Check for updates

EDUCATION FORUM

Where Have all the Women Gone?[1]

Where *have* all of the women gone? Some of us can remember years when computer science courses were quite well-populated with women. Indeed, one of our faculty at Montana State University is certain that the enrollment was about 40% women here at one point early in the '80s. Now we are lucky to have a handful of women in our introductory courses, and only a few in the upper division courses. Apparently, this phenomenon is mirrored across the country: a quick glance at the last three SIGCSE Conference proceedings reveals paper and panel titles such as "Where Have all the Women Gone and How Do We Keep Them from Going?" (1995), "Gender Issues" (1996), and "Gender Imbalance in Computer Science Programs, Etiology and Amelioration[2]: Views from U.S. Campuses and Elsewhere" (1997). In the 1997 panel just cited, the moderator states

> "...I've watched with growing alarm the drop in enrollment in our Computer Science programs from a ratio of 1:1 in 1983, to a current ratio of 5:1. The lack of interest of female students in Computer Science is still a very serious problem..."

What has happened? What can be done about it? Should anything be done about it?

This phenomenon has been a curiousity of mine for a long time, but it was only recently that it moved beyond reminiscing about the good old days over coffee—contemplating what has gone wrong—to first steps towards actual involvement. An NSF project meant to address the larger issue of women in all sciences was recently awarded to MSU entitled *Science and Engineering for All: Opening the Door for Rural Women*[3]. One of the activities of the project was a summer faculty institute, to which invitations were sent out to all science and engineering faculty, invitations that came with a tasty little honorarium as bait. I bit. And now it appears that I'm hooked.

Actually, my initial interest in the institute had less to do with the honorarium than with the animation projects our group is pursuing. We had been wondering just what we could do to make our projects gender sensitive given the low numbers of women in the discipline. We did get some helpful ideas for our projects, but we got more. For example, it was interesting, albeit disheartening, to hear firsthand of experiences of some women scientists regarding the gender discrimination they have faced during their careers as well as some of the blatant sexual harassment others have endured. In fact, for a number of years as a young assistant professor I was relatively ignorant of such problems. It never entered my mind that well-educated men would treat women colleagues with the crassness one came to expect in the Army. But, in retrospect, I have had to admit a memory of a few older male professors who evinced some disdain for women as peers and who seemed insensitive to them (witnessed by the occasional crude joke in faculty meetings). And not so long ago, we had to take disciplinary action against some of our male students who were using a programming trick to force the display of an explicit picture[4] on the monitor of a female student working at another station.

---

[1] ...long time passing...

[2] ...large words that translate into "something has caused things to get out of whack, so let's beat on the problem for a while to see if we can pound things back into whack."

[3] ...hmmm, I suppose that this is a dead giveaway that the PIs are MSU women scientists; I'm not sure any of us men could have gotten away with a title that involved opening doors for any woman, let alone the highly capable women of rural Montana...

[4] Well, I guess all pictures are explicit. Let's see...oh, you know what I mean!

Of course, bringing gender awareness and sensitivity into everyday affairs has also had its awkward moments. I remember not so long ago the pendulum swinging so far to the side of political correctness that speech became an inherent impediment for nearly everyone. Here's a good example. On my first search committee assignment at Montana State University, I was involved in a concerted effort to help establish the Department. I was really hoping that we could attract a female candidate to one of our open positions and was ecstatic to see one day an application from a young lady whose name I recognized (and who has since made a name recognizable by nearly everyone in the computer science theory community). "We just received an application from an outstanding girl!" I said. A female colleague on the committee with whom I had gotten along very well since joining the faculty said, "Did you say we got an application from a *girl*?" Noticing the emphasis she placed on *girl* I charged ahead in my enthusiasm: "Yup, a girl from X[5] University." "Did you say, *GIRL*?" she asked once more, giving me a piercing stare that clearly indicated that she, too, could not believe our evident luck in attracting a female candidate from such a prestigious institution. I was beginning to think, though, that I must be slurring my words. Later, an older and wiser faculty member took me aside to inform me that even though I regularly used the word "guy" in reference to our male candidates, the appropriate word when referring to a female candidate—at least as far as this committee was concerned—was "woman[6]."

