My PL journey goes through Benjamin

Teacher, Role Model, Collaborator, Friend

Michael Hicks

Senior Principal Scientist, Amazon

Professor Emeritus, University of Maryland

All content and opinions are my own; they do not represent my employer

About me

- **Professor** in Computer Science at the **University of Maryland** from 2002-2021
- **Senior Principal Scientist** at **AWS** since 2022
- Professional interests in **Programming Languages**, **Software Engineering** and **Software Security**

My career path was shaped in many ways by the good fortune I had in meeting Benjamin

Pre-PhD

- My first real job was in 1993, to build IT and comms for the airlines.

ARINC

- I was a bit shocked at how badly written some of the code was. Could it be dangerous to have bugs?

Grad school

- Make software better! Started 1995
- My advisor was Scott Nettles, who worked on garbage collection
- My first paper was a systems-oriented study on GC mechanisms

Uniprocessor Garbage Collection Techniques
[Submitted to ACM Computing Surveys]

Paul R. Wilson

Abstract

We survey basic garbage collection algorithms, and variations such as incremental and generational collection; we then discuss low-level implementation considerations and the relationships between storage management systems, languages, and compilers. Throughout, we attempt to present a unified view based on abstract traversal strategies, addressing issues of conservatism, opportunism, and immediacy of reclamation; we also point out a variety of implementation details that are likely to have a significant impact on performance.

Contents

The Measured Cost of Copying Garbage Collection Mechanisms

Michael W. Hicks, Jonathan T. Moore, and Scott M. Nettles
Computer and Information Science Department
University of Pennsylvania
Philadelphia, PA 19103
{mwh,jonm,nettles}@dsl.cis.upenn.edu

ICFP 1997

Abstract

We examine the costs and benefits of a variety of copying garbage collection (GC) mechanisms across multiple architectures and programming languages. Our study covers both low-level object representation and copying issues as well as the mechanisms needed to support more advanced techniques such as generational collection, large object spaces, and type-segregated areas.

Our experiments are made possible by a novel performance analysis tool, *Oscar*. Oscar allows us to capture

1 Introduction

Garbage collection (GC) is an important feature of many programming languages, "functional" ones in particular. Despite the importance of GC performance, there is surprisingly limited information available to guide implementors in how to design their languages, data representations, and runtime systems so that the collector has good performance. In practice, implementors either tune their systems based on their own experiments and guesses, or worse, just ignore the performance of the collector.

Focused on Systems-style PL

Oscar - The GC TestBed

Oscar is a tool for measuring the performance of copying garbage collection in a controlled, repeatable environment. This is acheived by capturing snapshots of programming language heaps that may then be used to replay garbage collections. The replay program is self-contained and written in C, which makes it easy to port to other architectures and to analyze with standard performance analysis tools. Furthermore, it is possible to study additional programming languages simply by instrumenting existing implementations to capture heap snapshots.

This research is being conducted by Michael Hicks, Jonathan Moore, Luke Hornof, and Scott Nettles in the University of Pennsylvania's Department of Computer and Information Science.

The initial findings of the study were presented at ICFP '97. Here are the slides.

Recently we have expanded our study to look more closely at questions regarding large object spaces. Our findings are presented in ISMM '98.

The SwitchWare Project

Active Network Research at Penn and Bellcore

> *Active networks explore the idea of allowing routing elements to be extensively programmed by the packets passing through them. This allows computation previously possible only at endpoints to be carried out within the network itself, thus enabling optimizations and extensions of current protocols as well as the development of fundamentally new protocols.*

Welcome to the SwitchWare home page, describing the Active Networks research effort underway in the Penn Department of Computer and Information Science and Bellcore as well as pointers to related material.

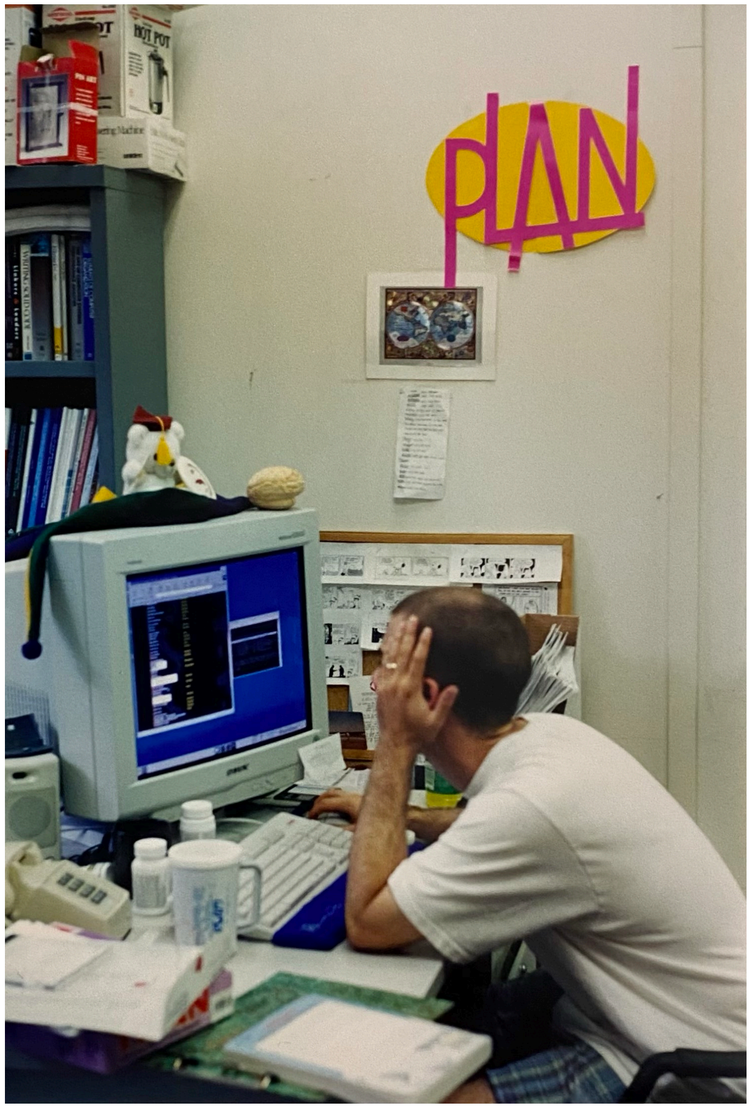
Activities

- The KeyNote Trust-Management System *(software available)*
- Packet Language for Active Networks: PLAN *(software available!)*
- Secure Active Network Environment (SANE) *(software available)*
- Active Bridge *(software available!)*
- Active Network Encapsulation Protocol (ANEP): ANEP *(software available!)*
- Security infrastructure based on Query Certificate Managers.
- SANE O/S
- Cyclone certifying run-time code generator

Garbage collection

and

Active
(Programmable)
Networks



PL/semantics non-match until …

My initial exposure to PL theory in Spring 1996 was unsuccessful

Math-oriented intuitions didn't land for me

Full Abstraction and the Context Lemma[1]

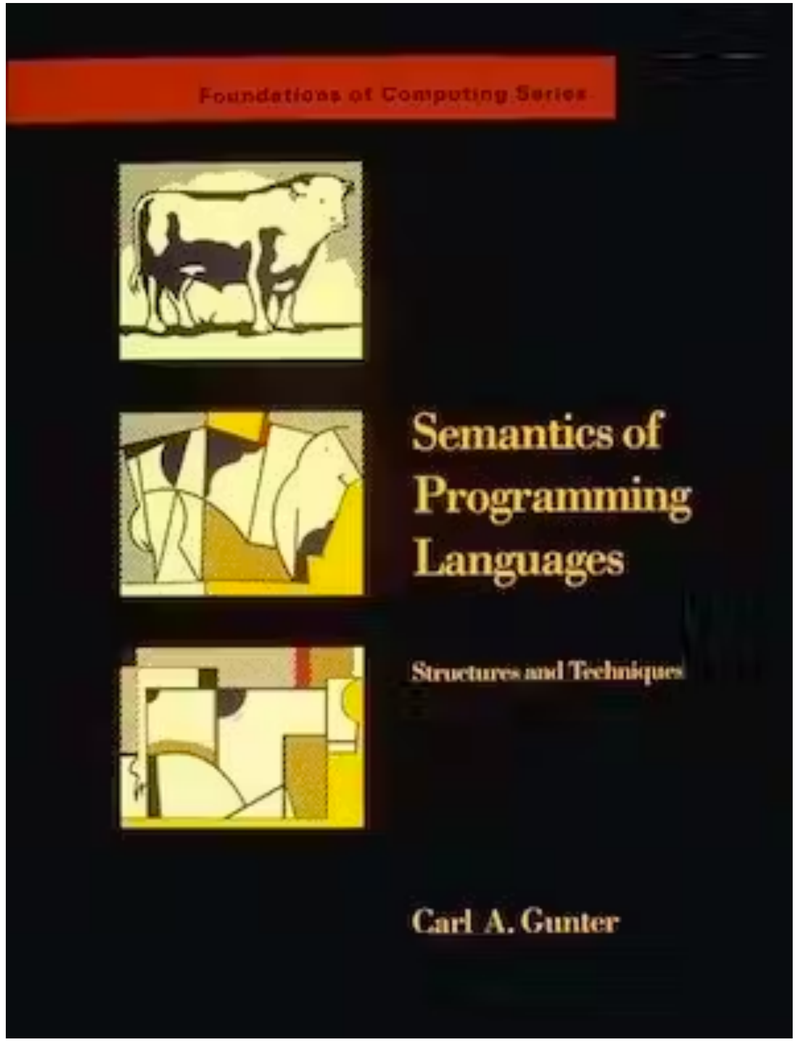
Trevor Jim[2] and Albert R. Meyer[3]
MIT Laboratory for Computer Science

