Teaching and Mentoring

Thomas Gilray

As an educator, I seek to share my deep enthusiasm for both the big ideas of our field, and their technical details, and to develop in students a self-reliance on core problem-solving skills, teaching them to learn through critical thinking and problem solving rather than to absorb rote facts. I believe strongly in the mutual impact that the teaching and research endeavors should have on one another. Research keeps the pursuit of knowledge and a deeper understanding vital and current. Likewise, teaching exciting topics with unveiled passion for learning is the best possible outreach to the next generation of computer scientists. My own interest in research was nurtured through a handful of particularly engaging and well-taught classes as an undergraduate and I seek in my own teaching to pay that forward. For my own sake, regular teaching experience has done wonders for my own sense of how to communicate ideas to others and consider them from a broader variety of perspectives.

Teaching Experience I have experience teaching four broad class topics in my positions at UMD, UAB, and WSU over the past eight years: programming language (PL) theory and paradigms, compilers and PL implementation, automata and theory of computation, and automated planning and reasoning (symbolic AI). I have experience teaching each of these topics multiple times and have developed a cutting-edge curriculum for each that I innovate upon with each iteration. My discipline is to develop two weeks of new and refined material for each offering as a way to build on what has worked well in the past while keeping my material fresh and updated. For example, in my programming languages class I focus especially on functional programming, semantics, and interpreters, and have recently expanded my focus on surveying language paradigms, type theory, and logic programming—in the last iteration I replaced the unit on C++ with one on Rust. In my (symbolic) AI course, I traditionally focused a bulk of the term on constraint solving, logic, SAT, DPLL, CDCL, and SMT, and in the last instance replaced and expanded a unit on game playing where students built Reversi players individually with a unit where students build adversarial agents playing a simulated capture-the-flag game in groups, which I found generated a lot of productive engagement among students. In my automata course, students would build a top-down parser and their own regular-expression library in Python that compiles REs to NFAs, NFAs to DFAs, and that minimizes DFAs.

In my Compilers course, which I've now taught once at WSU (CptS 452), students build a nanopass-style compiler for Scheme as a series of bite-sized projects; e.g., that implement the call stack using continuation-passing and then implement lambdas and continuations as objects. These passes aggregate to a working implementation that targets LLVM or C++ using Boehm GC (I have done both)—in the future, I would like to have students target WASM. In my

Advanced Programming Languages course at WSU (CptS 580), I developed my undergraduate course into a more advanced and expedited PL-theory curriculum for the first half of the course. Then, in the second half of this course we study reasoning about programs, first by studying type theory and Hindley-Milner let polymorphism and then by studying compositional analysis via summaries and whole-program analysis via abstract interpretation. In this second half of the course, we read relevant classic papers and students lead class discussions in small groups.

Teaching Philosophy My teaching philosophy emphasizes lower-pressure, more-granular exams and practical coding projects. I believe it's necessary to have rigorous in-class testing for assessment, but find that having only a single midterm and final puts too much pressure on those two test days and promotes "cramming" for these exams. Instead I prefer to hold a smaller half-size exam at the end of each unit, spreading out the pressure of these exams over more testing days and encouraging continuous studying across the term. I give my students a practice exam to take home for each in-class exam, give a few bonus points for students who complete this as a take-home and turn it in, and ask students to write a single page of notes that they may bring in for the exam itself, to further encourage active reading and studying by permitting notes generated in this process to be used during the in-class exam.

I also put a focus on the practice of programming in my courses. One of the most useful practical outcomes for many students will be the code they are required to write during a course and the necessary struggle and learning this catalyzes. For coding projects, I provide students with a starter and a set of public tests. Students use Git to submit their code to an autograder system developed by myself and two former undergraduate students who spent a summer enthusiastically building on a more basic system I had used for their term, fleshing it out into a fully-featured web-based application that integrates with Git for easy submission. The autograder will permit students to repeatedly submit code for testing against both public tests and hidden server tests (i.e., that are not included with the starter and that provide only a name as a hint of their substance), rate limiting these submissions using tokens that regenerate daily. The server allows me to set both "late" and "very late" submission periods after the nominal deadline, which apply increasing penalties, on a per-test basis, so that I can enforce a meaningful deadline during the term while also permitting students to earn most of the points by continuing to improve their projects until the last week of class.

