Job Application Package

Harrison Goldstein

October 4, 2024

Contents

Harrison Goldstein

Postdoctoral Fellow

I am a postdoc working with Leo Lampropoulos and Benjamin Pierce. My work combines programming languages, software engineering, and human-computer interaction to improve the tools that developers use to build software. The bulk of my work so far has focused on *property-based testing*.

🌐 harrisongoldste.in

✉️ me@harrisongoldste.in

🐦 @hgoldstein95

Education

Ph.D. in Computer Science
University of Pennsylvania
2019-2024

M.Eng. in Computer Science
Cornell University
2018 — GPA 4.02

B.S. in Computer Science
Cornell University
2014-2018 — GPA 4.08

Teaching

Penn CIS 552
Head TA 2020-2021

Penn CIS 810
Head TA 2021

Cornell CS 3110
Head TA 2017

Cornell Eng. Leadership
Head TA 2017

Cornell CS 2112
TA 2016

Awards and Honors

Victor Basili Postdoctoral Fellowship
University of Maryland

Certificate in Engineering Leadership
Cornell ECLP

1st Place, Business Plan Competition
Cornell Sch. of Hotel Admin.

Selected Publications

Property-Based Testing for the People
Dissertation

Tyche: Making Sense of Property-Based Testing Effectiveness
UIST 2024 Research Paper

Property-Based Testing in Practice   DISTINGUISHED PAPER
ICSE 2024 Research Paper

Reflecting on Random Generation   DISTINGUISHED PAPER
ICFP 2023 Research Paper

Parsing Randomness
OOPSLA 2022 Research Paper

Grants

NSF #2402449 SHF: Medium: Usable Property-Based Testing
National Science Foundation

TYCHE: An IDE for Property-Based Testing
Amazon Research Award 2023

Employment

University of Maryland, College Park
Victor Basili Postdoctoral Fellow 2024-

Galois, Inc.
Research Intern 2023

Amazon Web Services
Research Intern 2020

Selected Service Roles

Steering Committee Member NJPLS Ongoing	**Research Mentor** DeepSpec REU
Speaker PLMW (POPL'24, ICFP'24)	**Organizer** NJPLS 2023
Research Mentor REPL REU 2023, 2024	**Social Chair** POPL 2021
PC Member HATRA 2023	**A/V Coordinator** ICFP 2021

Harrison Goldstein
Postdoctoral Fellow

🌐 harrisongoldste.in
✉ me@harrisongoldste.in
🐦 @hgoldstein95

I am a postdoc working with Leo Lampropoulos and Benjamin Pierce. My work combines programming languages, software engineering, and human-computer interaction to improve the tools that developers use to build software. The bulk of my work so far has focused on *property-based testing*.

Publications, Talks, and Grants

Dissertation

Property-Based Testing for the People
Harrison Goldstein
advised by Benjamin C. Pierce
approved by Stephanie Weirich (committee chair), Andrew Head, Mayur Naik, Hila Peleg
Defended May 24, 2024

Refereed Conference Publications

UIST'24	**Tyche: Making Sense of Property-Based Tesing Effectiveness** **Harrison Goldstein**, Jeffrey Tao, Zac Hatfield-Dodds, Benjamin C. Pierce, Andrew Head User Interface Software and Technology (UIST) 2024. 13 pages.
PLDI'24	**Stream Types** J. W. Cutler, C. Watson, E. Nkurumeh, P. Hilliard, **H. Goldstein**, C. Stanford, B. C. Pierce Programming Language Design and Impl. (PLDI) 2024. 24 pages.
PLDI'24	**Daedalus: Safer Document Parsing** I. Diatchki, M. Dodds, **H. Goldstein**, B. Harris, D. Holland, B. Razet, C. Schlesinger, S. Winwood Programming Language Design and Implementation (PLDI) 2024. 24 pages.
ICSE'24 **Distinguished**	**Property-Based Testing in Practice** **Harrison Goldstein**, Joseph W. Cutler, Daniel Dickstein, Benjamin C. Pierce, Andrew Head International Conference on Software Engineering (ICSE) 2024. 23 pages.
ICFP'23 **Distinguished**	**Reflecting on Random Generation** **Harrison Goldstein**, Samantha Frohlich, Meng Wang, Benjamin C. Pierce International Conference on Functional Programming (ICFP) 2023. 34 pages.
OOPSLA'22	**Parsing Randomness** **Harrison Goldstein**, Benjamin C. Pierce Object-Oriented Programming, Systs., Langs., and Apps. (OOPSLA) 2022. 25 pages.
ESOP'21	**Do Judge a Test by its Cover: Combining Combinatorial and Property-Based Testing** **Harrison Goldstein**, John Hughes, Leonidas Lampropoulos, Benjamin C. Pierce European Symposium on Programming (ESOP) 2021. 27 pages.

Workshop Papers and Experience Reports

OCaml'24	**Mica: Automated Differential Testing for OCaml Modules** Ernest Ng, **Harrison Goldstein**, Benjamin C. Pierce OCaml Workshop 2024. 2 pages.

ICFP'23	**Etna: An Evaluation Platform for Property-Based Testing (Experience Report)** J. Shi, A. Kelles, **H. Goldstein**, B. C. Pierce, L. Lampropoulos International Conference on Functional Programming (ICFP) 2023. 17 pages.
HATRA'22	**Some Problems with Properties: A Study on Property-Based Testing in Industry** **Harrison Goldstein**, Joseph W. Cutler, Adam Stein, Benjamin C. Pierce, Andrew Head Human Aspects of Types and Reasoning Assistants (HATRA) 2022. 8 pages.
SysML'18	**Programming Language Support for Natural Language Interaction** Alex Renda, **Harrison Goldstein**, Sarah Bird, Chris Quirk, Adrian Sampson Conference on Machine Learning and Systems (SysML) 2018. 3 pages.

Demos

SCF'24	**Demonstrating FEDT: Supporting Characterization Experiments in Fabrication Research** V. Savage, N. Püsök, **H. Goldstein**, C. Nandi, J. Yi Ren and L. Oehlberg Symposium on Computational Fabrication (SCF) 2024.
UIST'23	**Tyche: In Situ Analysis of Random Testing Effectiveness** **Harrison Goldstein**, Benjamin C. Pierce, Andrew Head User Interface Software and Technology (UIST) 2023.