December 20, 1991

[1]Extended version of the paper appearing in the proceedings of TACS '91 (Meyer and Ito, Eds., volume 526 of *Lecture Notes in Computer Science*, pages 131–151, Springer-Verlag, 1991).

[2]E-mail: trevor@theory.lcs.mit.edu. Work supported by ARO grant DAAL03-89-G-0071.

[3]E-mail: meyer@theory.lcs.mit.edu. Work supported by ONR grant N00014-89-J-1988 and NSF grant 8819761-CCR.



64°F

Searchers: Google Yahoo Altavista Excite DejaNews Four11

News: Slashdot NYT IHT Freshmeat LinuxToday ZDNet CNet

People/papers: PL folks Hypatia CS bibs Citeseer Cora

Misc: Amazon IMDB MapsOnUs

Images: icons more icons backgrounds

Home
Coordinates
Papers/software
Courses

Benjamin C. Pierce

Associate professor
Department of Computer and Information Science
University of Pennsylvania

Research

- **Outline**
- **Papers and software**
- **Current projects**
  - the **Unison** file synchronizer
  - the **XDuce** XML processing language
  - **TinkerType**, a framework for modular development of type systems

Teaching

- **Course materials** (including Languages for Programming the Web)

Professional Activities

- **Current/recent conferences:** TACS 2001 (PC co-chair), ICFP 2001 (general chair), DBPL 2001 (PC), WWW 10 (PC), FST & TCS 2000 (PC), TCS 2000 (PC), OOPSLA 2000 (PC), LICS 2000 (PC), Mobile99 (PC), ICFP '99 (PC), MOS '99 (PC), FOOL '99 (PC), POPL '99 (PC)
- **Editorial boards:** ACM Transactions on Programming Languages and Systems (TOPLAS), Formal Aspects of Computing (FAC), Discrete Mathematics & Theoretical Computer Science (DMTCS)
- **Logic and Computation** group (Logic and Computation Seminar)
- **Types** Forum (send subscription requests to types-request@cis.upenn.edu)

Information

- **Coordinates** and PGP key
- **Capsule bio**

1999

Cutting edge
Web 1.0 design!

https://www.cis.upenn.edu/~bcpierce/

Benjamin C. Pierce

2024

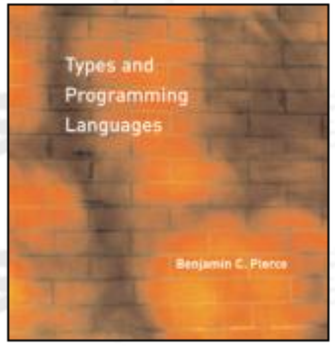
Cutting edge

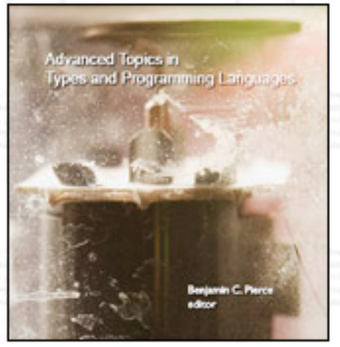
Web 1.0 design!

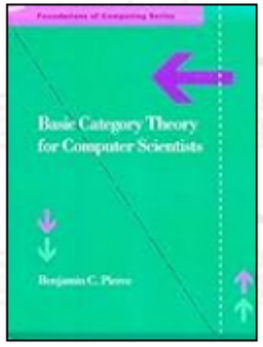
Professor
Department of Computer and Information Science
University of Pennsylvania

Information

- **Coordinates**, telephone numbers, schedule, and office hours
- **Capsule bio**







Research

- **Recent publications and talks** NEW
- **All papers and software**
- **Current/Recent Projects:**
  - **Property-based testing** NEW
  - **DeepSpec: The Science of Deep Specification**
  - **Distributed differential privacy**
  - **Unison**: A robust, portable file synchronizer
- **Books:**
  - ***Software Foundations*** series NEW
  - ***Types and Programming Languages***
  - ***Advanced Topics in Types and Programming Languages***
  - ***Basic Category Theory for Computer Scientists***
- The Penn ***PL Club***

deep spec

Climate Change

- **At Penn:**
  - **CIRCE:** Faculty Senate Select Committee on the Institutional Response to the Climate Emergency
  - **Clio:** Climate Impact Offset charge
  - Penn Faculty Climate Pledge
- **Conferences and air travel:**

https://www.cis.upenn.edu/~bcpierce/

1999

Benjamin Pierce's Plan File

```
Address:
  University of Pennsylvania
  Dept. of Computer & Information Science
  200 South 33rd Street
  Philadelphia, PA  19104-6389

Office:         Room 368 Moore Bldg.-GRW

Office phone:   +1 215 898-2012
Office fax:     +1 215 898-0587

Assistant:      Cheryl Hickey
                cherylh@central.cis.upenn.edu
                +1 215 898-3538

Email:          bcpierce@cis.upenn.edu
www:            http://www.cis.upenn.edu/~bcpierce
```

https://www.cis.upenn.edu/~bcpierce/plan.html

2004

Benjamin Pierce's Plan File

```
Address:
  University of Pennsylvania
  Dept. of Computer & Information Science
  3330 Walnut Street
  Philadelphia, PA  19104-6389

Office:         Levine Hall, Room 304

Office phone:   +1 215 898-2012
Office fax:     +1 215 898-0587

Office hours:   (SEMI-)REGULAR HOURS FOR SPRING, 2004:
                   I am NORMALLY in my office on Thursdays from 4:00 to 5:30,
                   but please drop me a note if you intend to come by so that
                   I can make sure to be there.

Assistant:      Jennifer Finley
                +1 215 898-8560

Email:          bcpierce ATSIGN cis DOT upenn DOT edu
www:            http://www.cis.upenn.edu/~bcpierce
```

sophisticated anti-spam measure

https://www.cis.upenn.edu/~bcpierce/plan.html

2024

Benjamin Pierce's Coordinates

Address: University of Pennsylvania
Dept. of Computer & Information Science
3330 Walnut Street
Philadelphia, PA 19104-6389

Office: Levine Hall, Room 562

Office phone: +1 215 898-6222
Office fax: +1 215 898-0587

Office hours: By appointment

Email: bcpierce ATSIGN cis DOT upenn DOT edu
www: ~~http://www.cis.upenn.edu/~bcpierce~~

~~sophisticated~~
“ anti-spam ”
measure

https://www.cis.upenn.edu/~bcpierce/plan.html

PL/semantics match!

My exposure to PL theory in Benjamin's class, based on notes predating TAPL, was a hit!

