The Book Review Column[1]

by William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742
email: `gasarch@cs.umd.edu`

Welcome to the Book Reviews Column. We hope to bring you at least two reviews of books every month. In this column three books are reviewed.

1. **Distributed Computing** by Attiya and Welch, reviewed by Maurice Herlihy. This is a textbook aimed at grad students, and advanced undergrads, for a course on Distributed Computing.
2. **Hilbert's Tenth Problem** , by Yuri Matiyasevich reviewed by Randall Pruim. This book provides a systematic treatment of the solutions to Hilbert's Tenth Problem, including historical notes and additional applications of the techniques used in the proof.
3. **Lambda Calculi: A Guide for Computer Scientists** by Chris Hankin, reviewed by Christopher League. This is a book on Lambda calculus, for grad students and advanced undergrads, aimed at computer scientists (as opposed to logicians).
4. **Systems that Learn (second edition)** by Jain, Osherson, Royer, Sharma, reviewed by Carl Smith. This is a book on Inductive Inference, which is the study of models of learning with a computability angle. The book starts from the beginning, and hence could be used a text in an intro graduate course, or as a supplment to a course on PAC and other learning models.

I am looking for reviewers for the following books

If you want a FREE copy of one of these books in exchange for a review, then email me at gasarchcs.umd.edu If you want more information about any of these books, again, feel free to email me.

1. *Data Structures and Algorithms in Java* by Goodrich and Tamassia.
2. *Modern Graph Theory* by Bollobas.
3. *Control Flow Semantics* by Jaco de Bakker and Erik de Vink
4. *Complexity and Real Computation* by Blum, Cucker, Shub, and Smale.
5. *The Functional Approach to Programming* by Cousineau and Mauny.
6. *Computational Geometry* by de Berg, van Kreveld, Overmars, and Schwarzkpf
7. *Term Rewriting and all that* by Baader and Nipkow

The following are DIMACS workshop books which are collections of articles on the topic in the title.

1. Randomization Methods in Algorithm Design.

[1]© William Gasarch, 2000.

2. Microsurveys in Discrete Probability.
3. Mathematical Support of Molecular Biology.
4. Multichannel Optical Networks: Theory and Practice.
5. Networks in Distributed Computing.
6. Advances in Switching Networks.
7. Network Design: Connectivity and Facilities Location.
8. External Memory Algorithms

Review of[2]

Distributed Computing
Authors: Attiya and Welch
Publisher: McGraw Hill, 1998
Paperback: ISBN 0077093526, $75.00
464 Pages

Reviewer: Maurice Herlihy
Department of Computer Science
Brown University

Last Spring, I used this book as the text for an upper-division undergraduate "Introduction to Distributed Computing" course, so I am reviewing this book primarily as a textbook.

This text provides a broad-ranging introduction to the modern theory of Distributed Computing. It provides background sufficient to introduce advanced undergraduates or beginning graduate students to the kind of research that typically appears in symposiums such as the ACM Principles of Distributed Computing (PODC) or the International Symposium on DIStributed Computing (DISC).

The first section, entitled "Fundamentals", introduces a number of classical problems from distributed computing, familiarizing the student with the kinds of problems, models, and techniques needed to reason about distributed problems. The section on basic network algorithms is a gentle introduction to a number of basic networking problems. Both the problems and solutions are simple, and clearly stated, and I found that students enjoyed this section because it built up their confidence. The section on leader election in rings introduces three themes central to distributed computing. First, the importance of models: synchronous and asynchronous rings behave quite differently. Second, even though the algorithms considered here are quite simple, they still require a non-trivial analysis. Finally, this section provides students with the first example of how to exploit asynchrony and faults to prove lower bounds. Even though the subject itself is no longer a primary focus of research, I nevertheless found leader election to be an effective way of conveying these fundamental points to my students. The section on mutual exclusion in shared memory provides another opportunity to reason about concurrent processes, and introduces various notions of fairness. For each mutual exclusion algorithm, my students typically started by asserting that it "obviously" works. After some discussion, they would be convinced that correctness was far from obvious, and after additional thought, they would decide it was indeed obvious after all.

