Jeremy G. Siek

Indiana University
Department of Computer Science
Luddy Hall
700 N. Woodlawn Ave.
Bloomington, IN 47408-3901

Email: jsiek@iu.edu
Website: https://jsiek.github.io/home/

Research Interests

Programming language design and implementation with a focus on *gradual typing*, that is, integrating static and dynamic checking of program properties.

Education

2005 **Indiana University**
Ph.D. Computer Science, *A Language for Generic Programming*
Advisor: Andrew Lumsdaine

1999 **University of Notre Dame**
M.S. Computer Science and Engineering
A Modern Framework for Portable High Performance Numerical Linear Algebra
Advisor: Andrew Lumsdaine

1997 **University of Notre Dame**
B.S. Mathematics

Professional Experience

2019–present **Indiana University**, Bloomington, IN
Professor

2013–2019 **Indiana University**, Bloomington, IN
Associate Professor

2007–2013 **University of Colorado**, Boulder, CO
Assistant Professor

Honors

2022 **Indiana University Trustees Teaching Award**

2019 **Distinguished Professor Fellowship**
Foundation Sciences Mathématiques de Paris (FSMP), January–May

2018 **Most Notable Paper Award for Dynamic Lang. Symposium 2008**
Gradual Typing and Unification-based Inference

2010 & 2015 **Distinguished Visiting Fellowship**
Scottish Informatics & Computer Science Alliance (SICSA)

2009-2014 **NSF Faculty Early Career Development (CAREER) Award**
Bridging the Gap Between Prototyping and Production

Software

Deduce Proof Checker. Available at
`https://jsiek.github.io/deduce/`
Developed a proof checker for use in education.

Carbon Explorer. Available at
`https://github.com/carbon-language/carbon-lang`
Collaborated with the C++ compiler team at Google to build a parser and interpreter for the new Carbon language.

Build to Order BLAS. Available at `https://github.com/nelsonth/btoblas`.
Developed a domain-specific compiler for basic linear algebra that applies automatic tuning to search for high-performance implementations.

Peer-reviewed Boost C++ libraries: Graph, Concept Check, Dynamic Bitset, Iterator Adaptor, Operator, and Property Map. The Boost Library Collection is available at `www.boost.org`.

The Matrix Template Library and the Iterative Template Library.
C++ template libraries for basic linear algebra and solving sparse systems of equations.

Publications

Books

1. Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda*. August 2022. URL `https://plfa.inf.ed.ac.uk/22.08/`.
2. Jeremy G. Siek. *Essentials of Compilation: An Incremental Approach in Python*. MIT Press, August 2023.
3. Jeremy G. Siek. *Essentials of Compilation: An Incremental Approach in Racket*. MIT Press, February 2023.
4. Jeremy G. Siek, Lee-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

Book Chapters

1. Jeremy G. Siek and Andrew Lumsdaine. *Advances in Software Tools for Scientific Computing*, chapter A Modern Framework for Portable High Performance Numerical Linear Algebra. Springer, 2000.

Journal Articles

1. Jeremy G. Siek, Peter Thiemann, and Philip Wadler. Blame and coercion: Together again for the first time. *Journal of Functional Programming*, 31:e20, 2021. doi: 10.1017/S0956796821000101.

2. Jeremy G. Siek and Tianyu Chen. Parameterized cast calculi and reusable meta-theory for gradually typed lambda calculi. *Journal of Functional Programming*, 31:e30, 2021. doi: 10.1017/S0956796821000241.

3. Wen Kokke, Jeremy G. Siek, and Philip Wadler. Programming language foundations in Agda. *Science of Computer Programming*, 194:102440, 2020.

4. Jeremy Siek. Declarative semantics for functional languages. *Archive of Formal Proofs*, 2017. ISSN 2150-914x. `http://isa-afp.org/entries/Decl_Sem_Fun_PL.shtml`, Formal proof development.

5. Thomas Nelson, Geoffrey Belter, Jeremy G. Siek, Elizabeth Jessup, and Boyana Norris. Reliable generation of high-performance matrix algebra. *ACM Trans. Math. Softw.*, 41(3):18:1–18:27, 2015.

6. Erik Silkensen and Jeremy G. Siek. Well-typed islands parse faster. *Lecture Notes in Computer Science*, Volume 7829:69–84, 2013. Trends in Functional Programming (TFP).

7. Jeremy G. Siek. The C++0x "concepts" effort. *Lecture Notes in Computer Science*, 7470:175–216, 2012. Special Issue Generic and Indexed Programming.

8. Daniel P. Friedman, Abdulaziz Ghuloum, Jeremy G. Siek, and Onnie Lynn Winebarger. Improving the lazy krivine machine. *Higher Order and Symbolic Computation*, 20(3):271–293, 2007. ISSN 1388-3690.

9. Jeremy G. Siek and Andrew Lumsdaine. A language for generic programming in the large. *Science of Computer Programming*, 76:423–465, 2011.

10. Ronald Garcia, Jaakko Järvi, Andrew Lumsdaine, Jeremy G. Siek, and Jeremiah Willcock. An extended comparative study of language support for generic programming. *Journal of Functional Programming*, 17(2):145–205, 2007.

11. Ian Karlin, Elizabeth Jessup, Geoffrey Belter, and Jeremy G. Siek. Parallel memory prediction for fused linear algebra kernels. *SIGMETRICS Perform. Eval. Rev.*, 38:43–49, March 2011.

Conference Papers

1. Jeremy G. Siek. Gradual guarantee via step-indexed logical relations in Agda. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 413, August 2024. Festschrift in Honor of Peter Thiemann's Sixtieth Birthday.

