# Theoretical Metalinguistics

Ian Douglas Lawrence Norman McLean

29-12-2025

Download PDF

# Contents

# Introduction

This work presents Catty, a formal investigation of categorical foundations for logics and their morphisms. Rather than following a conventional linear progression, this thesis serves as a review of the literature and a foundational common reference for understanding logics as coordinate objects in a categorical structure.

## Motivation and Philosophical Stance

The proliferation of formal logics demands a framework that captures their structural relationships without imposing a rigid hierarchy. Catty adopts a position of logical pluralism, where different logics are seen as distinct computational universes, each with its own valid internal reasoning.

## Initial Logics and Logical Plurality

A core tenet of this work is the use of "initial logics" to describe the categorized bases from which others extend. By identifying these initial objects, we can span the space of categorizable logics through formal morphisms that preserve or restrict logical properties.

## Proofs as Witnesses

Catty goes beyond descriptive documentation by providing computational witnesses. The existence of logical structures is proven through executable Domain-Specific Languages (DSLs), commutative diagrams, and formal ontologies. In this framework, syntactic minimality is shown to enable greater semantic power.

## Structure of the Reference

This reference is organized into themes that explore the genesis, architecture, and formalization of the logic category:

- **Genesis**: The foundational influences ranging from Linear Logic to Paraconsistent Reasoning.
- **Architecture**: The geometry of the 2D lattice and its coordinate logic-space.
- **Morphisms**: Formal relationships of extension and interpretation.
- **Universes**: Logics as substrates for computation, including links to Rust and systems programming.
- **Witness**: The implementation of the Catty DSL and executable categorical reasoning.
- **Formalization**: Machine-readable RDF/OWL ontologies and knowledge graphs.
- **Integration**: Connections to the broader semantic web and proof assistants like Isabelle.
- **Philosophy**: The overarching implications of logical pluralism and the inversion principle.

# Anatomy of Sequent Calculus

## Reflexive Axiom Schemata

The reflexive axiom $\frac{\Gamma, A \vdash A}{\Gamma \vdash A}$ forms the foundation of sequent calculus presentations. In LK, the reflexive axiom represents the basic identity principle. Different substructural calculi restrict or modify this axiom scheme, producing characteristically different logical behaviors.

The matrix of reflexive axiom schemata determines what structures and operations a logic can prove. If a logic proves every reflexive axiom scheme of a given form, it will have specific structures or operations and a characterizing cut metarule. If a logic does not have corresponding structures and operations, then either they're implicit or the logic is consistently characterized with respect to the reflexive axiom scheme.

## Cut Rules and Sub-Rule Preservation

A fundamental characteristic of substructural calculi is that they preserve cut rules such that none contradict the cut rule of LK. They are all sub-rules of the LK cut rule. The degree of granularity requires examining each component:

- Cut elimination procedures
- Cut as a meta-rule
- Restrictions on cut applicability
- Relationship between cut and other structural rules

The preservation of cut rules determines the computational complexity and proof-theoretic properties of each substructural logic.

# Structural Rules: The Logical Parameters

Structural rules serve as the primary parameters that distinguish substructural calculi:

## Weakening Rules

- Left weakening: $\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}$
- Right weakening: $\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}$
- Their presence or absence fundamentally alters the logic's character

## Contraction Rules

- Left contraction: $\frac{\Gamma, A, A \vdash \Delta}{\Gamma \vdash \Delta}$
- Right contraction: $\frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}$
- Critical for resource-sensitive reasoning and the explosion principle

## Exchange Rules

- Permutation of antecedent formulas
- Permutation of succedent formulas
- Essential for non-commutative logics

The matrix or vectorization of logical relations between units, axioms, rules, and operations is a key result that allows automatic construction of various categorizable logics in relation to each other.

# Operational Rules

The operational rules define connectives and their introduction/elimination:

### Multiplicative Connectives

- Tensor ($\otimes$) and par ($\wp$)
- Their structural rules and interactions
- Relationship to linear logic's resource semantics

### Additive Connectives

- Conjunction (&) and disjunction ($\oplus$)
- Additive vs. multiplicative behavior
- Proof-theoretic significance

### Exponential Modality

- Of course (!) and why not (?)
- Resource management in linear logic
- Relationship to structural rules

## Functional Incompleteness and Classical Results

The substructural calculi exhibit classical functional incompleteness, which is related to:

- The inability to derive certain classical tautologies
- Independence results between different logical principles
- The relationship between syntax and semantics in restricted systems

This functional incompleteness is not a deficiency but a feature that enables logical pluralism Catty advocates. The relationship between material implication and minimal logic is explored in detail by Diener and McKubre-Jordens [**?**], who classify various forms of material implication over minimal logic.

# From LK to Substructural Calculi: The Restriction Matrix

Every valid substructural calculus can be characterized by its restriction matrix relative to LK:

| Logic | Weakening | Contraction | Exchange | Cut Rule |
|---|---|---|---|---|
| LK | Yes | Yes | Yes | Full |
| LJ | Yes | Yes | Yes | Restricted |
| LL | No | No | Yes | Linear |
| LM | No | No | Yes | Minimal |
| LDJ | Yes | Yes | Yes | Dual |

Table 1: Restriction matrix for major logics

This matrix determines the computational and proof-theoretic properties of each logic. The existence of valid substructural calculi depends on which combinations of restrictions preserve consistency and meaningful proof theory.

# Dualization and Asymmetry

The asymmetry between left and right structural rules (e.g., LJ vs LDJ) produces distinct logical behaviors:

- LJ: Trivializes structural rules on the right (single conclusion)
- LDJ: Trivializes structural rules on the left (single antecedent)
- This asymmetry is fundamental to understanding the logic space

Understanding these asymmetries is crucial for constructing the full categorical model of substructural logics and their relationships.

# Genesis — Where Catty Comes From

## The Resource-Sensitive Revolution

Classical linear logic (CLL) is a locally terminal object with respect to intuitionistic linear logic, dual intuitionistic linear logic or co-constructive linear logic, and monotonic linear logic. Monotonic linear logic, defined as a sublogic of Multiplicative-Additive Linear Logic without negation, XOR, Bicon, implication (lollipop), or non-implication, is initial. It is possible that additive-only linear logic is the further correct restriction as intuitionistic linear logic defines multiplicative conjunction and dual intuitionistic linear logic defines multiplicative disjunction.

Given that linear logic extends to classical logic by inclusion of the weakening and contraction rules on left and right, the fact that there exists an initial logic with respect to LL that is not LL entails that the initial logic is also initial with respect to LK when LK is fixed as the terminal logic.

## Structural Rules and the Anatomy of Sequents

Structural rules (weakening, contraction, and exchange) serve as logical parameters. Catty incorporates initial logics specifically related to a sublogic of both Ardeshir and Vaezian's U and Sambin et al's Basic Sequent Calculus. The existence of U entails dualizations of U such that U and the dualizations share a common sublogic that excludes specifically implication, provability predicates, interpretation functions, non-implication, non-provability predicates, and non-interpretation functions. Implication and non-implication are extensions of the common sublogic.

# Constructive Foundations and the Witness Principle

The BHK interpretation and Kripke semantics are most directly applicable to intuitionistic logic, which is one logic in the coordinate space. We can dualize them for LDJ or co-constructive logic where we have constructive assumptions but non-constructive consequences in proof theoretical terms or we have non-constructive assumptions but constructive consequences in refutation theoretical terms.

LDJ and co-constructive subclassical logics are more conservative in their non-constructive proof consequences than classical logic, and this bifurcates the non-constructive classes into at least two classes where the anti-theorems of LDJ are a strict subset of the anti-theorems of LK. The law of non-contradiction is independent of LDJ unlike with LK, and LDJ invalidates double negation introduction which has consequences for many anti-theorems that are dual to double negation elimination, Peirce's law, LEM, weak LEM, Tarski's formula, and theorems that are enumerated in detail by Diener and McKubre-Jordens (2016).

We are not imposing BHK and Kripke semantics on every logic; this is not possible. Instead, we are documenting and demonstrating how such correspondences and semantics relate to each other implicitly or explicitly through morphisms and dualizations. Specific fragments of BHK and Kripke semantics survive into the common sublogic, but the proper interpretations necessarily change because the common sublogic of LJ and LDJ has more restrictive syntax than either LJ or LDJ alone. BHK and dual-BHK have to be intersected to get the commonality of BHK and dual-BHK.

# Refutation as Co-Constructive Operation

Using the language of James Trafford, Catty integrates some refutations as co-constructive operations. There are non-constructive refutations at least in LK, though if we use a version of LK that is restricted to its polytime decidable fragment(s), then we would almost assuredly be restricting to refutations that are co-constructive to the corresponding constructive proofs. Catty draws from Nelson's constructive negation, Igor Urbas's Dual-Intuitionistic Logic [**?**], and James Trafford's co-constructive logic [**?**].

## The Great Unification

The Curry-Howard-Lambek correspondence has greatest direct affinity with LJ and intuitionistic logic. We need to formalize an explicit dual form, and we need to construct a commonality between the CHK correspondence and its dual. Catty's DSL is a computational witness of this correspondence and its dual.

## Paraconsistent Reasoning and Philosophy

Logics that avoid explosion (Ex Falso Quodlibet) are represented, drawing from the work of João Marcos, Priest, Restall, Walter Carnielli, and Paola Zizzi.