[2]© Maurice Herlihy 2000

The sections on consensus and on causality and time are both pitched at an appropriate level for advanced undergraduates.

The second section of the book concerns how to simulate one kind of system with another. This section starts by introducing some formal machinery, and proceeds to consider how to implement broadcast and multicast, correctness conditions for shared memory, implementations of read/write variables, and "compiling" algorithms designed to tolerate one kind of failure into algorithms that tolerate other kinds of failures. I covered the multicast algorithms and correctness conditions. Because of the emphasis on formalism, and the focus on subtle distinctions among failure models, the remaining material struck me as perhaps more appropriate for first-year graduate students.

The third and final section of the book covers advanced topics. Students reacted well to the section on wait-free simulation of arbitrary objects (many were excited by applications to concurrent data structures), and to the section on asynchronously solvable systems. The section on randomization interested students with a sufficiently strong background in probability theory. Problems such as bounded timestamping elicited less enthusiasm.

The book includes a number of exercises, which were quite useful in preparing homeworks and exams. Many of the exercises have bugs, however, so it is prudent to consult the authors' errata page at

http//www.cs.technion.ac.il/~hagit/DC/errata.html

before making assignments.

It should be mentioned that this text is about fundamentals of distributed *computing*, not distributed *systems*. It does not cover deadlock detection, two-phase locking and serializability theory, distributed commitment protocols, checkpointing, practical mutual exclusion algorithms, and other theoretical topics related to distributed systems.

One mild criticism of the presentation is that the authors provide little or no explicit motivation for the models and problems considered in the text. In particular, the authors draw no distinction between problems interesting primarily as mathematical exercises, and problems interesting because they tell us something about computation as it is practiced. To pick one example, when I taught the course, I felt it important to warn students that the problem of mutual exclusion using read-write variables is a valuable theoretical exercise, training them to think rigorously about concurrency, fairness properties, and so on, but that there are fundamental architectural reasons (which every computer scientist should understand) why it makes no sense to use read-write variables for mutual exclusion in real systems. By contrast, lower bounds such as the impossibility of fault-tolerant consensus in asynchronous systems do have wide-ranging applications for real distributed systems.

To summarize, this book provides a useful and well-written introduction to distributed computing, especialy for advanced undergraduates and beginning graduate students.

Review of[3]

Hilbert's Tenth Problem
Author: Yuri Matiyasevich
Publisher: The MIT Press, 1993
Hardcover: ISBN 0-262-13295-8
$xxii + 264$ **pages**

Reviewer: Randall Pruim
Department of Mathematics and Statistics
Calvin College
Grand Rapids, MI 49506

[3]© Randall Pruim, 2000

rpruim@calvin.edu

1 Overview

In 1900, the mathematician David Hilbert gave an address in which he provided a collection of problems "that the nineteenth century left for the twentieth century to solve." [Matiyasevich, page *xix*] This address was subsequently expanded into a version that appeared as an article and has since been translated and reprinted several times (see, for example, *Mathematical Developments Arising from Hilbert's Problems*, vol. 28 of *Proceedings of Symposia in Pure Mathematics*, pp. 1–34, AMS, 1976). Of Hilbert's problems, one (the tenth) was what we now refer to as a decision problem, namely:

> Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers. [Hilbert, 1900]

A diophantine equation is specified by a multi-variate polynomial with integer coefficients $D(x_1, \dots, x_k)$ and a domain X from which a solution $x_1, \dots, x_k \in X^k$ to the equation

$$D(x_1, \dots, x_k) = 0$$

is sought. Using this notation, a more modern formulation of Hilbert's Tenth Problem is the following:

> Devise an algorithm that on input of (a code for) a diophantine equation $D(x_1, \dots, x_k)$ determines (deterministically, in finitely many steps) whether or not $D(x_1, \dots, x_k) = 0$ is solvable over the integers, that is with $X = Z$.

