Sasha Rubin
✉ *rubin@unina.it*
Teaching Portfolio

Hiring Committee, Department of Computer Science
University of Auckland

December, 2017

This document outlines my teaching and supervision experience and philosophy.

Teaching

My own teachers

The most influential course I've attended was a graduate level introduction to mathematical logic at the University of Auckland. It was taught by a topologist, David McIntyre, based on Moore's method — we were given basic definitions, followed by statements of fundamental theorems that we were to prove ourselves and present the following lesson. The material was to form the background of my graduate study. I probably did my best learning when my classmates presented a proof that I was unable to produce beforehand. Identifying the points in the proof that I did not think of trained me to get a focus on why and where my own reasoning had fallen short. It is these sorts of insights that I try generate in my own classroom.

I also had memorable courses by Christian Calude (algorithmic information theory), Bakhadyr Khoussainov (automata theory), and Michael Dinneen and Peter Gibbons (advanced artificial intelligence).

My teaching experience

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.

Example Experience – Cornell University

I taught three semesters of Calculus for Engineers at Cornell University, essentially a second course in calculus for first year students that follows a textbook very closely.

After each lecture I reflected on how I could improve presentation of the material. This usually involved 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.

My students were generally capable of acquiring, on their own, the skills to work routine problems. Consequently my main goal was to get them to reason mathematically, both verbally and in writing. I noticed that students are very sensitive to the wording I use. I kept a list of phrases to which they seem to respond, such as 'can anyone help A with her answer?', 'can you explain B's idea to me?', ‘what do you mean by X?’, ‘are you sure?’, ‘who will summarise today’s class?’, ‘if all I do is teach you computational procedures I’m short-changing you’.

I worked hard finding angles on the material that would engage my students. This invariably involved finding a question that sounded fun to try to solve. My favourite moments in class were those that involved discussing the big ideas in calculus. After writing a theorem on the board I asked: ‘why should we believe this?’ (a soft version of ‘how do we prove it?’) and the often overlooked ‘how is this theorem useful?’.

I also learned to ask questions that probe student knowledge and understanding. For instance ‘why would you say that?’ and ‘tell me more’ helps to diagnose their logic, and asking easy recall-level questions let me see whether students had been listening.

Although my students were most comfortable with being given algorithms to solve problems I instead focused on problem solving techniques *ala* Pólya: 1) identify the unknown, 2) if you can’t solve the problem find a problem that you can solve that has a similar unknown.

I experimented with small group work in class to alter the pacing of lectures and encouraged my students to think and reflect on what they had and had not understood.

Here is some student feedback:

```
- 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.
```

Example Experience – University of Cape Town

Early in 2010 I designed, taught and administered the undergraduate course on logic and computation in the mathematics department at the University of Cape Town. Student abilities in the class were very mixed which meant I had to structure the course and pace very carefully. I slowed the lectures down a bit and moved harder questions to the tutorials. One of the strongest students sent me an email at the end of the term:

```
- 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.
```

Example experience – Technical University of Vienna

In 2017 I designed and taught an advanced course “Milestones in Solving Games on Graphs” at the Technical University of Vienna (2017). The course attracted a mix of advanced undergraduates, Msc students, and PhD students. The course started from first principles, included many motivating examples, and also included lectures on my own recently published research.