Speaking

Talk Invited	**My PhD Compass: 6 Ways to Guide a PhD Towards Success** **Harrison Goldstein** PL Mentoring Workshop @ ICFP, September 2024
Podcast Invited	**Harry Goldstein | Property-Based Testing | #55** Jack Waudby, **Harrison Goldstein** Disseminate Podcast, June 2024
Talk	**Usable Property-Based Testing** **Harrison Goldstein** Amazon Web Services, June 2024
Talk Invited	**Consider Collaboration** Samantha Frohlich, **Harrison Goldstein** PL Mentoring Workshop @ POPL, January 2024
Talk Invited	**Advancing Property-Based Testing in Theory and Practice** **Harrison Goldstein** Microsoft Research, UC Berkeley, Galois, Inc., UCSC, University of Bristol
Talk	**Property-Based Testing in Practice** **Harrison Goldstein** Jane Street Programming Languages Colloquium, December 2023
Podcast Invited	**TheForkJoin Episode 2** Oliver Flatt, Rachit Nigam, **Harrison Goldstein** TheForkJoin
Talk	**Some Problems with Properties** **Harrison Goldstein** NJPLS October 2022
Talk	**Reflecting on Random Generation** **Harrison Goldstein** NJPLS May 2022

Posters

ICFP'21   **Ungenerators**
Harrison Goldstein
ICFP 2021

POPL'20   **Algebraic Combinatorial Testing**
Harrison Goldstein
POPL 2020

Drafts and Blog Posts

Delimited Continuations and Monads
Harrison Goldstein
Unpublished PhD Milestone Draft, April 2021

Confernces after COVID: An Early Career Perspective
Joseph W. Cutler, **Harrison Goldstein**, Andrew K. Hirsch, Jaemin Hong, Chandrakana Nandi
SIGPLAN PL Perspectives Blog, March 2021

Funded Grant Proposals

NSF #2402449 SHF: Medium: Usable Property-Based Testing
2023-2024
Made significant contributions, in the form of both research project ideas and grant writing, to help secure a NSF Medium grant. This grant helped to fund my dissertation work, and it will continue to fund follow-on projects.
Team: Benjamin Pierce (co-PI), Andrew Head (co-PI), **Harrison Goldstein** (contributor)

Amazon Research Award: TYCHE: An IDE for Property-Based Testing
Spring 2023
Secured funding from AWS to fund my ongoing work on user interfaces for property-based testing. This award includes the opportunity to collaborate with contacts at AWS, including Michael Hicks, on interfaces that support real industrial workloads.
Team: **Harrison Goldstein** (primary author), Benjamin Pierce (co-PI), Andrew Head (co-PI)

Education

Ph.D. in Computer Science, University of Pennsylvania
2019–2024

M.Eng. in Computer Science, Cornell University
2018 — GPA 4.02

B.S. in Computer Science, Cornell University
2014–2018 — GPA 4.08

Teaching and Advising

Courses

Penn CIS 552 "Advanced Programming"
Prof. Stephanie Weirich
Head TA 2020-2021

Penn CIS 810 "Writing and Speaking with Style"
Prof. Benjamin C. Pierce
Head TA 2021

Cornell CS 3110 "Functional Programming"
Prof. Nate Foster
Head TA 2017

Cornell Engineering Leadership
Profs. Erica Dawson and Werner Zorman
Head TA 2017

Cornell CS 2112 "Object-Oriented Design and Data Structures (Honors)"
Prof. Dexter Kozen
TA 2016

Graduate Advisees

Collaborator and Research Mentor, Joseph W. Cutler

Collaborator and Research Mentor, Jessica Shi

Co-Advisor, Ernest Ng

Service

Conference Organizing and Reviewing

Steering Committee Member, NJPLS — Ongoing

Reviewer, PLATEAU 2024

Co-Organizer, Celebration for Benjamin Pierce's 60th Birthday

PC Member, HATRA 2023

Organizer, NJPLS 2023

Social Chair, POPL 2021

A/V Coordinator, ICFP 2021

Mentorship

Speaker, PL Mentoring Workshop @ ICFP 2024

Ph.D. Mentor, SIGPLAN-M

Ph.D. Mentor, Penn CIS Mentoring

Speaker, PL Mentoring Workshop @ POPL 2024

Research Mentor, REPL REU 2023

Research Mentor, DeepSpec REU

Misc.

Editor in Chief, PLClub Blog 2021-2024

Awards and Honors

Victor Basili Postdoctoral Fellowship
University of Maryland, Computer Science Department

Certificate in Engineering Leadership
Cornell Engineering Leadership Certification Program

1st Place, Business Plan Competition
Cornell School of Hotel Administration

Employment

University of Maryland, College Park
Victor Basili Postdoctoral Fellow 2024-
Working with Prof. Leonidas Lampropoulos on topics related to usable property-based testing.

Galois, Inc.
Research Intern 2023
Worked on two research papers related to the SafeDocs DARPA program. Provided expertise on testing that helped situate the papers in the broader research context.

Amazon Web Services
Research Intern 2020
Worked on Zelkova, a tool for analyzing and proving properties about AWS access policies. Encoded logical constraints from access policies as SMT formulas, in order to infer policy implications.

Broadway Technology
Software Engineer and Software Engineering Intern 2017–2019
Built mission-critical internal tools for the company's financial personnel, in particular facilitating a transition to new financial tracking software. Designed and implemented data connectors, financial calculations, web interfaces, and more.

Susquehanna International Group, LLC
Technology Intern 2016
Helped to implement a safety system, protecting the firm from anomalous trading behaviors. Built an engine for executing business rules as monitors for live trading activities.

Last Second Beach, LLC
Chief Technology Officer 2015–2016
Helped to lead a small, early stage start-up, focused on providing one-price vacations. Built a demo iOS application, helping the company to win a $25,000 grant as part of a Business Plan Competition.

Research Statement: Usable Programming Tools with Formal Foundations

HARRISON GOLDSTEIN, University of Maryland, USA and University of Pennsylvania, USA

Over the years, much has been said about the tremendous cost of incorrect and low-quality software [19, 22]. Now, in an era where AI systems like GitHub's Copilot [7] make buggy code easier than ever to produce, it is critical that we have equally powerful systems that make code easier to trust.