This problem of Hilbert led to 70 years of research that eventually culminated in its negative solution: *There is no such algorithm.* This book presents a complete, self-contained proof of this fact, as well as a number of applications and extensions of the techniques involved from the perspective of one of the key researchers involved.

2 Summary of Contents

The content of the book is divided into two halves of approximately 100 pages each. The first five chapters (and the brief appendices presenting some results from number theory) provide a self-contained, complete proof of the negative solution to Hilbert's Tenth Problem. Chapters 6 through 10 introduce various extensions and applications of the results and techniques used in solving Hilbert's Tenth Problem. There is no mention, however, of any partial results which arose in the failed attempts to find a positive solution to Hilbert's Tenth Problem.

Chapter 1 is a short and straightforward chapter in which Matiyasevich presents the definitions and basic properties of the main "diophantine objects" of study in the book: diophantine equations (and systems of equations), sets, relations, and functions. Several equivalent formulations of Hilbert's Tenth Problem are also presented, including considering diophantine equations over the domain $X = N$.

A diophantine set A (of singletons) is a set defined by

$$A = \{a \mid \exists x_1, \dots, x_k D(a, x_1, \dots, x_k) = 0\} ,$$

where D is a polynomial with integer coefficients, and a and the x_i range over the natural numbers. With this definition, a stronger result that implies the negative solution to Hilbert's Tenth Problem can be stated: *The diophantine sets are precisely the computably enumerable sets.* For those familiar with some computability theory, this stronger form of Hilbert's Tenth Problem can serve as a nice framework for remembering the results presented here in terms of diophantine sets. (It also can serve to dull some of the mystery, since one needs to remember that for a long time this equivalence was suspected by many *not* to hold.) Definitions of the other diophantine objects are analogous.

Chapter 2 is the most difficult chapter in the first part of the book and presents a proof that exponentiation is a diophantine function of two variables. Chronologically, finding a diophantine relation of exponential growth was "the last missing link in establishing the algorithmic unsolvability of Hilbert's Tenth Problem," [page 38], and the original proof by Matiyasevich used a relation derived from the Fibonacci sequence. The improvements of this original technique to show (directly) that exponentiation is diophantine simplify the remaining argument, since there are many things that are quite naturally expressed using *exponential* diophantine equations that are not easily expressed using only ordinary diophantine equations. (Imagine trying to formulate Fermat's Last Theorem, for example, as an ordinary diophantine equation rather than as an exponential diophantine equation.)

The proof presented in sections 2.1–2.4 is quite technical and the motivation for some of the steps is not immediately clear, but these sections can easily be skipped on a first reading and the remainder of the first part of the book is considerably easier to read.

Chapter 3 deals with three possible ways to code sequences of numbers as single numbers: Cantor codes, Gödel codes (based on the Chinese Remainder Theorem), and positional codes (based on exponentiation of a constant base). The latter of these is the most useful in the remainder of the text (although the other two reappear from time to time as well) largely because certain operations on codes (such as concatenation, even with mixed bases) can more easily be demonstrated to be (exponential) diophantine using positional coding.

Using the results on exponential diophantine equations from the previous chapter, Matiyasevich uses positional coding to demonstrate that factorial, binomial coefficients and *prime numbers* are exponential diophantine and hence diophantine. The fact that the set of primes is a diophantine set is equivalent to the statement that there is a multivariate polynomial with integer coefficients such that the (positive) natural numbers in its range (over inputs from the natural numbers) are precisely the set of primes. In the commentary to this chapter, Matiyasevich points out that an exponential diophantine representation for the primes was already known in 1952 (the result of work by Julia Robinson), and at the time was considered to provide evidence that exponentiation was probably not diophantine, since the existence of such a polynomial seemed unlikely.

Chapter 4 discusses the construction of *universal diophantine equations* and uses these to demonstrate that there are diophantine sets with non-diophantine complement. In particular, it is shown by diagonalization that the set of all codes of *solvable* diophantine equations (H) is a diophantine set, but that its complement is not.