2. Tianyu Chen and Jeremy G Siek. Quest complete: The holy grail of gradual security. *Proceedings of the ACM on Programming Languages*, 8(PLDI):1609–1632, 2024.

3. Matteo Cimini, Dale Miller, and Jeremy G. Siek. Extrinsically typed operational semantics for functional languages. In *International Conference on Software Language Engineering*, pages 108–125. Association for Computing Machinery, 2020.

4. Michael M. Vitousek, Jeremy G. Siek, and Avik Chaudhuri. Optimizing and evaluating transient gradual typing. In *Dynamic Languages Symposium*, DLS, October 2019.

5. Giuseppe Castagna, Guillaume Duboc, Victor Lanvin, and Jeremy G. Siek. A space-efficient call-by-value virtual machine for gradual set-theoretic types. In *Symposium on Implementation and Application of Functional Languages*, IFL, September 2019.

6. Andre Kuhlenschmidt, Deyaaeldeen Almahallawi, and Jeremy G. Siek. Toward efficient gradual typing for structural types via coercions. In *Conference on Programming Language Design and Implementation*, PLDI. ACM, June 2019.

7. Giuseppe Castagna, Victor Lanvin, Tommaso Petrucciani, and Jeremy G. Siek. Gradual typing: a new perspective. In *Symposium on Principles of Programming Languages*, POPL. ACM, 2019.

8. David Broman and Jeremy G. Siek. Gradually typed symbolic expressions. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM '18, pages 15–29, New York, NY, USA, 2018. ACM. doi: 10.1145/3162068. URL `http://doi.acm.org/10.1145/3162068`.

9. Spenser A. Bauman, Sam Tobin-Hochstadt, Jeremy G. Siek, and Carl Friedrich Bolz-Tereick. Sound gradual typing: Only mostly dead. In *Conference on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA, 2017.

10. Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. Theorems for free for free: Parametricity, with and without types. In *International Conference on Functional Programming*, ICFP, 2017.

11. Michael M. Vitousek, Cameron Swords, and Jeremy G. Siek. Big types in little runtime. In *Symposium on Principles of Programming Languages*, POPL, 2017.

12. Matteo Cimini and Jeremy G. Siek. Automatically generating the dynamic semantics of gradually typed languages. In *Symposium on Principles of Programming Languages*, POPL, 2017.

13. Jeremy G. Siek and Sam Tobin-Hochstadt. The recursive union of some gradual types. In Sam Lindley, Conor McBride, Don Sannella, and Phil Trinder, editors, *Wadler Fest*, LNCS. Springer, 2016.

14. Matteo Cimini and Jeremy G. Siek. The gradualizer: a methodology and algorithm for generating gradual type systems. In *Symposium on Principles of Programming Languages*, POPL, 2016.

15. Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. Refined criteria for gradual typing. In *SNAPL: Summit on Advances in Programming Languages*, LIPIcs: Leibniz International Proceedings in Informatics, 2015.

16. Jeremy G. Siek, Peter Thiemann, and Philip Wadler. Blame and coercion: Together again for the first time. In *Conference on Programming Language Design and Implementation*, PLDI. ACM, 2015.

17. Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, Sam Tobin-Hochstadt, and Ronald Garcia. Monotonic references for efficient gradual typing. In *European Symposium on Programming*, ESOP, 2015.

18. Spenser Bauman, Carl Friedrich Bolz, Robert Hirschfeld, Vasily Kirilichev, Tobias Pape, Jeremy Siek, and Sam Tobin-Hochstadt. Pycket: A tracing JIT for a functional language. In *ICFP: International Conference on Functional Programming*, 2015.

19. Eric Holk, Ryan Newton, Jeremy Siek, and Andrew Lumsdaine. Region-based memory management for GPU programming languages. In *Conference on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA. ACM, 2014.

20. Michael M. Vitousek, Jeremy G. Siek, Andrew Kent, and Jim Baker. Design and evaluation of gradual typing for Python. In *DLS: Dynamic Languages Symposium*, 2014.

21. Justin Gottschlich, Maurice Herlihy, Gilles Pokam, and Jeremy G. Siek. Visualizing transactional memory. In *PACT: International Conference on Parallel Architectures and Compilation Techniques*, 2012.

22. Devin Coughlin, Bor-Yuh Evan Chang, Amer Diwan, and Jeremy G. Siek. Measuring enforcement windows with symbolic trace interpretation. In *ISSTA: International Symposium of Software Testing and Analysis*, 2012.

23. Wieyu Miao and Jeremy G. Siek. Pattern-based traits. In *Symposium on Applied Computing*, 2012.

24. Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. Blame for All. In *Symposium on Principles of Programming Languages*, POPL. ACM, 2011.

25. Weiyu Miao and Jeremy G. Siek. Incremental type-checking for type-reflective metaprograms. In *GPCE '10: Proceedings of the International Conference on Generative Programming and Component Engineering*, 2010.

26. Justin E. Gottschlich, Manish Vachharajani, and Jeremy G. Siek. An efficient software transactional memory using commit-time invalidation. In *Proceedings of the 8th annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 101–110. ACM, 2010.

27. Jeremy G. Siek and Philip Wadler. Threesomes, with and without blame. In *Symposium on Principles of Programming Languages*, POPL, pages 365–376. ACM, 2010.

28. Geoffrey Belter, E. R. Jessup, Ian Karlin, and Jeremy G. Siek. Automating the generation of composed linear algebra kernels. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, 2009.