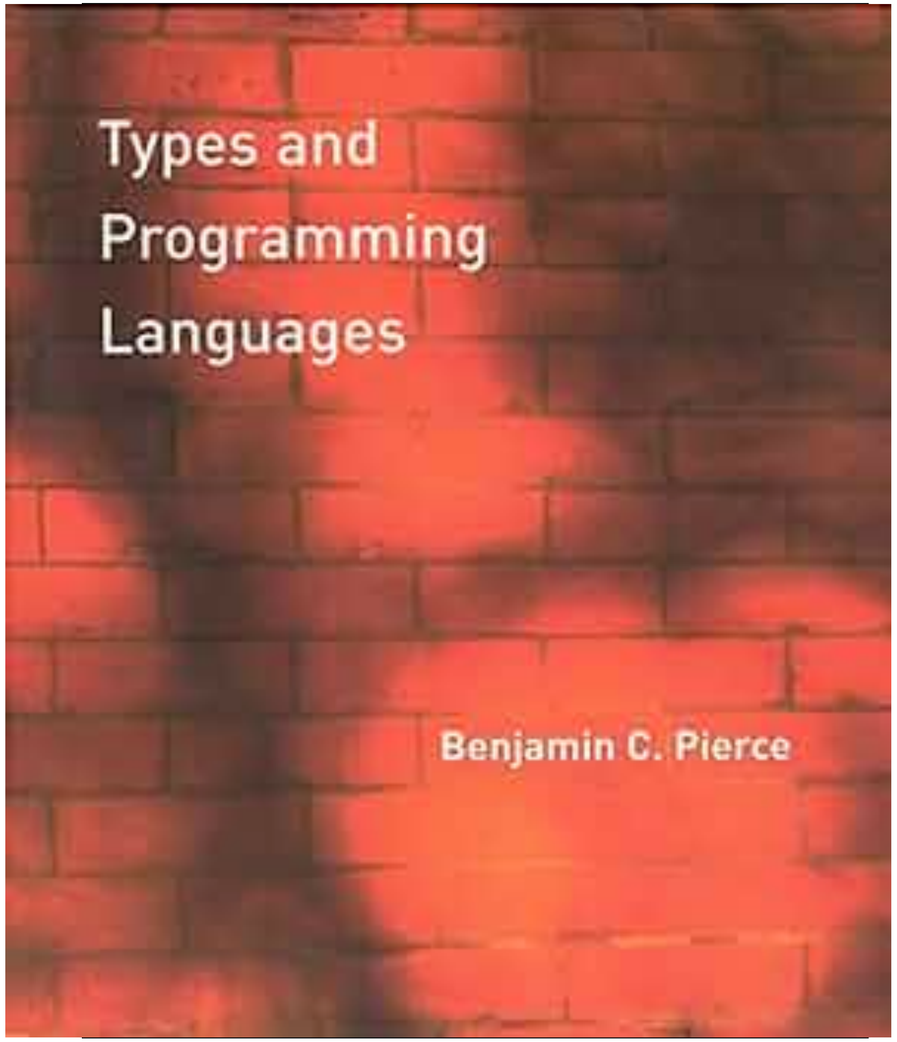
Its **programming-oriented intuitions** landed **much** better for me

Our first job is to define a type of OCaml values representing terms. OCaml's datatype definition mechanism makes this easy: the following declaration is a straightforward transliteration of the grammar on page 24.

```
type term =
    TmTrue of info
  | TmFalse of info
  | TmIf of info * term * term * term
  | TmZero of info
  | TmSucc of info * term
  | TmPred of info * term
  | TmIsZero of info * term
```

The constructors `TmTrue` to `TmIsZero` name the different sorts of nodes in the abstract syntax trees of type `term`; the type following `of` in each case specifies the number of subtrees that will be attached to that type of node.

Each abstract syntax tree node is annotated with a value of type `info`,



Thesis Committee

DYNAMIC SOFTWARE UPDATING

Michael Hicks

A DISSERTATION
in

Computer and Information Science

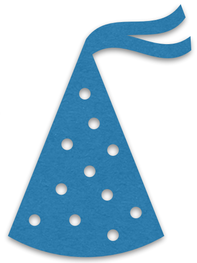
Acknowledgements

…

I must also thank my committee for their input and feedback. Any competence I have in programming language theory I owe to Benjamin Pierce. His wonderful course on type systems provided insight as well as information, and greatly broadened my understanding in the area.

…

Thesis defense after-party at Benjamin's house!



Writing the PL way

- Benjamin made me rewrite chapters 5 and 6 of my dissertation, “for my own good”
- It was fun to do, and illuminating! Felt great to get Benjamin’s seal of approval

Type Environment Values χ

Operators		
meet	$\chi_1 \sqcap \chi_2$	$\top \sqcap \chi = \chi \quad \chi \sqcap \top = \chi \quad \chi \sqcap \chi = \chi$
approximates	$\chi_1 \leq \chi_2$	$\chi \leq \top \quad \chi \leq \chi$
Relations		
compatible	$\chi_1 \diamond \chi_2$	$\chi_1 \leq \chi_2$ or $\chi_2 \leq \chi_1$

Type Environments X

Operators		
restriction	$X_1 - X_2$	X_1 restricted to labels not in $\mathtt{dom}(X_2)$
disjoint union	$X_1 \uplus X_2$	Union of disjoint maps, defined if $X_1 \mid X_2$
merge	$X_1 \oplus X_2$	Union of compatible maps (defined if $X_1 \diamond X_2$), maps $n \in \mathtt{dom}(X_1) \cap \mathtt{dom}(X_2)$ to $X_1(n) \sqcap X_2(n)$
Relations		
disjoint	$X_1 \mid X_2$	$\mathtt{dom}(X_1)$ and $\mathtt{dom}(X_2)$ are disjoint
link compatible	$X_1 \precsim X_2$	For n in $\mathtt{dom}(X_1) \cap \mathtt{dom}(X_2)$, $X_1(n) \leq X_2(n)$
compatible	$X_1 \diamond X_2$	For n in $\mathtt{dom}(X_1) \cap \mathtt{dom}(X_2)$, $X_1(n) \diamond X_2(n)$

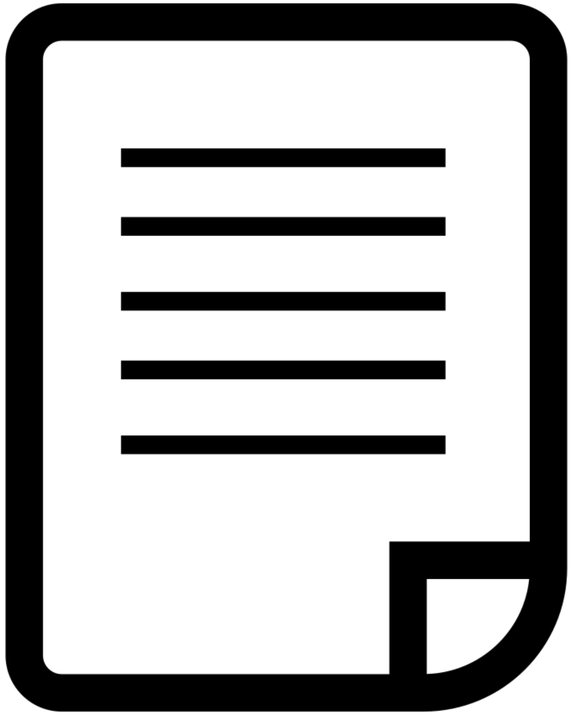
Figure 5.10: Type environments and type environment values: operators and relations

same names defined in the exports. We can do this by merging the imports ($X_I^1 \oplus X_I^2$), and then subtracting any labels mentioned in the combined exports: $(X_I^1 \oplus X_I^2) - (X_H^1 \uplus X_H^2)$. These operations are only well-formed if (1) the two export environments are disjoint (*i.e* $X_H^1 \mid X_H^2$), and (2) if the two imports are compatible (*i.e.* $X_I^1 \diamond X_I^2$). We also add the restrictions $X_H^1 \precsim X_I^2$ and $X_H^2 \precsim X_I^1$. This prevents the import environment from one interface from replacing $\top$ mapped to by some name in the export environment of the other interface. Doing so would break the abstraction enforced by `hide` and `reveal`.

Definition 5.3.2 (Type Interface Linking)

$$\frac{X_I^1 \diamond X_I^2 \qquad X_H^1 \precsim X_I^2 \qquad X_H^2 \precsim X_I^1 \qquad X_H^1 \mid X_H^2}{(X_I^1, X_H^1)\ \mathtt{link}\ (X_I^2, X_H^2) \Rightarrow (X_I^3, X_H^3)} \left(\begin{array}{c} X_H^3 = X_H^1 \uplus X_H^2 \\ X_I^3 = ((X_I^1 \oplus X_I^2) - X_H^3) \end{array} \right)$$