Chapter 5 puts the final pieces together to demonstrate that the diophantine sets are precisely the computably enumerable sets. This chapter begins with a nice development of Turing machines and quickly develops a large "tool chest" of useful machines by introducing two operations for constructing new machines from old ones, denoted as **if** M_1 **then** M_2, and **while** M_1 **do** M_2 **od**. From here it is a simple matter to show that every diophantine set is Turing semi-decidable (i.e., computably enumerable) by showing that Turing machines are capable of searching for solutions to polynomial equations. More interestingly, it is also the case that every computably enumerable set is (exponential) diophantine, i.e., that diophantine equations can “simulate” Turing machines. Once the equivalence of computably enumerable and diophantine has been established, the only remaining piece is the familiar result from computability theory that a computably enumerable set is computable if and only if its complement is also computably enumerable. Then, since H is diophantine (= computably enumerable), but its complement is not, H is not computable, so there is no algorithm for determining in finitely many steps whether or not a given diophantine equation has a solution.

Having presented a complete proof of the unsolvability of Hilbert’s Tenth Problem, Matiyasevich turns in the second part of the book to applications. Chapter 6 deals with the elimination of bounded universal quantifiers in expressions like

$$\exists x \forall y \le x \exists z_1 \ldots z_k [D(x, y, z_1, \ldots, z_k) = 0] \, .$$

Once one knows the equivalence of diophantine and Turing representations from the previous chapter, it is clear that such bounded quantifiers can be eliminated, but historically, other methods for eliminating these bounded quantifiers were used in the original proof that exponentiation is diophantine. Two such methods are presented in this chapter. The chapter concludes with a discussion of simple sets (from a diophantine point of view).

Chapters 7 and 8 deal with various generalizations of the diophantine problem in which one is interested in such things as whether a diophantine equation has finitely or infinitely many solutions, diophantine equations over the Gaussian integers and over the rationals, and relationships between the number of variables used in diophantine representations and the degree of the representations. In particular, it is shown that every diophantine set has an exponential representation that uses only three unknowns.

The final two chapters consider decision problems from outside number theory. Chapter 9 considers decision problems for real numbers and from calculus and shows, for example, that the problems of determining whether a computable real is rational or determining whether a function (from a suitably nice class of functions) has an anti-derivative in another (suitably nice) class of functions are both undecidable. Chapter 10 considers decision problems for two kinds of games: a “diophantine game” that is somewhat like an Ehrenfeucht-Fraise game and generalized versions of knights on a chessboard.

3 Style

The style of the book is narrative but terse. The reader will immediately notice that the book contains no theorems, at least none that are typographically distinguished in the usual manner of mathematics books. Instead, results are developed and stated within the narrative. Fortunately the statements of most of the important results are italicized and the sections are generally short and well-named, so it is still possible to locate results fairly quickly.

The book is a translation (from Russian), but it reads as if it were not, although it does use “semi-decidable” throughout instead of the more common “computably enumerable” (or “recursively enumerable”).

The book makes extensive use of numbered equations, which is perhaps unavoidable with this material, but requires a good deal of back-referencing by the reader. Consistent use of notational conventions simplifies this somewhat, but most references to equations are made without any explicit indication of the content of the cited equation, and many pages include a dozen or more such references, often from previous chapters.

Each chapter of the book is concluded with sections of exercises, open questions, unsolved problems and commentary. The preface indicates that the exercises contain problems of "varying difficulty," and many are difficult or at least technically involved. The hints provided in the back are sufficient for the more routine exercises, but often consist solely of references to the literature for the more difficult problems, which the author admits "amount to small research problems" intended to "present diverse results but without proofs." [page *xx*] The commentary of each chapter seeks to place the results of the chapter in context, relating it to the historical development of the proof, unsuccessful attempts and partial results which came along the way and further developments and applications of the techniques presented.

The bibliography contains an extensive (35 pages) list of references to literature related to the negative solution of Hilbert's Tenth Problem and other applications of the techniques involved. There are indexes organized by notation, names and subject, but the latter of these is quite short.

