Reading for graduate students

These recommendations are sorted from general to specific: from resources for any kind of graduate student to resources for graduate students in my own field, static analysis.

My criteria for inclusion in this list are readability, self-containment, brevity and coverage.

If you're a grad student at Utah, feel free to come by my office and borrow anything below for a day or two.

Update: Computer scientists, you might also be interested in my post on what every CS major should know.

Jump to

For grad students in any field

Resources for writing

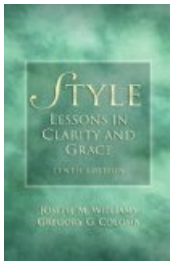
Writing is the default activity in graduate school.

A discovery isn't a discovery unless you can communicate that discovery.

A lot of academic writing is horrible, and it tends to be horrible in multiple ways: presentation, ordering, clarity, style, and sometimes even grammar and punctuation. I've written my fair share of unreadable papers, but writing better is something I've begun to take seriously.

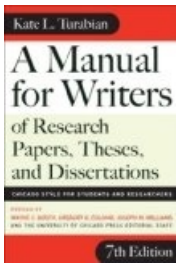
Better writing makes peer reviewers inclined to invest time in it.

On writing style, Style: Lessons in Clarity and Grace simply nails it.



The Chicago Manual of Style is an indispensable reference:

I didn't find A Manual for Writers of Research Papers, Theses and Dissertations until after my defense, but it is relevant to any kind of academic or technical writing. It answered long-held questions I had about issues like the use of the passive voice and the use of first-person pronouns in academic writing.

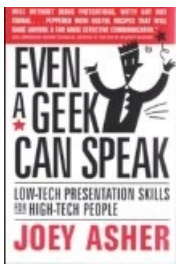


Like the Chicago book, it's a superb reference tome.

Resources for presenting (yourself)

Graduate students can't avoid giving presentations.

Once again, most academics give awful presentations. When it comes to giving presentations on technical topics, Even a Geek Can Speak will make every presentation you give better at the cost of just one afternoon's reading.

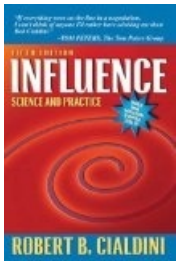


This book covers every aspect of a presentation, from purpose and audience to structuring and slides. It even contains effective techniques for dealing with nervousness about public speaking. I bought this book after I botched my first job talk, and it really turned things around for me.

Simon Peyton Jones, in addition to being a brilliant scientist, is also a gifted public speaker. Simon's advice on how to give a good research talk (co-offered by John Hughes and John Launchbury) should be required reading for all graduate students. Simon offers advice on the other non-technical research skills as well.

Robert Cialdini's Influence is a modern classic in the art of persuasion, with lots of evidence and amusing anecdotes from psychology to back it up. I read this book over my wife's shoulder while she was reading it for her M.B.A.

It's a fun read, and it will make you reconsider how you're conveying your message to your audience. Small tweaks can make a big difference.



A good remote

Though not a text, I highly recommend a *good* presentation remote. It's aggravating to have to give a talk no more than arm's length from your laptop. It just reinforces bad presentation body language. From extensive field testing, my recommendation on remotes is the Kensington.

It's small and well-designed; and it runs on easy-to-find AAA batteries.

This remote has actually won industrial design awards from IDSA.

It deserves them.

Practice

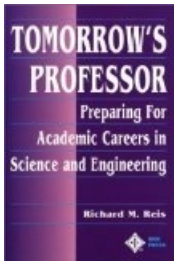
Ultimately, if you want to get good at public speaking, you have to practice. In my experience, it gets easier every time you do it. So, sign up for classes or seminars which will force you to give a presentation. And, don't just give the bare minimum: use the resources above and commit to giving the best presentation that you can. Your public reputation as a scientist is determined to a large degree by the quality of your talks.

For near-terminal grad students

The academic job hunt is a brutal process. Your first year as an assistant professor is busy. Being prepared helps. Fortunately, there are some books.

Tomorrow's Professor should be in every grad student's hands at least three years before they graduate. It explains the graduate-student-to-junior-faculty metamorphosis with a lot of examples and details.

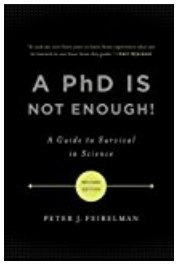
Most importantly, it covers construction of the standard materials required to perform an academic job search (cover letter, *curriculum vitæ*, research statement and teaching statement).



As a Ph.D. student, it feels at times like getting your Ph.D. will be the pinnacle of your existence; the closer you get finishing your Ph.D., the more a Ph.D. seems akin to acquiring mutant powers.

In fact, after you get your Ph.D., nothing changes. And yet, everything changes. A PhD Is Not Enough!: A Guide to Survival in Science is well-titled. It explains why getting your Ph.D. is only the *start* of your career.