Communication ~~Optional?~~ Essential

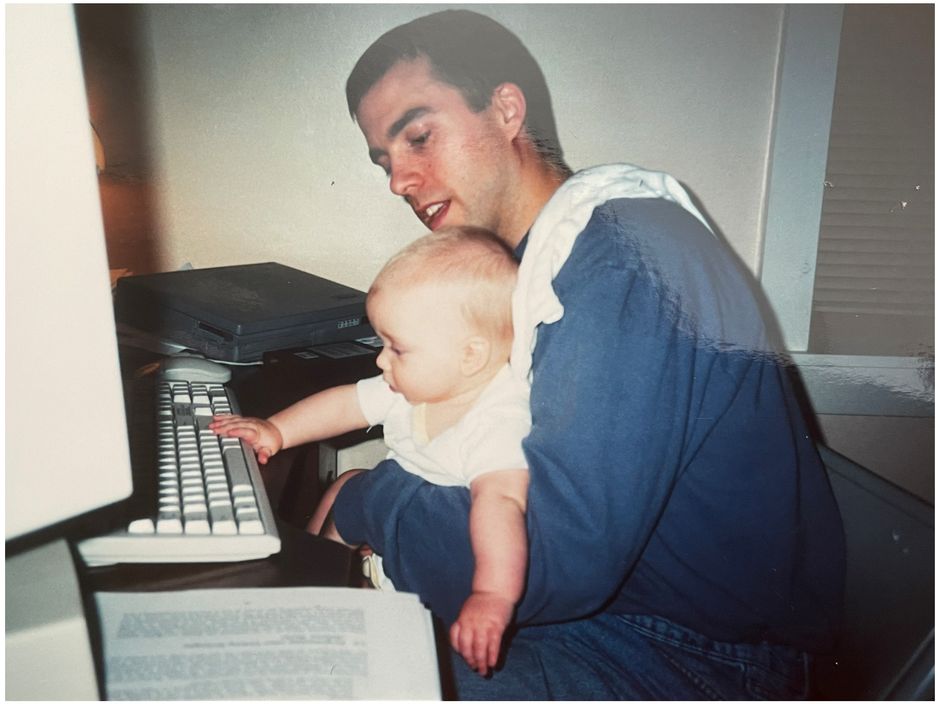


Work rarely speaks for itself. You have to speak for it

https://docs.google.com/document/d/1_vBXbugoLjO171w3wovs3ugmRQI-O6EcSVFDBF7eUzE/edit

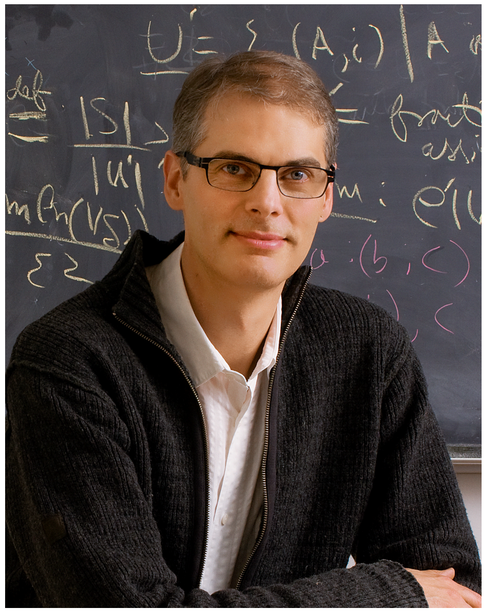
PhD!

- After six years at Penn I graduated
- On to professor life!



Role models

- Benjamin, Scott, Greg, and Jonathan set a fantastic example for me, showing how to be a successful Prof
- As any parent knows, your kids do what you do, not what you say. And they say what you say (so do you do it?)

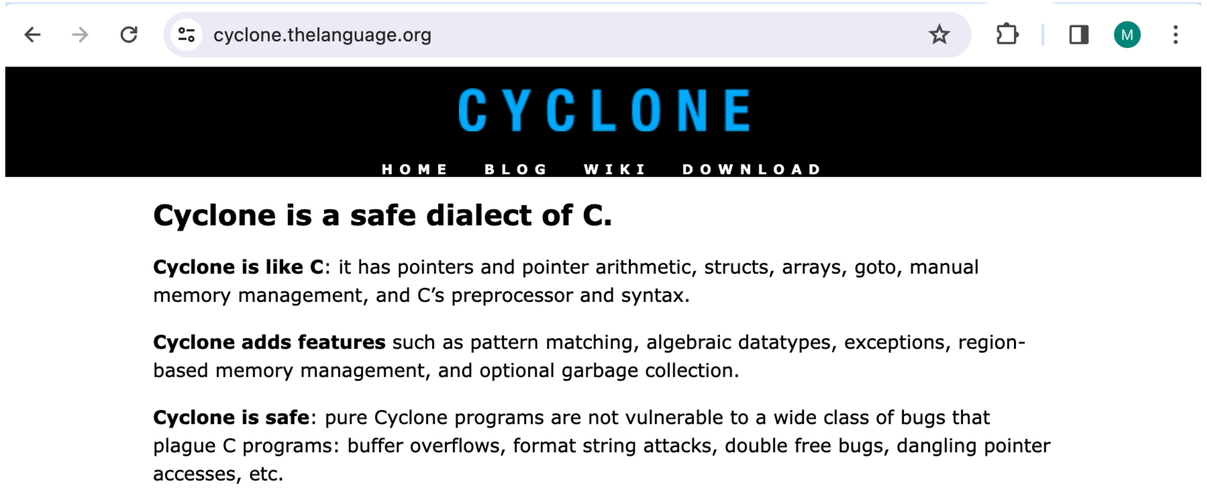


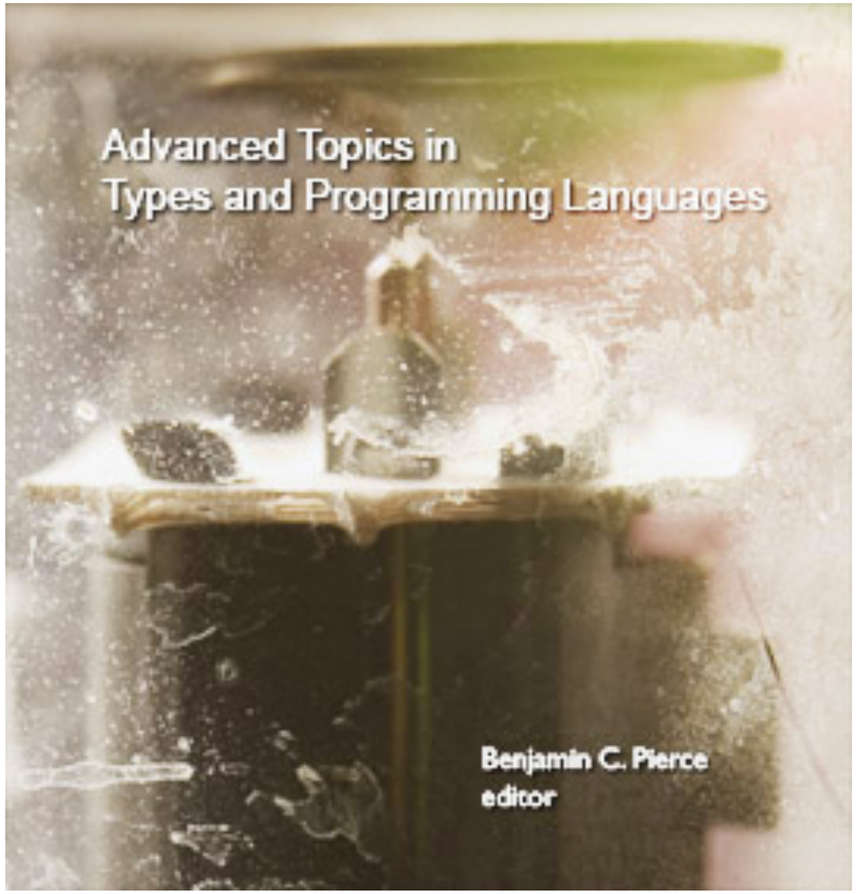

Path to POPL

- In 2002: ICFP, OOPSLA, PLDI, … This is my comfort zone. But publishing in POPL: no way!
- Benjamin helped me reach that milestone (mostly without knowing it)

Safe Memory Management

- Work on Cyclone safe memory management involves linearity





- **Benjamin's ATTAPL** (Dave Walker's chapter) was a big help
  - But too intimidated to formalize it myself

Qualified Types for Java

- First formalization and soundness proof on my own: OOPSLA'04
- Formalization extended **Benjamin's** co-authored work on **Featherweight Java**

Terms:

$$
\begin{array}{rcl}
CL & ::= & \textbf{class } C \textbf{ extends } C \ \{ \ \bar{T} \ \bar{f}; K \ \bar{M} \ \} \\
K & ::= & C(\bar{T} \ \bar{f}) \ \{ \ \textbf{super}(\bar{f}); \ \textbf{this}.\bar{f} = \bar{f}; \ \} \\
M & ::= & T \ m(\bar{T} \ \bar{x}) \ \{ \ \textbf{return } e; \} \\
\mathcal{E} & ::= & x \mid e.f \mid e.m(\bar{e}) \mid \textbf{new } C(\bar{e}) \mid (C)e \\
 & \mid & \textbf{let } x = e \textbf{ in } e \mid \textbf{makeproxy } e \\
 & \mid & \textbf{if } e = e \textbf{ then } e \textbf{ else } e \\
e & ::= & \mathcal{E}^l
\end{array}
$$

Types:

$$
\begin{array}{rcl}
C, D, E & & \text{class names} \\
Q & ::= & \textsf{proxy} \mid \textsf{nonproxy} \mid \kappa \\
\varphi & ::= & \{C_1, \ldots C_n\} \mid \alpha
\end{array}
$$

Transparent Proxies for Java Futures

Polyvios Pratikakis
polyvios@cs.umd.edu

Jaime Spacco
jspacco@cs.umd.edu

Michael Hicks
mwh@cs.umd.edu

Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

A *proxy* object is a surrogate or placeholder that controls access to another target object. Proxies can be used to support distributed programming, lazy or parallel evaluation, access control, and other simple forms of behavioral reflection. However, *wrapper proxies* (like *futures* or *suspensions* for yet-to-be-computed results) can require significant code changes to be used in statically-typed languages, while proxies more generally can inadvertently violate assumptions of transparency, resulting in subtle bugs.