To address this challenge, my work provides programmers with tools that help them to build confidence in the correctness of their software. My approach is guided by three core principles:

(1) Programming tools should be built on *solid formal foundations*.
(2) Programming tools should have impact for *real software engineers*.
(3) Programming tools should be *usable and understandable*.

I follow these principles by drawing on and collaborating in three different sub-areas of computer science: programming languages (PL) grounds my work in formalism, software engineering (SE) keeps me aligned with the needs of real developers, and human-computer interaction (HCI) ensures that the things I build are human-usable.

While this philosophy can be applied broadly, the bulk of my research has focused on tools for *property-based testing*. PBT is an approach to randomized testing that encourages rigorous reasoning about software safety without requiring developers to apply advanced formal methods. Software engineers fundamentally want to build good software, but they have limited time and energy with which to do so; by making principled testing techniques more powerful and more usable, my research tangibly impacts the overall quality of the software that developers produce.

My work so far has already made progress towards making PBT more powerful and usable. This progress was guided in part by an ambitious qualitative study [**ICSE'24***] that we did in collaboration with Jane Street. The study helped us to understand how developers use PBT tools and what they need from them. That understanding enabled us to make theoretical advances around languages for random data generation [**ICFP'23***, OOPSLA'22*] and build practical user interfaces for testing evaluation [UIST'24*]. I have also collaborated on benchmarks [ICFP'23], novel evaluation metrics [ESOP'21*], and automation [OCaml'24] for testing. My research in this area is ongoing, with a number of new threads of active research. Our study showed that developers need better ways to obtain random data generators, so I am currently working on semi-automatic techniques that can improve that process for developers. I also plan to improve the user experience in other parts of the PBT process—e.g., property creation and testing evaluation—and build a comprehensive IDE for PBT.

Beyond testing, I have mentored students working on languages for stream processing [PLDI'24] and usability aspects of proof assistants [PLATEAU'23]. Most recently, I have begun collaborating on a language for specifying and carrying out scientific experiments [SCF'24]. While these projects do not directly include testing, each one leverages skills that I developed in my work with PBT and each contributes to my vision of increasing the quality of software by applying PL, SE, and HCI.

My past and current projects have been funded through grants and awards that I helped to secure. I was awarded the Victor Basili Postdoctoral Fellowship to fund my postdoctoral research, I was the primary author on an Amazon Research Award proposal that funds my work on user interfaces for testing evaluation, and I was an equal co-author on a funded NSF Medium proposal for work on usable testing tools.

Over the next 10 years, I will grow my career, emphasizing collaboration with amazing students and colleagues. Together, we will create formally-justified and human-usable software tools, achieving significant reach and impact both in the research community and for software engineers.

---

*Indicates a first-author publication. **Bold** indicates a distinguished paper.
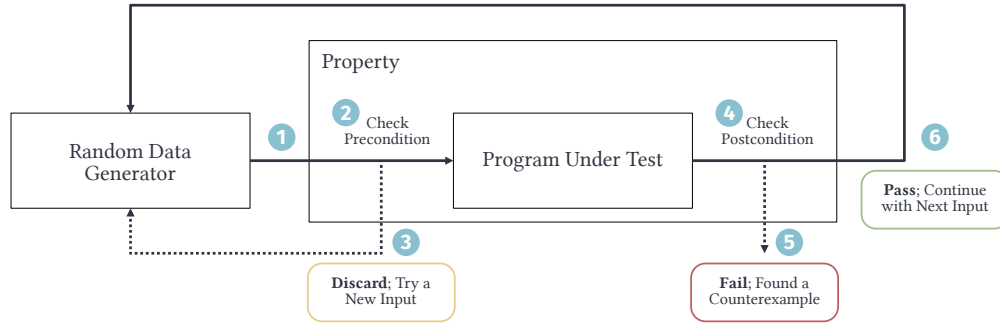Portions of this proposal are drawn from content in my dissertation [Goldstein, 2024*].

Fig. 1. The PBT process. (1) A random value is generated as an input to the property. (2) The property's *precondition* is checked, determining if the input is a valid input to the program under test. (3) If the input is invalid, it is discarded and the process restarts. (4) Once the program under test is executed, the property's *postcondition* is checked, determining if the program behaved correctly. (5) If the postcondition fails, we have found a counterexample and the program has a bug. (6) Otherwise the process restarts.

PRIOR WORK

My work so far, much of which appears in my dissertation [Goldstein, 2024*], has focused on understanding and improving the state of the art for property-based testing. Figure 1 shows a simplified overview of the PBT process.

Property-Based Testing in Practice

As mentioned above, I want my research to have "impact for real software engineers;" this requires a deep understanding of the needs, behaviors, and opinions those engineers. I obtain this understanding from the source—I talk to real software engineers and collect concrete data about what kinds of research will make a difference.

Much of the work I have done around PBT was motivated by a qualitative interview study called *Property-Based Testing in Practice* [**ICSE'24***], which appeared at ICSE in 2024 as a distinguished paper. In that study, we spoke with 30 software engineers at Jane Street, a financial technology company that makes extensive use of PBT. We learned about developers' experience with PBT to learn about how and why they use it and what we as researchers could do to address its shortcomings. Our analysis of the 20+ hours of interview transcripts highlighted six *observations* that both confirm and question conventional wisdom on real-world PBT use and seven *research opportunities* that translate into potential future projects in PBT.

This project has been invaluable for grounding and informing other PBT research. Projects like Mica [OCaml'24] and Tyche (see below) grew directly out of the research opportunities highlighted by the study. Meanwhile, our observations about developers' use of random data generators has clarified priorities around languages for random generators (see below), and motivated ongoing and future work in that space.

Abstractions for Random Data Generation

Empirical research is one tool in my research toolbox, but in order to ensure that my work is "built on solid formal foundations" I also rely heavily on theory. I use my background in programming languages to understand problems mathematically and develop solutions that are robust.

My work on languages for random data generators follows this model. In *Parsing Randomness* [OOPSLA'22*] we present *free generators*, an abstraction that formalizes a connection between random data generators and parsers, clarifying folklore knowledge and connecting the PBT literature to ideas from fuzz testing [27]. Then, in *Reflecting on Random Generation* [**ICFP'23***], which appeared as a distinguished paper at ICFP 2023, we extend free generators to *reflective generators*. Reflective generators use ideas from bidirectional programming [40] to essentially "run in reverse," enabling novel testing algorithms.