This illustrates an important aspect of my teaching philosophy: I aim to erect a high bar for students to strive toward, with an objective assessement process, while offering guidance and support in this process, extra credit opportunities, and partly flexible deadlines. By avoiding subjective processes for grading I am able to serve as an ally to students in their learning goals without it having an impact on assessment outside the student's own demonstrable accomplishments. By offering opportunities for extra credit, more granular exams, and soft deadlines for coding projects, I am able to ensure I don't lose students who suffer a disappointing setback on an exam or are late finishing a required project. Not having deadlines means students will procrastinate and end up losing out due to the best intentions of their instructor, but having strict deadlines means that more students will check out of class part way through the term

after a demoralizing setback.

Student Course Evaluations I have received sometimes polarized feedback in my course evaluations, but have sought to incrementally improve each of my courses at each iteration and pay close attention to constructive feedback I receive. The most common criticism is that the course is too challenging, but sometimes that the information I've provided is unclear or could be organized better. I've worked to optimize my syllabus and approach based on this feedback and there are a many examples of adjustments I've made based on feedback from students, including several of the elements I've presented above as my teaching philosophy, such as using more granular exams and the rules used in the autograder system.

In my most recent course (CptS 580), a student said what aided learning most was *"reading materials, and professors teaching"* and another responded *"I appreciated the whiteboard teaching style. Its much easier for me to take notes and learn when the professor is writing everything out in real–time, rather than rushing through pre–made slides."* A student said of my teaching that *"he was knowledgeable in this area, and he was able to speak very fluently even without the use of PowerPoint. I like this course and professor."* However, a student also suggested *"I expect more reading materials or shared class notes, which can help us learn better. Because we are not native speaker, and sometimes will miss some details although listening carefully in class"*, which I feel is not in contradiction with the comments above. Although slides can serve as an offline reference, they can also be a crutch during lecture, making is harder to be in the moment, responsive to students and working out material live by coding or through work at the whiteboard. My goal has been to focus on the latter, while having students take notes they can use on exams. This way, active note-taking forms an important part of their learning process. I also agree that offline resources are helpful and perhaps especially for students who are non-native speakers, who do not learn as well during lecture and prefer to study with a book, or who are not so adept at taking notes while actively listening and thinking in the moment (I relate well to this myself). My plan is to use a mix of approaches during lecture, focused on having students taking notes during class, but I will release more supplemental study materials and will consider placing students in study groups to encourage students to share notes and help one another to understand material from class better. I did (informally) encourage study groups in the last term and heard from two students that this was helpful. My complete student evaluations from this term are included as supplemental materials.

I have frequently heard from students that my courses are among their favorite. I have also heard back from several former students that thoughtfully discussing a project built in my class during an interview has helped them win a job offer at a big tech company. One of these was a student, Clark Ren, who had built a vizualization system for program analyses as part of an independent study course, and who received a job offer at Amazon. The others were students who talked about their projects in my Automata course. I regard this kind of positive impact I can have on young computer scientists to be the most important part of my service, outreach, and educational roles as a professor.

Mentoring Experience I've been fortunate to closely collaborate with and mentor PhD-student advisees of collaborators. I'm especially proud of a paper developing a new approach to all-to-all communication for variadic workloads that was primarily the work of PhD student Ke Fan—we published this algorithm at HPDC 2022. I taught Datalog foundations to Arash Sahebolamri and together we developed some novel extensions to Datalog, and a methodology for implementing a highly extensible Datalog via Rust macros—work we published at CC 2022. We published our Rust framework, Ascent, as an open-source tool and have since produced a scalable pointer analysis for LLVM, called Yapall, developed with our collaborators at Galois, inc. Arash graduated with his PhD from Syracuse University in 2023. I've worked on monotonic aggregation for data-parallel Datalogs with PhD student Yihao Sun, and we published at CLUSTER 2023, ASPLOS 2025, and AAAI 2025. I've also worked with PhD Student Akmedur Rahman Shovon on GPU-based relational algebra, and published at USENIX ATC 2023. He has also recently published a paper with me at ICS 2025.