29. Angela Yun Zhu, Walid Taha, Robert Cartwright, Matthieu Martel, and Jeremy G. Siek. In pursuit of real answers. In *Proceedings of the 2009 International Conference on Embedded Software and Systems*, pages 115–122. IEEE Computer Society, 2009.

30. Boyana Norris, Albert Hartono, Elizabeth Jessup, and Jeremy G. Siek. Generating empirically optimized composed matrix kernels from matlab prototypes. In *International Conference on Computational Science*, 2009.

31. Jeremy G. Siek, Ronald Garcia, and Walid Taha. Exploring the design space of higher-order casts. In *European Symposium on Programming*, 2009.

32. Jeremy G. Siek and Manish Vachharajani. Gradual typing and unification-based inference. In *Dynamic Languages Symposium*. AITO, 2008. **Most Notable Paper Award in 2018 for 2008**.

33. Jeremy G. Siek and Walid Taha. Gradual typing for objects. In *ECOOP 2007*, volume 4609 of *LCNS*, pages 2–27. Springer Verlag, 2007.

34. Douglas Gregor, Jaakko Järvi, Jeremy G. Siek, Gabriel Dos Reis, Bjarne Stroustrup, and Andrew Lumsdaine. Concepts: Linguistic support for generic programming in C++. In *OOPSLA: Conference on Object-oriented Programming, Systems, Languages, and Applications*. ACM, 2006.

35. Jeremy G. Siek and Walid Taha. A semantic analysis of C++ templates. In *ECOOP: European Conference on Object-Oriented Programming*, 2006.

36. Jaakko Järvi, Douglas Gregor, Jeremiah Willcock, Andrew Lumsdaine, and Jeremy G. Siek. Algorithm specialization in generic programming - challenges of constrained generics in C++. In *Conference on Programming Language Design and Implementation*, PLDI. ACM, 2006.

37. Jeremy G. Siek and Andrew Lumsdaine. Essential language support for generic programming. In *Conference on Programming Language Design and Implementation*, PLDI, pages 73–84. ACM, 2005.

38. Jeremy G. Siek and Andrew Lumsdaine. Language requirements for large-scale generic libraries. In *GPCE: International Conference on Generative Programming and Component Engineering*, September 2005.

39. Lie-Quan Lee, Jeremy G. Siek, and Andrew Lumsdaine. The generic graph component library. In *OOPSLA: Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 399–414. ACM, 1999.

40. Lie-Quan Lee, Jeremy G. Siek, and Andrew Lumsdaine. Generic graph algorithms for sparse matrix ordering. In *ISCOPE'99*, LNCS. Springer-Verlag, 1999.

41. Jeremy G. Siek and Andrew Lumsdaine. The matrix template library: A generic programming approach to high performance numerical linear algebra. In *ISCOPE'98*, LNCS. Springer-Verlag, 1998.

42. Ronald Garcia, Jaakko Järvi, Andrew Lumsdaine, Jeremy G. Siek, and Jeremiah Willcock. A comparative study of language support for generic programming. In *OOPSLA '03: Conference on Object-oriented Programing, Systems, Languages, and Applications*, pages 115–134. ACM, 2003.

Refereed Workshop Papers

1. Tianyu Chen and Jeremy G. Siek. Mechanized type safety for gradual information flow. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 194–206, July 2021. doi: 10.1109/SPW53761.2021.00033.

2. Deyaaeldeen Almahallawi and Jeremy G. Siek. Space-efficient monotonic references. In *Workshop on Gradual Typing*. ACM, 2020.

3. Kuang-Chen Lu, Jeremy G. Siek, and Andre Kuhlenschmidt. Hypercoercions and a framework for equivalence of cast calculi. In *Workshop on Gradual Typing*. ACM, January 2020.

4. Andre Kuhlenschmidt, Deyaaeldeen Almahallawi, and Jeremy G. Siek. An efficient compiler for the gradually typed lambda calculus. In *Scheme and Functional Programming Workshop*, SFP, September 2018.

5. Weixi Ma, Jeremy G. Siek, David Christiansen, and Daniel Friedman. Efficiency of a good but not linear nominal unification algorithm. In *International Workshop on Unification*, UNIF, July 2018.

6. Andre Kuhlenschmidt, Deyaaeldeen Almahallawi, and Jeremy G. Siek. Towards absolutely efficient gradually typed languages. In *Scripts to Programs Workshop*, STOP, 2015.

7. Carl Friedrich Bolz, Tobias Pape, Sam Tobin-Hochstadt, and Jeremy G. Siek. Meta-tracing makes a fast Racket. In *Workshop on Dynamic Languages and Applications*. ACM, 2014.

8. Weiyu Miao and Jeremy G. Siek. Compile-time reflection and metaprogramming for Java. In *PEPM: Workshop on Partial Evaluation and Program Manipulation (PEPM '14)*, 2014.

9. Christopher Schwaab and Jeremy G. Siek. Modular type-safety proofs in Agda. In *PLPV: Workshop on Programming Languages meets Program Verification*, 2013.

10. Jeremy G. Siek and Ronald Garcia. Interpretations of the gradually-typed lambda calculus. In *Workshop on Scheme and Functional Programming*. ACM, 2012. 13 pages.

11. Seth Fogarty, Emir Pasalic, Jeremy G. Siek, and Walid Taha. Concoqtion: Indexed types now! In *PEPM: Workshop on Partial Evaluation and Program Manipulation*, 2007.

12. Geoffrey Belter, Jeremy G. Siek, Ian Karlin, and E. R. Jessup. Automatic generation of tiled and parallel linear algebra routines. In *Fifth International Workshop on Automatic Performance Tuning (iWAPT)*, June 2010.