Fig. 2. A diagram of the Tyche interface in use.

Both free and reflective generators are primarily theoretical contributions; they connect ideas across areas of computer science, adding to the base of knowledge that future researchers can build on. Concretely, these abstractions continue to inform my ongoing work on automatically synthesizing and tuning random data generators (see below).

User Interfaces for Random Testing

The third major kind of work I engage in is building real tools for software developers. Tool-building is sometimes purely a matter of engineering, but ensuring that these tools are "usable and understandable" for real people often requires significant research. I design and evaluate tools in collaboration with real users, increasing the likelihood of potential impact.

Tyche [UIST'24*] is a great example of this kind of work. In *PBT in Practice*, we identified a gap in developers' use of PBT—they were missing good tools for evaluating whether their testing had been successful. To fill this gap, we worked with expert users of PBT tools to iteratively design an interface that provides key visual feedback and highlights potential problems during the testing process. We evaluated our final design in a study with 40 programmers, and validated that Tyche helps users to gauge the quality of a given testing setup. A diagram of the interface in use is shown in Figure 2.

Tyche is available across a variety of PBT frameworks (5 at the time of writing) and can easily be integrated into others. The interface has already helped to identify problems in testing setups, including problems in PBT frameworks themselves—the Hypothesis framework in Python has had four different efficiency bugs patched as a direct result of the better feedback provided by Tyche.

ONGOING AND FUTURE WORK

In the remainder of this statement, I discuss plans for future work that leverages PL, SE, and HCI to help programmers have confidence in their software. I start with ongoing work that focuses on improving technology for random data generation, thereby making PBT far more accessible to developers; I discuss the core problem in some detail, since it is especially important. Next, I discuss other future work in PBT, focused around software specification and testing evaluation. Finally, I discuss a project that actually has nothing to do with testing—it focuses on languages for designing scientific experiments—but still aligns with the themes of my research.

Synthesizing and Tuning Random Data Generators

In my opinion, the *most important open problem in PBT* is the following:

Definition (The Constrained Generator Problem). Given a logical predicate representing the precondition of a property, generate random data that satisfies that predicate.

This problem is NP-hard in general, since it reduces to iterated 3-SAT, but anecdotal evidence suggests that efficient solutions are available in the vast majority of realistic programming situations. Solving this problem would be incredibly significant for developers: many developers we have spoken to have expressed that the cost of writing generators can dissuade them from using PBT or significantly reduce its net value.

Existing Solutions and their Limitations. Of course, I am far from the first person to recognize that constrained random generation is an important issue for PBT; many have proposed algorithms to address this problem. A popular option is to assist the NP-hard search with SMT-solvers [33, 37], invoking the solver to help satisfy constraints during generation. Others have used clever applications of lazy evaluation [1], backtracking search [21], and reinforcement learning [31] to guide generation. There are also a wealth of approaches from the fuzz testing literature that use code coverage to find useful inputs [5, 6, 8, etc.]. Indeed, my own work on free and reflective generators (see above) has presented algorithms that try to improve the state of constrained random generation. But discussions from *PBT in Practice* and my own use of PBT over time suggest that these solutions do not suffice for all development situations.

The main problem is efficiency. The above techniques can all be effective when there plenty of time available for testing, but, in our study, we found that some developers test properties for only a few seconds at a time. This makes sense, since PBT is highly effective as a form of unit-testing, and unit-tests run many times during the development process. Doing any kind of expensive search during generation, whether with an SMT solver, machine learning, or coverage guidance, can slow the generator down and keep it from finding important bugs in the allotted time. Another issue is control: developers often have an idea of the kind of data that they want to test their code with, and many of the above approaches do not allow the developer to refine their generation strategy over time.

With these constraints in mind, I propose a refinement of the Constrained Generator problem:

Definition (The Constrained **Efficient** Generator **Synthesis** Problem). Given a logical predicate representing the precondition of a property, *synthesize* an *efficient* random data generator that satisfies that predicate.

In this version of the problem, generators are discouraged from searching during generation—instead, the search happens ahead of time during synthesis, and the final generator must be as efficient as possible. Also, since the generator is synthesized as a program, the programmer has maximal control, and they can modify the generator to match their idea of good test coverage.

Proposed Work. My experience working with real PBT users gives me a unique perspective on the challenges around constrained generation; I also have unique perspective on potential solutions. In particular, I envision a semi-automatic approach to synthesis: generators will be produced mostly automatically, but with small amounts of user interaction and guidance that will alleviate some tractability issues while also improving the quality of the final synthesized generators. The automatic parts of the synthesis algorithm will need to draw on results from across the literature— ideas from formal methods, compilers, program synthesis, logic programming, and probabilistic programming will be necessary. And the interactive parts will be designed to match the users' mental models of the synthesis process in a way that is hopefully much more intuitive than writing a generator from scratch.

The proposed synthesis pipeline is shown in Figure 3.

Step 1. The programmer writes a predicate constraining their data. In many cases, the programmer will already have a predicate expressed as part of their program.

Step 2. The predicate is compiled into a logical specification (if necessary). The compilation process can leverage techniques from symbolic execution [18], bounded model-checking [3], and solver-aided languages [38].
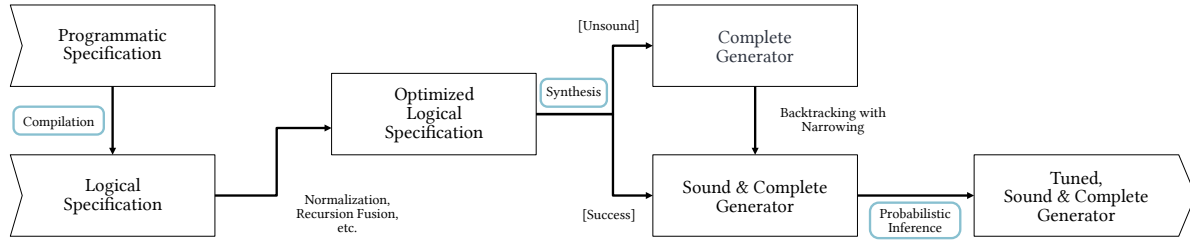
Fig. 3. A proposed process for synthesizing and tuning generators. Blue rectangles indicate opportunities for user interaction.