But really, our problem with the number of women in the curriculum has little, if anything, to do with gender discrimination, sexual harassment, or even speech impediments. The fact is, the women[7] aren't even appearing in the entry-level class to be discriminated against, harassed, or denigrated with unfortunately chosen words. So, while I appreciated the information learned in the summer institute about how to make a classroom "female friendly," or, as the moderators were quick to point out, "just plain friendly," my question was more fundamental: What's happened to the pipeline? Somewhere as far back as grade school or junior high, girls are learning that science isn't cool. My own daughters, both quite capable and bright, made a shift sometime during middle school from thinking that a career in science (even computer science!) was attractive, to "ugh."

In fact, I have often pointed my students with pride to the fact that computer science has no historical bias towards women, as some engineering fields have had. I personally have heard of no cases where women have consistently received lower starting salaries than men, or where they were not paid equivalently for equivalent work in mainstream computing companies. I have also noted, again with satisfaction, that among the prominent names in the computer science theory community are those of many women. I have certainly been aware of no gender discrimination in theoretical computer science[8]. In spite of these successes, however, the pipeline into the university barely dribbles women into the computer science curriculum at the present.

So, what can we do? Well, I don't know. As noted, the problem has not gone unnoticed or unstudied, although I know of no one who has a clear grasp of it. For my part, I think—as noted earlier—it has much to do with what is deemed cool at a certain stage of life. In that case, we need to make computer science cool, or at least attractive, to girls at an early age. The NSF grant alluded to at the beginning is quite cleverly conceived in this regard. For three years, it will gather university faculty, four year college faculty, tribal college faculty (the program has a purposeful focus on Native Americans as well), and high school teachers for summer institutes, and then provide a competition for a number of mini-grants to try some of the ideas generated by the

[5]We must protect the innocent, you know (and to be honest, I don't *remember* which university).

[6]One of the benefits of having a PhD degree and being trained in the ways of logic, I have found, is that it takes me only two or three days of meditation and reflection to figure out the blatantly obvious.

[7]See? I've learned!

[8]But, sigh, I have been known to be blind at times; let me know if I'm wrong!

institutes. This is a good way to try many different ideas for relatively little cost. And the ideas vary widely, from taking girls on field trips to see women scientists in action to taking the action to the girls by way of a funky show-and-tell traveling van crammed full of women scientists and whiz-bang experiments (does anyone remember the traveling Bookmobiles?).

Our idea, conceived by me and my long time research associate, Frances Goosey, is more modest, and yet it has the potential to reach nearly every girl in the state (and beyond): A computer science Web page. The ideas are too involved to present here in detail, but bootstrapping from our work in animation, we envision a page that allows kids to explore some of the fundamental aspects of computer science in an intuitive, fun, and informative way. The page would also allow the kids to interact with scientists through such links as *Ask a Scientist* (an idea we picked up from someone else) in which a student could pose a question that was read and responded to by someone at the university. Special links for girls would give snapshots of women who have made an impact on the science, such as Ada Lovelace, Grace Hopper, and others. Current women scientists would also be featured, perhaps with pictures (if they'll let us post them!) that show them both in current settings and at an age similar to the target age of the girls using the page. The *Ask a Scientist* link idea would also be expanded to include links to *Ask a Student*, *Ask a Woman Student*, *Ask a Woman Scientist*, *Ask a Native American Student*, and so forth. Parents also play a crucial role in how girls perceive future careers, so the Web page would also have a link for parents to follow for information and personal contact.

Of course, the page would have links to other relevant sites, as well. For example, MSU has a special page for Native American students that would be very beneficial for Native American pre-college students accessing our page. On the national level, there is an official ACM effort already underway to reach out to girls that provides valuable resources. It's The Ada[9] Project (TAP) and can be found at

```
http://www.cs.yale.edu/HTML/YALE/CS/HyPlans/tap/purpose.html
```

And really, the Web is just the beginning. At MSU, and many other institutions, it is now possible to deliver interactive two-way video presentations or courses, a medium we are sure to exploit in this regard. In any case it sounds like a fun project. If you are interested in joining in a collaborative effort, we'd be glad to hear from you.