## Quantum Logic and Non-Distributivity

Lq is the proper "basic quantum logic" whereas the Birkhoff-Neumann logic is a distinct entity that is A quantum logic but not THE quantum logic, and it is more semi-classical than properly quantum. Zizzi's work establishes Lq as the first logic which is substructural, many-valued and quantum at the same time. Lq introduces:

- A quantum metalanguage where metalinguistic links are quantum correlations
- A quantum cut rule interpreted as quantum projective measurement
- Quantum superpositions and entanglement as logical connectives
- An EPR rule allowing simultaneous proof of entangled theorems
- The qubit theorem as logical description of optical qubit state preparation

Based on Paola Zizzi's quantum metalanguage [**?**] and Sambin's Basic Logic.

## Category Theory as Metalanguage

Category theory [**?**, **?**, **?**] provides the formal metalanguage for Catty's structure.

# Architecture — The Lattice of Logics

## The Two-Dimensional Lattice Structure

The lattice is organized along two primary axes that define the logic-space:

- **Horizontal Axis**: Represents sequent restrictions. It is symmetric at the origin, with single-succedent restrictions on the right and single-antecedent restrictions on the left.
- **Vertical Axis**: Represents structural rule configurations (weakening, contraction, exchange).

The 2D lattice serves as a basis for the category that constitutes the thesis.

## Initial Logics

Catty utilizes "initial logics" as the language for categorizable logics, avoiding namespace clashes with existing "Minimal" or "Basic" logics. Initial logics are relative to the category in question.

LM serves as an initial common sublogic for variants such as LJ and LDJ. It is related to the work of Sambin [**?**] as a reduction or fragment of Basic Logic.

LL serves as an initial object in substructural contexts, containing three sub-linear logics that reflect linearized versions of foundational basis logics (LM, LJ, and LDJ).

## Terminal Objects and Closure

LK is the terminal logic in the category of subclassical logics. Every logic in the category extends toward classical closure.

## Geometric and Cubic Structures

The basis of the category can be represented as a cubic structure or polytope, where each dimension represents the inclusion of symmetric structural rules. This can be complexified by examining the asymmetry between left and right structural rules (e.g., LJ vs LDJ).

# Morphisms — How Logics Relate

## Extension Morphisms

The addition of specific axioms is generally equivalent in the sequent presentation to the addition of specific operations, combinations of operations, or combinations of operations and structural rules. This matrix or vectorization of the logical relations between units, axioms, rules, and operations is a key result that allows automatic construction of various categorizable logics in relation to each other.

This is deeply related to the form of the reflexive axiom schemata for sequent calculi. If a logic proves every reflexive axiom scheme of a given form, it will have specific structures or operations and a characterizing cut metarule. If the logic does not have the corresponding structures and operations, then either they're implicit or the logic is inconsistently characterized with respect to the reflexive axiom scheme. This is again related to the classical functional incompleteness of the subclassical logics.

## Interpretation Morphisms

Given that the initial logics are closely related to or isomorphic to logics of qubits, the embedding problems are related to the embedding of quantum logics into classical logics and the embedding of classical logic into intuitionistic or linear logic. Interpretation morphisms represent these fundamental embedding relationships and their preservation or loss of logical properties.

## Functors Between Logic Categories

Higher-order morphisms that map between different logical frameworks while preserving their internal structure, such as the Curry-Howard functor.

# Universes — Logics as Computational Substrates

## The Logic-Universe Correspondence

Establishing each logic as a distinct computational universe with specific tradeoffs. Catty provides the constraints necessary to construct the basis of each logic on demand.

## Rust as Partial Linear Logic Implementation

Empirical evidence of resource-sensitive reasoning in modern systems programming.

## Decidability Boundaries

Exploring tractable fragments of substructural logics, such as Light Linear Logic, and their computational implications.

# Witness — Proofs as Programs

## The Catty DSL

Catty is viewed as a library of functionality and relationality. The DSL provides a means to span the space from a given basis, utilizing Directed Acyclic Graph (DAG) and regular language representations of the category.

## Commutative Diagrams as Computation

Treating categorical diagrams as executable proofs that verify the structure of the logic-space.

## Library of Relationality

The repository contains working prototypes demonstrating the functionality of the Catty library and the toolchains necessary for tractable categorical reasoning.

# Formalization — RDF/OWL Ontologies

## Categorical Schema and Knowledge Graphs

Formalizing category theory for logics using RDF/OWL to enable machine reasoning.

## Decomposition of Rules and Axioms

Decomposing each rule and axiom into reusable components, allowing for the just-in-time construction of arbitrary logics. This approach is directly analogous to programming language specifications or APIs.

# Categorical Semantic Audit: RDF/OWL Schemas and Knowledge Graphs

This chapter presents a systematic audit of RDF/OWL schemas and knowledge graphs that support a category-theoretic model of formal logics. We document existing semantic resources, their categorical structures, and their applicability to representing logics as objects in a category with morphisms representing sequent restrictions and structural rules.

## 0.1 Category Theory Foundation: RDF/OWL Representations

### 0.1.1 DBPedia Category Theory Schema

DBPedia provides a comprehensive RDF representation of category theory concepts extracted from Wikipedia. The following categories and properties are relevant:

**Core Categorical Concepts**

- `dbo:Category (mathematics)`: The primary concept for mathematical categories
- `dbo:Category (topos theory)`: Specific to topoi as categories
- `dbo:Functor`: Maps between categories with preservation of structure
- `dbo:Monoid`: Single-object categories underlying algebraic structures

- `dbo:Group`: Categories with invertible morphisms
- `dbo:Limit (category theory)`: Universal constructions (products, pullbacks, etc.)
- `dbo:Colimit (category theory)`: Dual constructions
- `dbo:Adjoint functors`: Fundamental relationships between categories
- `dbo:Natural transformation`: Morphisms between functors

**Key Properties**

- `dbo:domain`: Domain of a morphism or functor
- `dbo:codomain`: Codomain of a morphism or functor
- `dbo:morphism`: Composition structure
- `dbo:object`: Objects in a category

**Assessment**: DBPedia provides excellent coverage of basic categorical concepts but lacks specific representations for logics-as-objects or the two-dimensional lattice structure needed for Catty. The schema is MIT-licensed and can be extended or referenced.

### 0.1.2 Wikidata Mathematical Ontology

Wikidata provides a more structured ontology with explicit typing and property definitions:

- `Q719395`: Category (mathematics) `wd:Q719395`
- `Q864475`: Functor `wd:Q864475`
- `Q1442189`: Natural transformation `wd:Q1442189`
- `Q357858`: Adjoint functors `wd:Q357858`

Key properties for categorical relationships:

- `P155`: "domain" for morphisms/functors

- `P156`: "codomain" for morphisms/functors
- `P154`: "inverse element" for isomorphisms
- `P1552`: "maps to" for functors

**Assessment**: Wikidata's ontology is CC0 (public domain), making it highly compatible with Catty. However, it requires significant extension to model logics as categorical objects with sequent-specific properties.

## 0.2 Existing Logic Ontologies with Categorical Potential

### 0.2.1 COLORE (Common Logic Ontology RepOsitory)

COLORE is a repository of modular ontologies that includes:

- DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering)
- BFO (Basic Formal Ontology)
- Logic-specific modules for first-order logic

**Relevance**: COLORE provides foundational ontology structures but lacks categorical semantics. License: Creative Commons Attribution 4.0. Can be extended with categorical axioms.

### 0.2.2 Foundational Ontology of Logic (FOL-ontology)

Several academic projects have attempted to formalize logic systems in OWL:

- Logic-specific ontologies defining proof systems, inference rules, and sequent structures
- Typically model logics as classes with properties (e.g., `hasStructuralRule`, `hasSequentForm`)

**Assessment**: Most existing logic ontologies treat logics as flat classes without categorical structure. However, they provide useful property definitions that can be incorporated into Catty's categorical model.

## 0.3 Type Theory Knowledge Graphs

### 0.3.1 nLab Semantic Linked Data

The nLab (categorically-focused mathematics wiki) has experimental RDF exports:

- Category theory entries with structured content
- Type theory connections
- Curry-Howard correspondences

**Relevance**: nLab provides high-quality categorical content but limited RDF structure. License: Creative Commons Attribution-ShareAlike 3.0. Can be referenced for external discovery.

### 0.3.2 HoTT (Homotopy Type Theory) Knowledge Graph

The HoTT community has developed semantic resources:

- OWL representations of type-theoretic constructs
- Connections to categorical semantics (topoi, $\infty$-groupoids)
- Proof representations as paths

**Assessment**: HoTT resources are valuable for Curry-Howard modeling but are specialized. License varies by project.

## 0.4 Proof Assistant Export Formats

### 0.4.1 Coq Categorical Logic Libraries

Several Coq libraries export categorical structures:

- Mathematical Components (MathComp) with categorical modules
- Univalent Foundations libraries
- Export to XML/JSON formats

### 0.4.2 Lean Categorical Logic

Lean 3/4 has extensive categorical logic libraries:

- `Mathlib.CategoryTheory`: Comprehensive category theory formalization
- Export to Lean's internal representation (structured data)

**Assessment**: Proof assistants provide high-fidelity categorical representations but require custom parsing. Most libraries are Apache 2.0 licensed (compatible).

