Michael M. Vitousek

Contact Information

mvitouse@indiana.edu
mmvitousek@gmail.com
http://homes.soic.indiana.edu/mvitouse
http://github.com/mvitousek

3025J Luddy Hall, IUB
Bloomington, IN 47408
(650) 269–3518

Research Interests

I am interested in designing programming languages and techniques that support users throughout the development cycle, from prototyping to production, and that reduce barriers to entry while guaranteeing properties programmers rely on. My research so far has focused on soundly combining static and dynamic typing within the same programming language, also known as *gradual typing*, and I'm especially interested in techniques for applying gradual typing to existing programming languages practically, soundly, and efficiently.

Education

Indiana University

Ph.D. candidate, Computer Science, 2013–present
*Formerly **University of Colorado Boulder**, 2011–2013*

- Research committee: Jeremy G. Siek (chair), Sam Tobin-Hochstadt, Amr Sabry, Lawrence S. Moss
- Dissertation: *Gradual Typing for Python, Unguarded*
- Expected defense date: August 2018

Willamette University

B.A., Computer Science and History (minor in Mathematics), *magna cum laude*, December 2010

Employment History

Indiana University

Graduate research assistant (with Jeremy G. Siek), June 2015–present
Associate instructor (with Jeremy G. Siek), January 2016–May 2016
Graduate research assistant (with Jeremy G. Siek), August 2013–December 2015

Mozilla

Research intern (with Luke Wagner), May 2013–August 2013

Adobe Systems

Research intern (with Avik Chaudhuri), May 2012–August 2012

University of Colorado Boulder

Graduate research assistant (with Jeremy G. Siek), September 2012–May 2013
Graduate research assistant (with Jeremy G. Siek), June 2011–April 2012

Honors and Awards

2015–16	Nominated, associate instructor of the year, IU School of Informatics and Computing
2013	Honorable mention, NSF Graduate Research Fellowship
2013	Google Lime Scholarship
2012	Third place, PLDI Student Research Competition
2011–13	Chancellor's Fellowship, University of Colorado Boulder
2011–12	University Fellowship, University of Colorado Boulder
2010	Phi Beta Kappa, Willamette University
2009	Meritorious Winner, Mathematical Competition in Modeling
2008	Kenneth G. Batchelder Memorial Scholarship, Willamette University

Conference Publications

M.M. Vitousek, C. Swords, J.G. Siek. *Big types in little runtime: open world soundness and collaborative blame for gradual type systems*. POPL '17: Symposium on Principles of Programming Languages (January 2017).

J.G. Siek, M.M. Vitousek, M. Cimini, J.T. Boyland. *Refined criteria for gradual typing*. SNAPL '15: Summit on Advances in Programming Languages (May 2015).

J.G. Siek, M.M. Vitousek, M. Cimini, S. Tobin-Hochstadt, R. Garcia. *Monotonic references for efficient gradual typing*. ESOP '15: European Symposium on Programming (April 2015).

M.M. Vitousek, A.M. Kent, J.G. Siek, J. Baker. *Design and evaluation of gradual typing for Python*. DLS '14: Symposium on Dynamic Languages (October 2014).

Workshop Appearances

M.M. Vitousek, J.G. Siek. *From optional to gradual typing via transient checks*. STOP '16: Script To Program Evolution Workshop (July 2016).

J.G. Siek, M.M. Vitousek, J.D. Turner. *Effects for funargs*. HOPE '12: Workshop on Higher-Order Programming with Effects (September 2012).

M.M. Vitousek, S. Bharadwaj, J.G. Siek. *Towards gradual typing in Jython*. STOP '12: Script To Program Evolution Workshop (June 2012).

Technical Reports and Work in Progress

M.M. Vitousek, J.G. Siek, A. Chaudhuri. *Optimizing and evaluating transient gradual typing*. Draft (March 2018).

M.M. Vitousek, J.G. Siek. *Gradual typing in an open world*. Indiana University Technical Report TR729 (October 2016).

J.G. Siek, M.M. Vitousek, J.D. Turner. *Effects for funargs*. Draft (December 2011).

Selected Talks

Optimizing and evaluating transient gradual typing with trusted type inference. Facebook, Inc. (December 2017).

Big types in little runtime: open world soundness and collaborative blame for gradual type systems. POPL '17: Symposium on Principles of Programming Languages (January 2017).

From optional to gradual typing via transient checks. STOP '16: Script To Program Evolution Workshop (July 2016).

Lightweight gradual typing with transient checks. PI Meeting, NSF Award 1518844, SHF Large: Gradual Typing Across the Spectrum (May 2016).

Design and evaluation of gradual typing for Python. DLS '14: Symposium on Dynamic Languages (October 2014).

Empirical analysis of megamorphic property accesses. Mozilla (August 2013).

Gradual typing with efficient object casts. PLDI Student Research Competition, final round (June 2012).

Towards gradual typing in Jython. STOP '12: Script To Program Evolution Workshop (June 2012).

