Requiem: Academic Bibliography

Technical Foundations for Dependent Type Theory Implementation

December 20, 2025

Abstract

This bibliography provides the academic foundations for Requiem, a minimal dependently-typed language implementing Higher-Order Abstract Syntax (HOAS), Normalization by Evaluation (NbE), bidirectional type checking, and Martin-Löf identity types. Papers are organized by topic with annotations explaining their relevance to the implementation.

Contents

1 Normalization by Evaluation (NbE)

1.1 Definitive References

- **Abel, Andreas** (2013). *Normalization by Evaluation: Dependent Types and Impredicativity*. Habilitation thesis.

  Annotation: The definitive work on NbE for dependent type theory. Covers predicative universe hierarchies (directly applicable to Requiem's `Type_0 : Type_1 : Type_2` hierarchy), eta-equality for functions and pairs (implemented in Requiem's `raise`/`lower`), and semantic domains. Essential for understanding the theoretical foundations.

- **Altenkirch, Thorsten and Kaposi, Ambrus** (2016). "Normalisation by Evaluation for Dependent Types." *FSCD 2016.*

  Annotation: Modern formulation using presheaf categories and quotient inductive types. Includes partial Agda formalization showing type-directed reflection/reification (the `raise`/`lower` pattern). Provides categorical semantics justifying Requiem's semantic domain separation.

- **Abel, Andreas; Vezzosi, Andrea; Winterhalter, Théo** (2017). "Normalization by Evaluation for Sized Dependent Types." *ICFP 2017.*

  Annotation: Extends NbE to handle sized types while maintaining decidability. The type-directed reflection/reification algorithms are directly implemented in Requiem's `raise` and `lower` functions. Shows how to handle neutral terms (stuck computations).

1.2 Practical Tutorials

- **Christiansen, David** (2019). "Checking Dependent Types with Normalization by Evaluation: A Tutorial."

  Annotation: Literate Racket implementation building incrementally from untyped lambda calculus to dependent types. Excellent for understanding the evaluation-quotation cycle. The tutorial's semantic domain design directly inspired Requiem's structure.

- **Christiansen, David** (2013). "Normalization by Evaluation (Haskell version)."

  Annotation: Haskell implementation with clear separation of syntax, semantics, and normal forms. Shows how to implement NbE without dependent types in the metalanguage (unlike Agda implementations).

2 Higher-Order Abstract Syntax (HOAS)

2.1 Foundational Papers

- **Pfenning, Frank and Elliot, Conal** (1988). "Higher-Order Abstract Syntax." *PLDI 1988.*

  Annotation: The original paper introducing HOAS representation using typed lambda calculus. Shows how to delegate substitution to the metalanguage. Requiem uses this approach: Janet functions represent binders, avoiding explicit variable manipulation.

- **Chlipala, Adam**. "Certified Programming with Dependent Types" (CPDT), Chapter on HOAS.

Annotation: Discusses parametric HOAS (PHOAS) and weak HOAS as practical alternatives in proof assistants like Coq. Explains the trade-offs between full HOAS (exotic terms problem) and PHOAS (additional safety). Requiem uses weak HOAS since Janet's functions are unrestricted.

- **Poswolsky, Adam and Schürmann, Carsten** (2008). "Practical Programming with Higher-Order Encodings and Dependent Types." *ESOP 2008*.

  Annotation: The Delphin system demonstrates programming over HOAS encodings without explicit contexts. Shows how to handle pattern matching and recursion over HOAS terms. Relevant for understanding Requiem's context-free evaluation.

3 Bidirectional Type Checking

3.1 Tutorial Introduction

- **Christiansen, David** (2013). "Bidirectional Typing Rules: A Tutorial."

  Annotation: Accessible introduction to synthesis ($\Rightarrow$) and checking ($\Leftarrow$) modes. Explains why lambdas require type annotations (checking mode) while applications can synthesize types (inference mode). Directly explains Requiem's `infer` and `check` functions.

3.2 Comprehensive Survey

- **Dunfield, Joshua and Krishnaswami, Neelakantan R.** (2022). "Bidirectional Typing." *ACM Computing Surveys*, Vol. 54, No. 5.

  Annotation: Definitive 62-page survey covering history, applications, and relationship to other type system techniques. Includes coverage of dependent types, higher-rank polymorphism, and subtyping. Essential reference for justifying the bidirectional approach.

- **Dunfield, Joshua and Krishnaswami, Neelakantan R.** (2013). "Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism." *ICFP 2013*.

  Annotation: Shows how bidirectional typing handles complex features like rank-N types. Demonstrates completeness and decidability results. Provides theoretical backing for bidirectional approach.