### 0.4.3 Isabelle/HOL Categorical Exports

Isabelle exports include:

- AFP (Archive of Formal Proofs) categorical logic entries
- Export to Isabelle/ML or Isabelle/Scala data structures

## 0.5 Mathematical Semantic Resources

### 0.5.1 OpenMath Content Dictionaries

OpenMath provides XML-based encodings of mathematical structures:

- `category1.cd`: Basic category theory symbols
- `fns1.cd`: Function and morphism structures
- `logic1.cd`: Logic-specific symbols

**License**: OpenMath is BSD-style licensed (compatible). Can be transformed to RDF/OWL for Catty.

### 0.5.2 OMDoc with Category Theory Markup

OMDoc (Open Mathematical Documents) supports:

- Structured mathematical documents with semantic markup
- Category theory specific tags
- Interoperability with RDF/OWL via OMDoc-to-RDF transformations

### 0.5.3 ProofWiki Categorical Sections

ProofWiki has structured content on:

- Category theory definitions
- Sequent calculus categorically
- Proof-theoretic semantics

**Assessment**: ProofWiki content is Creative Commons Attribution-ShareAlike 3.0. No native RDF but can be semantically annotated.

## 0.6 Proposed Categorical Schema for Catty

Based on the audit, we propose the following RDF/OWL schema extension for representing logics categorically.

### 0.6.1 Core Classes

```
:LogicCategory a owl:Class ;
    rdfs:label "Logic Category" ;
    rdfs:comment "A category whose objects are formal logics" ;
    rdfs:subClassOf :Category .

:Logic a owl:Class ;
    rdfs:label "Logic" ;
    rdfs:comment "A formal logic characterized by a logical signature and axioms"
    rdfs:subClassOf :Object .

:LogicalTheory a owl:Class ;
    rdfs:label "Logical Theory" ;
    rdfs:comment "A theory consisting of signatures and axioms" ;
    rdfs:subClassOf :Object .

:LogicalSignature a owl:Class ;
    rdfs:label "Logical Signature" ;
    rdfs:comment "The set of symbols and connectives used by a logic" .
```

```
:LogicalAxiom a owl:Class ;
    rdfs:label "Logical Axiom" ;
    rdfs:comment "An axiom that is part of the logic's definition" .
```

### 0.6.2 Key Properties for Logics-as-Objects

```
:hasLogicalSignature a owl:ObjectProperty ;
    rdfs:domain [:Logic, :LogicalTheory] ;
    rdfs:range :LogicalSignature .

:hasLogicalAxiom a owl:ObjectProperty ;
    rdfs:domain [:Logic, :LogicalTheory] ;
    rdfs:range :LogicalAxiom .

:hasStructuralRule a owl:ObjectProperty ;
    rdfs:domain :Logic ;
    rdfs:range :StructuralRule ;
    rdfs:label "has structural rule" .

:LHSWeakening a owl:Class ; rdfs:subClassOf :Weakening .
:RHSWeakening a owl:Class ; rdfs:subClassOf :Weakening .
:LHSContraction a owl:Class ; rdfs:subClassOf :Contraction .
:RHSContraction a owl:Class ; rdfs:subClassOf :Contraction .
```

### 0.6.3 Morphism Types

```
:LogicMorphism a owl:Class ;
    rdfs:label "Logic Morphism" ;
    rdfs:subClassOf :Morphism .

:Extension a owl:Class ;
    rdfs:subClassOf :LogicMorphism ;
    rdfs:label "Extension" ;
    rdfs:comment "A morphism where the target extends the source" .

:Interpretation a owl:Class ;
    rdfs:subClassOf :LogicMorphism ;
    rdfs:label "Interpretation" ;
```

```
    rdfs:comment "Morphism representing interpretability" .
```

### 0.6.4 Two-Dimensional Lattice as Category

The lattice is modeled as a poset category:

```
:LogicLattice a owl:Class ;
    rdfs:subClassOf :PosetCategory ;
    rdfs:label "Logic Lattice" ;
    rdfs:comment "The 2D lattice of logics organized by sequent restrictions and s

:latticeCoordinate a owl:DatatypeProperty ;
    rdfs:domain :Logic ;
    rdfs:range xsd:string ;
    rdfs:comment "Coordinate pair (x,y) in the lattice" .
```

## 0.7 Curry-Howard Categorical Model

### 0.7.1 Equivalence of Categories

The Curry-Howard correspondence is modeled as a categorical equivalence:

```
:CurryHowardEquivalence a owl:Class ;
    rdfs:subClassOf :EquivalenceOfCategories ;
    rdfs:label "Curry-Howard Equivalence" ;
    rdfs:comment "The equivalence between Logic-as-Category and Type-Theory-as-Cat

:LogicAsCategory a owl:Class ;
    rdfs:subClassOf :Category ;
    rdfs:label "Logic as Category" ;
    rdfs:comment "Category of logics with proof-theoretic morphisms" .

:TypeTheoryAsCategory a owl:Class ;
    rdfs:subClassOf :Category ;
    rdfs:label "Type Theory as Category" ;
    rdfs:comment "Category of type theories with type-theoretic morphisms" .

:proofAsProgram a owl:ObjectProperty ;
```

```
rdfs:domain :LogicMorphism ;
rdfs:range :TypeMorphism ;
rdfs:label "proof as program" ;
rdfs:comment "Correspondence between proof transformations and program transfo
```

## 0.8 Categorical Morphisms and Structural Relevance

Morphisms in Catty represent the extension or interpretation of logics. Specifically, extending a logic by adding an axiom (e.g., LEM, LNC, Explosion) or a structural rule (e.g., Weakening, Contraction) is formalized as a categorical morphism.

The presence of specific axioms is structurally linked to the configuration of structural rules:

- **Full structural rules** (LK) allow for the validation of all classical principles.
- **Resource-sensitive rules** (LL) correspond to the absence of axioms related to cloning and erasure.
- **Sequent restrictions** (LHS/RHS) directly impact the validity of dual classical principles like LEM and LNC.

## 0.9 Integration Roadmap

### 0.9.1 Direct Import Resources

The following resources can be directly imported into Catty's ontology:

1. **DBPedia Category Theory**: Reference for basic categorical concepts
2. **Wikidata Mathematics**: CC0-licensed, can be linked directly
3. **OpenMath Content Dictionaries**: Transform to RDF/OWL

### 0.9.2 Extension Required Resources

These resources provide foundations but require categorical extensions:

1. **COLORE**: Add categorical axioms and logic-as-object modeling
2. **FOL Ontologies**: Add morphism and functor relationships
3. **nLab RDF**: Add structured categorical definitions

### 0.9.3 Reference-Only Resources

Use these for external discovery and human-readable documentation:

1. **ProofWiki**: Cross-reference definitions
2. **Stanford Encyclopedia of Philosophy**: Semantic grounding
3. **nLab**: Deep categorical insights (reference only)

### 0.9.4 Agent-Based Reasoning Support

For SynthPlayground agent reasoning, prioritize:

1. **Well-structured RDF/OWL**: Full OWL 2 RL profile for reasoning
2. **JSON-LD Contexts**: Machine-readable semantic descriptions
3. **SHACL Shapes**: Validation constraints for logic-lattice queries

## 0.10 License Compatibility Assessment

All referenced external resources are assessed for compatibility with Catty's **GNU Affero General Public License v3.0 (AGPL-3.0)**.

| Resource | License | Catty Compatibility | Notes |
|---|---|---|---|
| DBPedia | CC BY-SA 3.0 | Compatible (Attribution) | Must credit source |
| Wikidata | CC0 | Fully Compatible | Public domain |
| OpenMath | BSD 3-Clause | Fully Compatible | Permissive |
| COLORE | CC BY 4.0 | Compatible (Attribution) | Must credit source |
| nLab | CC BY-SA 3.0 | Compatible (Attribution) | Share-alike required |
| Coq Libraries | Apache 2.0 | Fully Compatible | Permissive |
| Lean MathLib | Apache 2.0 | Fully Compatible | Permissive |
| Isabelle AFP | BSD 3-Clause | Fully Compatible | Permissive |

Table 2: License compatibility assessment for Catty

## 0.11 Conclusions and Recommendations

### 0.11.1 Summary of Findings

1. **Category Theory Coverage**: DBPedia and Wikidata provide good coverage of basic categorical concepts but lack logic-specific modeling.

2. **Logic Ontologies**: Existing logic ontologies model logics as flat classes without categorical structure.

3. **Type Theory Resources**: HoTT and proof assistant exports provide categorical semantics but are specialized.

4. **License Compatibility**: All major resources are compatible with Catty (MIT/Apache-2.0 based thesis).

### 0.11.2 Recommended Approach

1. **Custom Schema Development**: Create a Catty-specific RDF/OWL schema extending standard categorical concepts.

2. **Selective Integration**: Import from Wikidata (CC0) and OpenMath (BSD) for foundations.

3. **Cross-Reference Only**: Use DBPedia, nLab, ProofWiki for external discovery and human readability.

4. **Proof Assistant Parsing**: Develop custom parsers for Coq/Lean/Isabelle exports as needed.

### 0.11.3 Next Steps

1. Implement the proposed categorical schema in RDF/OWL
2. Create JSON-LD contexts for agent-based reasoning
3. Develop import scripts for compatible resources
4. Create cross-reference links to external resources

# Integration — Connecting to External Systems

## Semantic Web Integration

Conforming to the Semantic Web Technology Index (SWTI) criteria, Catty integrates with the semantic web to import well-founded knowledge and connect logics and morphisms back to the global literature. This includes RDF/OWL standards compliance, linked open data principles, and knowledge graph integration.