Opportunity for Interactivity: Some program constructs cannot be encoded logically (e.g., calls to a database and other side-effects). In these cases, the system can prompt the user to provide logical approximations that are tractable for the downstream synthesis process, thus providing the user a knob for control rather than giving up.

Step 3. The logical specification is optimized to be more amenable to synthesis. Certain kinds of recursive predicates (e.g., ones that build intermediate data structures or jointly constrain the same part of a structure) are difficult to turn into a generator. Interestingly, these same predicates can often be *optimized* [25, 30, 39] to be both more efficient and better for generation.

Step 4. The optimized logical specification is used to synthesize a generator. My collaborators and I propose using deductive synthesis techniques [28, 29] to produce a generator from the formula. Generators will be synthesized by assembling generators from a library of base components.
Opportunity for Interactivity: If the library of components is incomplete, users can interactively add to the library and fill the gaps. This is particularly useful because it lets the user address ambiguities from the original specification.

Step 5. Backtracking search is added to the synthesized generator to ensure soundness (if necessary). The synthesis process may still fail, even with help from the user. Some predicates cannot be optimized into a generator that gets every constraint right on the first try. In these cases, the synthesizer will err on the side of *completeness*—generating all possible predicate-satisfying values—instead of *soundness*—generating *only* predicate-satisfying values. With many of the harder generation problems optimized away, we think we can achieve good performance by bringing back search procedures inspired by prior work [1, 21] along with new insights from logic programming [36] to backtrack and filter out the invalid values, restoring soundness.

Step 6. The final sound and complete generator is tuned to have an interesting distribution for testing. In this case, "tuning" means weighting the random choices the generator makes to achieve a more interesting good distribution of values. I am currently working with collaborators on techniques that use probabilistic programming [16] to tune a generator to optimize various metrics.
Opportunity for Interactivity: Users can specify the metrics that the distribution should optimize, based on their understanding of what makes test inputs interesting.

Step 7. The programmer uses the generator to find bugs in their program. Further iteration on the generator based on testing performance (e.g., as measured by Tyche) is easy, since the generator is expressed as a program.

This end-to-end, interactive process for generator synthesis will be the first of its kind, providing users with a clear path from specification to testing, without needing to manually write data generators.

Other Advances in PBT

Constrained random generation may be the Mt. Everest of challenges in PBT, but there are a number of shorter peaks that are still worth climbing in the meantime. The two most prominent are helping developers to *specify* their code and helping them to *evaluate* their testing.

Specification. Specifying program behavior is difficult. Existing tools [2, 15] can suggest properties based on the shape and behavior of users' code, but these tools are not general enough to apply reliably in real programming workflows. One might be tempted to use generative AI to help solve this problem, but using a large language model to generate properties directly could be disastrous [17]: test suites are part of an application's *trusted computing base* and, as such, must be carefully and thoughtfully chosen. I propose a hybrid approach: AI can help the user to identify complex areas of their code base and brainstorm natural language correctness conditions; then classical tools can assist programmers in actually expressing their ideas as code.

Users may also need better ways to express their specifications as concrete properties. For example, current ways of expressing properties require clear software boundaries: PBT is hard to apply to programs with poorly encapsulated global state or with leaky or overly complex abstraction boundaries. A potential way around this limitation is to provide ways to express properties about the program's internal behavior. Following work on linear logic properties [26] and trace contracts [23], properties could be written declaratively over sequences of events that track the program's execution. This would allow for much more flexible specification, while still adhering to the basic principles of PBT.

Evaluation. The evaluation metrics displayed in Tyche are a good start, but there are likely even better ways for developers to gauge the quality of their testing. One path I would like to explore further are alternative metrics for "coverage." Informal conversations with software developers indicate that black-box coverage metrics like *combinatorial coverage* [20] may be valuable: developers want to understand how well they cover the possibility space of program behaviors without necessarily instrumenting their whole codebase.

I also want to continue to make sure that PBT evaluation is understandable, including by parties beyond the original test author. Indeed, in domains with strict acceptance requirements for software, it is often necessary to persuade auditors of the value of a property-based test. Tools like Tyche can help here too, providing data that can be compiled into reports and other documents that clarify the impact of a property as part of a test suite.

An IDE for PBT. New tools for specification, generation, and evaluation should not be developed independently. A unified environment for PBT technologies would encourage adoption and provide a platform for user-driven research. I envision an integrated development environment (IDE) that helps throughout the PBT process.

Concretely, I propose extending the basic platform provided by Tyche to incorporate a variety of interactive tools that the community develops for PBT. This can include the above ideas—generator advances, specification aids and evaluation metrics—in addition to other advances that benefit from living in and among user code. This unified platform will give would-be users an easy way to discover tools as they are developed, opening channels for feedback and impact. And as user research on PBT progresses, a unified set of tools can serve as a controlled environment in which to study interface design.

Languages for Experiment Design

While testing is my main focus at the moment, I am also interested in projects that are farther afield. One exciting project that I am currently collaborating on is the Fabrication Experiment Design Tool (FEDT). FEDT is a domain-specific programming language for designing and executing scientific experiments, and specifically experiments that involve computational fabrication like 3D printing and laser cutting. The language helps scientists to express their

experiment design unambiguously—this makes it easier to communicate methods within a lab, and it ensures that the experiment can be replicated by other researchers. So far, we presented a demo of FEDT at a workshop [SCF'24], we have a conference paper under submission, and we are currently looking into future directions for language design.

It might seem like this project is a total departure from my other research, but it is actually a more natural step than it seems. My work on free and reflective generators gave me a strong background in domain-specific languages, my human-centered projects positioned me well to help design tools with real users in mind, and my extensive work on random generation may eventually help us to address important questions about experiment simulation. In general, this kind of project, where I can bring some expertise and combine it with the orthogonal expertise of my fantastic collaborators, is one of my favorite ways to do research.