4 Opinion

The author intends this book to be accessible to a "broad readership, especially to younger mathematicians" and he succeeds in providing a self-contained, complete and systematic presentation of the negative solution of Hilbert's Tenth Problem that requires essentially no prerequisite knowledge. The outline of the main proof is nicely motivated and clearly presented throughout the 100 pages of the first half of the book. Motivation for the specific number-theoretic techniques involved along the way is not always as clear, but most of the individual steps do not involve much more than rudimentary knowledge in number theory (facts about prime numbers, congruences, and the like), and the few more advanced results from number theory are nicely presented in the appendices.

While proofs of the main result of the book have appeared elsewhere, most of them either presume more prerequisite knowledge or omit details of some of the number theory. So for those interested in a complete proof, this is likely the best source available. In particular, Matiyasevich has included several new proofs which streamline the argument.

For those whose appetite is not satisfied by the first half of the book, there are more than enough additional applications presented in the second half, which according to the author have not appeared in any collected form before. And the bibliography and commentaries together provide a very thorough account of all the work that led up to the final solution to Hilbert's Tenth Problem.

Lambda Calculi: A Guide for Computer Scientists[§]
by Chris Hankin

Review by: Christopher League
Yale University Computer Science

[§]Graduate Texts in Computer Science, Vol. 3; Oxford, 1994 (162 pages)

1 Overview

The λ calculus is an elegant computational model first described by Alonzo Church in 1932, and now used extensively in the formal semantics of programming languages. Originally the inspiration for the Lisp programming language, its typed variants are now the foundation for modern type-safe languages such as Haskell and ML.

Terms in the λ calculus may be defined inductively as follows. Any variable x is a term. If M is a term, then $(\lambda x.M)$ is also a term, called an *abstraction.* An abstraction is analogous to a *function* in programming languages; x is the *formal parameter* and M is the *body.* Finally, if M and N are terms, then $(M\,N)$ is also a term. This term is called an *application*, and roughly corresponds to a *function call* in a programming language. We can also define relations on λ terms to represent various ways of *reducing* or converting one term to another. A standard rule called β reduces terms of the form $(\lambda x.M)\,N$ to $M[x := N]$, a *substitution* in which we replace all free occurrences of the variable x in M with the term N.

The λ calculus is of fundamental importance to computer science; it plays a rôle not only in programming languages, but also in the theory of computation. Since recursion and numerals can both be encoded as terms, the λ calculus can simulate a Turing machine. And since the λ calculus is simpler to describe and more programming-oriented than a Turing machine, perhaps it should supplant (or at least supplement) Turing machines in the computer science curriculum.

This book is a short introduction to the basic theory of the type-free λ calculus, intended for graduate students and advanced undergraduates. The only prerequisite, according to the author, is a standard discrete mathematics course, covering logic and set theory. He intends for the topics and presentation to be especially relevant to computer scientists (as opposed to, say, mathematical logicians).

Chapters 1 through 3 cover the basics that any presentation of λ calculi must include: definitions, substitution, reduction, normal forms, and the Church-Rosser theorem. The rest is a series of loosely-related additional topics: combinatory logic, computability, types, practical issues such as strictness analysis and type inference, and an introduction to other calculi used in computer science. In the next section, I will briefly sketch each chapter, and conclude with my personal impressions of the text.

2 Content summary

Chapter 1 is a 5-page overview of formal systems, BNF grammars, and mathematical induction. It is probably insufficient if the reader is not already familiar with these topics. A more substantial introduction may be found in Chapter 2. Using the propositional logic as an analogy, the author introduces inductive definitions, axiom schemata, and inference rules. He then uses these tools to define the lambda calculus, which includes syntax of terms and contexts, definitions of free and bound variables, and an equivalence relation on terms.

Chapter 2 then considers substitution in more detail. Above, I said $M[x := N]$ means to replace free occurrences of x in M with the term N. We also must ensure that free variables in N do not become bound in M. This may be accomplished by 1) renaming bound variables in M, 2) using a convention wherein all bound variables are chosen to be different from free variables, or 3) using de Bruijn notation, which eliminates variables altogether in favor of natural numbers. de Bruijn notation, while difficult for humans to manipulate, is often used in implementations and machine-checkable proofs.

