Teaching Statement

Klaus v. Gleissenthall

Teaching Philosophy I have been extremely lucky to have had some incredible teachers who made me love computer science and are ultimately responsible for my pursuing a career in research. One of the first courses that really piqued my interest was a seminar on machine learning algorithms that were able to compose music. I was drawn in by the intriguing idea of machines being able to achieve such an innately human and creative task as composition. What ultimately captivated me, though, was the beauty and simplicity of the underlying mathematics which enabled it.

My biggest goal in teaching is to convey the same enthusiasm and love for our field to my students. From this point of view, teaching is ultimately not about getting students to learn and apply some particular language or algorithm or mathematical technique; to me, it is about helping them understand and see the beauty in the reasoning principles that underpin it.

At the core of this beauty is always clarity: in science, no definition or algorithm has to be accepted as a given, in its own right, but it must always earn its place. Why is always a good question!

One of the things I find most rewarding is to help students dissect a problem until all its part fit together. The way to achieve this is often through examples which I always aim to draw from real world problems that students care about. Finally, I think that the best way to understand a method is by implementing it.

I find that teaching also presents a great opportunity to make students think about program correctness, not as an afterthought, but as a design principle. This is closely tied to my research interests: the philosophy behind pretend synchrony is that one can often structure algorithms in a way to make reasoning about their correctness easy. This is not only helpful when formally verifying their correctness but even when manually reasoning about them. I aim to incorporate these principles into my teaching.

Teaching Experience I have helped teach two verification ("model checking") courses at Technische Universität München and tutored a Prolog course at the University of Cambridge. The verification course involved giving lectures, while the Prolog course consisted of giving tutorials to small groups of students. I enjoyed teaching these courses and think that teaching presents an essential way to have impact as a researcher: by passing on ones way of thinking.

Mentoring I have advised several PhD students who worked with me during my time as a post-doc at UCSD. My style of mentoring is collaborative: my main goal is to realize interesting projects and to do cutting-edge research together – while teaching some important technical and research skills along the way. An essential ingredient for this is the friendly and positive environment that results from working on a common goal. I aim to strike a balance between helping when help is needed and giving enough autonomy. A quote that got stuck in my head sums up this philosophy nicely: "when they're stuck writing the code, I commit the first line; when they're stuck writing the proof, I write the first lemma".

Future Teaching My training has put me in a position to teach undergraduate courses on programming languages, logic, formal methods, compilers, software engineering and introductory programming. I would also be happy to teach undergraduate classes outside my immediate areas of interest. I am excited to teach seminars and courses at graduate level. A natural topic for me would be the verification of distributed systems and hardware security. I would also enjoy teaching courses on related topics like verification, logic, the internals of SAT and SMT solvers and concurrency theory. For example, I would love to teach a course that builds a verification method from first principle, starting with a simplex solver for satisfiability of linear arithmetic formulas, instrumenting the solver to turn it into an interpolating solver, and finally embedding the solver in a CEGAR loop to find invariants for simple programs.