REFERENCES

[1] Koen Claessen, Jonas Duregård, and Michal H. Palka. 2015. Generating Constrained Random Data with Uniform Distribution. *J. Funct. Program.* 25 (2015). https://doi.org/10.1017/S0956796815000143

[2] Koen Claessen, Nicholas Smallbone, and John Hughes. 2010. QuickSpec: Guessing Formal Specifications Using Testing. In *Tests and Proofs*, Gordon Fraser and Angelo Gargantini (Eds.). Springer, Berlin, Heidelberg, 6–21. https://doi.org/10.1007/978-3-642-13977-2_3

[3] Edmund M. Clarke and Jeannette M. Wing. 1996. Formal Methods: State of the Art and Future Directions. *Comput. Surveys* 28, 4 (Dec. 1996), 626–643. https://doi.org/10.1145/242223.242257

[4] Joseph W. Cutler, Christopher Watson, Emeka Nkurumeh, Phillip Hilliard, Harrison Goldstein, Caleb Stanford, and Benjamin C. Pierce. 2024. Stream Types. *Proc. ACM Program. Lang.* 8, PLDI (June 2024), 204:1412–204:1436. https://doi.org/10.1145/3656434

[5] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. {AFL++} : Combining Incremental Steps of Fuzzing Research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*.

[6] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2013. Does Automated White-Box Test Generation Really Help Software Testers?. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA 2013)*. Association for Computing Machinery, New York, NY, USA, 291–301. https://doi.org/10.1145/2483760.2483774

[7] GitHub. 2024. Copilot.

[8] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. 2008. Grammar-Based Whitebox Fuzzing. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '08)*. Association for Computing Machinery, New York, NY, USA, 206–215. https://doi.org/10.1145/1375581.1375607

[9] Harrison Goldstein. 2024. *Property-Based Testing for the People.* Ph. D. Dissertation. University of Pennsylvania. https://harrisongoldste.in/papers/dissertation.pdf

[10] Harrison Goldstein, Joseph W. Cutler, Daniel Dickstein, Benjamin C. Pierce, and Andrew Head. 2024. Property-Based Testing in Practice. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24, Vol. 187)*. Association for Computing Machinery, Lisbon, Portugal, 1–13. https://doi.org/10.1145/3597503.3639581

[11] Harrison Goldstein, Samantha Frohlich, Meng Wang, and Benjamin C. Pierce. 2023. Reflecting on Random Generation. In *Proc. ACM Program. Lang.*, Vol. 7. Association for Computing Machinery, Seattle, WA, USA, 322–355. https://doi.org/10.1145/3607842

[12] Harrison Goldstein, John Hughes, Leonidas Lampropoulos, and Benjamin C. Pierce. 2021. Do Judge a Test by Its Cover. In *Programming Languages and Systems (Lecture Notes in Computer Science)*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 264–291. https://doi.org/10.1007/978-3-030-72019-3_10

[13] Harrison Goldstein and Benjamin C. Pierce. 2022. Parsing Randomness. *Proceedings of the ACM on Programming Languages* 6, OOPSLA2 (Oct. 2022), 128:89–128:113. https://doi.org/10.1145/3563291

[14] Harrison Goldstein, Jeffrey Tao, Zac Hatfield-Dodds, Benjamin C. Pierce, and Andrew Head. 2024. Tyche: Making Sense of Property-Based Testing Effectiveness. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. 16. https://doi.org/10.1145/3654777.3676407

[15] Zac Hatfield-Dodds. 2023. Ghostwriting Tests for You — Hypothesis 6.82.0 Documentation. https://hypothesis.readthedocs.io/en/latest/ghostwriter.html.

[16] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (Nov. 2020), 1–31. https://doi.org/10.1145/3428208 arXiv:2005.09089 [cs]

[17] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[18] James C. King. 1976. Symbolic execution and program testing. *Commun. ACM* 19, 7 (July 1976), 385–394. https://doi.org/10.1145/360248.360252

[19] Herb Krasner. 2022. The Cost of Poor Software Quality in the US: A 2022 Report.

[20] D. Richard Kuhn, James M. Higdon, James Lawrence, Raghu Kacker, and Yu Lei. 2012. Combinatorial Methods for Event Sequence Testing. In *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012*, Giuliano Antoniol, Antonia Bertolino, and Yvan Labiche (Eds.). IEEE Computer Society, 601–609. https://doi.org/10.1109/ICST.2012.147

[21] Leonidas Lampropoulos, Diane Gallois-Wong, Catalin Hritcu, John Hughes, Benjamin C. Pierce, and Li-yao Xia. 2017. Beginner's Luck: A Language for Property-Based Generators. *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017* (2017), 114–129.

[22] N.G. Leveson and C.S. Turner. 1993. An Investigation of the Therac-25 Accidents. *Computer* 26, 7 (July 1993), 18–41. https://doi.org/10.1109/MC.1993.274940

[23] Cameron Moy and Matthias Felleisen. 2023. Trace contracts. *Journal of Functional Programming* 33 (Jan. 2023), e14. https://doi.org/10.1017/S0956796823000096

[24] Ernest Ng, Harrison Goldstein, and Benjamin C. Pierce. 2024. Mica: Automated Differential Testing for OCaml Modules. https://doi.org/10.48550/arXiv.2408.14561 arXiv:2408.14561 [cs].

[25] László Németh. 2000. *Catamorphism-based program transformations for non-strict functional languages.* PhD. University of Glasgow. https://eleanor.lib.gla.ac.uk/record=b1990140

[26] Liam O'Connor and Oskar Wickström. 2022. Quickstrom: Property-Based Acceptance Testing with LTL Specifications. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 1025–1038. https://doi.org/10.1145/3519939.3523728

[27] Rohan Padhye, Caroline Lemieux, Koushik Sen, Mike Papadakis, and Yves Le Traon. 2019. Semantic Fuzzing with Zest. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 329–340. https://doi.org/10.1145/3293882.3330576

[28] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. 2016. Program Synthesis from Polymorphic Refinement Types. https://doi.org/10.48550/arXiv.1510.08419 arXiv:1510.08419 [cs]

[29] Nadia Polikarpova and Ilya Sergey. 2019. Structuring the synthesis of heap-manipulating programs. *SuSLik, Tool Implementation for Article: Structuring the Synthesis of Heap-Manipulating Programs* 3, POPL (Jan. 2019), 72:1–72:30. https://doi.org/10.1145/3290385

[30] Jacob Prinz and Leonidas Lampropoulos. 2023. Merging Inductive Relations. *Reproduction Package for "Merging Inductive Relations"* 7, PLDI (June 2023), 178:1759–178:1778. https://doi.org/10.1145/3591292

[31] Sameer Reddy, Caroline Lemieux, Rohan Padhye, and Koushik Sen. 2020. Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1410–1421. https://doi.org/10.1145/3377811.3380399

[32] Valkyrie Savage, Nóra Püsök, Harrison Goldstein, Chandrakana Nandi, Jia Yi Ren, and Lora Oehlberg. 2024. Demonstrating FEDT: Supporting Characterization Experiments in Fabrication Research.