Next we encounter the fixed point theorem, a rather surprising first result, crucial for the account of computability in Chapter 6. The theorem states that for any term F there exists a term X such that the application of F to X is equivalent to X. The proof is constructive; it shows how to create the fixed point of any term. Fixed points always involve self-application, as in the term $Y \equiv (\lambda x.(x\,x))\,(\lambda x.(x\,x))$. Finally, Chapter 2 defines consistency and completeness, arguing that the λ calculus enjoys both.

Chapter 3 covers *reduction*, a reflexive and transitive (but not symmetric) relation between λ terms that intuitively corresponds to computation. A reduction should be *compatible* as well, meaning that it behaves the same way in any context. The β rule above induces one such reduction. We then explore confluence (the Church-Rosser property) which states that any two reduction sequences starting from equivalent terms have a common reduct. A *normal form* is also defined as a term which contains no redexes.

Now that the very basics of the λ calculus have been introduced, the author turns to Schönfinkel and Curry's Combinatory Logic in Chapter 4. This is a calculus in which two so-called combinators, S and K (with "hard-coded" equivalence rules), are sufficient in place of λ abstraction. Surprisingly, the fixed point theorem still holds. The author then relates CL to the λ calculus: S and K can be defined using λ terms, and a new abstraction operator λ^* may be built using only S and K. Finally, he defines a *compiler* from λ terms to combinatory logic, and shows how to optimize the resulting code using CL equivalences.

Chapter 5 investigates the model theory of the λ calculus. First, to introduce the idea, the author describes a set-theoretic model for propositional logic. Unfortunately, a model for λ calculus is not so simple due to self-application. It would require a set D which is isomorphic to its own function space, $D \to D$. Thus the formulation of the λ model is difficult to understand without some background in domain theory.

Chapter 6 gives an account of computability based on the λ calculus. We first revisit the fixed point theorem; fixed point combinators are essential for computability because they enable recursive definitions. Next we explore two ways of encoding numeral systems as λ terms. The first scheme pairs a number with a place-holder to represent its successor. This way we can retrieve the predecessor in constant time. Addition and multiplication can be defined as recursive functions. The second scheme uses the more familiar Church numerals. Here, finding the predecessor is linear, but addition and multiplication can be encoded without recursion.

Given recursion and numerals, the author proposes a class of numeric functions that are *λ-definable.* He then establishes a link between λ-definable and partial recursive functions, which are known to be Turing computable. Finally, we see that Gödel's incompleteness theorem has an analog in the λ calculus. Determining whether an arbitrary λ term has a normal form is undecidable (analogous to the halting problem for Turing machines).

Chapter 7 introduces a few *typed* λ calculi. In the simply typed λ calculus, every term has a single known type associated with it. λ abstractions have arrow types ($\sigma \to \tau$); if x has type ($\sigma \to \tau$) and y has type σ, then the application ($x\,y$) has type τ. Many terms from the type-free λ calculus cannot be assigned types in this system; most notably, self application is banned because a term cannot be both a function type and the correct argument type. This means we cannot construct fixed points; since there is no recursion, all reductions terminate with a normal form. There exists an algorithm to infer the type of any term.

Next we see the polymorphically-typed λ calculus of Girard and Reynolds. Types can contain variables, bound by the logical quantifier $\forall$ (e.g., $\forall\alpha.\alpha \to \alpha$ is the type of the polymorphic identity function). The quantifier is introduced by a new *type abstraction* Λ. Since type inference for this calculus is undecidable, we annotate the argument of each λ abstraction with its type; then type checking is decidable. Finally, the author introduces a calculus with intersection types in which a term can be considered to have two types simultaneously. This system can express the type of self application: $(\lambda x.(x\,x))$ has type $(\sigma \cap (\sigma \rightarrow \tau)) \rightarrow \tau$ for all types σ and τ. Now, all terms of the type-free λ calculus can be assigned types, but type checking is undecidable.