13. Elizabeth R. Jessup, Ian Karlin, Erik Silkensen, Geoffrey Belter, and Jeremy Siek. Understanding memory effects in the automated generation of optimized matrix algebra kernels. In *Automated Program Generation Workshop at ICCS*, May 2010. doi: DOI:10.1016/j.procs.2010.04.209.

14. Justin E. Gottschlich, Jeremy G. Siek, Manish Vachharajani, Dwight Y. Winkler, and Daniel A. Connors. An efficient lock-aware transactional memory implementation. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, ICOOOLPS '09, pages 10–17. ACM, 2009.

15. Jeremy G. Siek and Philip Wadler. Threesomes, with and without blame. In *Proceedings for the 1st workshop on Script to Program Evolution*, STOP '09, pages 34–46. ACM, 2009.

16. Justin E. Gottschlich, Jeremy G. Siek, and Daniel A. Connors. C++ move semantics for exception safety and optimization in software transactional memory libraries. In *Proceedings of the Third International Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS). In conjunction with ECOOP.* jul 2008.

17. Jeremy G. Siek, Ian Karlin, and E. R. Jessup. Build to order linear algebra kernels. In *Workshop on Performance Optimization for High-Level Languages and Libraries (POHLL 2008)*, pages 1–8, April 2008.

18. Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, pages 81–92, September 2006.

19. Jaakko Järvi, Andrew Lumsdaine, Jeremy Siek, and Jeremiah Willcock. An Analysis of Constrained Polymorphism for Generic Programming. In Kei Davis and Jörg Striegnitz, editors, *Multiparadigm Programming 2003: Proceedings of the MPOOL Workshop at OOPSLA'03*, John von Neumann Institute of Computing series, pages 87–107, October 2003.

20. Doug Gregor, Brian Osman, David R. Musser, Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. Concept-based component libraries and optimizing compilers. In *NSF Next Generation Systems Program Workshop at International Parallel and Distributed Processing Symposium, IPDPS 2002*, pages 174 –181, 2002.

21. David Abrahams and Jeremy G. Siek. Policy adaptors and the Boost Iterator Adaptor Library. In *Second Workshop on C++ Template Programming*, October 2001.

22. Jeremiah Willcock, Jeremy Siek, and Andrew Lumsdaine. Caramel: A concept representation system for generic programming. In *Second Workshop on C++ Template Programming*, October 2001.

23. Jeremy G. Siek and Andrew Lumsdaine. Concept checking: Binding parametric polymorphism in C++. In *Proceedings of the First Workshop on C++ Template Programming*, 2000. URL `citeseer.nj.nec.com/siek00concept.html`.

24. Jeremy G. Siek and Andrew Lumsdaine. Mayfly: A pattern for lightweight generic interfaces. In *Pattern Languages of Programs*, July 1999.

25. Jeremy G. Siek, Andrew Lumsdaine, and Lie-Quan Lee. Generic programming for high performance numerical linear algebra. In *Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)*. SIAM Press, 1998.

26. Jeremy G. Siek and Andrew Lumsdaine. The Matrix Template Library: A unifying framework for numerical linear algebra. In *Parallel Object Oriented Scientific Computing*. ECOOP, 1998.

Funding

Acceleration of Feature Engineering. Funding Agency: Meta, Amount: $100,000. Duration: 12/30/24 – 12/30/25.

Programming Language Research. Funding Agency: Google, Amount: $136,000. Duration: 5/10/22 – 8/21/23.

Programming Language Research. Funding Agency: Google, Amount: $86,000. Duration: 8/15/21 – 5/9/22.

Revisiting Elementary Denotational Semantics. Principal Investigator: Jeremy G. Siek. Funding Agency: NSF, Amount: $380,714. Duration: 8/1/2018 – 7/31/2021.

Performant Sound Gradual Typing. Principal Investigator: Sam Tobin-Hochstadt. Co-PI: Jeremy G. Siek. Funding Agency: NSF, Amount: $1,192,054. Duration: 7/1/2018 – 6/30/2021.

Facebook Faculty Research Award. Principal Investigator: Jeremy G. Siek. Funding Agency: Facebook. Amount: $30,000. Duration: 7/1/2017-6/30/2018.

Gradual Typing Across the Spectrum. Principal Investigator: Matthias Felleisen (Northeastern). IU PI: Jeremy G. Siek. Funding Agency: NSF, Amount: $584,715 (IU's portion). Duration: 7/1/2015 – 6/20/2018.

Evaluating Exascale Execution Models. Principal Investigator: Thomas Sterling. Co-PI: Jeremy G. Siek. Funding Agency: DOE, Amount: $800,000. Duration: 5/1/15–4/30/17.

Unifying Modular Abstractions for Chapel, Part 2. (NSF CAREER Award Supplement) Principal Investigator: Jeremy G. Siek. Funding Agency: DOD, Amount: $89,906, Duration: 8/27/2013–8/26/2014.

Modular Reflection. Principal Investigator: Bor-Yuh Chang. Co-PI: Jeremy G. Siek. Funding Agency: NSF. Amount: $125,000 (Siek's part), Duration: 10/1/2012–9/30/2015.

Unifying Modular Abstractions for Chapel. (NSF CAREER Award Supplement) Principal Investigator: Jeremy G. Siek. Funding Agency: DOD, Amount: $88,816, Duration: 8/27/2012–8/26/2013.

REU for CAREER: Bridging the Gap Between Prototyping and Production. Principal Investigator: Jeremy G. Siek. Funding Agency: NSF. Amount: $21,000, Duration: 6/1/2012–5/31/2013.