[33] Eric L. Seidel, Niki Vazou, and Ranjit Jhala. 2015. Type Targeted Testing. In *Programming Languages and Systems (Lecture Notes in Computer Science)*, Jan Vitek (Ed.). Springer, Berlin, Heidelberg, 812–836. https://doi.org/10.1007/978-3-662-46669-8_33

[34] Jessica Shi, Alperen Keles, Harrison Goldstein, Benjamin C Pierce, and Leonidas Lampropoulos. 2023. Etna: An Evaluation Platform for Property-Based Testing (Experience Report). *Proc. ACM Program. Lang.* 7 (2023). https://doi.org/10.1145/3607860

[35] Jessica Shi, Benjamin Pierce, and Andrew Head. 2023. Towards a Science of Interactive Proof Reading. (2023).

[36] Robert J. Simmons, Michael Arntzenius, and Chris Martens. 2024. Finite-Choice Logic Programming. https://doi.org/10.48550/arXiv.2405.19040 arXiv:2405.19040 [cs].

[37] Dominic Steinhöfel and Andreas Zeller. 2022. Input Invariants. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 583–594. https://doi.org/10.1145/3540250.3549139

[38] Emina Torlak and Rastislav Bodik. 2013. Growing Solver-Aided Languages with Rosette. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013)*. Association for Computing Machinery, New York, NY, USA, 135–152. https://doi.org/10.1145/2509578.2509586

[39] Nicolas Wu and Tom Schrijvers. 2015. Fusion for Free. In *Mathematics of Program Construction*, Ralf Hinze and Janis Voigtländer (Eds.). Springer International Publishing, Cham, 302–322. https://doi.org/10.1007/978-3-319-19797-5_15

[40] Li-yao Xia, Dominic Orchard, and Meng Wang. 2019. Composing Bidirectional Programs Monadically. In *European Symposium on Programming*. Springer, 147–175. https://doi.org/10.1007/978-3-030-17184-1_6

Teaching Statement

HARRISON GOLDSTEIN, University of Maryland, USA and University of Pennsylvania, USA

While research brought me to academia, I have stayed because of education and, in particular, advising.

During my Ph.D., my advisor Benjamin took two new students, Jessica Shi and Joseph W. Cutler. Both Joe and Jessica were initially interested in working on projects related to my work, so it was natural for me to join their weekly meetings to help guide the process. These PBT-oriented collaborations were quite successful—we published one PBT paper with each of the two students [ICSE'24, ICFP'23]— but I did not expect how deeply invested I would become in the students' continued academic journeys. When Jessica and Joe moved onto their own topics, I stayed involved in their work; I remained in meetings to help guide projects at a high level and provided low-level feedback on writing, talk slides, mathematical formalism, etc. I was a co-author on Joe's paper on *Stream Types* [PLDI'24] and I have helped out with two of Jessica's projects on proof assistant usability that are currently under submission.

Also while at Penn, I was also lucky enough to work with an M.S. student, Ernest Ng. Ernest worked with Benjamin and me on a poster that won second place at the ICFP SRC in 2023 and a paper that was published at the OCaml workshop in 2024 [OCaml'24]. While working with us, Ernest was accepted to Cornell's Computer Science Ph.D. program, which he is now attending.

My experiences with these students was one of the most rewarding parts of the Ph.D. process. I learned many ways that advising is different from other academic collaboration—for example, in advising, solving the problem at hand is less important than helping the student learn and grow through the process—and I learned to appreciate the challenge of managing those nuances. It was also a joy to watch students go from helping on my project to executing on their own projects.

In hindsight, it should have been obvious that advising would be the core of my academic journey. Before I found computer science research, I considered pursuing engineering leadership. I am certified by Cornell's Engineering Leadership Certification Program and still meet regularly with Rob Parker who helps administer the program. Thanks to this background, my philosophy on advising is focused on communication and intentional leadership; I feel strongly that an advising relationship requires openness and honesty, and that my goal as an advisor is first and foremost to do right by the student. As faculty, I plan to grow a small- to medium-sized lab that can be a productive engine for research without sacrificing a personal connection with students.

Advising is high-impact on a small scale, but I also find it rewarding to have larger-scale impact while teaching in a classroom setting. I have been involved in a number of courses on programming, starting during my undergraduate education (Cornell CIS 2112), through my masters (Cornell CIS 3110), and into my Ph.D. (Penn CIS 5520). I was also a Head TA for Benjamin's course on Writing and Speaking with Style, and I have been invited to speak twice as a guest lecturer in Molly Feldman's course on Programming Abstractions. For me, public speaking, and lecturing in particular, is an exciting challenge—tailoring content to an audience is difficult, but it is incredibly rewarding when a room full of students is engaged in the lecture topic. Outside the classroom, I always seek the "aha moment" that a student has during office hours when a concept finally "clicks."

My philosophy for classroom instruction is similar to my advising philosophy: it is focused on clear communication and doing right by students. While I understand the institutional need for evaluation in undergraduate education, teaching for evaluation alone can lead instructors to treat students as statistics rather than individuals. In my teaching, I treat students as people first and foremost and look for ways to prioritize intellectual safety and genuine engagement; I see exams and other evaluation as a way to measure those primary goals, not as an end in and of themselves. In the same vein, I see students' failures as, at least in part, my own. For this reason, I am open-minded and feedback-oriented in my teaching. I will regularly ask students how I can improve, and I will take that advice to heart.