Chapter 8 addresses an assortment of practical issues, including evaluation order, reduction machines, strictness analysis, and polymorphic type inference. In Chapter 2, evaluation order was left undefined; but when implementing a programming language based on the λ calculus, this is an important decision. Theoretically, *normal order* is traditionally used because it ensures that a normal form is reached if one exists. It is not practical, however, because it evaluates arguments every time they appear in the body of a function. Applicative order, used by ML, evaluates arguments exactly once, but might not reach a normal form, even if one exists; consider a constant function $(\lambda x.3)$ applied to a non-terminating argument. Lazy evaluation, used in Haskell, evaluates arguments *at most* once, and would seem to be the best of both worlds. However, it is not as efficient to implement as applicative order, and is somewhat trickier to describe formally as well.

Reduction machines are useful for formally describing the differences between these evaluation orders; some of the ideas from combinatory logic reappear here. Strictness analysis determines statically that particular sub-terms must be evaluated; this information is useful for avoiding the overhead of lazy evaluation when an argument is certain to be used. Finally, Chapter 8 investigates polymorphic type inference. In the previous chapter, we learned that type inference is undecidable in the polymorphically-typed λ calculus. Milner discovered, however, that if we restrict how polymorphism is introduced, we can formulate an implicitly typed polymorphic calculus and give an algorithm for type inference. This calculus is often called Mini-ML.

The last chapter introduces a few additional calculi. Abramsky's Lazy λ calculus more accurately describes a lazy implementation of plain λ. Boudol's γ calculus is process-based, extending λ with non-determinism and concurrency. It is similar in flavor to Milner's Calculus of Communicating Systems. Finally, the $\lambda\sigma$ calculus is equivalent to plain λ, but it explicitly includes substitution machinery as part of the calculus. In plain λ, even with de Bruijn indices, substitution is meta-linguistic.

3 Opinion

It was the author's intent to make this topic comprehensible and relevant to computer scientists. He states in the preface:

> I strongly believe that there is a distinct cultural difference between computer science students and their mathematically trained counterparts. The former have sound computational intuitions but are generally unfamiliar and uncomfortable with formalism.

I certainly agree with the distinction. Judging from the text, however, I think the author is somewhat optimistic about the formal background of computer scientists, even those bound for graduate school.

Chapter 2, for example, contains at best a perfunctory explanation of all the fundamentals: syntax, equivalence, three approaches to substitution, fixed points, consistency, and completeness are squeezed into only 18 pages. Where appropriate, examples from programming languages are inserted, but these feel like concessions to the non-mathematical segment of the audience in an otherwise mathematical text.

Similarly, the connection between the λ calculus and the theory of computation in Chapter 6 requires a thorough understanding of partial recursive functions, recursively enumerable sets, and

Gödel numbering. I think an account of the theory of computation using the λ calculus as the primary model would be much more interesting.

Nevertheless, the book does succeed in pulling together a collection of topics that are *relevant* to computer science. The long-standing canonical reference[¶] does not include the topics of Chapters 8 and 9, for instance, and is generally less concerned with implementation issues (reduction machines, lazy evaluation, and explicit substitutions). With the help of a motivated instructor, this book could be the primary text for an effective course on the λ calculus.

Review of[‖]

Systems that Learn (second edition)
Authors: Jain, Osherson, Royer, Sharma
Publisher: The MIT Press, 1999
Hardcover: ISBN 0-262-100770
317 **pages**

Reviewer: Carl Smith

While this new edition shares much in common with the first edition, it is hardly fair to call it the "second edition." Firstly, only one of the four authors was among the triumvirate that produced the first edition. Secondly, although the new *Systems that Learn* is 50% larger than the first, it is not just an augmentation of more recent material. For sure, new material has been added, but more importantly, there has been a serious reorganization reflecting the codification of the fundamental core results in the intervening years. Furthermore, while the first edition focused primarily on language, as opposed to function, learning, the recent edition presents a balanced treatment of both threads. Like the first edition, there are numerous exercises, many of which point to extension of the material covered, making the book suitable for use as a graduate course text.