Modern Object-Oriented Chapel. (NSF CAREER Award Supplement) Principal Investigator: Jeremy G. Siek. Funding Agency: DOD, Amount: $39,905, Duration: 7/1/2011–6/30/2012.

Remodel ECEE 1B61A Graduate Student Office Space. Principal Investigator: Jeremy G. Siek. Funding Agency: CU CEAS and the CU ECEE Dept, Amount: $55,120, June 2011.

Interfaces and Modular Generics for Chapel. Principal Investigator: Jeremy G. Siek. Funding Agency: DOD, Amount: $103,735, Duration: 9/1/2010–8/31/2012.

EAGER: Exploratory Research on Gradual Programming. Principal Investigator: Jeremy G. Siek. Funding Agency: NSF. Amount: $81,748,
Duration: 8/1/2009–7/31/2010.

CAREER: Bridging the Gap Between Prototyping and Production. CCF 0846121. Principal Investigator: Jeremy G. Siek. Funding Agency: NSF. Amount: $481,910, Duration: 3/1/2009–2/28/2014.

REU for CAREER: Bridging the Gap Between Prototyping and Production. Principal Investigator: Jeremy G. Siek. Funding Agency: NSF. Amount: $10,500, Duration: 3/1/2009–2/28/2010.

Matching funds from the University of Colorado for the CAREER award. Amount: $60,000.

Test and Evaluation of Architecture-Aware Compiler Environments. Principal Investigator: Jeremy G. Siek. Funding Agency: DARPA. Amount: $1,146,195, Duration: 2/1/2009–7/1/2013.

Modular Metaprogramming. Principal Investigators: Jeremy G. Siek (Colorado)

and Andrew Lumsdaine (Indiana). Funding Agency: NSF. Amount: $340,000 (Colorado's portion), Duration: 7/1/2007–7/1/2010.

Presentations

Invited Presentations

The Metatheory of Gradual Typing: State of the Art and Challenges. Conference on Algebra and Coalgebra in Computer Science, June 2023.

Efficient Gradual Typing. University of Minnesota, November 2021.

Progress on Compiler Correctness via Filter Models. Workshop on Intersection Types and Related Systems (ITRS), July 2021.

Toward a Mechanized Encyclopedia of Gradual Typing. University of Chile, Santiago, July 2019.

Toward Efficient Gradual Typing. INRIA Paris, Gallium, February 2019.

State of the Art in Gradual Typing. University of Paris Diderot, January 2019.

Challenges and Progress Toward Efficient Gradual typing. Dynamic Languages Symposium, October 2017.

State of the Art in Gradual Typing. University of Paris Diderot, April 2017.

Pycket A Tracing JIT For a Functional Language. Advances in Programming Languages and Systems, University of Frankfurt, December 2015.

The Polymorphic Blame Calculus and Parametricity. University of Strathclyde, August 2015.

The State of the Art in Gradual Typing. SICSA Summer School on Practical Types. University of St. Andrews, August 2015.

Design and Evaluation of Gradual Typing for Python. NII Shonan Meeting on Software Contracts. May 2014.

A Mechanized Semantics for System F via the F Machine. IU Logic Seminar. April 2014.

Reliable Generation of High-Performance Matrix Algebra. Indiana University, CREST. January 2014.

Reliable Generation of High-Performance Matrix Algebra. University of Utah. May 2013.

The gradual typing approach to mixing static and dynamic typing. Keynote at the Trends in Functional Programming Conference. May 2013.

Foundations for Gradually Typed Languages. Indiana University, February 2013.

Interpretations of the Gradually-Typed Lambda Calculus. Distilled Tutorial at the ACM SIGPLAN Workshop on Scheme and Functional Programming. September 2012.

Reliable Generation of High-Performance Matrix Algebra. NII Shonan Meeting on Bridging the Theory of Staged Programming and the Practice of High-Performance Computing. May 2012.

Gradual Typing Roundup. Dagstuhl Seminar on Foundations for Scripting Langauges. January 2012.

Composable DSLs. IFIP WG 2.11, September 2011.

Build to Order Linear Algebra Kernels. University of Glasgow, June 2010.

General Purpose Languages Should be Metalanguages. University of Edinburgh, Laboratory for Foundations in Computer Science, June 2010.

The C++0x Concept Effort. University of Washington. May, 2010.

Concepts in C++. Spring School on Generic and Indexed Programming. Oxford University, UK. March, 2010.

General Purpose Languages Should be Metalanguages. Keynote at the ACM SIGPLAN 2010 Workshop on Partial Evaluation and Program Manipulation. Madrid, Spain. January, 2010.

Build to Order Linear Algebra Kernels. NSF-NAIS Workshop, Intelligent Software: the interface between algorithms and machines. Edinburgh, UK. October, 2009.

Build To Order BLAS. CScADS Workshop on Libraries and Autotuning for Petascale Applications. Tahoe, CA. August, 2009.

Space-Efficient Blame Tracking for Gradual Typing. The University of Edinburgh, March 2009.

Type Safe Reflective Metaprogramming. Microsoft Research Redmond, RiSE team, December 2008.

Panel: The Future of Programming Languages. Microsoft's Professional Developers Conference, October 2008.

Gradual Typing with Inference. Foundations of Object-Oriented Languages (FOOL'08), January, 2008. San Francisco.

Knights' Tours, Religious Wars, and Built to Order BLAS. University of Colorado at Boulder, April 2007.

Build to Order Linear Algebra Kernels. Tech-X Corporation, March 2007.

Language Support for Generic Programming, C++200X and Beyond. Rensselaer Polytechnic Institute, March 2007.