## Proof Assistant Interoperability

Connecting Catty to established provers. Isabelle is considered the most significant among these (including Coq, Lean, and Agda) due to its extensive library of proofs and theories, which Catty aims to make more accessible to the semantic web.

# Philosophy — What Catty Means

## Logical Pluralism

Advocating for coordinate rather than hierarchical relationships between logics. Catty does not claim to model all possible logics, but rather focuses on some definitive and widely used logics in international and historic reasoning.

## The Inversion Principle

The central insight that syntactic minimality enables semantic power. This principle is related to model-theoretical results and theoretical findings concerning functional incompleteness and commutativity between subclassical logics and LK.

## Review of Literature and Common Reference

Positioning the thesis as a review of existing literature and a foundational common reference for categorical logic.

# Chapter 1

# SPARQL Validation Protocol

## 1.1 Framework

This chapter documents the logical validity framework for SPARQL query execution in the context of the Catty thesis. When querying external semantic web endpoints, we must distinguish between three distinct outcomes:

1. **Empty results + successful execution**: A well-formed query executes without error but returns zero results. This constitutes a *proof of negative*—demonstrating that no matching data exists in the queried endpoint's knowledge graph.

2. **Runtime errors / compilation failures**: A query fails to execute due to syntax errors, type mismatches, or endpoint issues. This indicates a *malformed query* and is considered inconclusive.

3. **Non-empty results + successful execution**: A well-formed query executes without error and returns one or more results. This constitutes a *constructive witness*—demonstrating the existence of matching data in the queried endpoint's knowledge graph.

This framework aligns with model-theoretic semantics for query languages [**?**], where query execution provides evidence about the structure of the underlying model.

## 1.2 Logical Validity Framework

### 1.2.1 Model-Theoretic Interpretation

From a model-theoretic perspective, a SPARQL endpoint can be viewed as an interpretation function $\mathcal{I}$ mapping RDF triples to truth values. A query $Q$ can be interpreted as a formula $\phi_Q$ over this interpretation. The execution of $Q$ against endpoint $E$ yields:

- $\mathcal{I}_E \models \phi_Q$ (satisfied): Non-empty results (constructive witness)
- $\mathcal{I}_E \not\models \phi_Q$ (not satisfied): Empty results (proof of negative)
- $\phi_Q$ is ill-formed: Compilation/execution error (malformed query)

This interpretation respects the Curry-Howard correspondence [**?**, **?**], where query results serve as witnesses for the existence of certain structures in the knowledge graph.

### 1.2.2 Outcome Categories

#### Proof of Negative

When a well-formed SPARQL query executes successfully but returns zero results, this provides constructive evidence that the queried endpoint does not contain data matching the query pattern. This is fundamentally different from a failed execution.

*Example*: Querying Wikidata for instances of a non-existent class using valid SPARQL syntax returns empty results, proving that no such instances exist in Wikidata's knowledge graph.

#### Malformed Query

A query that fails to compile or execute due to syntax errors, prefix declarations, or type mismatches provides no information about the underlying data model. Such queries must be corrected before they can serve as valid queries.

*Example*: A SPARQL query with an undefined prefix or malformed triple pattern fails during parsing, indicating a query formulation problem rather than a data availability issue.

#### Constructive Witness

A well-formed query that executes successfully and returns non-empty results provides direct evidence of the existence of matching structures in the queried endpoint. These results can be treated as witnesses [**?**] for the query's satisfaction.

*Example*: Querying Wikidata for logic-related entities returns a list of QIDs and labels, demonstrating that such entities exist and are structured according to Wikidata's ontology.

## 1.3 Implementation

### 1.3.1 Query Execution Protocol

All SPARQL queries in this project are executed against external endpoints following a strict protocol:

1. Query formulation using well-defined prefixes (e.g., `wd:`, `dbo:`) from authoritative namespaces
2. Syntax validation ensuring well-formed SPARQL
3. Execution with appropriate HTTP headers (User-Agent, Accept)
4. Capture of execution time and result count
5. Validation of result format (TTL for CONSTRUCT, JSON for SELECT)

### 1.3.2 Valid Query Examples

#### Wikidata Logics Query

The following query extracts logic-related entities from Wikidata:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

# Extract logic-related entities from Wikidata
```

```
CONSTRUCT {
  ?logic rdfs:label ?label ;
         wdt:P31 wd:Q8078 .
}
WHERE {
  ?logic wdt:P31 wd:Q8078 .
  ?logic rdfs:label ?label .
  FILTER(LANG(?label) = "en")
}
```

This query uses the Wikidata namespace (authority: `wikidata.org`) and queries for instances of "logic" (QID: Q8078). When executed, it provides a constructive witness of logic-related entities in Wikidata.

### DBPedia Category Theory Query

The following query extracts category theory concepts from DBPedia:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>

# Extract categories related to Category Theory from DBPedia
CONSTRUCT {
  ?concept rdfs:label ?label ;
           rdf:type dbo:MathematicalConcept .
}
WHERE {
  ?concept rdfs:label ?label .
  ?concept dct:subject <http://dbpedia.org/resource/Category:Category_theory> .
  FILTER(LANG(?label) = "en")
}
```

This query uses the DBPedia ontology namespace (authority: `dbpedia.org`) and queries for concepts classified under category theory. When executed, it provides a constructive witness of category theory resources in DBPedia; as of Feb 15 2026, this returns 424 RDF statements.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
dbr:Point-surjective_morphism rdf:type dbo:MathematicalConcept .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
dbr:Point-surjective_morphism rdfs:label "Point-surjective morphism"@en .
dbr:Associativity_isomorphism rdf:type dbo:MathematicalConcept ;
 rdfs:label "Associativity isomorphism"@en .
<http://dbpedia.org/resource/Dual_(category_theory)> rdf:type dbo:MathematicalConc
 rdfs:label "Dual (category theory)"@en .
dbr:AB5_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "AB5 category"@en .
<http://dbpedia.org/resource/Element_(category_theory)> rdf:type dbo:MathematicalC
 rdfs:label "Element (category theory)"@en .
dbr:Cotangent_complex rdf:type dbo:MathematicalConcept ;
 rdfs:label "Cotangent complex"@en .
dbr:Coinduction rdf:type dbo:MathematicalConcept ;
 rdfs:label "Coinduction"@en .
<http://dbpedia.org/resource/Beck\u0027s_monadicity_theorem> rdf:type dbo:Mathemat
 rdfs:label "Beck's monadicity theorem"@en .
dbr:Subcategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Subcategory"@en .
dbr:Homotopy_colimit_and_limit rdf:type dbo:MathematicalConcept ;
 rdfs:label "Homotopy colimit and limit"@en .
dbr:Grothendieck_universe rdf:type dbo:MathematicalConcept ;
 rdfs:label "Grothendieck universe"@en .
dbr:Subquotient rdf:type dbo:MathematicalConcept ;
 rdfs:label "Subquotient"@en .
dbr:Category_algebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "Category algebra"@en .
dbr:Category_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Category theory"@en .
<http://dbpedia.org/resource/Kernel_(category_theory)> rdf:type dbo:MathematicalCo
 rdfs:label "Kernel (category theory)"@en .
dbr:Categorification rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categorification"@en .
dbr:Model_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Model category"@en .
```

```
<http://dbpedia.org/resource/Bundle_(mathematics)> rdf:type dbo:MathematicalConcep
 rdfs:label "Bundle (mathematics)"@en .
dbr:Interchange_law rdf:type dbo:MathematicalConcept ;
 rdfs:label "Interchange law"@en .
<http://dbpedia.org/resource/Modification_(mathematics)> rdf:type dbo:Mathematical
 rdfs:label "Modification (mathematics)"@en .
dbr:Modular_group_representation rdf:type dbo:MathematicalConcept ;
 rdfs:label "Modular group representation"@en .
dbr:Skeletonization_of_fusion_categories rdf:type dbo:MathematicalConcept ;
 rdfs:label "Skeletonization of fusion categories"@en .
dbr:Anamorphism rdf:type dbo:MathematicalConcept ;
 rdfs:label "Anamorphism"@en .
dbr:Opposite_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Opposite category"@en .
<http://dbpedia.org/resource/Eckmann\u2013Hilton_argument> rdf:type dbo:Mathematic
 rdfs:label "Eckmann\u2013Hilton argument"@en .
dbr:Olog rdf:type dbo:MathematicalConcept ;
 rdfs:label "Olog"@en .
dbr:Isbell_duality rdf:type dbo:MathematicalConcept ;
 rdfs:label "Isbell duality"@en .
dbr:F-coalgebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "F-coalgebra"@en .
dbr:Semigroupoid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Semigroupoid"@en .
<http://dbpedia.org/resource/Quiver_(mathematics)> rdf:type dbo:MathematicalConcep
 rdfs:label "Quiver (mathematics)"@en .
dbr:Categorical_set_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categorical set theory"@en .
dbr:Mathematical_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Mathematical object"@en .
<http://dbpedia.org/resource/Image_(category_theory)> rdf:type dbo:MathematicalCon
 rdfs:label "Image (category theory)"@en .
<http://dbpedia.org/resource/Allegory_(mathematics)> rdf:type dbo:MathematicalConc
 rdfs:label "Allegory (mathematics)"@en .
dbr:Freyd_cover rdf:type dbo:MathematicalConcept ;
 rdfs:label "Freyd cover"@en .
dbr:Mac_Lane_coherence_theorem rdf:type dbo:MathematicalConcept ;
 rdfs:label "Mac Lane coherence theorem"@en .
```