To solve these problems, we have designed and implemented a simple framework for proxy programming that employs a static analysis based on qualifier inference, but with additional novelties. Code for using wrapper proxies is automatically introduced via a classfile-to-classfile transformation, and potential violations of transparency are signaled to the programmer. We have formalized our analysis and proven it sound. Our framework has a variety of applications, including support for asynchronous method calls returning futures. Experimental results demonstrate the benefits of our framework: programmers are relieved of managing and/or checking proxy usage, analysis times are reasonably fast, overheads introduced by added dynamic checks are negligible, and performance improvements can be significant. For example, changing two lines in a simple RMI-based peer-to-peer application and then using our framework resulted in a large performance gain.

Keywords

Java, future, proxy, type inference, type qualifier

1. INTRODUCTION

A *proxy* object is a surrogate or placeholder that controls access to another object. One example of a proxy is a *future*, popularized in MultiLisp [23]. In MultiLisp, the syntax `(future` e`)` designates that expression e should be evaluated concurrently. A future for it is returned, and some time later the program *claims* the future, possibly blocking until the result of evaluating e is available. For example, in the following code, the two lists `x` and `y` are sorted in parallel, the former in a new thread, and the latter in the parent thread:

```
(merge (future (mergesort x)) (mergesort y))
```

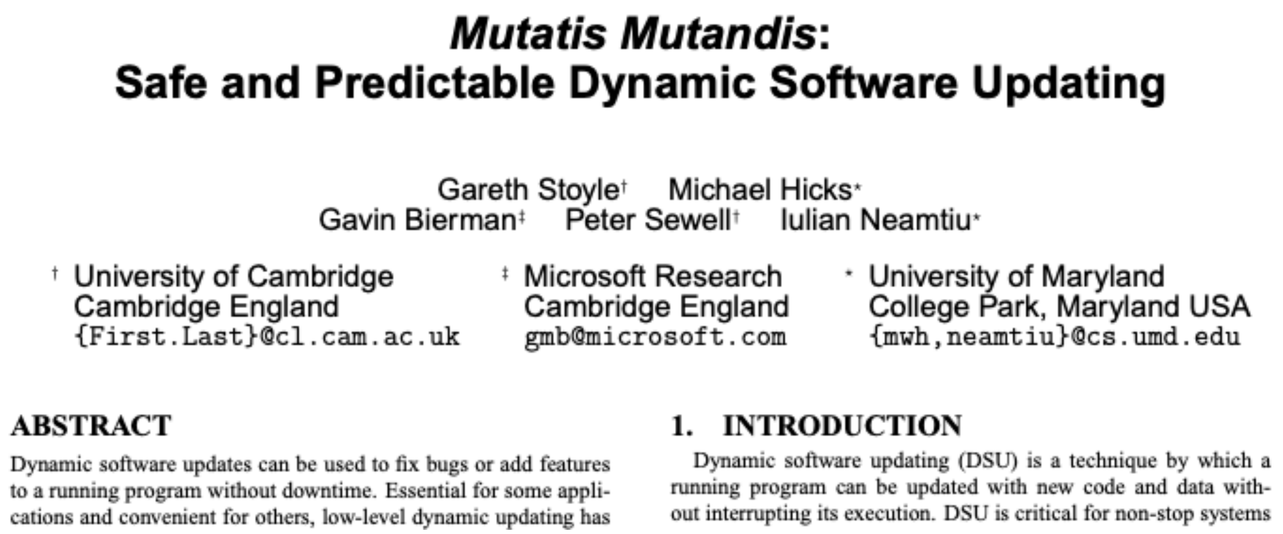
The results of both `mergesort` computations are passed to the `merge` routine; the first argument will be a future while the second argument will be a sorted list.

In MultiLisp, claims are performed transparently by the interpreter. In our example, this allows the programmer to write `merge` as if it takes two sorted lists as arguments, and the interpreter will perform claims as necessary. In general, the programmer simply inserts `future` annotations in the program and the runtime transparently takes care of the rest.[1] This makes the use of futures simple and lightweight.

A future is an example of a *wrapper proxy* in that it wraps the actual result; whenever the actual result is needed, the future must be unwrapped to retrieve it. Other examples of wrapper proxies include *suspensions*, which are wrappers

POPL: achieved!

- Formalized approach to dynamic updating: POPL'05
  - Work with **Benjamin's collaborator**, Peter Sewell (yes, a tenuous connection …)
- Continued gaining experience and then …
- Paper on contextual effects at POPL'08 was well received, which was a big confidence boost: *Have I figured this stuff out?*



***Mutatis Mutandis*:**
Safe and Predictable Dynamic Software Updating

Gareth Stoyle† Michael Hicks⋆
Gavin Bierman‡ Peter Sewell† Iulian Neamtiu⋆

† University of Cambridge
Cambridge England
{First.Last}@cl.cam.ac.uk

‡ Microsoft Research
Cambridge England
gmb@microsoft.com

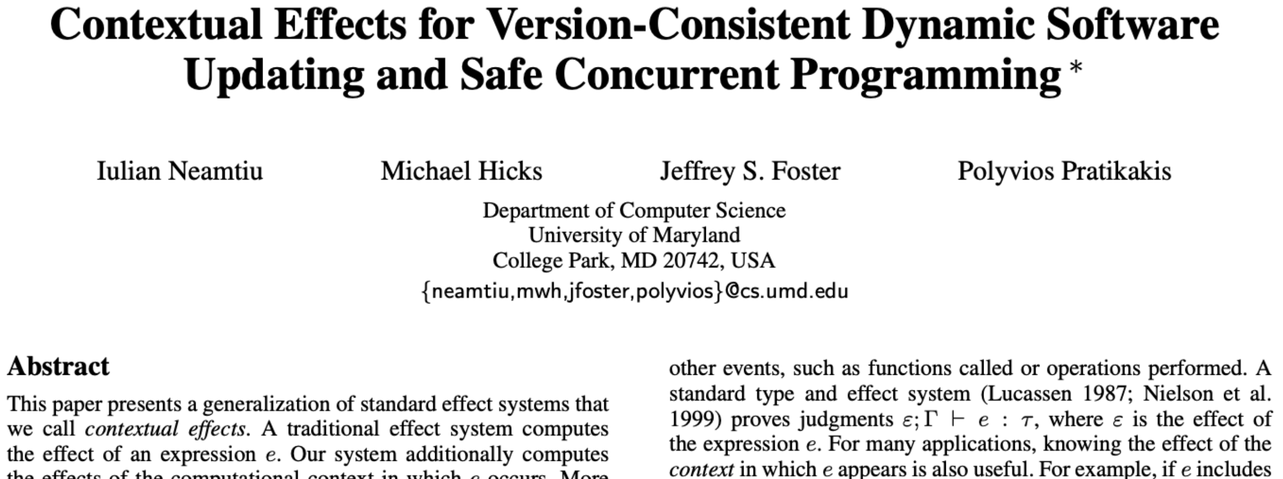
⋆ University of Maryland
College Park, Maryland USA
{mwh,neamtiu}@cs.umd.edu

ABSTRACT

Dynamic software updates can be used to fix bugs or add features to a running program without downtime. Essential for some applications and convenient for others, low-level dynamic updating has

1. INTRODUCTION

Dynamic software updating (DSU) is a technique by which a running program can be updated with new code and data without interrupting its execution. DSU is critical for non-stop systems



Contextual Effects for Version-Consistent Dynamic Software Updating and Safe Concurrent Programming*

Iulian Neamtiu Michael Hicks Jeffrey S. Foster Polyvios Pratikakis

Department of Computer Science
University of Maryland
College Park, MD 20742, USA
{neamtiu,mwh,jfoster,polyvios}@cs.umd.edu

