Teaching Statement

Benjamin Kiesl

November 2019

This teaching statement consists of two parts. In the first part I outline my teaching philosophy, and in the second part I discuss possible courses I can teach.

1 Teaching Philosophy

"*How did he do this?*" That was my first question after I had just finished the book and laid it down, convinced that I'd finally dive deep into computer science. The book? "*What is Computer Science?*" by Peter Rechenberg—a retired professor with a strong focus on compilers and formal languages. Until that point, my fascination for computer science had been limited, to put it mildly. But Rechenberg's book has changed my view of computer science completely, awakening a kind of enthusiasm that hadn't been there before. So, *how did he do this?*

On my quest for answering this question, I've learned several lessons that should not only turn me into a better writer but also into a better teacher. The details of the journey are not so relevant, but what is relevant is that it had me read various books about *learning* and *writing*, including one by Rechenberg himself. The main lessons I learned—which form the corner stones of my teaching philosophy—can be summarized as follows: *awareness of the curse of knowledge*, *importance of context*, *deliberate practice*, and *openness to criticism.*

The Curse of Knowledge. You surely know that one person who is extremely knowledgeable about a certain topic but is just not good at explaining it. Why? The reason is often *the curse of knowledge*: it's when we've become so familiar with a topic that we forget that the people around us don't even understand half of the words we use. We are all victims of the curse of knowledge, but I believe that if we want to be good teachers, we need to fight it, and to fight it, we need to become aware of it. Whenever I have to explain a complex topic—no matter if in a talk, a personal discussion, or a paper—I try to question the words I use as much as possible. For example, before I use a word like *foobar* (i.e., a word that not everybody might know), I ask myself, does my audience understand what a *foobar* is? Can I maybe use a different word, or can I first explain in an intuitive way what a *foobar* is, before I use the word? Should I maybe avoid explaining the part about the *foobar* at all because it anyhow requires too much background knowledge? Should I assume less background knowledge to make myself understandable? Awareness of the curse of knowledge is, in my opinion, one of the most important qualities of a good teacher, and it has served me well over the years.

Context. I strongly believe that efficient learning is only possible if the learners are aware of the context surrounding a particular course or topic. As a student, you deserve answers to several questions: Why is the course I teach you even supposed to be interesting? Why did people decide to make it part of your curriculum? How are you going to benefit from taking this course? This becomes especially important for the more theoretical subjects I'm particularly interested in teaching. How, for instance, is a student supposed to be interested in mathematical logic if she doesn't know what it's good for? Just going over the basic theory of logic and proving some theorems is surely not going to do the job. There is so much more an efficient teacher needs to discuss: Why logic builds the very foundation on which computer science is built. How it helps us gain a better understanding of computer hardware, software, and even our own ways of thinking. That it can be used to prove the correctness of safety-critical computer programs or security protocols. One could go on here forever, but what should become clear is that *context matters*, since without the necessary context students might not want to listen to what you teach them.

Deliberate Practice. This is an approach I got familiar with not too long ago. It's described in detail, for instance, in the book *Peak: Secrets From the New Science of Expertise* by Anders Ericsson and Robert Pool, which convinced me that many of their ideas can also be used effectively in teaching at the university level. A simplified summary of deliberate practice is this: If a student wants to become more skilled at something, a teacher should constantly challenge her by proposing exercises at the exact right level of difficulty—not too easy but also not too hard. Deliberate practice can be applied to skills reaching from chess over golf to mathematics or programming. I believe that as teachers we can adopt ideas behind deliberate practice by carefully choosing the exercises we hand out to our students. If we manage to find the right levels of difficulty and if we can increase the complexity of the exercises over time without making them too hard, students can benefit tremendously. I have, for instance, tried to apply the ideas behind deliberate practice when creating a set of exercises that serves as a tutorial for the tool Tamarin (a tool that allows users to verify the correctness of security protocols) and so far I have received very positive feedback from people who used my tutorial. In the future, I also want to incorporate it into possible lectures I give. I feel that especially for the more theoretical courses and for programming, deliberate practice can be very beneficial.

Openness to Criticism. I once read that what distinguishes the good from the best is the openness to criticism. I've seen this myself with a professor back when I was a student: After one semester, the professor received some negative feedback about his lecturing style. The professor never complained about this feedback; instead, the next semester he gave another course where it was obvious that he had incorporated the feedback—and the course was fantastic. Especially for younger researchers like me, I believe it is important to ask for feedback and to incorporate it as much as possible.

2 Courses I Can Teach

So far in my career I haven't held lectures. I was involved in teaching at the graduate and undergraduate level as a teaching assistant though. I'm also currently supervising two students, which I enjoy a lot.

In general, I'm very happy to teach any courses for which there is currently a need for teaching, but here I want to highlight courses that I would be especially willing to teach. Considering my background in theoretical computer science and formal methods, I can identify the following courses (contained in the computer science curriculum at TU Graz) as the ones that align most with my interests:

- Theoretical Computer Science
- Logic & Computability
- Foundations of Artificial Intelligence & Logic
- Foundations of Computer Science

Considering my experience in software development, I would also be very open to teaching classes that are more directly related to programming, algorithms, and data structures. This includes courses on topics such as the following:

- Object-oriented Programming
- Algorithms & Data Structures
- Design & Analysis of Algorithms

On the graduate level, also when it comes to more advanced courses, I would be very happy to teach courses on one or more of the following topics:

- Security Protocols
- Model Checking
- SAT Solving
- Computability Theory
- Complexity Theory

These are just the most obvious choices. As I mentioned in the beginning, I'm very happy to help out wherever there is a need as long as I believe that my expertise on a topic suffices to teach a certain course.