I plan to develop two courses relatively early in my faculty career. One will be a smaller graduate-level course on *Languages and Strategies for Test Data Generation*; through this course, I hope to get students interested in the "Constrained Generation Problem" (as I define it in my research statement) and explore the connections between the various random, enumerative, and search-based approaches to obtaining large amounts of data for testing. The second course will be much more ambitious—and take longer to develop. It will teach mid-to-late sequence undergraduate and masters students about both applying and developing *Advanced Testing Technologies*. The course will cover basic testing concepts like unit and integration testing, but it will quickly move towards more advanced topics, focusing on the ways that testing changes across language paradigms and software engineering methodologies. The course will culminate in a final project that asks the students to build their own testing framework, use it to catch bugs in provided software, and argue how and why it would fit into a particular kind of software engineering workflow.

Besides my own courses on programming languages and software engineering, I would be excited to help teach established courses in those areas. I can also teach or co-teach a course on HCI, provided it focuses on programming tools or another sub-area of HCI that I have experience in.

REFERENCES

[1] Joseph W. Cutler, Christopher Watson, Emeka Nkurumeh, Phillip Hilliard, Harrison Goldstein, Caleb Stanford, and Benjamin C. Pierce. 2024. Stream Types. *Proc. ACM Program. Lang.* 8, PLDI (June 2024), 204:1412–204:1436. https://doi.org/10.1145/3656434

[2] Harrison Goldstein, Joseph W. Cutler, Daniel Dickstein, Benjamin C. Pierce, and Andrew Head. 2024. Property-Based Testing in Practice. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24, Vol. 187).* Association for Computing Machinery, Lisbon, Portugal, 1–13. https://doi.org/10.1145/3597503.3639581

[3] Ernest Ng, Harrison Goldstein, and Benjamin C. Pierce. 2024. Mica: Automated Differential Testing for OCaml Modules. https://doi.org/10.48550/arXiv.2408.14561 arXiv:2408.14561 [cs].

[4] Jessica Shi, Alperen Keles, Harrison Goldstein, Benjamin C Pierce, and Leonidas Lampropoulos. 2023. Etna: An Evaluation Platform for Property-Based Testing (Experience Report). *Proc. ACM Program. Lang.* 7 (2023). https://doi.org/10.1145/3607860

Diversity Statement

HARRISON GOLDSTEIN, University of Maryland, USA and University of Pennsylvania, USA

People like me are well represented in academia, especially in computer science. I have never struggled with discrimination in the way that many of my friends and colleagues have. Knowing this, I consider it my duty and responsibility to leverage the privilege I have to make my university, my subject area, and academia as a whole a better and more diverse place.

My approach to DEI is to start by educating myself on the challenges that others face. Since I may not intuitively understand others' perspectives, I try to default to curiosity and support, rather than rejecting new information. I talk with my friends and colleagues about their experiences, and I seek out media that helps keep me informed.[1]

Next, I encourage and support organizations and initiatives that try to address the challenges that I am aware of. One important initiative that I have supported and participated in is SIGPLAN's Programming Languages Mentoring Workshops (PLMWs). While PLMW is available to the whole community, not exclusively underrepresented students, one of its major goals is to broaden participation in programming languages research. Of the students funded to attend the most recent PLMW at ICFP 2024, 33% identified as female, 13% as Latino, 46% as LGBTQ+, 20% as first-generation in higher ed, and 20% as from institutions that do not award PhDs in their field.[2] I have given two invited talks at PLMW workshops, one on collaboration in collaboration with Samantha Frohlich and one on navigating the Ph.D. process. Informal feedback from both talks indicated that they helped to make underrepresented students more comfortable in the research environment and that the advice in those talks was useful. I intend to co-chair a PLMW during my time as a junior faculty member.

Another initiative I have given my time to is the Research Experiences for Undergraduates in Programming Languages (REPL) at the University of Pennsylvania, led by Joey Velez-Ginorio. Joey recently co-authored a CACM article on why *Research Experiences for Undergraduates Are Necessary for an Equitable Research Community* [2], and I am honored to have been able to help out in the program's first couple of years. While the program is still new, it has already provided value to students: every student from the first year of the program who applied to graduate school was accepted into a top-tier university like Northeastern University, the University of Toronto, and the University of Pennsylvania. Also, two students from the first year collaborated on successful publications. My participation in the program has been as both an administrator and as a mentor. In the first year of the program I reviewed applications and provided small amounts of input and guidance for the program's design. And during both years, I mentored undergraduate students.

Finally, I make myself available as a mentor and supporter to help mitigate challenges for individual students. As my teaching statement emphasizes, advising is the part of the academic process that I value most, and I use that affinity to improve the lives of underrepresented students. For example, I have been involved in the SIGPLAN-M program that matches mentors with mentees within the programming languages community; I have had three mentees, two of whom were members of groups underrepresented in computer science. I have also provided informal advising and mentorship to many other underrepresented students, including other PhD students, REPL students that I wasn't officially working with, and a handful of others who have reached out via email over the years for advice. My involvement has ranged from simple advice to interceding in difficult situations on students' behalf (e.g., helping them communicate more productively with their advisor).

I hope that by following this approach—staying educated, supporting positive institutions, and helping through mentorship—I can help make academia a more diverse and inclusive place.

---

[1] For example, I recently read *Invisible Women: Data Bias in a World Designed for Men* [1], which is an eye-opening look at how the way we collect and analyze data can re-enforce societal biases.

[2] Data provided by the PLMW organizers.

REFERENCES

[1] Caroline Criado Perez. 2019. *Invisible women: Data bias in a world designed for men.* Abrams.

[2] Joshua Sunshine and Joey Velez-Ginorio. 2024. Research Experiences for Undergraduates Are Necessary for an Equitable Research Community. *Commun. ACM* 67, 8 (Aug. 2024), 26–28. https://doi.org/10.1145/3665517

Letters of Recommendation

These wonderful colleagues have agreed to write letters that speak to my strengths. I highly recommend that interested hiring committees read all five letters, but if your university's policy is to read three or four, you may contact a prefix of the list. Contact information is available by request.

Benjamin C. Pierce	University of Pennsylvania	Ph.D. Advisor
Leonidas Lampropoulos	University of Maryland	Postdoc Advisor
Andrew Head	University of Pennsylvania	Collaborator
John Hughes	Chalmers University	Collaborator
Hila Peleg	The Technion	Collaborator