Computational Linguistics as Language Science

Spring 2020

Thomas Graf

About this book

What?

This textbook is a graduate-level introduction to proof-based (rather than modeling-based) computational linguistics for linguists. The focus is on how a computationally informed perspective can illuminate aspects of language that theoretical linguists and cognitive scientists care about. A concerted effort has been made to keep the material as approachable as possible without sacrificing exactness. To get the most out of the book, readers have to be willing to engage with mathematical notation, but each unit is designed in a modular fashion so that the less mathematically inclined can skip the parts they find too tedious.

Why?

Like most textbooks, this one was written because of a picky instructor who wasn't quite happy with the existing options. It grew out of my lecture notes for *Computational Linguistics 2* at Stony Brook University's Department of Linguistics. Within linguistics, computational linguistics tends to be geared towards model-building and simulations: MaxEnt learners, corpus-based techniques, and so on. These topics are covered in a different course at Stony Brook, though, with Computational Linguistics 2 exploring the proof-based side of the field: formal language theory, subregular complexity, logic, string and tree automata/transducers, learnability, parsing theory, algebra, plus some algorithms and data structures. There is a number of excellent textbooks that cover a subset of these topics, but they all fell short in some respect:

- Marcus Kracht's *The Mathematics of Language* is a treasure trove and still one of the most rewarding textbooks ever written, but it is too hard and theoretical for the average linguistics student.
- *Speech and Language Processing* by Jurafsky and Martin is an excellent reference book, but it is clearly aimed at computer science students. The focus is on engineering techniques rather than investigating linguistic questions from a computational perspective.
- András Kornai's *Mathematical Linguistics* presents a more appropriate compromise between linguistic inquiry and applications, but is still too far removed from linguistics and cognition for my taste.
- *Mathematical Methods in Linguistics* by Partee, ter Meulen & Wall is a classic, but as the title implies it mostly focuses on mathematics, in particular those areas that are needed for semantics (set theory, algebra, and predicate logic). Its final

chapter on formal language theory is a commendable inclusion, but does not make a strong case for why linguists should care about this perspective.

- Ed Keenan and Larry Moss have produced an impressive and very comprehensive introduction to mathematical linguistics with *Mathematical Structures in Language* (and I'm not just saying that because of the many fond memories I have as a teaching assistant for the course that their book sprang from). But overall it is a mathematical methods book, not an introduction to proof-based computational linguistics as a means to study language.

- Some subtopics have dedicated textbooks, e.g. Laura Kallmeyer's *Parsing Beyond Context-Free Grammars*, *Categorial Grammar* by Glyn Morrill, *The Logic of Categorial Grammars* by Richard Moot and Christian Retoré, or my colleague Jeff Heinz's book *Grammatical Inference for Computational Linguistics*, co-authored with Colin de La Higuera and Menno van Zaanen. I have happily used them in special topics courses, but they proved difficult to sample from for a broad introduction course like Computational Linguistics 2.

I am sure I have forgotten a few others, all of them deserving of a shout-out here. But the bottom-line is that none fit my vision of proof-based computational linguistics as a way of studying language. So here we are.

How?

Computational linguistics is a very interdisciplinary field, and this is also reflected in the potential readership for a book like this: full-fledged theoretical linguists, computer scientists with no background in linguistics at all, or cognitive scientists that care about language only to the extent that it reveals deeper principles of human cognition. It is impossible to serve all of them equally well, so I decided to keep the focus on what would benefit and interest linguists the most. But I did make minor additions to include other groups — that is why the linguistically trained reader will sometimes encounter brief sections on very basic material they obviously know about already. I hope they won't feel patronized.

Even the core audience of linguistics grad student is far from homogeneous, though. Depending on which classes they have taken before, their mathematics and programming background may range from non-existant to far above average. For this reason each unit of the textbook adopts the non-linear design indicated in Fig. 1. This should allow readers (and instructors) to tailor the book to their specific background.

So what do I read?