Before we quit here, though, there is one nagging question that begs to be posed. Is all of this worth the effort? If women entering the University make free will decisions about which fields to study, who are we to try to artificially bolster the number of women pursuing careers in science? Somehow, I guess we have the gut feeling that the reason for the low numbers of female students entering science curricula is one of image. Projects, such as our Web page idea, that provide a better image of the discipline in an interactive and fun way, may provide young girls with more motivation to enter science fields, still of their own free will. I guess we'll see what happens.

Textbooks

This time there are three textbooks that have been previewed. One is *Algorithmic Number Theory* by **Eric Bach** and **Jeffrey Shallit**. Then there are two compiler books (quite a number of compiler books have been introduced in the past few years). One is *Modern Compiler Implementation in C* by **Andrew W. Appel** and the other is *Compilers & Compiler Generators* by **P. D. Terry**.

[9]Having no reference to the programming language by that name, but to the woman after whom the programming language was named.

These are books that just happened to cross my desk. Remember, if you are an author and would like to have your new book of interest to the theory community previewed here, please contact me.

Logout

From Bozeman, where the heavy snows from a long winter still leave their traces amongst a silent cacophony of riotously colored and fragrantly scented wild alpine flowers, the offspring of the Rockies and a long, wet spring...

Rocky Ross		
Computer Science Department	E-mail:	ross@cs.montana.edu
Montana State University	URL:	http://www.cs.montana.edu/~ross
Bozeman, MT 59717	Phone:	(406) 994-4804

Algorithmic Number Theory
Volume 1—Efficient Algorithms

Eric Bach and Jeffrey Shallit

The MIT Press, 1996
ISBN: 0-262-02405-5

Preface (Abridged)

This is the first volume of a projected two-volume set on algorithmic number theory, the design and analysis of algorithms for problems from the theory of numbers. This volume focuses primarily on those problems from number theory that admit relatively efficient solutions. The second volume will largely focus on problems for which efficient algorithms are not known, and applications thereof.

Prerequisites

We hope that the material in this book will be useful for readers at many levels, from the beginning graduate student to experts in the area. The early chapters assume that the reader is familiar with the topics in an undergraduate algebra course: groups, rings, and fields. Later chapters assume some familiarity with Galois theory. A good text is Herstein.

We assume the reader is familiar with the analysis of algorithms, as treated, for example, in Aho, Hopcroft, and Ullman, or Cormen, Leiserson and Rivest and with the language of complexity theory, as treated, for example, in Garey and Johnson. However, we have tried to make our discussion of complexity theory relatively self-contained.

Finally, we assume the reader has had the equivalent of an undergraduate course in probability theory. A good text in this area is Feller. For some sections a knowledge of more advanced mathematics will be useful.

Goals of This Book

As stated above, this book discusses the current state of the art in algorithmic number theory. This book is *not* an elementary number theory textbook, and so we frequently do not give detailed proofs of results whose central focus is not computational. Choosing otherwise would have made this book twice as long.

However, we firmly believe that a mere list of proved theorems is not particularly enlightening or useful. Therefore, we endeavor to prove as much as is feasible. Some proofs are left to the reader as exercises, with outlines suggested in Appendix A. And every theorem which is not proved in the text or left as an exercise has a reference in the "Notes" section that appears at the end of each chapter.

This book is also not intended solely as a practical guide for those who wish to implement number-theoretic algorithms. The variety of architectures of modern machines, and the profusion of languages and operating systems, frequently make specific remarks on running times useless. On the other hand, computational theory without the influence of practice seems artificial. Thus the "Notes" section of each chapter contains remarks on practical implementations of the algorithms discussed.

We might mention several other books that cover approximately the same material. The first was Knuth [1969]; a second edition appeared as Knuth [1981]. More recently, there have appeared texts such as Riesel [1985]; Kranakis [1986]; Koblitz [1987]; Bressoud [1989]; H. Cohen [1993]; and Zippel [1993]. We hope that our book will be useful in addition to these, because of its somewhat different emphasis.