Languages and Libraries for High-Performance Computing. U. of Wyoming, March 2007.

Gradual Typing. Oxford University, February 2007.

Languages and Libraries for High-Performance Computing. Oxford Univ., Nov. 2006.

Languages and Libraries for High-Performance Computing. University of Colorado at Boulder, May 2006.

Languages and Libraries for High-Performance Computing. University of California, Merced, May 2006.

Languages and Libraries for High-Performance Computing. Virginia Polytechnic Institute and State University, February 2006.

Languages and Libraries for High-Performance Computing. MathWorks, February 2006.

The Design and Implementation of the Boost Graph Library. The Association of C and C++ Users (ACCU) Spring Conference 2003.

A Language for Generic Programming. April, 2005. Rice University.

Modular Generics. The Adobe Systems Technical Summit, April 2004.

Generic Programming and Numerics. Center for Theoretical Studies, ETH Zürich. 2002.

Generic Software Components for Scientific Computing. The Institute for Scientific Computing Research at Lawrence Livermore National Lab. November 1999.

Conference Presentations

Toward Efficient Gradual Typing for Structural Types via Coercions. PLDI: ACM Conference on Programming Language Design and Implementation, Phoenix Arizona, June 2019.

Back to the Future with Denotational Semantics. Off the Beaten Track. (workshop at POPL) January 2018.

Back to the Future with Denotational Semantics. Midwest Programming Languages Summit. December 2017.

Declarative Semantics: perspectives and applications. IU Logic Seminar. September 2017.

Simple Models of the Lambda Calculus, and Intersection Types. IU Logic Seminar. February 2017.

Refined Criteria for Gradual Typing. SNAPL: Summit on Advances in Programming Languages, May 2015.

Monotonic References for Efficient Gradual Typing. ESOP: European Symposium on Programming, April 2015.

Linking isn't Substitution. ACM SIGPLAN Workshop on Higher-Order Programming with Effects, September 2013.

Effects for Funargs. ACM SIGPLAN Workshop on Higher-Order Programming with Effects, September 2012.

Monads for Relations. Scottish Programming Languages Seminar, June 2010.

Threesomes, With and Without Blame. Symposium on Principles of Programming Languages, January, 2010.

Blame Tracking for Gradual Types. JVM Language Summit, September, 2009. Sun Microsystems, Santa Clara, CA.

Threesomes, With and Without Blame. 1st International Workshop on Script to Program Evolution, July, 2009. Genoa, Italy.

Exploring the Design Space of Higher-Order Casts. European Symposium on Programming, March 2009. York, United Kingdom.

Gradual Typing for Python. JVM Language Summit, September, 2008. Sun Microsystems, Santa Clara, CA.

Build to Order Linear Algebra Kernels. Front Range Architecture Compilers Tools and Languages Workshop (FRACTAL), April, 2008.

Build to Order Linear Algebra Kernels. Joint Workshop on High-Level Parallel Programming Models and Supportive Environments and Performance Optimization for High-Level Languages and Libraries, April, 2008.

Gradual Typing with Inference. PC Meeting for the 2008 European Conference on Object-Oriented Programming. February, 2008.

Gradual Typing for Objects. European Conference on Object-Oriented Programming, July, 2007, Berlin.

Effectively writing about your research. ECOOP 2007 Doctoral Symposium and Ph.D. Students Workshop, July, 2007, Berlin.

Generic Programming and the Boost Graph Library. The Boost Libraries Conference. May 2007, Aspen.

Gradual Typing for Functional Languages. Scheme and Functional Programming Workshop, September 2006.

Language Requirements for Large-Scale Generic Libraries. The conference on Generative Programming and Component Engineering. September 2005.

Essential Language Support for Generic Programming. The ACM SIGPLAN 2005 conference on Programming Language Design and Implementation. June, 2005.

Modular Generics. The OOPSLA 2004 doctoral symposium.

Policy Adaptors and the Boost Iterator Adaptor Library. The Second Workshop on C++ Template Programming, October 2001.

Concept checking: Binding parametric polymorphism in C++. The First Workshop on C++ Template Programming, Erfurt, Germany, 2000.

Generic Programming for High Performance Numerical Linear Algebra. SciTools in Oslo, Norway. 1998. Award for best presentation.

The Matrix Template Library: A Generic Programming Approach to High Performance Numerical Linear Algebra. The *International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE)*. December 1998.

The Matrix Template Library: A Unifying Framework for Numerical Linear Algebra. The Parallel and High-Performance Object-Oriented Computing workshop. July 1998.

Generic Programming for High Performance Numerical Linear Algebra. The SIAM Workshop on OO Methods for Interoperable Scientific and Engineering Computing, 1998.

Teaching

Undergraduate Courses

1. CSCI C343/H343: Data Structures, 2013-2017, 2019-2025.
2. ECEN 2703: Discrete Math for Computer Eng., 2010, 2011.
3. CSCI 2830: Computer Science as a field of work and study. Fall 2006.

Cross-listed Graduate and Undergraduate Courses

1. CSCI P423/P523: Compilers, 2014, 2016, 2018, 2020-2024.
2. ECEN 4553/5523: Compiler Construction, 2009-2012.
3. CSCI 4448/6448: Object-Oriented Analysis and Design. 2007.

