# Lecture 17

# Epilogue

## 1 Planned Improvements for the Following Years

This course is an integral part of the upcoming Computational Linguistics MA program here at Stony Brook, so I want it to be top-notch. In combination with the NLP course in the computer science department, it also has to get students to a level where they can sign up for internships over the summer. There are several respects in which it currently falls short of this goal, in my opinion. In no specific order:

- *Practical Tools*
  I missed several opportunities to present practical tools such as commonly used implementations of two-level morphology or Python's libraries for FSAs and FSMs. In future years, these will be included as parts of homework exercises. I will not discuss them in class because
  - it is more important to understand the concepts than the quirks of a particular implementation,
  - these tools are well-documented,
  - libraries come and go and often change,
  - odds are that if you land an industry job, you'll have to use a different toolkit,
  - the most important skill in life is autonomous learning, in particular when it comes to computers. Any kind of practical skills you learn in college will be outdated within a couple of years, and in the real world nobody is going to take time out of their schedule to teach you new tricks.
- *Probabilistic Models*
  The section on monoids was actually intended to start off an ongoing comparison of how categorical models can be enriched with probabilities without altering their basic operation. Standard presentations of probabilistic models are needlessly messy, heavy in notation, and focused on extraneous details. It is preferable to have a well-behaved model, whatever it may be (strictly local string grammar, regular tree languages, etc.), that can then be combined with a few standard techniques to create a probabilistic model. This also makes it clearer how the workload is distributed between the model and the probabilities — remember, factorization is key for understanding! Unfortunately there is not

enough time to fully develop this perspective, and I'm not sure what could be cut to make room for it.

- *Algorithms*
  We initially covered a fair share of algorithms and related concepts like asymptotic complexity, which then never made an appearance again in the rest of the course. I had actually planned to introduce a few more algorithms and basic techniques via linguistic examples. For example, Riggle's implementation of OT relies on Dijkstra's shortest path algorithm, which is used in a tremendous number of optimization problems. Parsing is a good opportunity to discuss dynamic programming, i.e. strategies to avoid computing the same value multiple times in different functions by memorizing intermediate results and making them accessible to specific subroutines. Once again this had to be cut for time reasons.

- *Python Implementation*
  Python was dropped after the move to tree languages. That is due to the increased complexity that comes with switching from strings to trees. Trees, unlike strings, are not a standard data structure in programming languages, so one can either define their own standard or rely on a library. Both options come with significant programming overhead. My hope is that this can be offset in future years by making programming with trees a central part of CompLing 1.

- *Focus on Theory*
  The course is very theory-heavy by design. In contrast to tools and NLP techniques, theory does not become outdated. FSAs are over 50 years old, and they are still exactly the same kind of mathematical object. Theory is also important for coming up with new solutions rather than simply reusing somebody else's ideas. In addition, there is already tons of material that focuses on more practical matters, spanning the whole range from textbooks to MOOCs. So there is little point in offering yet another class along these lines (which would also overlap quite a bit with the CompLing courses in the CS department).

  That said, the lecture notes should probably be refactored to allow for a more personalized reading experience, where students that are scared of formalism or have no experience with proofs can follow a "safe road" that covers all essential points while the more adventurous can dive into the nitty-gritty of equivalence proofs and closure properties.

- *Presentation*
  I am pretty happy with the first seven chapters, which offered plenty of code, examples, figures, images, boxes with background information, and generous color coding to keep things approachable. The margin notes were also a lot more light-hearted and tried to instill a general passion for science and computers rather than being simply footnotes expounding on some arcane point. After that, things got a lot less visual as more and more of my time was being consumed by other duties.

- *Readings*
  I tried to offer a fair number of accessible readings on various topics, and overall I am pretty happy with the mix. However, I did not present a worked-out plan

for self-study, i.e. a progression of increasingly difficult papers on a given topic that take you all the way from beginner to expert. For some topics this is simply impossible due to enormous difficulty gaps — the literature on logical graph transductions goes from "simple intuitive sketch" straight to "brutal tour de force". But many of the more linguisticy topics could be mapped out very nicely.

- *The Syntax Part*
  Given my research interests, it is somewhat ironic that the phonology part seems to have worked out much better than the syntax part. The latter ended up touching only briefly on several topics of great importance, such as Tree Adjoining Grammar (TAG), Categorial Grammar (CG), Dependency Grammar, feature percolation, crossing dependencies, machine translation, and synchronous grammars. I also omitted several areas of application; for example, tree transductions are used nowadays to automatically translate annotated corpora into the right format for a given grammar formalism. And while the phonology chapters had some nice data sets to establish typological diversity and how it relates to formal models, the syntax part mostly just claimed that things are dandy without ever carefully analyzing specific pieces of data. To some extent this simply reflects differences in how computational research is conducted in phonology and syntax — the former often looks at specific surface processes rather than what analysis has been proposed in the literature, while the latter takes a specific syntactic proposal for granted without worrying about the actual data.

- *General Computer Skills*
  When you look at jobs in the industry, they usually require a fair share of general computer skills, such as familiarity with the Linux command line and shell scripting, SSH and VPN, working with databases (MySQL, MariaDB) and version control systems (git,svn), markup languages, and collaborative work flows (e.g. via wikis). I tried to bring some of that to the class by using github and wikis, but that is not enough. We are currently developing an infrastructure of virtual machines for the computational linguistics lab, and those virtual machines will be available for students to run on their own computers. That way, we can easily deploy a fully configured Linux environment, and hopefully that will give students more exposure to some of these things.

One very obvious criticism is that the two CompLing courses are not well-coordinated at this point. My hope is that some of the concepts of the first six chapters can already be covered in CompLing 1 (without definitions, theorems and proofs), and that CompLing 1 will also include some algorithms and general computer science topics like the binary number system and how to estimate memory usage. But this is not a one-way road, and CompLing 2 will also have to adapt.

One thing that will not change is the difficulty of the course and the weekly workload — if anything, both will go up quite a bit. This is ultimately a boot camp for professional computational linguists, and boot camps have to push you to your breaking point.