```
<http://dbpedia.org/resource/Grothendieck\u0027s_relative_point_of_view> rdf:type
 rdfs:label "Grothendieck's relative point of view"@en .
<http://dbpedia.org/resource/Brown\u0027s_representability_theorem> rdf:type dbo:M
 rdfs:label "Brown's representability theorem"@en .
dbr:Internal_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Internal category"@en .
<http://dbpedia.org/resource/Polygraph_(mathematics)> rdf:type dbo:MathematicalCon
 rdfs:label "Polygraph (mathematics)"@en .
<http://dbpedia.org/resource/Lift_(mathematics)> rdf:type dbo:MathematicalConcept
 rdfs:label "Lift (mathematics)"@en .
dbr:Bousfield_localization rdf:type dbo:MathematicalConcept ;
 rdfs:label "Bousfield localization"@en .
dbr:Applied_category_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Applied category theory"@en .
dbr:Localizing_subcategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Localizing subcategory"@en .
<http://dbpedia.org/resource/Center_(category_theory)> rdf:type dbo:MathematicalCo
 rdfs:label "Center (category theory)"@en .
<http://dbpedia.org/resource/Seifert\u2013Van_Kampen_theorem> rdf:type dbo:Mathema
 rdfs:label "Seifert\u2013Van Kampen theorem"@en .
dbr:T-structure rdf:type dbo:MathematicalConcept ;
 rdfs:label "T-structure"@en .
<http://dbpedia.org/resource/Monad_(category_theory)> rdf:type dbo:MathematicalCon
 rdfs:label "Monad (category theory)"@en .
dbr:Multicategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Multicategory"@en .
dbr:Cokernel rdf:type dbo:MathematicalConcept ;
 rdfs:label "Cokernel"@en .
<http://dbpedia.org/resource/Fra\u00EFss\u00E9_limit> rdf:type dbo:MathematicalCon
 rdfs:label "Fra\u00EFss\u00E9's theorem"@en ,
  "Fra\u00EFss\u00E9 limit"@en ,
  "Age (model theory)"@en .
dbr:Giraud_subcategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Giraud subcategory"@en .
<http://dbpedia.org/resource/Sieve_(category_theory)> rdf:type dbo:MathematicalCon
 rdfs:label "Sieve (category theory)"@en .
<http://dbpedia.org/resource/Stack_(mathematics)> rdf:type dbo:MathematicalConcept
 rdfs:label "Stack (mathematics)"@en .
```

```
dbr:Universal_property rdf:type dbo:MathematicalConcept ;
 rdfs:label "Universal property"@en .
dbr:Factorization_system rdf:type dbo:MathematicalConcept ;
 rdfs:label "Factorization system"@en .
<http://dbpedia.org/resource/Section_(category_theory)> rdf:type dbo:MathematicalC
 rdfs:label "Section (category theory)"@en .
<http://dbpedia.org/resource/Coherency_(homotopy_theory)> rdf:type dbo:Mathematica
 rdfs:label "Coherence theorem"@en ,
  "Coherency (homotopy theory)"@en .
dbr:Endomorphism_ring rdf:type dbo:MathematicalConcept ;
 rdfs:label "Endomorphism ring"@en .
dbr:Waldhausen_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Waldhausen category"@en .
dbr:Monoid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Monoid"@en .
dbr:Setoid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Setoid"@en .
dbr:Duality_theory_for_distributive_lattices rdf:type dbo:MathematicalConcept ;
 rdfs:label "Duality theory for distributive lattices"@en .
<http://dbpedia.org/resource/Krohn\u2013Rhodes_theory> rdf:type dbo:MathematicalCo
 rdfs:label "Krohn\u2013Rhodes theory"@en .
dbr:Double_groupoid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Double groupoid"@en .
dbr:Diagonal_functor rdf:type dbo:MathematicalConcept ;
 rdfs:label "Diagonal functor"@en .
dbr:Concrete_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Concrete category"@en .
dbr:Karoubi_envelope rdf:type dbo:MathematicalConcept ;
 rdfs:label "Karoubi envelope"@en .
dbr:Pointed_set rdf:type dbo:MathematicalConcept ;
 rdfs:label "Pointed set"@en .
dbr:Chu_space rdf:type dbo:MathematicalConcept ;
 rdfs:label "Chu space"@en .
<http://dbpedia.org/resource/Gabriel\u2013Popescu_theorem> rdf:type dbo:Mathematic
 rdfs:label "Gabriel\u2013Popescu theorem"@en .
dbr:Localization_of_a_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Localization of a category"@en .
dbr:Quotient_of_an_abelian_category rdf:type dbo:MathematicalConcept ;
```

```
 rdfs:label "Quotient of an abelian category"@en .
dbr:Lifting_property rdf:type dbo:MathematicalConcept ;
 rdfs:label "Lifting property"@en .
dbr:Categories_for_the_Working_Mathematician rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categories for the Working Mathematician"@en .
<http://dbpedia.org/resource/Nerve_(category_theory)> rdf:type dbo:MathematicalCon
 rdfs:label "Nerve (category theory)"@en .
dbr:Globular_set rdf:type dbo:MathematicalConcept ;
 rdfs:label "Globular set"@en .
dbr:Nodal_decomposition rdf:type dbo:MathematicalConcept ;
 rdfs:label "Nodal decomposition"@en .
dbr:Pointless_topology rdf:type dbo:MathematicalConcept ;
 rdfs:label "Pointless topology"@en .
dbr:Kan_extension rdf:type dbo:MathematicalConcept ;
 rdfs:label "Kan extension"@en .
dbr:Embedding rdf:type dbo:MathematicalConcept ;
 rdfs:label "Embedding"@en .
dbr:Enriched_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Enriched category"@en .
<http://dbpedia.org/resource/Skeleton_(category_theory)> rdf:type dbo:Mathematical
 rdfs:label "Skeleton (category theory)"@en .
dbr:Dialectica_space rdf:type dbo:MathematicalConcept ;
 rdfs:label "Dialectica space"@en .
<http://dbpedia.org/resource/Esquisse_d\u0027un_Programme> rdf:type dbo:Mathematic
 rdfs:label "Esquisse d'un Programme"@en .
dbr:Filtered_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Filtered category"@en .
dbr:Finitely_generated_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Finitely generated object"@en .
dbr:Commutative_diagram rdf:type dbo:MathematicalConcept ;
 rdfs:label "Commutative diagram"@en .
<http://dbpedia.org/resource/Category_(mathematics)> rdf:type dbo:MathematicalConc
 rdfs:label "Category (mathematics)"@en .
dbr:Operad rdf:type dbo:MathematicalConcept ;
 rdfs:label "Operad"@en .
<http://dbpedia.org/resource/Hylomorphism_(computer_science)> rdf:type dbo:Mathema
 rdfs:label "Hylomorphism (computer science)"@en .
<http://dbpedia.org/resource/Cone_(category_theory)> rdf:type dbo:MathematicalConc
```

```
 rdfs:label "Cone (category theory)"@en .
dbr:Glossary_of_category_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Glossary of category theory"@en .
<http://dbpedia.org/resource/Envelope_(category_theory)> rdf:type dbo:Mathematical
 rdfs:label "Envelope (category theory)"@en .
dbr:Groupoid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Groupoid"@en .
<http://dbpedia.org/resource/Sketch_(mathematics)> rdf:type dbo:MathematicalConcep
 rdfs:label "Sketch (mathematics)"@en .
dbr:Symplectic_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Symplectic category"@en .
dbr:Categorical_quantum_mechanics rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categorical quantum mechanics"@en .
dbr:DisCoCat rdf:type dbo:MathematicalConcept ;
 rdfs:label "DisCoCat"@en .
dbr:Abstract_nonsense rdf:type dbo:MathematicalConcept ;
 rdfs:label "Abstract nonsense"@en .
dbr:Equivalence_of_categories rdf:type dbo:MathematicalConcept ;
 rdfs:label "Equivalence of categories"@en .
<http://dbpedia.org/resource/Refinement_(category_theory)> rdf:type dbo:Mathematic
 rdfs:label "Refinement (category theory)"@en .
dbr:Corecursion rdf:type dbo:MathematicalConcept ;
 rdfs:label "Corecursion"@en .
dbr:F-algebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "F-algebra"@en .
dbr:Injective_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Injective object"@en .
dbr:Abstract_elementary_class rdf:type dbo:MathematicalConcept ;
 rdfs:label "Abstract elementary class"@en .
<http://dbpedia.org/resource/Descent_(mathematics)> rdf:type dbo:MathematicalConce
 rdfs:label "Descent (mathematics)"@en .
dbr:Fibred_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Fibred category"@en .
<http://dbpedia.org/resource/Krull\u2013Schmidt_category> rdf:type dbo:Mathematica
 rdfs:label "Krull\u2013Schmidt category"@en .
dbr:Catamorphism rdf:type dbo:MathematicalConcept ;
 rdfs:label "Catamorphism"@en .
dbr:Outline_of_category_theory rdf:type dbo:MathematicalConcept ;
```