- **Felicissimo, Thiago** (2024). "Generic bidirectional typing for dependent type theories." PhD Thesis, ENS Lyon.

  Annotation: Recent work on theory-independent formulation of bidirectional typing with machine-checked decidability proofs. Shows how to abstract over specific type theories while maintaining soundness.

4 Martin-Löf Type Theory and Identity Types

4.1 Foundational Works

- **Martin-Löf, Per** (1975). "An Intuitionistic Theory of Types: Predicative Part." In *Logic Colloquium '73*.

  Annotation: Where the J eliminator first appeared. Introduces intensional identity types with computation rule $J(A, x, P, d, x, refl_x) \equiv d$. This is the foundation for Requiem's identity type implementation.

- **Martin-Löf, Per** (1984). *Intuitionistic Type Theory*. Bibliopolis.

  Annotation: The definitive book on MLTT. Presents the complete system including Pi, Sigma, identity types, and universe hierarchy. Requiem's type system is a direct implementation of this theory (without inductive types or W-types).

4.2 Modern Explanations

- **nLab contributors**. "Martin-Löf dependent type theory."

  Annotation: Formal inference rules with modern notation. Includes detailed formation, introduction, elimination, and computation rules for all type formers. Excellent reference for checking rule implementations.

- **nLab contributors**. "Identity type."

  Annotation: Comprehensive coverage of identity types including J vs K axioms, intensional vs extensional equality, and connections to homotopy type theory. Explains why Requiem uses J (intensional) rather than K (UIP).

- **Licata, Dan** (2011). "Just Kidding: Understanding Identity Elimination in Homotopy Type Theory."

  Annotation: Explains the distinction between J (path induction) and K (uniqueness of identity proofs). Shows why intensional type theory with J is weaker than extensional type theory with K. Clarifies why Requiem can't prove UIP.

- **Götz, Lennard** (2018). "Martin-Löf's J-Rule." Bachelor thesis, LMU München.

  Annotation: Proves equivalence of various formulations of the J eliminator. Useful for understanding why different presentations (transport, path induction, etc.) are computationally equivalent.

4.3 Philosophical Context

- **Rathjen, Michael and Coquand, Thierry** (2024). "Intuitionistic Type Theory." *Stanford Encyclopedia of Philosophy.*

  Annotation: Covers meaning theory, BHK interpretation, and philosophical foundations. Explains the constructive mathematics perspective underlying Requiem's design.

5 Complete Tutorial Implementations

- **Löh, Andres; McBride, Conor; Swierstra, Wouter** (2010). "A Tutorial Implementation of a Dependently Typed Lambda Calculus." Revised version. (PDF)

  Annotation: Literate Haskell implementation building from simply-typed lambda calculus to dependent types with Pi, application, and checking. Incremental development similar to Requiem's structure. Includes executable code and detailed explanations.

- **Löh, Andres; Conor McBride; Wouter Swierstra** (2008). "Simply Easy! (An Implementation of a Dependently Typed Lambda Calculus)."

  Annotation: Short paper with a complete, minimal implementation. Focuses on simplicity and accessibility, making it an excellent starting point for understanding how the core algorithms (NbE, bidirectional checking) fit together.

6 Proof Theory and Foundations

- **Girard, Jean-Yves; Lafont, Yves; Taylor, Paul** (1989). *Proofs and Types.*
  Cambridge University Press. [PDF Available]

  Annotation: Introduction to proof theory via typed lambda calculus. Covers Curry-Howard correspondence, system F, and cut elimination. Provides theoretical foundations for understanding propositions-as-types, which underlies Requiem's identity types (proofs of equality are inhabitants of Id types).

- **Pierce, Benjamin C.** (2002). *Types and Programming Languages* (TAPL).
  MIT Press.

  Annotation: Standard textbook covering lambda calculus, simple types, polymorphism, and subtyping. Chapters 9-11 (Simply Typed Lambda Calculus) and Chapter 23 (Universal Types) provide foundations for understanding Requiem's function types. Essential background reading.

- **Pierce, Benjamin C.** (ed.) (2004). *Advanced Topics in Types and Programming Languages* (ATTAPL).
  MIT Press.

  Annotation: Chapter 2 (Dependent Types) by David Aspinall and Martin Hofmann directly covers dependent type systems. Chapter 1 (Substructural Type Systems) relevant for understanding linearity. Advanced companion to TAPL.

- **Henk P. Barendregt** (1992). "Lambda calculi with types." In *Handbook of Logic in Computer Science.*

  Annotation: Comprehensive reference on typed lambda calculi and the foundation of the lambda cube.

- **Rob Nederpelt and Herman Geuvers** (2014). *Type theory and formal proof: an introduction*. Cambridge University Press.

