8785, Internet Engineering Task Force, 2020.

[16] Patrick Schultz, David I. Spivak, Christina Vasilakopoulou, and Ryan Wisnesky. Algebraic databases. *Theory and Applications of Categories*, 32(16):547–619, 2017.

[17] Patrick Schultz, David I. Spivak, and Ryan Wisnesky. Algebraic model management: A survey. In *Recent Trends in Algebraic Development Techniques*, volume 10644 of *Lecture Notes in Computer Science*, pages 56–69. Springer, 2017.

[18] Patrick Schultz and Ryan Wisnesky. Algebraic data integration. *Journal of Functional Programming*, 27:e24, 2017.

[19] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012.

[20] David I. Spivak and Robert E. Kent. Ologs: A categorical framework for knowledge representation. *PLOS ONE*, 7(1):e24274, 2012.

[21] David I. Spivak and Ryan Wisnesky. Relational foundations for functorial data migration. In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 21–28, 2015.

[22] W3C OWL Working Group. Owl 2 web ontology language document overview (second edition). W3c recommendation, World Wide Web Consortium, December 2012.

[23] W3C RDB2RDF Working Group. R2rml: Rdb to rdf mapping language. W3c recommendation, World Wide Web Consortium, September 2012.

[24] W3C RDF Working Group. Rdf 1.1 concepts and abstract syntax. W3c recommendation, World Wide Web Consortium, February 2014.

[25] W3C SPARQL Working Group. Sparql 1.1 query language. W3c recommendation, World Wide Web Consortium, March 2013.

[26] Ryan Wisnesky, David I. Spivak, Patrick Schultz, and Eswaran Subrahmanian. Functorial data migration: From theory to practice. *arXiv preprint*, 2015.