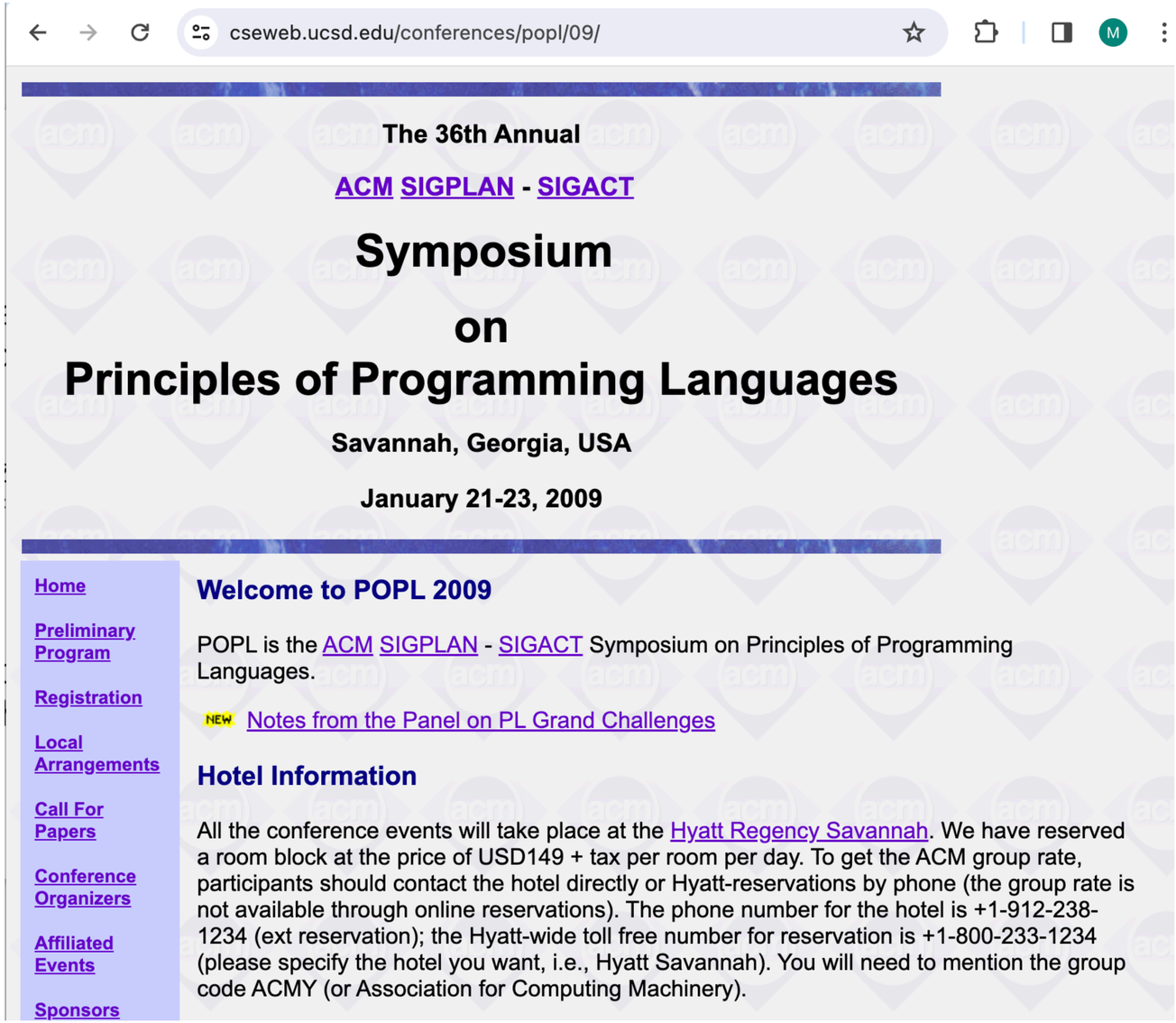
Abstract

This paper presents a generalization of standard effect systems that we call *contextual effects*. A traditional effect system computes the effect of an expression e. Our system additionally computes the effects of the computational context in which e occurs. More

other events, such as functions called or operations performed. A standard type and effect system (Lucassen 1987; Nielson et al. 1999) proves judgments $\varepsilon; \Gamma \vdash e : \tau$, where ε is the effect of the expression e. For many applications, knowing the effect of the *context* in which e appears is also useful. For example, if e includes

POPL'09 PC

- **Benjamin invited me to be on the POPL'09 PC**
  - Thrilled and anxious!
- He set an amazing standard — read and left a review on *all* of the papers!
- Deliberately and carefully guided the discussion, but did not put his thumb on the scale too much



POPL is now a comfortable home

- In 2012, I was POPL PC Chair (pleased and surprised!)
- I've co-authored more papers in POPL than any other venue

[–] **2020 – today**

2023

[j36] Finn Voichick, Liyi Li, Robert Rand, Michael Hicks:
Qunity: A Unified Language for Quantum and Classical Computing. Proc. ACM Program. Lang. 7(POPL): 921-951 (2023)

2021

[j29] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, Michael Hicks:
A verified optimizer for Quantum circuits. Proc. ACM Program. Lang. 5(POPL): 1-29 (2021)

2020

[j26] David Darais, Ian Sweet, Chang Liu, Michael Hicks:
A language for probabilistically oblivious computation. Proc. ACM Program. Lang. 4(POPL): 50:1-50:31 (2020)

[–] **2010 – 2019**

2019

[j22] Shih-Han Hung, Kesha Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, Xiaodi Wu:
Quantitative robustness analysis of quantum programs. Proc. ACM Program. Lang. 3(POPL): 31:1-31:29 (2019)

[j21] James Parker, Niki Vazou, Michael Hicks:

POPL'12 Program Chair's Report
(or, how to run a medium-sized conference)

Michael Hicks
Department of Computer Science
University of Maryland, College Park, USA

1. Introduction

It was a pleasure and a privilege to serve as the program committee (PC) chair of the 39th Symposium on the Principles of Programming Languages (POPL). This paper describes the review process we used, why we used it, and an assessment of how it worked out.[1]

We made some substantial changes to the review process this year, most notably by incorporating a form of double-blind reviewing. These and other changes were made in an POPL'12 with graphs illustrating trends from this data, and some scripts and other code is available at my website [3]. I present overall results from the surveys in the context of discussion about the process throughout Sections 2–5.

1.1 Recommendations

Here I list each of the things we did for this year's POPL, and my recommendations for whether to do them again:

- I was pleased to serve on the POPL'24 PC
- **Thanks, Benjamin!**

Keeping up with Benjamin

- Over the years, Benjamin’s research and other activities frequently influenced my own

POPL mark

2006

Mechanized Metatheory for the Masses: The PoplMark Challenge

Brian E. Aydemir[1], Aaron Bohannon[1], Matthew Fairbairn[2], J. Nathan Foster[1], Benjamin C. Pierce[1], Peter Sewell[2], Dimitrios Vytiniotis[1], Geoffrey Washburn[1], Stephanie Weirich[1], and Steve Zdancewic[1]

[1] Department of Computer and Information Science, University of Pennsylvania
[2] Computer Laboratory, University of Cambridge

Abstract. How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?
We propose an initial set of benchmarks for measuring progress in this area. Based on the metatheory of System $F_{<:}$, a typed lambda-calculus with second-order polymorphism, subtyping, and records, these benchmarks embody many aspects of programming languages that are challenging to formalize: variable binding at both the term and type levels, syntactic forms with variable numbers of components (including binders), and proofs demanding complex induction principles. We hope that these benchmarks will help clarify the current state of the art, provide a basis for comparing competing technologies, and motivate further research.

Boomerang

v0.2

A ***bidirectional*** programming language for ad-hoc, textual data.