The obligatory introduction lays the groundwork for the technicalities to follow by formalizing somewhat the learning process. The first (of three) parts is rounded out by an in depth discussion of how to treat data, hypotheses and scientists as formal processes and some of the fundamental ramifications of doing so. Making these notions computable also received serious consideration here. The learning of languages and functions are considered in tandem. The presentation is coherent throughout, laying out the basic principles and paradigms considered in the rest of the book.

The second part of the book concerns various fundamental generalizations of the basic paradigm of learning. The discussion starts with various strategies for learning. These strategies revolve on around constraining the learning process in some way. For example, demanding that every hypothesis is consistent with all of the observed data produces an initially surprising constriction of what is learnable. The discussion of various constraints leads naturally to considering various criteria of learning. There are several notions of what it might mean for a scientist (modeled as an algorithm) to succeed at learning some function or language, given suitable examples. For example, does the learning happen only in the limit, or perhaps, after a fixed number of conjectures?

Another historically important issue that ramifies as an identification criteria is the proximity of the produced answer to the desired result. From the earliest days of the learning theory it was realized that science proceeds quite nicely utilizing only partially correct theories. The same phenomenon is also abundantly clear in human activities as well. Consequently, there has been much research on learning criteria where the learner is only required to come up with an "approximation" of the desired result. Different notions of "approximation" give rise to different identification criteria, most of which explained and discussed in the volume under review.

[¶]Barendregt, H.P. **The Lambda Calculus: Its Syntax and Semantics.** Elsevier: Amsterdam, 1984
[‖]© Carl Smith, 2000

The second part of the book is rounded out with a discussion of of various data environments. The early formal studies of learning assumed that the data arrived unstructured and unmolested. More recent studies reported on consider models of inaccurate data and data with some extra structure, as is the case when the data is delived in some regular order.

The third and final part of the book does what any book attempting to survey a research area must do, that is, consider the additional topics that seem to be important but have not yet been integrated into the fundamental core knowledge of the area. As is the nature of such adventures, there are some omissions, and the authors cannot be faulted for them. An entire chapter is devoted to team and probabilistic learning. Within the formal study of learning, it is rare when two different criteria turn out to capture the same collection of learnable phenomena. The deep relationships between learning by teams of algorithms and learning via probabilistic learning algorithms continues to stimulate much research. The presentation here is a excellent condensation of this line of inquiry.

The difficulty of learning persuaded many researchers to consider learning with some sort of additional information. Of course, there are a variety of ways to formalize the notion of "extra information." Among the techniques discussed are using bounds on the size of the answer sought and use of a known good approximation. A separate chapter is devoted to the use of an oracle during learning. Oracle learning gives the learning algorithm an arbitrary consultant to query. The results here focus on how the expressive power of the oracle trades off against the class of phenomena that is learnable.

Defining the complexity of learning has always been on onerous problem due to the fact the complexity of the learning algorithm, in the traditional complexity of algorithms sense, is much different from the intuitive notion of learning complexity. Many of the various and sundry notions of the complexity of learning are discussed in a chapter of the additional topics section.

The book concludes with a modern discussion of the enumeration technique. This technique dates to the earliest days of formal learning theory. The basic idea is to start with a list of possible answers, examine data to eliminate them one at a time, finally arriving at a potential answer that cannot be eliminated by the data. This final answer must be correct, since it is a program that agrees with all the data. A haunting question has been "Is this all there is to learning, the rest being just the complexity of the process?" The key to understanding the issues raised turned out to be one of transformations. The same issue seems to be at the core of the pedagogical technique of giving examples. If it is the case that some phenomenon can be learned, and all transformations of it can also be learned, then we say the original phenomenon can be learned *robustly.* In the classroom, we all hope that our students will learn from our examples to apply the techniques learned in new situations later in their lives. Are we teaching them robustly? This research area is fraught with difficulty and has been a catalyst to many elegant results. The third edition of *Systems that Learn* will undoubtedly contain more material on this topic.

Carl H. Smith
College Park, MD
November 4, 1999