  Annotation: Accessible textbook covering foundations from untyped lambda calculus to the Calculus of Constructions.

- **Wadler, Philip** (2015). "Propositions as Types." *Communications of the ACM*, Vol. 58, No. 12.

  Annotation: Historical and conceptual overview of the Curry-Howard correspondence. Explains how Requiem's identity types correspond to equality propositions, and how the J eliminator corresponds to proof by induction over equality.

7 Online Textbooks and Courses

- **Wadler, Philip; Kokke, Wen; Siek, Jeremy** (2024). *Programming Language Foundations in Agda* (PLFA).

  Annotation: Complete development of programming language theory in Agda, from lambda calculus through type soundness proofs. Part 1 (Logical Foundations) covers properties like confluence and normalization that Requiem's test suite verifies. Part 2 (Programming Language Foundations) covers type systems and metatheory.

- **Chlipala, Adam** (2013). *Certified Programming with Dependent Types* (CPDT).

  Annotation: Practical guide to proof engineering in Coq. Covers inductive types, dependent pattern matching, and proof automation. Shows how to build verified software using dependent types (different approach than Requiem's minimal core, but valuable for understanding applications).

8 Category Theory and Semantics

- **Seely, R. A. G.** (1984). "Locally Cartesian Closed Categories and Type Theory." *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 95, No. 1.

  Annotation: Original paper showing that dependent type theory is the internal language of locally cartesian closed categories. Provides categorical semantics for Pi, Sigma, and substitution. Theoretical foundation for understanding Requiem's context operations categorically.

- **Hofmann, Martin** (1997). *Syntax and Semantics of Dependent Types.* Chapter 2 in *Extensional Constructs in Intensional Type Theory.*
  Springer.

  Annotation: Detailed treatment of categorical semantics for dependent types. Clarifies the relationship between syntax (terms, types) and semantics (functors, natural transformations). Helps understand the semantic domain in Requiem.

9 Implementation Techniques

- **Hirrolot, Hirrolot** (2023). "How to Keep Lambda Calculus Simple."

  Annotation: Accessible blog post explaining de Bruijn indices vs levels, evaluation strategies, and normalization. Compares different representation choices. While Requiem uses HOAS instead of de Bruijn, this explains the alternatives and trade-offs.

- **Kovács, András**. *elaboration-zoo*. GitHub repository.

  Annotation: Collection of minimal implementations demonstrating elaboration from surface syntax to core. Shows practical techniques for implementing unification, implicit arguments, and metavariables. While Requiem omits elaboration, these examples show the next step in building a practical dependently-typed language.

10 Termination and Structural Recursion

- **Abel, Andreas and Altenkirch, Thorsten** (2002). "A Predicative Analysis of Structural Recursion." *Journal of Functional Programming*, Vol. 12, No. 1.

  Annotation: Introduces the `foetus` termination checker. Shows how to ensure termination through structural recursion checks on inductive types. Foundations for ensuring that Requiem's evaluation always terminates.

- **Lee, Chin Soon; Jones, Neil D.; Ben-Amram, Amir M.** (2001). "The Size-Change Principle for Program Termination." *POPL 2001.*

Annotation: Introduces a more general principle for termination based on tracking size changes across recursive calls. A key reference for more advanced termination analysis beyond simple structural recursion.

11 Research Context

- **Univalent Foundations Program** (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*.

  Annotation: The HoTT Book. While Requiem implements intensional MLTT (not HoTT), this book motivates why identity types are interesting beyond equality. Shows connections to topology and higher category theory. Relevant for understanding future directions.

12 Video Series and Video Lectures

- **Lambda Cube Unboxed**. Full Series Playlist.

  Annotation: 13-video series on typed and untyped lambda calculi. Covers the lambda cube, parametric polymorphism, and dependent types. Excellent visual introduction to the foundations.

- **Computerphile: Type Theory**. Playlist.

  Annotation: Featuring Thorsten Altenkirch and Philip Wadler. Includes "Propositions as Types" and "The Hardest Problem in Type Theory" (NbE).

13 Additional Courses and Communities

- **Oregon Programming Languages Summer School (OPLSS)**. Course Archive.

  Annotation: Annual summer school with archived lectures on dependent types, proof assistants, and verification (e.g., 2015, 2019 sessions).

- **HoTTEST Summer School 2022**. YouTube Playlist and GitHub Repository.

  Annotation: Comprehensive resources on Homotopy Type Theory and univalent foundations, featuring lectures by leading researchers. Covers both theoretical foundations and practical formalization.

- **Proof Assistants Stack Exchange**. Q&A Community.

  Annotation: Active community for technical questions on type theory and implementation.