We restrict our attention in this book to those concepts and algorithms that relate to what we see as algorithmic number theory. Thus, we do not discuss some very interesting topics that seem more algebraic than number-theoretic; algorithms on finite groups, Gröbner bases, etc. We also restrict our attention to problems from *elementary* number theory: e.g., primality testing, factorization, discrete logarithm, etc. Thus there is virtually no overlap between the present work and the books of Zimmer [1972] and Pohst and Zassenhaous [1089].

In writing this book, we have found many opportunities to extend or refine known results. This is particularly true with regard to algorithms, many of whose running times were not completely worked out in the literature. As an alternative to listing all such improvements (which is clearly impractical), we give references to earlier results, such as are known to us, in the appropriate places.

Table of Contents

1 Introduction 1

1.1 Number Theory and Complexity 1 / 1.2 Number Theory and Computation: A Brief History 4 / 1.3 Condensed History of the Theory of Computation 11 / 1.4 Notes on Chapter 1 13

2 Fundamentals of Number Theory 19

2.1 Notation, Definitions and Some Computational Problems 19 / 2.2 More Definitions 22 / 2.3 Multiplicative Functions and Möbius Inversion 23 / 2.4 Notation: Big-O, Little-o, Big-Omega, Big-Theta 25 / 2.5 Abel's Identity and Euler's Summation Formula 25 / 2.6 Asymptotic Integration 27 / 2.7 Estimating Sums over Primes 28 / 2.8 Basic Concepts of Abstract Algebra 29 / 2.9 Exercises 34 / 2.10 Notes on Chapter 2 37

3 A Survey of Complexity Theory 41

3.1 Notation 41 / 3.2 The Notion of "Step" 41 / 3.3 The Language Classes 44 / 3.4 Reductions and $\mathcal{NP}$-Completeness 47 / 3.5 Randomized Complexity Classes 50 / 3.6 A Formal Computational Model 52 / 3.7 Other Resources 55 / 3.8 Parallel Complexity Classes 57 / 3.9 Exercises 59 / 3.10 Notes on Chapter 3 63

4 The Greatest Common Divisor 67

4.1 The Euclidean Algorithm 67 / 4.2 The Euclidean Algorithm: Worst-Case Analysis 68 / 4.3 The Extended Euclidean Algorithm 70 / 4.4 The Euclidean Algorithm and Continuants 73 / 4.5 Continued Fractions 75 / 4.6 The Least-Remainder Euclidean Algorithm 79 / 4.7 The Binary gcd Algorithm 82 / 4.8 Constructing a gcd-Free Basis 84 / 4.9 Exercises 96 / 4.10 Notes on Chapter 4

5 Computing in Z/(n) 101

5.1 Basics 101 / 5.2 Addition, Subtraction, Multiplication 101 / 5.3 Multiplicative Inverse 102 / 5.4 The Power Algorithm 102 / 5.5 The Chinese Remainder Theorem 104 / 5.6 The Multiplicative Structure of $(\mathbf{Z}/(n))^*$ 108 / 5.7 Quadratic Residues 109 / 5.8 The Legendre Symbol 110 / 5.9 The Jacobi Symbol 111 / 5.10 Exercises 114 / 5.11 Notes on Chapter 5 120

6 Finite Fields 125

6.1 Basics 125 / 6.2 The Euclidean Algorithm 127 / 6.3 Continued Fractions 130 / 6.4 Computing in $k[X]/(f)$ 132 / 6.5 Galois Theory 133 / 6.6 The Structure of $k[X]/(f)$ 136 / 6.7 Characters 141 / 6.8 Exercises 143 / 6.9 Notes on Chapter 6 148