It also contains cautionary tales of how *not* to begin your scientific career.



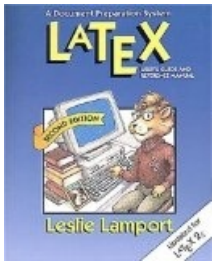
For graduate students in the sciences

If you're a graduate student in science, math or engineering, you will be writing technical papers with a lot of formal mathematics.

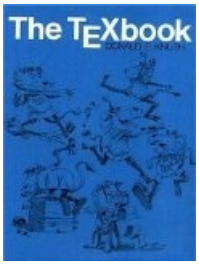
The *lingua franca* of the academic publishing community is LaTeX. LaTeX makes it possible for the motivated scientist to typeset technical documents so beautifully that I would call them art.

(*Possible* is the key word. There is some ugly LaTeX out there.)

When you're starting out with LaTeX, Leslie Lamport's LaTeX book covers all the basics, and it makes a good reference for all of the common things you'd like to do in LaTeX.

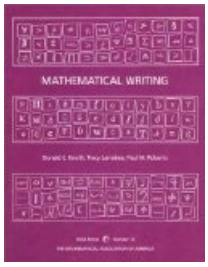


LaTeX, as it turns out, is a deep rabbit hole. (It's Turing-complete.) When you're ready for your black belt in TeX-fu, Donald Knuth's TeXbook is how you get there.



This is *not* an introductory book. This is for hard-core TeX users.

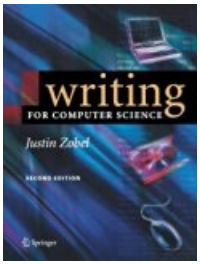
Donald Knuth once taught a class on mathematical writing, the result of which has been distilled in book form [pdf].



The first section of the book is a concise listing of the major *dos* and *don'ts*.

For graduate students in computer science

Justin Zobel's Writing for Computer Science is the "missing chapter" for computer scientists in Kate Turabian's Manual for Writers of Research Papers.

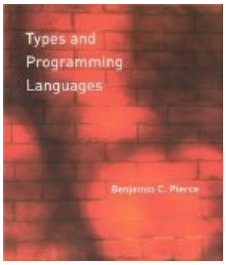


For computer science more generally, refer to my post on what every CS major should know.

For graduate students in programming languages

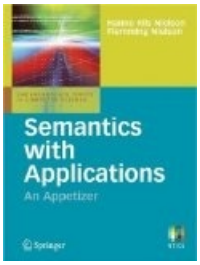
It's difficult to find a good book to get started in programming languages. Part of the problem is that the field has become so vast that no text can cover the entire field. The other part of the problem is that very few texts and papers are written for the introductory reader.

Benjamin Pierce's Types and Programming Languages really stands out.



It's a comprehensive, readable introduction to both λ-calculus *and* type theory. At the same time, the book holds up well as a reference for advanced research in the field.

The Nielsons' book Semantics with Applications [ps] [pdf] [course notes] is almost a perfect introduction to formal semantics. (From looking at the table of contents on amazon, the newer edition looks much better than the edition I have.) It gives a detailed account of the three major semantic paradigms: denotational, axiomatic and operational. A section on applying semantics to static analysis provides a nice gateway to the field. The principal strength of this book is its coverage of semantics for imperative languages due to its use and extension of the same While-based language throughout.



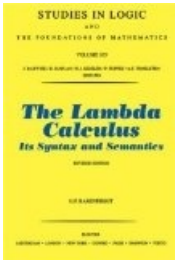
The exact same semantic techniques can model functional languages, but this is not covered by the book.

Shriram Krishnamurthi's Programming Languages: Application and Interpretation is also a solid introduction to the field, and it's *free*! This book works well for introducing new students to programming languages in part because it uses Scheme and S-Expression notation, which prevents syntax from distracting away from the core issue of semantics. Perhaps more importantly, S-Expression notation means it's very easy to fire up an interpreter and just try things out.

Mitch Wand and Dan Friedman's Essentials of Programming Languages is also a good introduction to programming languages and a great reference for formal semantics. Like Shriram's, this book also uses Scheme, so it inherits the same advantages.

In addition to being great scientists, Mitch and Dan are all-around good people, and in their writing, that comes across as precise yet friendly and approachable prose. I get the sense they actually want the reader to understand the material.

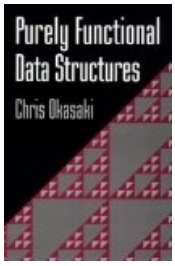
Barendregt's classic The Lambda Calculus is an encyclopedia of the type-free λ-calculus. There's a lot in here on long-forgotten aspects of the λ-calculus, including its role in logic and foundations and its relationship to topology. It's almost hard to believe the λ-calculus was so well studied *before* it was foundational to the theory of modern programming languages.



The book makes for dense reading. I would not recommend it as an introductory text, but as a handbook during research, it is indispensible. It's a superb complement to Benjamin Pierce's Types and Programming Languages.