Graduate Courses

1. CSCI B629: Topics in Programming Languages: Denotational Semantics, 2018.
2. CSCI B522: Programming Language Foundations, 2015.
3. ECEN 5533: Fundamentals of Programming Languages, 2011.
4. ECEN 5013: Theorem Proving in Isabelle, 2011.
5. ECEN 5013: Types and Programming Languages, 2010.
6. ECEN 5043: Software Engineering Reusable Components, 2009.
7. ECEN 5023/CSCI 7135-001: Types and Programming Languages, 2008.
8. CSCI 7000: Practical Theorem Proving with Isabelle/Isar. 2007.

Short Courses and Tutorials

1. Teaching and Learning Compilers Incrementally. ICFP 2023. September.
2. Teaching and Learning Compilers Incrementally. PLDI 2023. June.
3. Crash Course on Notation in Programming Language Theory. LambdaConf 2018. June.
4. Theory and Models of Lambda Calculus, Untyped and Typed. By Dana Scott and Jeremy G. Siek. LambdaConf 2018. June.
5. The State of the Art in Gradual Typing. POPL Tutorial. January 2017.
6. Gradual Type Systems. ECOOP Summer School. July 2016.
7. The State of the Art in Gradual Typing. SICSA Summer School on Practical Types. August 2015.
8. C++, Short and Sweet. Online video lecture available at Udemy.
9. Generic Programming and the Boost Graph Library. May 14, 2007. Tutorial at the Boost Libraries Conference 2007.
10. Generic Programming and the Boost Libraries. Short course taught at Engineering Dynamics, Inc. New Orleans. March 21-28, 2006.
11. The Practice of Generic Programming. With Jaakko Järvi. Invited tutorial at the Adobe Systems Technical Summit, April 2004.

Advising

- **Current Ph.D. Students**
  - Tianyu Chen
  - Darshal Shetty
- **Current B.S. Students**
  - Matei Cloteaux
  - Shulin Gonsalves

  - Calvin Josenhans

- **Mentored Post-docs**
  - Matteo Cimini, Univ. of Massachusetts Lowell
- **Graduated Ph.D. Students, Advised**
  - Weixi Ma (co-advised with Dan Friedman), Ph.D., 2021.
  - Deyaaeldeen Almahallawi, Ph.D, 2020.
  - Michael Vitousek, Ph.D., 2018.
  - Thomas Nelson, Ph.D., 2015.
  - Weiyu Miao, Ph.D., 2013.
  - Geoffrey Belter, Ph.D. 2012.
  - Justin E. Gottschlich, Ph.D, 2010.
- **Graduated M.S. and B.S. Students, Advised**
  - Matthew Heimerdinger, M.S. 2022.
  - Kuang-Chen Lu, M.S., 2020.
  - Christopher Wailes, M.S. 2018.
  - Di Zhong, B.S. Spring 2017.
  - Zeina Migeed, B.S. Spring 2017.
  - Andre Kuhlenschmidt, M.S. 2016.
  - Xing Jie Zhong, M.S. Spring 2012.
  - Shashank Bharadwaj, M.S. Fall 2012.
  - Neelam Agrawal, M.S. Spring 2012.
  - Jonathan Turner, M.S. Fall 2012.
  - Erik Silkensen, M.S. Spring 2012.
  - Moss Prescott, M.S. Fall 2010.
  - Sri Teja Basava, M.S. Spring 2011.
  - Christopher Schwaab, B.S. Spring 2010.
- **Graduated Students, Co-advised**
  - Ian Karlin, Ph.D. student, Spring 2011.
  - Scott Busch, M.S. Thesis, Fall 2009.
  - David Broman, Ph.D. at Linköping University, Spring 2010.
  - Tipp Moseley, Ph.D., Spring 2009.
  - Scott Williams, B.S., Independent Study Senior Project, Fall 2006.
  - Jeff Fifield, Ph.D. student, Fall 2011.

Mentoring

- REU 2016.
- Northeastern Research Co-op, 2016.
- IU SROC, Summer 2014.
- Google Summer of Code, Summer 2006, 2007, 2008.

Service

University

- Bloomington Faculty Council 2022–present.
- Member of CEWiT's Advocates for Gender Equity (lead instructor for one workshop each semester), 2017–2019.

School

- Chair of the Luddy Policy Committee, 2024–present.
- Director of the Center for Programming Systems, 2019–present.
- Chair of the Luddy Curriculum Committee, 2017–2018, 2019–2024.
- Member of the Luddy Budgetary Affairs Committee, 2020–2022.
- Member of the SICE CS-Info and CS-ISE Structure Committees 2016.
- Member of SICE Building Committee 2015–2016.
- Member of Ph.D. Outreach Commmittee 2015–2016.

Department

- Chair of the faculty search committee for Systems and Networks 2023–2024.
- Chair of the faculty search committee for Computer Science 2022–2023.
- Member of the Admissions and Awards Committee 2014–2015, 2021–2022.
- Chair of the faculty search committee for Systems and Networks 2021–2022.
- Director of the Computer Science Masters Program 2014–2018.
- Member of the Graduate Education Committee 2014–2017.
- Chair of a committee for a faculty tenure case 2016, 2018 ($\times$2).
- Chair of a two committees for faculty 3rd-year reviews 2015.
- Chair of Computer Science Colloquium Committee 2014–2015.
- Departmental Executive Committee AY 2009–2012.
- Computer Engineering Search Committee AY 2011-2012.
- Head of the Digital Core Curriculum Committee AY 2012–2013.

