Sasha Rubin, Teaching Statement, December 2017

As indicated in my CV, I have designed and taught courses at all academic levels:

- Graduate-level courses at Technical University of Vienna, University of Naples, Cornell University, and the European Summer School in Logic, Language and Information.
- Undergraduate-level courses at University of Naples, University of Cape Town, Cornell University, University of Auckland, and University of Wisconsin Madison.

I work hard at improving my teaching. In particular, I have sought out teaching mentors, notably Maria Terrell (Director of Teaching Assistant Programs) and David Way (Associate Director of Instructional Support) at Cornell University. Maria taught me, for instance, the importance of explicitly making material relevant to students' prior knowledge and experience, especially for first-year undergraduate students who may not have developed the stamina and motivation for engaging in more abstract/mathematical material. David taught me that improvement grows out of *reflection* on *feedback* about one's teaching. David also taught me the importance of monitoring student interest and comprehension and adjust one's teaching practice accordingly. I also made an effort to discuss teaching strategies with teachers that have won awards for their teaching (e.g., John Hopcroft), and attend their lectures in order to *experience* good teaching.

Here are my main teaching practices:

1. **I try make sure the material engages students:** This often involves asking questions that are fun for students to solve, or discussing the big ideas underlying the technical content. Consequently, students were more engaged during class which made the experience for everyone (including me) more enjoyable than it otherwise would have been.
2. **I formally and informally solicit feedback:** I ask for student feedback (positive and negative) after the course is finished in order to understand what worked well and what didn't (from the student point of view). Early on in a course I also ask for basic feedback to fix easy problems (e.g., blackboard writing should be bigger). I have also asked mentors and good teachers to visit my class and observe my teaching, especially if I am concerned about one or another aspect of my teaching or my students.
3. **I get to know my students:** For instance, while teaching at Cornell I scheduled a 10-15 minute meeting with each student (I had 30 students in each class) in order to find out why they were studying, what they planned to major in, and what (if any) apprehensions they had about the course. Consequently, I was able to incorporate examples that would interest students, as well as refine how I pitched higher-level ideas in class.
4. **I use technology when appropriate:** For instance, the second time I taught the Calculus course at Cornell I administered weekly short quizzes, using the online learning platform moodle (see `moodle.org`), explicitly designed to encourage students to read the relevant chapter of the textbook *before* coming to class. Consequently, the level of student understanding and questions during class were *much* higher than they were before.
5. **I give hints of interesting material that goes beyond the syllabus:** For instance, I connect material in the syllabus to material that will be covered in more advanced courses on the same topic, or I connect material to my own research.
6. **I try teach students how to identify, formalise and solve problems:** In other words, I try use the syllabus to arm students with general ways of thinking that are useful for solving any problem, not just the ones in the syllabus. This usually involves demonstrating my thought process, as well as giving students an opportunity to try solve problems in class.
7. **I encourage student interaction:** For instance, I sometimes break students into pairs so they can try solve a problem together or explain an issue to each other ("think-pair-share"). This interaction often helps students realise that they don't yet understand an idea or technique well enough to work with it on their own.
8. **After each lecture I reflect on how I could improve presentation of the material:** This sometimes involves improving the pace, refining what to write on the board, and finding better ways to break up the material into chunks that students could follow. This is especially useful if I teach a course more than once.
9. **I go to lectures of other academics:** This allows me to see what else is going on in the department, and sometimes serve as a master-class in how to teach.

Here is some student feedback, the first two from a stage 1 course on calculus at Cornell, the next from a stage 3 course on logic and computation at the University of Cape Town, and the remaining two from a course on graph-games (advanced) at TU Vienna.

```
Sasha was a wonderful lecturer -- very organized, clear, and willing to help anyone with difficulties
(proactively, as well -- he sought out people who were doing poorly and offered assistance, which was
extremely beneficial).
```

The class was interesting, fun, and easier. Professor was very well prepared and made us really understand what we were learning by having us write mathematical formulas in words. Instead of memorizing formulas, he helped us understand what we were learning, why we were learning it and why it was useful.