2010

CIS 500: Software Foundations - Fall 2007

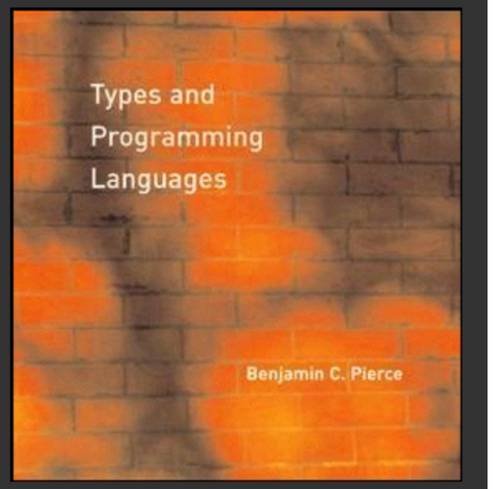
[Home | Schedule | Coq | Resources | Recitations | Syllabus | WPE]

Mon, Wed 12:00-1:30PM

Levine Hall Auditorium

Instructor
Benjamin C. Pierce
bcpierce AT cis.upenn.edu
office hours: TBA

Teaching Assistant
Leonid Spesivtsev
spesiv AT cis.upenn.edu
office hours: TBA



Putting Differential Privacy to Work

Penn University of Pennsylvania -- Home -- Contributors -- Software -- Publications --

A wealth of data about individuals is constantly accumulating in various databases in the form of medical records, social network graphs, mobility traces in cellular networks, search logs, and movie ratings, to name only a few. There are many valuable uses for such datasets, but it is difficult to realize these uses while protecting privacy. Even when data collectors try to protect the privacy of their customers by releasing anonymized or aggregated data, this data often reveals much

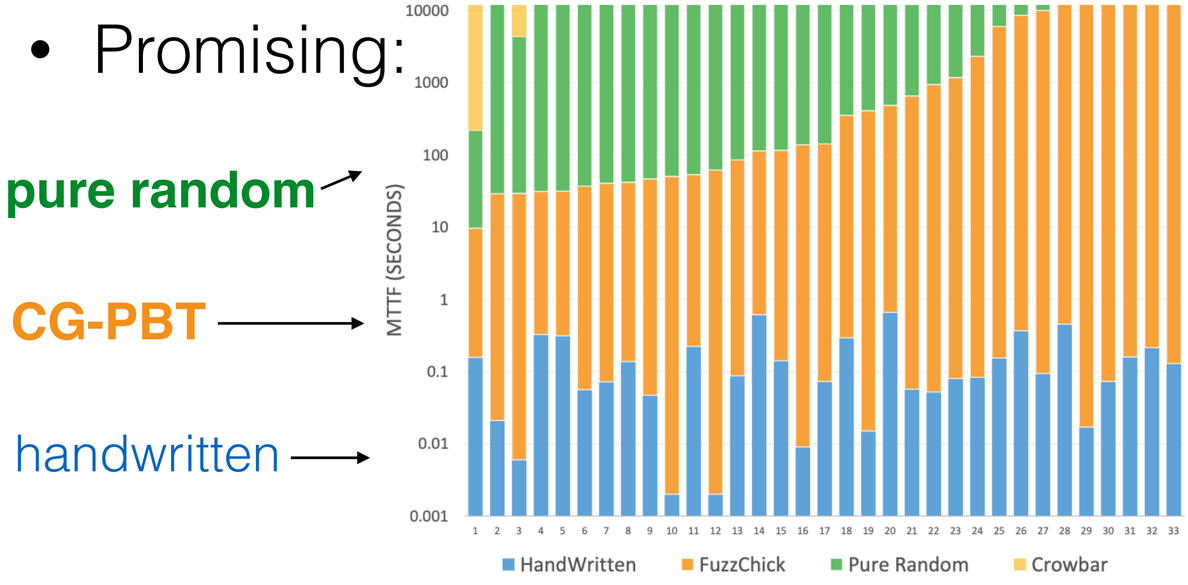
2012

Research Collaboration

- Never worked directly with Benjamin on research
  - Despite all of the influence and interaction I've just described!
- That finally came in 2018

FuzzChick

- Use ideas from **coverage-guided fuzzing** to generate inputs for **property-based tests**
  - Applied to PBT framework for Coq
- Promising:



- Influenced my AWS work on Cedar

OOPSLA'19

Coverage Guided, Property Based Testing

LEONIDAS LAMPROPOULOS, University of Maryland, USA and University of Pennsylvania, USA
MICHAEL HICKS, University of Maryland, USA
BENJAMIN C. PIERCE, University of Pennsylvania, USA

Property-based random testing, exemplified by frameworks such as Haskell's QuickCheck, works by testing an executable predicate (a *property*) on a stream of randomly generated inputs. Property testing works very well in many cases, but not always. Some properties are conditioned on the input satisfying demanding semantic invariants that are not consequences of its syntactic structure—e.g., that an input list must be sorted or have no duplicates. Most randomly generated inputs fail to satisfy properties with such *sparse preconditions*, and so are simply discarded. As a result, much of the target system may go untested.

We address this issue with a novel technique called *coverage guided, property based testing (CGPT)*. Our approach is inspired by the related area of coverage guided fuzzing, exemplified by tools like AFL. Rather than just generating a fresh random input at each iteration, CGPT can also produce new inputs by mutating previous ones using type-aware, generic mutation operators. The target program is instrumented to track which control flow branches are executed during a run and inputs whose runs expand control-flow coverage are retained for future mutations. This means that, when sparse conditions in the target are satisfied and new coverage is observed, the input that triggered them will be retained and used as a springboard to go further.

We have implemented CGPT as an extension to the QuickChick property testing tool for Coq programs; we call our implementation FuzzChick. We evaluate FuzzChick on two Coq developments for abstract machines that aim to enforce flavors of noninterference, which has a (very) sparse precondition. We systematically inject bugs in the machines' checking rules and use FuzzChick to look for counterexamples to the claim that they satisfy a standard noninterference property. We find that vanilla QuickChick almost always fails to find any bugs after a long period of time, as does an earlier proposal for combining property testing and fuzzing. In contrast, FuzzChick often finds them within seconds to minutes. Moreover, FuzzChick is almost fully automatic; although highly tuned, hand-written generators can find the bugs faster than FuzzChick, they require substantial amounts of insight and manual effort.

CCS Concepts: • **Software and its engineering → General programming languages**.

Additional Key Words and Phrases: random testing, property-based testing, fuzz testing, coverage, QuickChick, AFL, FuzzChick

ACM Reference Format:
Leonidas Lampropoulos, Michael Hicks, and Benjamin C. Pierce. 2019. Coverage Guided, Property Based Testing. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 181 (October 2019), 29 pages. https://doi.org/10.1145/3360607

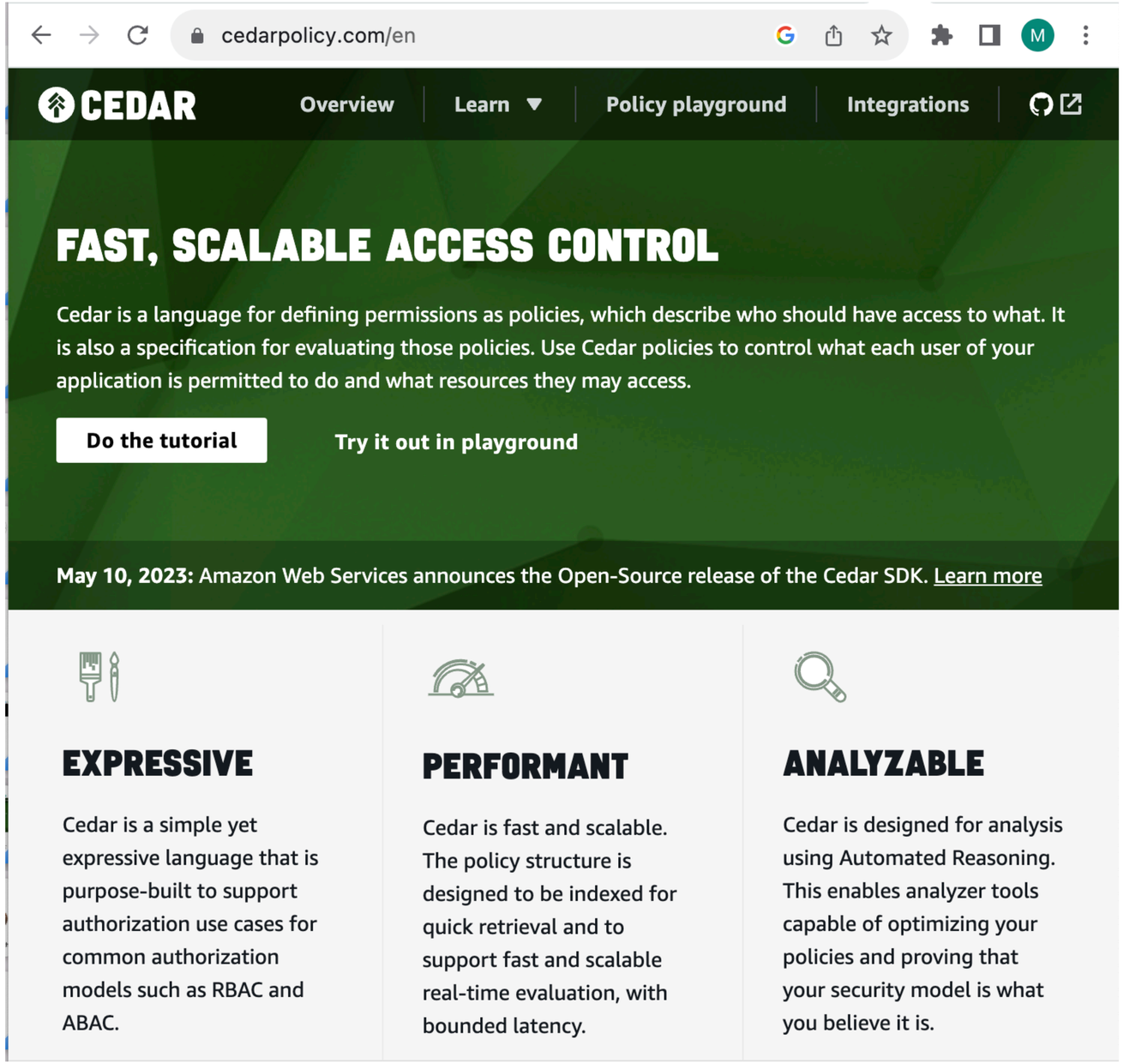
Authors' addresses: Leonidas Lampropoulos, University of Maryland, USA , University of Pennsylvania, USA, llamp@seas.upenn.edu; Michael Hicks, University of Maryland, USA, mwh@cs.umd.edu; Benjamin C. Pierce, University of Pennsylvania, USA, bcpierce@cs.upenn.edu.