Major Software

Reticulated Python: Gradual Typing for Python. December 2012–present.
`https://github.com/mvitousek/reticulated`

Research Experience

Runtime techniques for gradual typing

For interaction between statically and dynamically typed code to be safe with respect to the expected static types from the static code, runtime checks need to be performed. For higher-order types like functions and mutable objects, these checks cannot be performed eagerly (when values cross from static to dynamic or vice versa) but need to ensure that when the values are used, the result is of the expected type. Historically this has been performed by installing proxies (also called wrappers) on higher-order values, but this leads to problems in performance and compatibility. I am interested in alternative designs to runtime enforcement for gradual typing. I developed and formalized the *transient* approach to runtime checks (**DLS '14**, **POPL '17**), which uses pervasive shallow checks rather than proxies. This approach is promising for transitioning from optional (unsafe) typing to safe gradual typing (**STOP '16**), and it supports blame tracking using a side-channel strategy (**POPL '17**). I identified the *open-world soundness* property, supported by the transient approach, which allows gradually typed programs to safely be embedded in dynamic, open-world contexts (**POPL '17**). In upcoming work I show that, in combination with type inference, the run-time overhead of transient can be minimized.

With colleagues I also developed the *monotonic* approach, which "locks down" mutable values at their most specific static type for compiler optimizations (**STOP '12**, **ESOP '15**).

Gradual typing for Python (and other languages)

The Python programming language is dynamically typed, but many of its users desire the option to use static types where desired. I have worked on implementing gradual typing in the Python programming language, first by modifying the Jython compiler to support gradual typing (**STOP '12**), and then by developing a source-to-source translator, Reticulated Python, which typechecks gradually typed source programs and then translates them to valid Python 3 code with safety-preserving runtime checks inserted (**DLS '14**). I used Reticulated Python as an experimental platform in developing the transient and monotonic approaches described above. I also took the same approach in developing CheckScript, a version of TypeScript that uses the transient design to ensure runtime safety (**STOP '16**). I also assisted in the development of PEP 484, the Python standard for type annotations.

Gradual typing criteria

I worked with colleagues to introduce and develop the *gradual guarantee*, a crucial formal property for gradually typed languages that ensures that programs in such languages can be gradually evolved from dynamic to static as long as there exists a corresponding statically-typed version of the program (**SNAPL '15**). I also introduced *open-world soundness*, which allows gradually typed programs to safely be embedded in dynamic, open-world contexts (**POPL '17**).

Empirical analysis of JavaScript megamorphism

(at **Mozilla**, May–August 2013, with Luke Wagner.)
I worked at Mozilla as a research intern on the JavaScript engine team. I instrumented the Firefox JavaScript JIT, IonMonkey, to determine the overhead of fallback inline-caches at megamorphic property access sites, and I used this to determine the most important design patterns and use cases that result in this overhead (mixins and monolithic functions).

Joint dispatch for binary methods

(at **Adobe Systems**, May–August 2012, with Avik Chaudhuri.)
I worked at Adobe Systems as a research intern on the ActionScript team. I developed a new approach to the binary method problem, allowing for binary methods to be implemented in Java-like languages without requiring F-bounded polymorphism or exact This-types. I also worked with Avik Chaudhuri to design a system of bounded generics defined in terms of type intervals, and developed techniques to efficiently implement such a design.

Upwards funargs and effects
(at **University of Colorado Boulder**, June–November 2011, with Jeremy G. Siek and Jonathan D. Turner.)
I worked with colleagues to create a type-and-effect system which detects possible dangling references in stack-allocating languages with first-class functions (**HOPE '12**). This, combined with a kind of function-local storage, serves as a solution to the classic upwards funarg problem, and increases the viability of first-class functions in stack-allocating languages like Chapel.

Service

Reviewing for ESOP '16: European Symposium on Programming (April 2016).

Reviewing for OOPSLA '15: Conference on Object-Oriented Programming Systems, Languages, and Applications (October 2015).

Reviewing for POPL '14: Symposium on Principles of Programming Languages (January 2017).

Teaching Experience

Fall	2016	Mentor, visiting Northeastern University undergraduate researcher
Spring	2016	Associate instructor, programming language implementation
Summer	2014	REU mentor, dynamic analysis of gradually typed programs
Spring	2013	Lecturer, readings in gradual typing
Spring	2008	Teaching assistant, intro to programming
Fall	2007	Teaching assistant, intro to programming

Selected Graduate Coursework

- ❑ Programming Language Theory
- ❑ Homotopy Type Theory
- ❑ Program Analysis
- ❑ Software Engineering
- ❑ Theorem-proving using Isabelle
- ❑ Compilers
- ❑ Cyberphysical Systems
- ❑ Theory of Computation
- ❑ Analysis of Algorithms
- ❑ Operating Systems

Other Experience

Summers, 2008–10	Programming Languages Summer School, **University of Oregon**
Summers, 2006–08	Biogeochemistry lab and field assistant, **Stanford University**