7 Solving Equations over Finite Fields 155

7.1 Square Roots: Group-Theoretic Methods 155 / 7.2 Square Roots: Field-Theoretic Methods 157 / 7.3 Computing d-th Roots 160 / 7.4 Polynomial Factoring Algorithms 163 / 7.5 Other Results on Polynomial Factoring 168 / 7.6 Synthesis of Finite Fields 171 / 7.7 Hensel's Lemma 173 / 7.8 complexity-Theoretic Results 177 / 7.9 Exercises 188 / 7.10 Notes on Chapter 7 194

8 Prime Numbers: Facts and Heuristics 203

8.1 Some History 204 / 8.2 The Density of Primes 206 / 8.3 Sharp Estimates and the Riemann Hypothesis 211 / 8.4 Primes in Arithmetic Progressions and the ERH 215 / 8.5 Applications of the ERH 217 / 8.6 Other Conjectures about Primes 224 / 8.7 Extensions to Algebraic Numbers 227 / 8.8 Some Useful Explicit Estimates 233 / 8.9 Exercises 236 / 8.10 Notes on Chapter 8 245

9 Prime Numbers: Basic Algorithms 265

9.1 Primality Proofs and Fermat's Theorem 266 / 9.2 Primality Tests for Numbers of Special Forms 272 / 9.3 Pseudoprimes and Carmichael Numbers 275 / 9.4 Probabilistic Primality Tests 278 / 9.5 ERH-Based Methods 283 / 9.6 Primality Testing Using Algebraic Number Theory 285 / 9.7 Generation of "Random" Primes 293 / 9.8 Prime Number Sieves 295 / 9.9 Computing $\pi(x)$ and p_n 299 / 9.10 Exercises 303 / 9.11 Notes on Chapter 9 308

Modern Compiler Implementation in C Andrew W. Appel
Princeton University

Cambridge University Press, 1997
ISBN: 0-521-58653-4

Preface (Abridged)

Over the past decade, there have been several shifts in the way compilers are built. New kinds of programming languages are being used: object-oriented languages with dynamic methods, functional languages with nested scope and first-class function closures; and many of these languages require garbage collection. New machines

have large register sets and a high penalty for memory access, and can often run much faster with compiler assistance in scheduling instructions and managing instructions and data for cache locality.

This book is intended as a textbook for a one-semester or two-quarter course in compilers. Students will see the theory behind different components of a compiler, the programming techniques used to put the theory into practice, and the interfaces used to modularize the compiler. To make the interface and programming examples clear and concrete, I have written them in the C programming language. Other editions of this book are available that use the Java and ML languages.

The "student project compiler" that I have outlined is reasonably simple, but is organized to demonstrate some important techniques that are now in common use: Abstract syntax trees to avoid tangling syntax and semantics, separation of instruction selection from register allocation, sophisticated copy propagation to allow greater flexibility to earlier phases of the compiler, and careful containment of target-machine dependencies to one module.

This book, *Modern Compiler Implementation in C: Basic Techniques*, is the preliminary edition of a more complete book to be published in 1998, entitled *Modern Compiler Implementation in C*. That book will have a more comprehensive set of exercises in each chapter, a "further reading" discussion at the end of every chapter, and another dozen chapters on advanced material not in this edition, such as parser error recovery, code-generator generators, byte-code interpreters, static single-assignment form, instruction scheduling and software pipelining, parallelization techniques, and cache-locality optimizations such as prefetching, blocking, instruction-cache layout, and branch prediction.

Exercises. Each of the chapters in Part I has a programming exercise corresponding to one module of a compiler. Unlike many "student project compilers" found in textbooks, this one has a simple but sophisticated back end, allowing good register allocation to be done after instruction selection. Software useful for the programming exercises can be found at

```
http://www.cs.princeton.edu/~appel/modern/
```

There are also pencil and paper exercises in each chapter; those marked with a star * are a bit more challenging, two-star problems are difficult but solvable, and the occasional three-star exercises are not known to have a solution.