Every reader should be able to complete a unit's main path from the introduction to the summary. While some math is unavoidable even in those general sections, the more demanding parts are put in a separate section. All the math that is required to follow the main discussion is explained in separate background boxes:

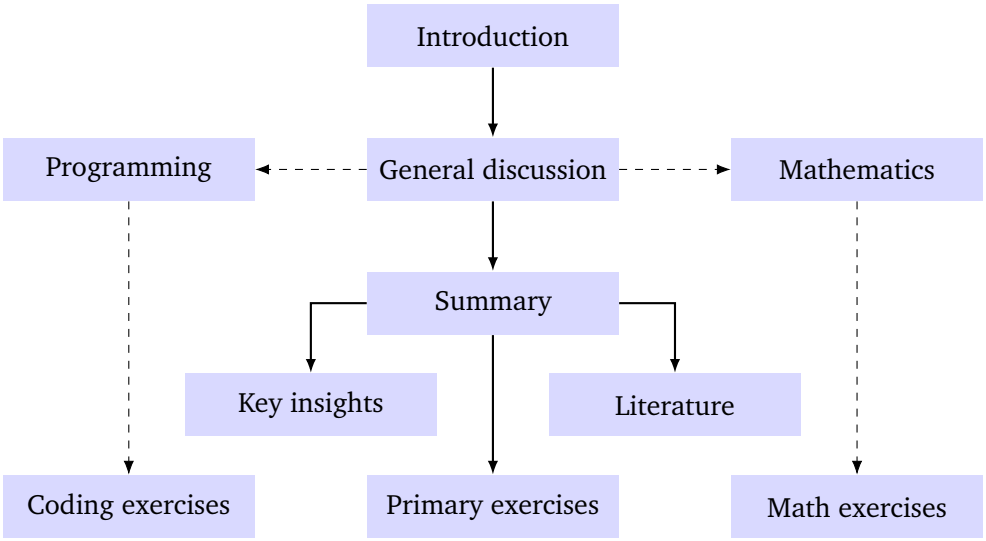

Figure 1: General structure of each unit; dashed paths are optional

Background

These boxes occur throughout the text and introduce basic concepts of mathematics and computer science.

As can be seen in Fig. 1, the book also includes optional discussions of programming issues. No, this is not a programming book. But it is often instructive to analyze how exactly formal ideas can be translated into concrete code. In particular, issues that linguists care about may appear under an entirely new light when one considers the concrete implementation challenges they pose. I decided to go with Python instead of pseudo code for these sections, simply because most students nowadays are more likely to have previous experience with Python than with reading pseudo code. To further increase the approachability of the code sections, I have adopted a beginner-friendly coding style. I err very much on the verbose side with line comments and docstrings (which document the purpose and overall design of specific pieces of code). I also make a deliberate effort to avoid advanced techniques such as decorators and generators. Error handling, which is very important for production-level software, is completely omitted. Similarly, efficiency is considered only to the extent that it illustrates a key pedagogical point. Quite simply, the code sections are of little interest on their own and are designed only to support the theoretical concepts covered in each unit.

Depending on which parts of a unit the reader decides to skip, the book can range from a light-weight introduction that conveys the key intuitions to a very formal *tour de force*.

No, which units should I read?

Why, all of them! There are several shorter paths through the book, but you'll have to figure out on your own which one suits your needs. All I can do is provide a Hasse diagram to track the dependencies between the units.

fixme: include diagram for whole book

Are there exercises? Are there solutions?

Exercises are included without solution so that they can be reused by instructors in their course. Each exercise is assigned one of four difficulty levels, indicated by asterisks:

Exercise 0.1 The lowest difficulty. Every student should be able to solve this by simply applying a key technique of the unit in a mechanical fashion.

Exercise 0.2* Requires some effort and creativity. The student cannot simply follow a recipe from the unit but has to build on their understanding of the material to come up with a solution of their own.

Exercise 0.3** Very difficult for students, and instructors might have to sit down for a minute or two to figure out the general strategy. Writing up the full solution might take quite a while, and/or the solution involves a lot of lateral, out-of-the-box thinking.

Exercise 0.4*** Open research problem. If somebody figures out the answer, they should write a paper about it. Or send me the solution and I'll handle the tedious paper publishing part for you.

What's next?

You tell me. Go find yourself a problem and solve it. If you need inspiration, pick some of the papers that are mentioned in the literature discussion sections — you should be able to follow along fairly well after having read this book. Maybe you're doing fieldwork on an understudied language that has a surprisingly complex phenomenon. You might feel the urge to explore the formal properties of a specific linguistic formalism that's dear to your heart but wasn't covered here. Perhaps you'll go and prove an open conjecture, or you find a crucial mistake in a published proof. Maybe, just maybe, you'll use an idea sketched in this book and expand it into a piece of technology that will make you millions (ideally in a currency where that is an above-average amount of money). You're the next generation. Surprise me. Surprise yourself.

Contents

Contents (detailed)