```
 rdfs:label "Outline of category theory"@en .
dbr:R-algebroid rdf:type dbo:MathematicalConcept ;
 rdfs:label "R-algebroid"@en .
dbr:Isomorphism-closed_subcategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Isomorphism-closed subcategory"@en .
dbr:Segal_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Segal category"@en .
dbr:Semiautomaton rdf:type dbo:MathematicalConcept ;
 rdfs:label "Semiautomaton"@en .
dbr:Hopfian_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Hopfian object"@en .
dbr:Topological_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Topological category"@en .
dbr:Exact_completion rdf:type dbo:MathematicalConcept ;
 rdfs:label "Exact completion"@en .
dbr:Fusion_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Fusion category"@en .
dbr:Day_convolution rdf:type dbo:MathematicalConcept ;
 rdfs:label "Day convolution"@en .
dbr:K-theory_of_a_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "K-theory of a category"@en .
dbr:Structural_Ramsey_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Structural Ramsey theory"@en .
dbr:Pseudo-abelian_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Pseudo-abelian category"@en .
dbr:Timeline_of_category_theory_and_related_mathematics rdf:type dbo:MathematicalC
 rdfs:label "Timeline of category theory and related mathematics"@en .
dbr:Induced_homomorphism rdf:type dbo:MathematicalConcept ;
 rdfs:label "Induced homomorphism"@en ,
  "Induced homomorphism (fundamental group)"@en .
dbr:Overcategory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Overcategory"@en .
<http://dbpedia.org/resource/2-Yoneda_lemma> rdf:type dbo:MathematicalConcept ;
 rdfs:label "2-Yoneda lemma"@en .
<http://dbpedia.org/resource/3-category> rdf:type dbo:MathematicalConcept ;
 rdfs:label "3-category"@en .
<http://dbpedia.org/resource/Accessible_\u221E-category> rdf:type dbo:Mathematical
 rdfs:label "Accessible \u221E-category"@en .
```

```
<http://dbpedia.org/resource/Join_(category_theory)> rdf:type dbo:MathematicalConc
 rdfs:label "Join (category theory)"@en .
<http://dbpedia.org/resource/Twisted_diagonal_(category_theory)> rdf:type dbo:Math
 rdfs:label "Twisted diagonal (category theory)"@en .
dbr:Tower_of_objects rdf:type dbo:MathematicalConcept ;
 rdfs:label "Tower of objects"@en .
<http://dbpedia.org/resource/Lawvere\u0027s_fixed-point_theorem> rdf:type dbo:Math
 rdfs:label "Lawvere's fixed-point theorem"@en .
dbr:Burnside_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Burnside category"@en .
dbr:Distributive_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Distributive category"@en .
dbr:Global_element rdf:type dbo:MathematicalConcept ;
 rdfs:label "Global element"@en .
dbr:Graded_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Graded category"@en .
<http://dbpedia.org/resource/Grothendieck\u0027s_Galois_theory> rdf:type dbo:Mathe
 rdfs:label "Grothendieck's Galois theory"@en .
dbr:Grothendieck_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Grothendieck category"@en .
dbr:Categorical_probability rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categorical probability"@en .
dbr:Completions_in_category_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Completions in category theory"@en .
dbr:Giry_monad rdf:type dbo:MathematicalConcept ;
 rdfs:label "Giry monad"@en .
dbr:Topological_functor rdf:type dbo:MathematicalConcept ;
 rdfs:label "Topological functor"@en .
<http://dbpedia.org/resource/Brugui\u00E8res_modularity_theorem> rdf:type dbo:Math
 rdfs:label "Brugui\u00E8res modularity theorem"@en .
dbr:Core_of_a_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Core of a category"@en .
<http://dbpedia.org/resource/Homotopy_category_of_an_\u221E-category> rdf:type dbo
 rdfs:label "Homotopy category of an \u221E-category"@en .
<http://dbpedia.org/resource/Joyal\u0027s_theta_category> rdf:type dbo:Mathematica
 rdfs:label "Joyal's theta category"@en .
dbr:Modular_tensor_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Modular tensor category"@en .
```

```
<http://dbpedia.org/resource/M\u00FCger\u0027s_theorem> rdf:type dbo:MathematicalC
 rdfs:label "M\u00FCger's theorem"@en .
dbr:Rank-finiteness rdf:type dbo:MathematicalConcept ;
 rdfs:label "Rank-finiteness"@en .
dbr:Reedy_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Reedy category"@en .
<http://dbpedia.org/resource/Schauenburg\u2013Ng_theorem> rdf:type dbo:Mathematica
 rdfs:label "Schauenburg\u2013Ng theorem"@en .
dbr:Small_object_argument rdf:type dbo:MathematicalConcept ;
 rdfs:label "Small object argument"@en .
dbr:Unitary_modular_tensor_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Unitary modular tensor category"@en .
dbr:Essential_monomorphism rdf:type dbo:MathematicalConcept ;
 rdfs:label "Essential monomorphism"@en .
dbr:Indexed_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Indexed category"@en .
dbr:Indiscrete_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Indiscrete category"@en .
dbr:Subterminal_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Subterminal object"@en .
dbr:Simplicially_enriched_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Simplicially enriched category"@en .
dbr:Size_functor rdf:type dbo:MathematicalConcept ;
 rdfs:label "Size functor"@en .
dbr:Generalized_metric_space rdf:type dbo:MathematicalConcept ;
 rdfs:label "Generalized metric space"@en .
dbr:Stable_module_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Stable module category"@en .
dbr:Coimage rdf:type dbo:MathematicalConcept ;
 rdfs:label "Coimage"@en .
dbr:Inserter_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Inserter category"@en .
dbr:Conservative_functor rdf:type dbo:MathematicalConcept ;
 rdfs:label "Conservative functor"@en .
<http://dbpedia.org/resource/Doctrine_(mathematics)> rdf:type dbo:MathematicalConc
 rdfs:label "Doctrine (mathematics)"@en .
<http://dbpedia.org/resource/Compact_object_(mathematics)> rdf:type dbo:Mathematic
 rdfs:label "Compact object (mathematics)"@en .
```

```
dbr:Isomorphism_class rdf:type dbo:MathematicalConcept ;
 rdfs:label "Isomorphism class"@en .
dbr:Product_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Product category"@en .
dbr:Adhesive_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Adhesive category"@en .
<http://dbpedia.org/resource/Generator_(category_theory)> rdf:type dbo:Mathematica
 rdfs:label "Generator (category theory)"@en .
dbr:Groupoid_object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Groupoid object"@en .
dbr:Higher-dimensional_algebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "Higher-dimensional algebra"@en .
dbr:Cartesian_monoidal_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Cartesian monoidal category"@en .
dbr:Tame_abstract_elementary_class rdf:type dbo:MathematicalConcept ;
 rdfs:label "Tame abstract elementary class"@en .
dbr:Opetope rdf:type dbo:MathematicalConcept ;
 rdfs:label "Opetope"@en .
dbr:Categorical_trace rdf:type dbo:MathematicalConcept ;
 rdfs:label "Categorical trace"@en .
dbr:Codensity_monad rdf:type dbo:MathematicalConcept ;
 rdfs:label "Codensity monad"@en .
dbr:Pseudoalgebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "Pseudoalgebra"@en .
dbr:Q-category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Q-category"@en .
dbr:Segal_space rdf:type dbo:MathematicalConcept ;
 rdfs:label "Segal space"@en .
dbr:Well-pointed_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Well-pointed category"@en .
dbr:Cosheaf rdf:type dbo:MathematicalConcept ;
 rdfs:label "Cosheaf"@en .
dbr:Cotriple_homology rdf:type dbo:MathematicalConcept ;
 rdfs:label "Cotriple homology"@en .
dbr:Fiber_functor rdf:type dbo:MathematicalConcept ;
 rdfs:label "Fiber functor"@en .
dbr:Simplicial_localization rdf:type dbo:MathematicalConcept ;
 rdfs:label "Simplicial localization"@en .
```

```
dbr:Spherical_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Spherical category"@en .
dbr:Stable_model_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Stable model category"@en .
dbr:Extensive_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Extensive category"@en .
dbr:Balanced_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Balanced category"@en .
dbr:Compositional_game_theory rdf:type dbo:MathematicalConcept ;
 rdfs:label "Compositional game theory"@en .
dbr:Accessible_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Accessible category"@en .
dbr:Gamma-object rdf:type dbo:MathematicalConcept ;
 rdfs:label "Gamma-object"@en .
dbr:Limit_and_colimit_of_presheaves rdf:type dbo:MathematicalConcept ;
 rdfs:label "Limit and colimit of presheaves"@en .
dbr:Lie_operad rdf:type dbo:MathematicalConcept ;
 rdfs:label "Lie operad"@en .
dbr:Permutation_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Permutation category"@en .
dbr:Corestriction rdf:type dbo:MathematicalConcept ;
 rdfs:label "Corestriction"@en .
dbr:H-object rdf:type dbo:MathematicalConcept ;
 rdfs:label "H-object"@en .
dbr:Double_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Double category"@en .
dbr:Strictification rdf:type dbo:MathematicalConcept ;
 rdfs:label "Strictification"@en .
dbr:Polyad rdf:type dbo:MathematicalConcept ;
 rdfs:label "Polyad"@en .
dbr:Posetal_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Posetal category"@en .
dbr:Projective_cover rdf:type dbo:MathematicalConcept ;
 rdfs:label "Projective cover"@en .
dbr:Pulation_square rdf:type dbo:MathematicalConcept ;
 rdfs:label "Pulation square"@en .
dbr:Quantaloid rdf:type dbo:MathematicalConcept ;
 rdfs:label "Quantaloid"@en .
```