Table of Contents

Preface ix

Part I Fundamentals of Compilation

1. Introduction 3
2. Lexical Analysis 16
3. Parsing 39
4. Abstract Syntax 80
5. Semantic Analysis 94
6. Activation Records 116
7. Translation to Intermediate Code 140
8. Basic Blocks and Trees 166
9. Instruction Selection 180
10. Liveness Analysis 206
11. Register Allocation 222
12. Putting It All Together 248

Part II Advanced Topics

13. Garbage Collection 257
14. Object-oriented Languages 283
15. Functional Programming Languages 299

16. Dataflow Analysis 333
17. Loop Optimizations 359
Appendix: Tiger Language Reference Manual 381

Compilers & Compiler Generators

An Introduction With C++

P. D. Terry
Rhodes University

Thomson Computer Press, 1997
ISBN: 1-85032-298-8

Preface (Abridged)

This book has been written to support a practically oriented course in programming language translation for senior undergraduates in Computer Science. More specifically, it is aimed at students who are probably quite competent in the art of imperative programming (for example in C++, Pascal, or Modula-2), but whose mathematics may be a little weak; students who require only a solid introduction to the subject, so as to provide them with insight into areas of language design and implementation, rather than a deluge of theory which they will probably never use again; students who will enjoy fairly extensive case studies of translators for the sorts of languages with which they are most familiar; students who need to be made aware of compiler writing tools, and to come to appreciate and know how to use them. It will hopefully also appeal to a certain class of hobbyist who wishes to know more about how translators work.

The reader is expected to have a good knowledge of programming in an imperative language and, preferably, a knowledge of data structures. The book is practically oriented, and the reader who cannot read and write code will have difficulty following quite a lot of the discussion. However, it is difficult to imagine that students taking courses in compiler construction will not have that sort of background!

There are several excellent books already extant in this field. What is intended to distinguish this one from the others is that it attempts to mix theory and practice in a disciplined way, introducing the use of attribute grammars and compiler writing tools, at the same time giving a highly practical and pragmatic development of translators of only moderate size, yet large enough to provide considerable challenge in the many exercises that are suggested.

Support Software

Appendix A gives instructions for unpacking the software provided on the diskette and installing it on a reader's computer. In the same appendix will be found the addresses of various sites on the Internet where this software (and other freely available compiler construction software) can be found in various formats. The software provided on the diskette includes:

- Emulators for the two virtual machines described in Chapter 4 (one of these is a simple accumulator-based machine, the other is a simple stack-based machine).
- The one- and two-pass assemblers for the accumulator-based machine, discussed in Chapter 6.
- A macro assembler for the accumulator-based machine, discussed in Chapter 7.
- Three executable versions of the Coco/R compiler generator[10] used in the text and described in detail in Chapter 12, along with the frame files that it needs. (The three versions produce Turbo Pascal, Modula-2, or C/C++ compilers.)

[10] A compiler generator based on L-attributed grammars.

- Complete source code for hand-crafted of each of the versions of the Clang compiler that is developed in a layered way in Chapters 14 through 18. This highly modularized code comes with an 'on the fly' code generator, and also with an alternative code generator that builds and then walks a tree representation of the intermediate code.
- Cocol grammars and support modules for the numerous case studies throughout the book that use Coco/R. These include grammars for each of the versions of the Clang compiler.
- A program for investigating the construction of minimal perfect hash functions (as discussed in Chapter 14).
- A simple demonstration of an LR parser (as discussed in Chapter 10).

Table of Contents

1. Introduction 1
2. Translator classification and structure 9
3. Compiler construction and bootstrapping 27
4. Machine emulation 35
5. Language specification 71
6. Simple assemblers 101
7. Advanced assembler features 125
8. Grammars and their classification 147
9. Deterministic top-down parsing 169
10. Parser and scanner construction 187
11. Syntax-directed translation 215
12. Using Coco/R — overview 233
13. Using Coco/R — case studies 257
14. A simple compiler — the front end 285
15. A simple compiler — the back end 319
16. Simple block structure 355
17. Parameters and functions 381
18. Concurrent Programming 413

Appendix A: Software resources for this book 439
Appendix B: Source code for the Clang compiler/interpreter 445
Appendix C: Cocol grammar for the Clang compiler/interpreter 499
Appendix D: Source code for a macro assembler 525
Bibliography 567
Index 573