Policies as Code

- **Cedar** is a language for writing **authorization policies** with which to implement permissions in custom applications

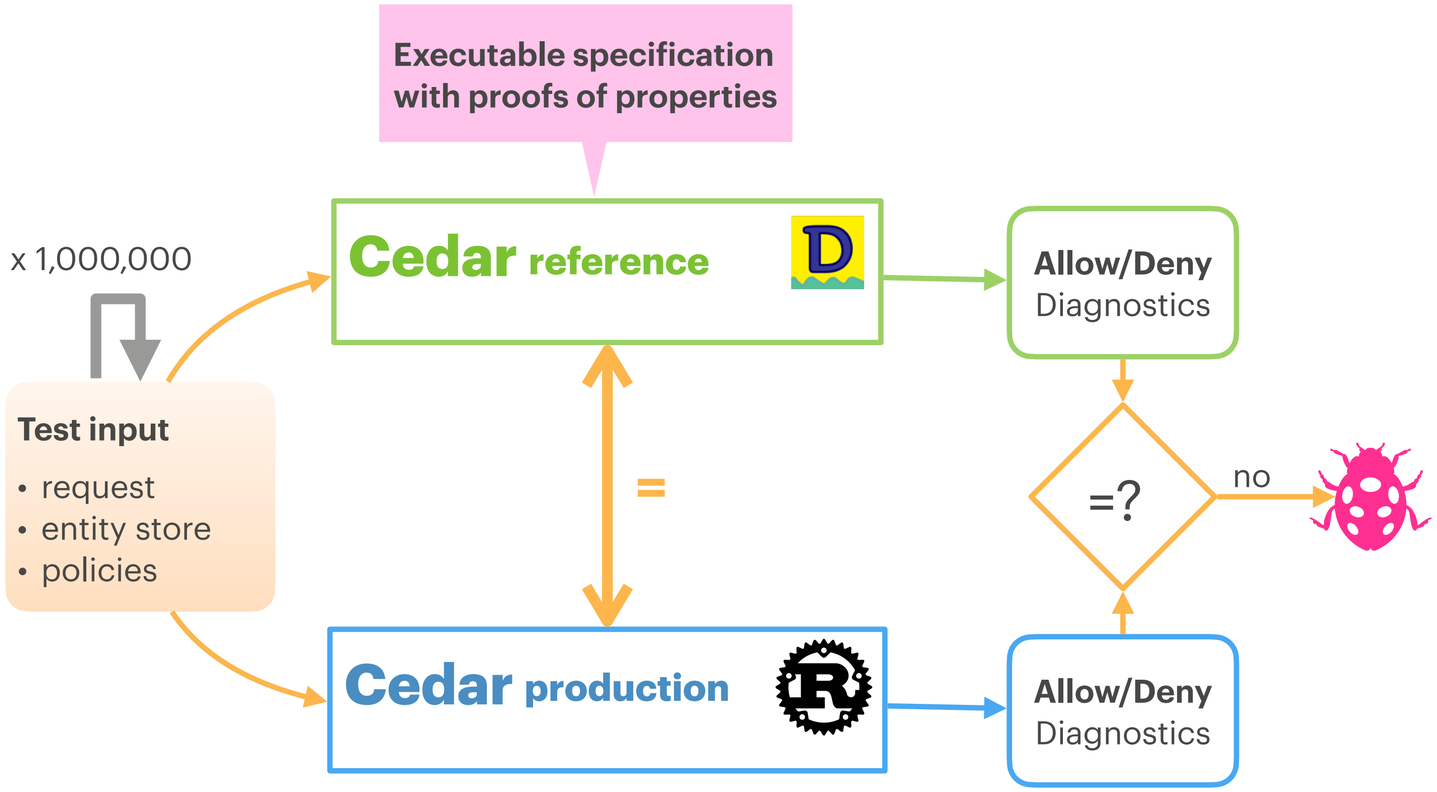
- Developed 2022 - present at AWS, open-sourced on May 10, 2023, and used by the **Amazon Verified Permissions** cloud-hosted authorization service

- Paper accepted to OOPSLA'24



Cedar DRT

- Use **differential random testing** to check correctness of Cedar production code
- **Challenge**: Effective input generation
  - Difficult, even when writing generators by hand



- **Opportunity**: New tech for property-based testing, applied to Cedar DRT
  - **Working with Benjamin and his team on this**

Writing Process, Firsthand

- Writing a paper (and grant proposal) with Benjamin was eye opening!
- He was **relentless** at finding the **clearest** and **most direct presentation**
  - Questioned every claim we made and the way we made it
  - Other authors ready to move on, but BCP poking at how to do it better

Climate

2016

BY MICHAEL HICKS | JUNE 8, 2016 · 1:00 PM

↓ Jump to Comments

Carbon Footprint of Conference Travel

Conferences are the heart of the PL research community. The best PL research is published at conferences, following a rigorous peer review process on par or better than the process of high-quality journals. Conferences are also where science gets done. As the respective community gathers to learn about the latest results, its members also network and interact, developing collaborations or carrying on projects that could produce the next breakthroughs. At conferences, students and young professors can rub elbows with luminaries, and researchers can develop problems and exchange ideas with practitioners.

Air travel warms the planet disproportionately

One drawback of conferences (compared to other forms of research publication) is their cost. The monetary cost is obvious: at least one of a paper's authors must pay to attend the conference, a cost that includes registration, travel, and hotel. Another cost that is less often considered is the **environmental cost**. In particular, I'm thinking of the impact that travel to/from the conference has on global warming. Most conference attendees travel great distances, and so travel by airplane. But air travel is particularly bad for global warming. So I wondered: what is the cost of conference travel, in terms of carbon footprint?

2017

sigplan.org/Resources/Climate/

ACM SIGPLAN

Home Conferences OpenTOC Awards Research Highlights Membership

SIGPLAN and Climate Change

Quick Links

- ACM's new (as of 2019) Carbon Offset Program
- ACM's CO2 Footprint Calculator for Conferences
- acm-climate mailing list

Introduction

Climate change due to human emissions of greenhouse gases is an urgent threat, one that requires massive efforts at every level of society. SIGPLAN (and ACM and other scientific societies more broadly) contribute disproportionately to this threat by hosting conferences that are responsible for large amounts of carbon emissions, primarily as a result of participants' air travel.

The SIGPLAN Climate Committee recognizes this dilemma and consequently seeks to study and address SIGPLAN and ACM's role in climate change.

About the SIGPLAN Climate Committee

In 2017, SIGPLAN formed an ad hoc committee to study the climate impact of conferences and possible steps that SIGPLAN might take in resp

The SIGPLAN Climate Com chaired by SIGPLAN Vice-Chair Benjamin Pierce

SIGPLAN Chair Jens Palsberg, Past EC member Crista Lopes, and Past SIGPLAN Chair Michael Hicks.

2019

provost.upenn.edu/senate/circe

Office of the Faculty Senate

SENATE SELECT COMMITTEE ON THE INSTITUTIONAL RESPONSE TO THE CLIMATE EMERGENCY

CIRCE

Penn Faculty Climate Pledge

- Learn about the Pledge (3-minute video)
- Read the Pledge
- Sign the Pledge
- Act on your Pledge: "Bring It Home" Manual for Individuals
- List of Signatories

Impact

- Remote PC meetings
- ACM CO2 offsets policy
- CACM paper, thorough report, awareness
- Virtual conference development

A source of positive impact

- **My PL journey goes through Benjamin**: Teacher, Role Model, Collaborator, Friend
- My research, teaching, service, and mindset all have experienced Benjamin's positive touch
  - I am one of many!
- Looking forward to what Benjamin does next!