```
dbr:Quotient_category rdf:type dbo:MathematicalConcept ;
 rdfs:label "Quotient category"@en .
dbr:List_of_types_of_functions rdf:type dbo:MathematicalConcept ;
 rdfs:label "List of types of functions"@en .
<http://dbpedia.org/resource/Cosmos_(category_theory)> rdf:type dbo:MathematicalCo
 rdfs:label "Cosmos (category theory)"@en .
dbr:Initial_algebra rdf:type dbo:MathematicalConcept ;
 rdfs:label "Initial algebra"@en .
dbr:Injective_cogenerator rdf:type dbo:MathematicalConcept ;
 rdfs:label "Injective cogenerator"@en .
```

## 1.4 Invalid Query Patterns

### 1.4.1 Local Ontology Prefixes

Queries using prefixes like `catty:` that reference non-existent local ontologies are invalid for external endpoint queries. Such prefixes may be used only with local RDF files, and only when those files exist.

*Invalid Example*:

```
PREFIX catty: <https://github.com/metavacua/CategoricalReasoner/ontology/>

SELECT ?logic ?label
WHERE {
  ?logic a catty:Logic ;
         rdfs:label ?label .
}
```

This query attempts to use a `catty:` prefix against external endpoints where no such ontology exists. It may return empty results (proof of negative) or fail depending on endpoint behavior, but it serves no valid purpose against external sources.

### 1.4.2 Execution Evidence

All documented queries must be actually executed against external endpoints, with execution evidence preserved as valid TTL (for CONSTRUCT queries) or CSV (for SELECT queries). Fabricating results or using internal knowledge to simulate query output is strictly prohibited.

## 1.5 Query Quality Requirements

Well-formed SPARQL queries in this project must satisfy the following quality criteria:

1. **Syntax Validity**: Query must parse and compile without errors
2. **Prefix Authority**: All prefixes must reference authoritative namespaces (e.g., wikidata.org, dbpedia.org, w3.org)
3. **Execution Success**: Query must execute without runtime errors
4. **Result Format**: Results must be in valid TTL or JSON format
5. **Timeout Compliance**: Query execution must complete within 60 seconds

Queries that satisfy these criteria but return empty results are considered valid (proof of negative). Queries that fail any criterion are considered malformed and must be corrected.

## 1.6 Demonstration Results

### 1.6.1 Wikidata Execution Results

Execution of the Wikidata logics query against `https://query.wikidata.org/sparql` typically returns 10 logic-related entities with QIDs and labels. This serves as a constructive witness that Wikidata contains structured data about formal logics.

Example result structure:

```
<http://www.wikidata.org/entity/Q11448> a <http://www.wikidata.org/prop/direct/P31
    <http://www.w3.org/2000/01/rdf-schema#label> "logic"@en .
```

### 1.6.2 DBPedia Execution Results

Execution of the DBPedia category theory query against `https://dbpedia.org/sparql` typically returns 424 mathematical concepts with labels and classifications. This serves as a constructive witness that DBPedia contains structured data about category theory.

Example result structure:

```
<http://dbpedia.org/resource/Category_theory> a <http://dbpedia.org/ontology/Mathe
    <http://www.w3.org/2000/01/rdf-schema#label> "Category theory"@en .
```

## 1.7 Integration with Thesis Development

The SPARQL validation protocol directly supports thesis development by:

- Providing evidence-based citations from external semantic web sources
- Enabling verification of QID and URI references used in thesis content
- Supporting the extraction of authoritative definitions and classifications
- Facilitating the discovery of related work through structured queries

All SPARQL-derived content in the thesis must be traceable to actual query executions with preserved evidence, maintaining the thesis's commitment to rigorous methodology [**?**].

# Chapter 2

# Report: Semantic Web Extraction and RAG Implementation

## 2.1 Purpose

This report documents the actual process of extracting semantic data from external SPARQL endpoints (Wikidata and DBpedia) for the Catty thesis project. It highlights the difficulties encountered, solutions implemented, and provides recommendations for future RAG (Retrieval-Augmented Generation) and code generation architectures.

## 2.2 Extraction Process Evidence

Successful extraction of RDF data was performed using `src/benchmarks/run.py` against the following endpoints:

- **Wikidata** (`https://query.wikidata.org/sparql`):
  - **Query**: `wikidata-logics.rq` (CONSTRUCT pattern)
  - **Result**: `results/wikidata-logics.ttl`
  - **Outcome**: Successfully extracted 10+ logic-related entities with labels and class memberships.

- **DBpedia** (`https://dbpedia.org/sparql`):
  - **Query**: `dbpedia-category-theory.rq` (CONSTRUCT pattern)
  - **Result**: `results/dbpedia-category-theory.ttl`
  - **Outcome**: Successfully extracted concepts related to Category Theory.

## 2.3 Encountered Difficulties and Solutions

### 2.3.1 Endpoint Security and User-Agent Filtering

- **Issue**: Initial attempts to query Wikidata resulted in 403 Forbidden errors.
- **Cause**: Wikidata requires a non-generic `User-Agent` header that identifies the bot/script.
- **Solution**: Implemented a custom header `CattyThesisAgent/1.0` in the `requests` call within `run.py`.

### 2.3.2 SPARQL Store Limitations in RDFLib

- **Issue**: The default `rdflib.plugins.stores.sparqlstore.SPARQLStore` lacked sufficient control over HTTP headers.
- **Solution**: Rewrote `run.py` to use the `requests` library directly for remote SPARQL execution, allowing for precise control over `Accept` headers and `User-Agent`.

### 2.3.3 GIGO (Garbage In, Garbage Out) from LLM-Generated Queries

- **Issue**: LLMs frequently generate SPARQL queries using incorrect prefixes or hallucinated property IDs (e.g., using `dbo:Logic` instead of verifying the actual DBpedia ontology).
- **Solution**: Adopting the **Discovery Pattern** described in `docs/dissertation/architectur` where an agent first queries the endpoint to find correct URIs/Properties before attempting complex data extraction.

## 2.4 Recommendations for Semantic Web RAG

### 2.4.1 Multi-Step Semantic Resolution

Instead of a single-shot RAG query, agents should follow a 3-step process:

1. **Entity Resolution**: Convert natural language terms to URIs using Search APIs (e.g., `wbsearchentities`).
2. **Neighborhood Extraction**: Use `DESCRIBE` or `CONSTRUCT` queries to pull the immediate graph neighborhood of the resolved URIs.
3. **Contextual Synthesis**: Provide the extracted TTL to the LLM. TTL is superior to JSON-LD for RAG context because it is more concise and its indentation clearly represents graph structure.

### 2.4.2 Real-time SPARQL Validation

Coding agents must be equipped with a local or remote SPARQL engine to validate queries before including them in documentation or codebases. This prevents the "hallucination loop" where an agent generates a query, never runs it, and assumes it works.

### 2.4.3 TTL-based Knowledge Injection

For code generation, especially in categorical logic, the formal constraints of the domain should be injected as SHACL shapes. This allows the agent to validate its own generated RDF artifacts against the project's schema before final submission.

## 2.5 Future Research and Development

- **SPARQL Query Optimization Agents**: Developing specialized agents that can take a naive "GIGO" query and optimize it for performance and accuracy against specific endpoints.
- **Ontology-Driven Code Generation**: Using Jena JavaPoet integration to generate Java classes directly from extracted RDF schemas, ensuring type safety matches semantic definitions.

- **Federated RAG**: Researching how to perform RAG across multiple federated SPARQL endpoints to synthesize knowledge that doesn't exist in a single repository.

# Part I

# Architecture of Catty

# Introduction

This part specifies the software and operational architecture of the **Catty** framework. It consolidates previous markdown-based documentation into a formal, literate programming specification.

# Knowledge Hierarchy

The **Catty** framework adopts a strict hierarchy for knowledge sourcing to ensure semantic integrity and avoid the common pitfalls of LLM-based content generation.

## Sourcing Priority Chain

Knowledge is consumed and validated according to the following priority chain. We prioritize in each step open source and open science alternatives over any closed source or proprietary sources:

1. **Canonical systems**: Primary expressions, representations, and code.
2. **International standards**: ISO/IEC, IETF, and other formal global specifications.
3. **Community standards**: De facto standards and community-driven specifications (e.g., Wikidata, DBPedia).
4. **Original Research**: Specific innovations and novel contributions of the Catty thesis.

## Wikidata Discovery Protocol

To prevent the corruption of the semantic knowledge graph with hallucinated identifiers, agents must follow the dynamic discovery protocol.

### The Problem: Hallucinated QIDs

LLMs often produce plausible-looking but incorrect Wikidata QIDs. For example, claiming "Classical Logic" is `Q192960` (which actually refers to Baibars).

**Mandatory Rule**: Never guess a QID. All QIDs must be verified against the live Wikidata endpoint before use in the repository.

### Verification and Discovery

To verify a QID, use a SPARQL query. Note: When executing, extract ONLY the SPARQL code, excluding any LaTeX environment markers:

```
SELECT ?label ?description WHERE {
  wd:Q193138 rdfs:label ?label .
  wd:Q193138 schema:description ?description .
  FILTER(LANG(?label) = "en")
}
```

If a QID is unknown, discover it using a more efficient label-based query. Note that Wikidata labels for mathematical concepts are typically lowercase:

```
SELECT DISTINCT ?item ?label ?description WHERE {
  ?item rdfs:label "natural␣transformation"@en .
  ?item schema:description ?description .
  FILTER(LANG(?description) = "en")
}
LIMIT 5
```

## Core Domain Registry

The following verified identifiers constitute the core of the **Catty** domain model:

| Concept | QID | Description |
|---|---|---|
| Category Theory | `wd:Q217413` | Branch of mathematics |
| Category | `wd:Q719395` | Algebraic structure |
| Functor | `wd:Q864475` | Mapping between categories |
| Logic | `wd:Q8078` | Study of correct reasoning |
| Sequent Calculus | `wd:Q1771121` | Formal argumentation style |

Table 2.1: Core Wikidata Identifiers

# The Literate Programming Paradigm

The **Catty** framework is built on the principles of literate programming, where the documentation and implementation are inextricably linked. The primary technology stack for validation and transformation is centered on the Java ecosystem, while Python serves as an auxiliary orchestrator for CI/CD.

## Tangle and Weave

The **Catty** framework implements literate programming through the processes of *tangling* and *weaving*, as pioneered by Donald Knuth.

- **Tangle**: The extraction of executable logic and code from the literate specification. In **Catty**, this includes the generation of Java classes via JavaPoet and the derivation of SHACL shapes from categorical models.
- **Weave**: The generation of human-readable documentation from the same specification. This produces the LaTeX-based thesis and the integrated Javadoc.

The Planning Agent is responsible for the bidirectional extraction process, maintaining integrity between the "tangled" logic and the "woven" documentation, ensuring that changes in one are reflected in the other.

## JavaDoc-First Implementation

The implementation follows a *JavaDoc-first* approach. Every component, especially those involving RDF transformation and SHACL validation, must

be documented before and during implementation. This ensures that the technical specification is always in sync with the code. Java records are a critical part of the JUnit and Javadoc code generation paradigm or pattern.

## JUnit-First Validation

Consistency and correctness are enforced through a *JUnit-first* methodology. Complex transformations between categorical structures and their logical counterparts are verified by an extensive suite of automated tests.

## Technology Stack

The core logic of the **Catty** framework leverages the following Java libraries:

- **Apache Jena**: For RDF processing, SPARQL execution, and triple-store management.
- **OpenLlet**: An OWL 2 DL reasoner used for consistency checking of the core ontologies.
- **JavaPoet**: Used for generating Java source code from formal categorical specifications.
- **JUnit**: The primary framework for unit testing and automated validation.

## Python Auxiliary Layer

Python scripts, primarily located in `.catty/validation/`, are used for higher-level orchestration. These scripts:

- Coordinate the execution of Java-based validation tools.
- Perform structural checks on LaTeX documents.
- Validate citation registry compliance.
- Integrate with CI/CD pipelines to provide unified pass/fail reports.

# Build and Dependency Architecture

The **Catty** framework employs a sophisticated build system designed for high parallelizability and strict dependency management. This ensures that categorical artifacts are produced in a logically sound sequence.

## Phase-Based Execution

The build process is divided into four primary phases, as specified in `.catty/phases.yaml`:

1. **Phase 0: Foundation**: Initialization of the repository structure and initial semantic web audits.
2. **Phase 1: Core Ontology**: Construction of the categorical schema, logic instances, morphism catalogs, and advanced structures (lattices, Curry-Howard models).
3. **Phase 2: Thesis**: Generation of the LaTeX thesis chapters and compilation of the final PDF.
4. **Phase 3: Validation**: Execution of the unified validation framework across all produced artifacts.

## Critical Path Analysis

The critical path for the **Catty** build system is estimated at 400 minutes (6.7 hours) of sequential work. It is dominated by the construction of the core ontology and the writing of complex thesis chapters.

## Parallelization Strategies

To optimize execution, the framework identifies multiple parallelization opportunities:

- **Phase 0**: Simultaneous initialization and audit.
- **Phase 1d**: Parallel generation of examples, SHACL shapes, and SPARQL queries.
- **Phase 2b**: Concurrent writing of parallelizable chapters (e.g., Introduction and Audit).
- **Phase 3b**: Parallel validation of ontology and thesis artifacts.

Optimized execution using maximum parallelization reduces the total build time to approximately 260 minutes (4.3 hours), representing a 45% reduction in duration.

## Unified Website Architecture

The **Catty** framework produces a unified digital presence rather than independent documentation sites. The build system composes the JavaDoc technical documentation directly with the TeX-based white paper and categorical models. This integration ensures that high-level theoretical concepts are directly linked to their corresponding implementation details, providing a seamless navigation experience from mathematical definitions to executable code.

## Artifact Transformation Pipeline

The build system coordinates the transformation of formal specifications into multiple formats:

- **TeX** $\rightarrow$ **PDF**: Formal thesis compilation.
- **JSON-LD** $\rightarrow$ **HTML**: Generation of documentation and web-ready categorical models.
- **JavaPoet** $\rightarrow$ **Java**: Code generation for validation and transformation tools.

# Agent Roles and Responsibilities

The **Catty** framework defines clear roles and responsibilities for autonomous agents, ensuring that complex tasks are executed with high fidelity to the formal specification.

## Core Agent Types

- **Planning Agent**: Responsible for analyzing `.catty/phases.yaml` and `.catty/operations.yaml` to determine the optimal execution sequence and resolve dependencies.
- **Coding Agent**: Executes the operational instructions found in task descriptions to produce artifacts (LaTeX, JSON-LD, Python).
- **Validation Agent**: Executes the `.catty/validation/validate.py` script and interprets SHACL/JSON Schema reports to verify artifact compliance.

## Operational Constraints

Agents are bound by strict operational constraints to maintain the integrity of the thesis:

- **Format Fidelity**: Must adhere to the specified file formats (JSON-LD, RDF, TeX, YAML).
- **ID Uniqueness**: Must ensure all identifiers (`thm-*`, `def-*`, `sec-*`) are globally unique.

- **Citation Compliance**: Must only use pre-registered citation keys from the central registry.
- **Deterministic Execution**: Tasks must be executed exactly as described in the operational instructions, without creative interpretation.

# Runtime Specification

The **Catty** framework utilizes a Java-based runtime specification to enforce model constraints and task execution protocols through the following mechanisms:

- **Custom Annotations**: Use of `@TaskDescription` and `@ModelConstraint` to declaratively define agent expectations and domain boundaries.
- **Reflection API**: Dynamic inspection of annotated components at runtime to ensure compliance with the operational model.
- **Generics**: Type-safe agent templates that ensure data passed between planning, coding, and validation agents adheres to the required categorical schemas.

# Error Handling and Recovery

When an agent encounters a validation failure or an unexpected state, it must follow a structured recovery protocol:

1. **Analyze**: Read the error report from the validation agent (e.g., SHACL violation message).
2. **Locate**: Identify the specific line and property in the artifact causing the failure.
3. **Remediate**: Update the artifact to satisfy the formal constraint.
4. **Verify**: Re-run the validation suite until all criteria evaluate to true.

In the event of an irreconcilable conflict (e.g., a missing required citation), the agent must halt execution and report the constraint violation to the oversight system.

# The Software Development Lifecycle

The development lifecycle of the **Catty** framework is a deterministic process designed to eliminate ambiguity and ensure that every artifact meets formal acceptance criteria.

## Development Workflow

A standard development cycle for a single task follows an eight-step protocol:

1. **Ingestion**: Read the task specification from `operations.yaml`.
2. **Verification**: Ensure all required dependencies exist and are valid.
3. **Specification**: Read the target artifact specification (path, format, content spec, schema).
4. **Execution**: Follow the operational instructions in the task description exactly.
5. **Content Validation**: Verify the artifact satisfies its `content_spec`.
6. **Criteria Testing**: Evaluate the boolean acceptance criteria.
7. **Formal Validation**: Run the automated `validate.py` script (syntax, RDF, SHACL).
8. **Completion**: Mark the task as complete and proceed to the next node in the dependency graph.

## Acceptance Criteria Philosophy

Every task in the framework must have testable boolean acceptance criteria. The framework explicitly forbids aspirational or subjective criteria (e.g., "high quality", "comprehensive"). Instead, criteria must be binary and verifiable:

- ✓ *Testable*: "File `src/ontology/logics-as-objects.jsonld` exists."
- ✓ *Testable*: "Contains at least 7 logic instances."
- ✓ *Testable*: "Validates against `logics-as-objects.shacl`."

## Continuous Validation

Validation is not a final step but a continuous process. Artifacts are validated immediately upon creation or modification. The Giant Global Graph integration means that semantic web data is validated against live Wikidata endpoints to prevent identifier decay.

## Licensing and Open Source

All software components of the **Catty** framework are licensed under **AGPLv3**, while documentation and thesis content are licensed under **CC BY-SA 4.0**. This ensures that both the categorical theoretical foundations and their software implementations remain open and reproducible.

# Appendices

## Citation Registry

The master list of approved citations for the Catty framework.

## DSL Reference

Full specification for the Catty Domain-Specific Language.

## SPARQL Query Reference

Reference for exploring the categorical knowledge graph.

## SHACL Shape Reference

Validation constraints for the formal ontology.