Here are some student comments:

```
- I especially liked the fact that you let us engage with the ideas.
  I think I was able to deepen my understanding 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 level 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 am open to teaching any course listed on the UoA website. I can immediately, or with a few weeks preparation, teach: COMPSCI 101, 105, 107, 111, 210, 220, 225, 320, 350, 367. Other undergraduate courses would require me more preparation time (e.g., 1 semester). Existing postgraduate courses that I would enjoy teaching include: COMPSCI 720, 750, 761, 765, 767. In particular, I would be able to integrate my current research into COMPSCI 761 and 767.

I can contribute to course-design and teaching on 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 offer the following advanced courses:

- **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.

- **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.

Finally, I have an interest in engaging in the scholarship of teaching and rigorously investigating how to improve student learning, especially of abstract and mathematical concepts. While at Cornell, I had regular meetings to discuss pedagogy and rigorous insights about teaching and learning with David Way, Associate Director of Instructional Support, Center for Teaching Excellence, Cornell University (dgw2@cornell.edu). In Auckland, I would be interested in pursuing this direction with Paul Denny and Andrew Luxton-Reilly in the computer-science education group.

Supervision

My own mentors

I did my Phd under the supervision of Bakhadyr Khoussainov. Bakh taught me how to do research: how to isolate interesting problems, how to write papers and how to give talks. Memorably, when Bakh understood that we needed to understand modal logic in order to tackle issues in verification, we worked our way through a number of chapters of a textbook of modal logic, meeting once a week to discuss the content. This was highly rewarding for me, and taught me how to process and internalise large amounts of mathematical material, as well as to understand that not all material is equally important.

During my PhD I spent one semester working with Moshe Vardi (Rice University) from whom I benefited immeasurably. For instance, Moshe taught me the importance of formalising intuitions.

Experience and philosophy

My usual approach to supervision is to discuss possible problems with students and let them pick one to work on. If the student lacks confidence or is unsure about how to proceed, I create a mini-proposal and timeline for them to follow. I meet with students once a week to discuss progress and troubleshoot. I consider undergraduate research successful if the student a) has fun, b) is challenged, and c) produces and publishes novel research.

In 2009 I supervised six undergraduate students for a two month research experience (REU). The student selection process was very competitive and so I received exceptionally talented undergraduates. The students formed two groups and worked on two projects. During this time I learned the value of giving students a few days to brew and filter their ideas before group discussions. Overall it was a rich experience for both me and for my students. One exceptional student expressed to me that the experience helped him decide to pursue a career in research. The results were subsequently published in the journal *Theoretical Computer Science* and the conference *GandALF'12*.

In 2012 I co-supervised an undergraduate summer project which led to a publication in the conference *LATA'13*. While the student was writing up I realised that a proof required some formalities that the student did not know. At that point the student was keen to learn *how* I came to realise there was a problem. This episode taught me the value of modelling good mathematical thinking for learners.

In 2017 I supervised an undergraduate thesis, that included theoretical and practical components, on "graphical games", a topic at the interface of game-theory and graph-theory. We are writing up this work.

I recently worked with four junior PhD students, resulting in papers published at *VMCAI'14, IJCAI'16, AAMAS'16*, and *VMCAI'18*. In all cases I learned the value of helping graduate students to structure their thinking so they could contribute more to the collaboration than they otherwise might.

Future plans

I am very interested in supervising undergraduate projects and MSc theses (COMPSCI 380, 789) and PhD students, particularly in topics in artificial intelligence. I list a number of project ideas, each having aspects that can suit students with a variety of interests and levels (pre-requisites are listed in parentheses):

1. **Formal methods for collaborative tasks of mobile-robots in partially-known environments.** Study, extend, implement and evaluate algorithms for describing and verifying co-operative mobile-agent tasks (such as patrolling, searching).
2. **Algorithms for strategic-epistemic analysis of Poker and Bridge.** Study, extend, implement and evaluate synthesis algorithms for multiplayer games of imperfect information with public actions and epistemic reachabilty goals, and test on Poker and Bridge scenarios (formal methods, artificial intelligence).
3. **Automatic programming.** Study methods based on synthesis and planning to automatically derive strategies/policies from declarative specifications of problems (artificial intelligence, formal methods).
4. **Describing very large game/MDP policies to users.** Study how to decomposed strategies/policies into components that are best described declaratively and best described operationally. Study how to use dimensionality reduction techniques to explain large game/MDP policies to users (machine-learning, knowledge-representation, formal methods).
5. **Optimal strategies in fully-observable non-deterministic planning problems.** Formalise and study finding optimal solutions to fully-observable non-deterministic planning problems using recent results in games on graphs (automated planning, formal methods)
6. **Analysing games on graphs with reinforcement learning.** Study to what extent reinforcement learning can be used to analyse classic objects in formal methods such as safety or liveness games (machine-learning, formal methods).