I have enjoyed your maths course the most of all the maths courses I've taken so far at UCT. Even more than the content, your delivery was excellent... I made the decision during one of your lectures to do honours in mathematics and computer science next year at UCT.

I especially liked the fact that you let us engage with the ideas. I think I was able to deepen my understand of graph games a lot because I never took an actual course in it.

During the lecture Sasha Rubin promotes logically and mathematically rigorous thinking and achieves a rare le of engagement among the students. This combination results in a very enjoyable and stimulating course.

Future teaching plans

I believe that computer-science curricula should provide students with a rigorous foundation in computational thinking and in the reasoning required to design and develop computational systems.

Computational thinking is rooted in mathematical logic. Thus, even if there are not many dedicated logic courses, I believe that one can and should inject computational aspects of logic into existing topics, e.g., Boolean logic and satisfiability in courses on discrete mathematics, unique-readability of logical formulas into courses on parsers, first-order logic and temporal logics in courses on databases, etc.

I can contribute to course-design and teach the following compulsory courses listed on the ICL website: 140 (Logic), 142 (Discrete structures), 145 (Mathematical methods), 141 (Reasoning about programs), 150 (Graphs and Algorithms), 165 (Presentation skills), 240 (Models of computation), 202 (Algorithms II), 231 (Introduction to Artificial intelligence), 233 (Computational Techniques), 303 (System Verification). With a little preparation time (e.g., 1 semester), I could help teach: 112 (Hardware), 120.1 and 120.2 and 120.3 (Programming), 113 (Architecture), 130 (Databases), 245 (Probability and Statistics), 276 (Introduction to Prolog), 337 (Simulation and Modelling), 349 (Information and Codes), 347 (Distributed Algorithms).

I can contribute to teaching aspects of the following topics which intersect with my research: probabilistic systems (Markov chains, Markov decision problems), database-theory (logics for querying, including fragments of first-order logic such as conjunctive queries, fixpoint queries, first-order temporal logics, DATALOG and its fragments), automata for applications (including tree automata for structured tree-data), game-theory (modeling and verification of rational agents), automated planning (including applications to multi-agent systems such as collaborative mobile robots, swarm protocols), and knowledge representation (design of expressive and computationally-tractable languages to describe systems and specifications).

I could immediately offer the following advanced courses:

- **The automata-theoretic approach to verification and synthesis**. Formal methods provide algorithms for automatically determining whether a mathematical model of a system satisfies a specification (verification) or, alternatively, to automatically construct a system that satisfies a given specification (synthesis). This field, for which the founders and proponents won two different Turing awards, has been used successfully for complex systems, and many hardware and software companies use these methods in practice, e.g., verification of VLSI circuits, communication protocols, software device drivers, real-time embedded systems, and security algorithms. In 2000, Moshe Vardi and Pierre Wolper won the Gödel Prize (an annual prize for outstanding papers in the area of theoretical computer science) for a 1994 paper that launched a unifying paradigm, i.e., the automata-theoretic approach to verification and synthesis. The power of the approach is that it reduces the problems to classic problems in automata-theory, and so neatly separates the encoding from the combinatorics. The course will provide an introduction to the theory of automata and demonstrate its applications to verification and synthesis.

- **Graph-games for Multi-agent Systems**. A central theme in artificial intelligence (AI) revolves around the concept of an agent, which is any entity that can interact with other agents and/or the environment using sensors and actuators. In many cases, one is interested in "rational agents" (which can include humans, robots, software agents), which try to achieve a specific goal, and whose actions are guided by this desire. A system that consists of a group of such interacting agents is called a multi-agent system (MAS). Examples include software agents on the Internet, driverless cars, humans or software playing multi-player card games, robots exploring new and dangerous environments, and biological systems such as cultures of bacteria and swarms of insects. In this course we will model MAS as games on graphs, a convenient mathematical model of many phenomena in mathematics and computer science that involve *interaction*. The course will provide students with the foundational mathematics and algorithmic tools for modelling and formally reasoning about MAS.