In Fall 2023, I hired Sowmith Kunapaneni, an MS student who took my class on programming languages the previous Spring. He was eager to work on additional PL projects outside of class, learned quickly, and converted to the PhD program. In the last 9 months at UAB, Sowmith made rapid progress in learning about Datalog, helping a group of BS and MS students with their compiler, learning program analysis theory, and studying our GPU-based join algorithms. Sowmith and I are developing a new version of our lab's logic programming language, Slog, and since joining WSU with me this year, he has published a paper at VLDB with myself and collaborators, and has made major progress toward publishing a paper on vectorized joins on the GPU. We plan to submit this work to SIGMOD or VLDB by the end of the summer.

In Fall 2024, I began working with Akash Rao, a new PhD student in my lab (co-advised with Ananth Kalyanaraman) studying parallel join algorithms and developing a new approach to applying the parallel-prefix algorithm to large N-way relational joins, or linear recursive Datalog rules that can be unrolled to large N-way joins, applying this technique to parallelize transitive closure in a communication-avoiding manner. We've submitted this work to SC 2025. As of Summer 2025 I have begun working with Henry Olson, who will be starting as a new PhD student in my lab Fall 2025.

Mentoring Philosophy My mentoring philosophy centers around the idea that for most people, the best way to learn a complex topic is by rebuilding the state of the art. I typically begin working with students by bringing them up to speed on foundational topics in my area, finding out what recent ideas are most exciting to the student, and guiding them to rebuild this state of art themselves.

With Sowmith, I guided him to develop components of our Datalog implementations, such as GPU-based relational algebra—both rebuilding some to better learn all the details himself, as well as developing our new Datalog compiler closely with me. With each new student, I attempt to manage the overwhelming amount of new material by orienting them toward the most important details, and scaffolding out a path for them to learn in a practical, on-the-ground manner through implementation.

I also feel strongly that a PhD student learns from a wide variety of influences in their lab and department, not only their advisor—my own best influences and resources in grad school were frequently the postdocs and other PhD students in the lab. I endeavor to ensure my students are supporting one another and constantly in a position to collaborate and learn from one another. My PhD students always have both a primary research project, along with supporting roles on other projects for which they are not primarily responsible. I also assign PhD students as mentors to my BS and MS students so that in addition to their meetings with me, the junior student also meets privately with a PhD student at least once a week. This provides the junior student another sounding board who they might feel more open to seeking certain kinds of help from, and as teaching can often be the most effective method for learning, provides the more senior student with a valuable experience communicating ideas they've recently learned with enough clarity to support someone still new to those ideas.

One of the challenges I've discovered in mentoring PhD students is keeping pace and setting expectations around the PhD process. New students will often need to be more one-track-minded in getting up to speed on a very technical topic, and will sometimes lack resourcefulness in taking concrete steps. I aim to guide a student's focus closely in the first year, and provide regular advise on making incremental progress and how to be resourceful in identifying these next steps themselves. As the student progresses, I move to add parallel tracks of work, a weekly paper to read with me, reading groups with collaborators, and supporting roles on other projects. If a student wants an academic career long term, I will emphasize the necessary challenge of publishing multiple papers at the top venues. I see it as part of my advising role to calibrate student expectations around the difficulty of doing this, and orient students to the social process of research and publishing, addressing the strategy inherent to clear technical communication and marketing ones ideas successfully. For example, I recently had a detailed discussion of a successful ICFP paper with Sowmith, comparing the accepted draft to a previously rejected ArXiv copy and prompting them to consider what improvements and repositioning of the paper contributed to its final acceptance at a selective conference.