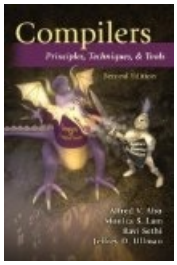
If writing high-performance, correct compilers for functional languages is a goal of yours, Greg Morrisett's Ph.D. thesis is a good read. Greg has done and continues to do a lot of great work, so I would actually recommend any of his papers to more advanced graduate students in programming languages. Greg tends to develop powerful machinery as a byproduct of reaching his underlying goal; I've found that machinery can often be adapted to other research problems.

Much research in programming languages gets implemented in (and for) Standard ML, OCaml and Haskell. One of the major challenges (and eventual joys) of using these languages for those more accustomed to imperative languages is learning to use functional data structures effectively. Chris Okasaki's Purely Functional Data Structures provides a full treatment of frequently used data structures in their purely functional form.

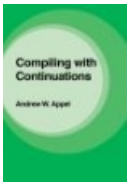


For graduate students in compilers

The recently revamped classic, "the dragon book," is a *great* reference for implementors that want to implement the standard analyses and optimizations that an industrial-strength compiler like GCC has:



In the 1980's, advanced compilers used continuation-passing-style (CPS) as their internal representation. In the 1990's and early 2000's, administrative-normal form (ANF) was in vogue, in part because it didn't require compiler writers to understand continuations. Lately, however, I see more and more rediscovering the unmatched power and simplicity of CPS. Andrew Appel's Compiling with Continuations was written at the zenith of the first CPS epoch, which makes it an unbeatable reference on CPS-based compilation even today:



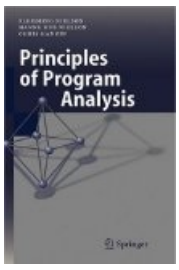
For graduate students in static analysis

Patrick and Rhadia Cousots' original paper on abstract interpretation, which set an entire paradigm in motion, is a good read even after becoming acquainted with the field. Patrick and Rhadia have shown themselves to be great minds of our time in static analyis, which means that their writing is excellent, but it is often unapproachable to casual readers given its level of sophistication. Their original paper is different: because they're starting a field, they cannot and do not assume any background knowledge.

My start in programming languages came from reading the first three chapters of my advisor's dissertation. Olin Shivers is a gifted writer, and it's always a pleasure to read his work. His dissertation covers *k*-CFA, an analytic platform from which a number of Ph.D.'s can be launched. In his dissertation, you get to see concepts like domain theory, denotational semantics and abstract interpretation in use. It's probably one of the last major works of that era to use denotational semantics.

This dissertation is best read after or during an intro-level programming languages course which covers formal semantics, but it is still a remarkably self-contained piece of work.

There are not a lot of books written on static analysis. The Nielsons/Hankin book Principles of Program Analysis is fairly comprehensive.



Those with some background in formal semantics should be able to use this book as a reference during their research.

In my static analysis seminar, I use a reading list of classic papers:

1. A lattice-theoretical fixpoint theorem and its applications. Tarski. 1955.
2. Assigning meaning to programs. Floyd. 1967.
3. A unified approach to global program optimization. Kildall. 1973.
4. Abstract interpretation: a unified lattice model for static analysis of

programs by construction or approximation of fixpoints. Cousot and Cousot. 1977.

5. Systematic Design of Program Analysis Frameworks. Cousot and Cousot. 1979.
6. Control-flow analysis in Scheme. Shivers. 1988.
7. Efficiently computing static single assignment form and the control dependence graph. Cytron, Ferrante, Rosen, Wegman, Zadeck. 1991.
8. Points-to analysis in almost linear time. Steensgaard. 1996.
9. Parametric shape analysis via 3-valued logic. Sagiv, Reps and Wilhelm. 2002.

Related posts

- Tips for defending a Ph.D.
- HOWTO: Respond to peer reviews
- Tips for work-life balance
- 12 resolutions for grad students
- HOWTO: Peer review scientific work
- Electric meat
- HOWTO: Get a great letter of recommendation
- Boost productivity: Cripple your technology
- HOWTO: Send and reply to email
- Classroom Fortress: Nine Kinds of Students
- The 5+5 Commandments of a Ph.D.
- 10 tips for academic talks
- 10 reasons Ph.D. students fail
- 6 tips for low-cost academic blogging
- Get The Illustrated Guide in print; fund Ph.D. students; save lives
- The illustrated guide to a Ph.D.
- A Ph.D. thesis proposal is a contract
- 3 qualities of successful Ph.D. students
- HOWTO: Get in to grad school
- Academic job hunt advice

[article index] [email me] [@mattmight] [+mattmight] [rss]

Latest: HOWTO: Get tenure
Next: Preventing and healing repetitive strain injury (RSI)
Prev: Academic job hunt advice
Rand: Least resistance weight loss