Research Community

- PC Member. ICFP 2025. The ACM SIGPLAN International Conference on Functional Programming.
- NSF Review Panel 2024, CISE Software and Hardware Foundations.
- PC Member. PLDI 2024. ACM Conference on Programming Language Design and Implementation.
- PC Member. POPL 2024. The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.
- NSF Review Panel 2023, CISE Software and Hardware Foundations.
- PC Member. PLDI 2023. ACM Conference on Programming Language Design and Implementation.
- PC Member. CC 2023. The ACM SIGPLAN 2023 International Conference on Compiler Construction.
- PC Member. ICFP 2022. The ACM SIGPLAN International Conference on Functional Programming.
- Organizer. ITRS 2021. Workshop on Intersection Types and Related Systems.
- PC Member. ITRS 2020. Workshop on Intersection Types and Related Systems.
- Organizer. WGT 2020. The 1st ACM SIGPLAN Workshop on Gradual Typing.
- PC Member. POPL 2020. The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.
- External Reviewer. POPL 2019. The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.
- PC Member. ECOOP 2018. European Conference on Object-Oriented Programming.
- ERC Member. PLDI 2018: ACM Conference on Programming Language Design and Implementation.
- ERC Member. OOPSLA 2016: ACM Conference on Object-Oriented Programming Systems, Languages, and Applications.
- PC Member. STOP 2016: Script To Program Evolution Workshop.
- PC Member. ESOP 2016: European Symposium on Programming.
- PC Member. OOPSLA 2015: ACM Conference on Object-Oriented Programming Systems, Languages, and Applications.
- PC Member. ICALP 2015: International Colloquium on Automata, Languages, and Programming.
- Program Chair. STOP 2015: Workshop on Script to Programs.
- ERC Member. PLDI 2015: The ACM Conference on Programming Language Design and Implementation.

- PC Member. PEPM 2015: the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation.
- PC Member. POPL 2014: The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.
- PC Member. GPCE 2013: The International Conference on Generative Programming: Concepts & Experiences.
- PC Member. PEPM 2013: The ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation.
- PC Member. WGP 2013: The 9th ACM SIGPLAN Workshop on Generic Programming.
- PC Member. PLDI 2013: The ACM Conference on Programming Language Design and Implementation.
- Review Committee Member. POPL 2013: The ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages.
- PC Member. ICSE 2012: NIER track of the International Conference on Software Engineering.
- PC Member. GPCE 2011: Generative Programming and Component Engineering.
- PC Member. DSL 2011: IFIP Working Conference on Domain-Specific Languages.
- PC Member. LCPC 2011: The International Workshop on Languages and Compilers for Parallel Computing.
- PC Member. Onward! 2011.
- Review Committee Member. ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2011).
- Program Chair. 2011 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM 2011).
- Program Chair. 2010 International Workshop on Foundations of Object-Oriented Languages (FOOL 2010).
- PC Member. The Fifth international Workshop on Automatic Performance Tuning (iWAPT 2010).
- PC Member. The ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI 2010).
- PC Member. 1st International Workshop on Script to Program Evolution (STOP 2009).
- General Chair. Generative Programming and Component Engineering (GPCE) 2009.
- PC Member. The ACM 2009 Conference on Programming Language Design and Implementation (PLDI).
- PC Member. IFIP Working Conference on Domain Specific Languages (DSL WC), July 2009.

- PC Member. ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2008)
- Program Chair. Generative Programming and Component Engineering (GPCE) 2008.
- PC Member. ACM SIGPLAN 2008 Workshop on Partial Evaluation and Program Manipulation (PEPM 2008).
- PC Member. First Workshop on Generative Technologies (WGT 2008).
- PC Member. European Conference on Object-Oriented Programming (ECOOP 2008).
- Panel member for the ECOOP 2007 Doctoral Symposium and Ph.D. Students Workshop.
- PC Member. Generative Programming and Component Engineering (GPCE) 2007.
- PC Member. The Boost Libraries Conference 2007.
- Co-Chair. ACM SIGPLAN 2007 Symposium on Library-Centric Software Design. Co-located with OOPSLA. I was responsible for obtaining ACM sponsorship.
- PC Member. Workshop on Generic Programming at ICFP 2006.
- Organizer. Library-Centric Software Design Workshop at OOPSLA 2005 and 2006.
- PC Member. Parallel Object-Oriented Scientific Computing Workshop at ECOOP 2005 and 2006.
- PC Member. The Second MetaOCaml Workshop at GPCE 2005.
- Reviewer. ESOP 2007, PARA 2006, POPL 2006, 2009, 2011, and 2017, PEPM 2009, ACM TOPLAS, ACM TOMS, ICFP 2003, Software Practice and Experience, Wiley Encyclopedia of Computer Science and Education (2008), Science of Computer Programming, Journal of Object Technology, Journal of Functional Programming..

Committees and Editing

- Associate Editor, ACM TOMS 2011–2018
  Transactions on Mathematical Software
- GPCE Steering Committee 2008–2013
  International Conference on Generative Programming and Component Engineering.
- FOOL Steering Committee (Chair) 2011–2014
  International Workshop on Foundations of Object-Oriented Languages.
- PEPM Steering Committee 2011–2014
  Workshop on Partial Evaluation and Program Manipulation.
- Guest Editor. Special issue of the journal Science of Computer Programming on Library-Centric Software Design.

- C++ ANSI/ISO Standards Committee 2001–2008
  Worked on the proposal for adding support for generic programming to C++ through the addition of special kinds of interfaces called "concepts". Served in the Library Working Group evaluating extensions to the C++ Standard Library. Worked on the iterator concept and iterator adaptor proposals for standard library extension.

Other

- Contributor to Boost C++ Open Source Group 1999–2005
  Contributed and maintained many software libraries including the Boost Graph Library. Served as review manager for the Boost Unit Test Library, the Boost Preprocessor Library, and the Boost MPI Library.