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

The prices I provided this time are from amazon.com; however, the website www.bestbookbuys.com gives a range of prices from place to purchase books on the web. (amazon.com is usually *not* the cheapest, and its usually not even close.).

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

1. **Proofs and Confirmations: the story of the alternating sign matrix conjecture** , by David Bressoud. This is the exciting story of how the alternating sign conjecture was solved. It gives a nice view of how mathematics is actually done. Reviewed by Jeremy Avigad.

2. **Proofs and Refutations** by Imre Lakatos. This is an classic work in the philosophy of mathematics. It forces us to think of what we do in a new light. It is reviewed here since it is a nice companion review to **Proofs and Confirmations**. Reviewed by William Gasarch.

3. **Dynamic Logic (Foundations of Computing)** by D. Harel, D. Kozen and J. Tiuryn. This is a book about assigning logic and semantics to programs. Reviewed by Riccardo Pucella.

4. **Analysis of Algorithms:An Active Learning Approach** by Jeffrey J. McConnell. Review by Christopher Jennings This is a undergraduate text in algorithms that encourages a more hands-on approach then most books.

Review of
Proofs and Confirmations: the story of the alternating sign matrix conjecture [2]
Author of Book:David Bressoud
Co-published by Cambridge University Press and MAA in 1999
256 pages, $23.00
Author of Review: Jeremy Avigad

1 Overview

An *alternating sign matrix* is a square matrix whose entries are all either 0, 1, or -1, with the properties that each row and each column sums to 1, and that the non-zero entries in each row and column alternate in sign. For example,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

is such a matrix. In the mid-1980's William Mills, David Robbins, and Howard Rumsey found themselves wondering how the number of $n \times n$ alternating sign matrices, A_n, grows as a function of n. On the basis of combinatorial fiddling and brute calculation, they came up with the following formula:

$$A_n = \prod_{j=0}^{n-1} \frac{(3j+1)!}{(n+j)!}.$$

This equation came to be known as the *ASM conjecture*, and computational verification of the first 20 cases made it practically certain to be correct. But a proof remained elusive, despite the simplicity of the conjecture and a good deal of effort from the community of algebraic combinatoricists. Doron Zeilberger finally announced a proof in 1992, but it was not until 1995 that all the gaps were filled in and all the details were checked carefully; certifying the resulting 71 page manuscript employed the efforts of 88 referees and a computer. Soon after, Greg Kuperberg discovered another proof, exploiting connections with physicists' models of "square ice." Zeilberger was then able to use this connection to establish refinements of the alternating sign matrix conjecture as well.

Proofs and Confirmations fills out this story gracefully, guiding the reader through the long and elaborate proofs. In the introduction, David Bressoud writes:

> My intention in this book is not just to describe this discovery of new mathematics, but to guide you into this land and lead you up some of the recently scaled peaks. This is not an exhaustive account of the marvels that have been uncovered, but rather a selected tour that will, I hope, encourage you to return and pursue your own exploration.

This passage is representative of the tenor of Bressoud's exposition.

2 Contents and audience

In many ways, Combinatorics is a black sheep among the subjects of pure mathematics. It is often viewed as a study of isolated puzzles, lacking a deep conceptual framework and unifying ideas, and poorly integrated with the broader history and culture of mathematics.[3] *Proofs and Confirmations* may not do much to dispel the notion that combinatoricists like puzzles: alternating sign matrices arose incidentally in the study of an algorithm for evaluating determinants, and in Bressoud's account mathematical curiosity and the spirit of a challenge are the main motivating factors. But, in the end, the connection with square ice shows that the research has applications to statistical mechanics; and the book's final chapter indicates that there are additional ties to representation theory and the study of Lie algebras.

Indeed, Bressoud's book goes a long way towards meeting the objection that combinatorics is a subject without a broader context. In working through the book, one finds a wealth of generally useful concepts (matrices and determinants, binomial coefficients, generating functions, hypergeometric series, Pfaffians), and fundamental methods (including various forms of induction and recursive definition, and the strategy of exploiting symmetries that become apparent when one varies the representation of a problem). One also encounters connections to a number of branches of mathematics, including linear algebra, the study of differential equations, and the study of elliptic functions.

Finally, Bressoud shows us that algebraic combinatorics has a long and rich history. In the course of the narrative we encounter figures ranging from the fourteenth century Chinese mathematician

[3]For a discussion of these views and a spirited response, see W. T. Gowers' "The two cultures of mathematics" in V. Arnold et al., eds., *Mathematics: Frontiers and Perspectives*, American Mathematical Society, 2000, pages 65–78.

Chu Shih-Chieh, to Viete, Newton, Gauss, Sylvester, Cauchy, Jacobi, Cayley, and Weyl, along with many others. These names are not dropped casually, but are, rather, introduced with brief contextual sketches that help the reader place them in the web of mathematical ideas. Even Charles Dodgson (better known as Lewis Carroll) makes an appearance, as the inventor of the algorithm for computing determinants that inspired the study of ASM's in the first place.

Proofs and Confirmations should appeal to a number of audiences. First, of course, are students and researchers in algebraic combinatorics and related fields. The book is streamlined to bring the reader to the forefront of contemporary research as quickly and smoothly as possible. The nature of the subject makes it impossible to avoid pages of detailed calculations, but Bressoud has a gift for framing these calculations with the ideas and intuitions that guide them.

The fact that the book requires nothing more than familiarity with linear algebra, however, makes it accessible to a much wider audience. Its particular focus renders it unsuitable as a textbook for an introductory course in, say, Discrete Mathematics; but it could well be used in an undergraduate or graduate seminar, where the emphasis is on fostering the methods of mathematical exploration rather than conveying a standard body of content. Each chapter is followed by a long list of thoughtful and illuminating exercises, many of which encourage further experimentation with a symbolic computation package like Mathematica or Maple.

More generally, the book will appeal to anyone who likes algebra and combinatorics, and is curious as to what is currently going on at the intersection of these two disciplines. (I should make it clear that it is this group to which I myself belong.) *Proofs and Confirmations* is not light recreational reading, and following the proofs through to the end requires effort and endurance. But many of the topics developed along the way also stand alone, and can be enjoyed in their own right. For example, the reader can flip open to page 208 and learn that the determinant of a skew symmetric matrix is equal to the square of the associated Pfaffian. The book sketches the history behind this discovery, due to Cayley, and provides exercises that guide the reader through an elegant proof.

Finally, Bressoud has something to say about the development of mathematical ideas, based not only on the book's historical allusions, but also on comments from contemporary researchers. Bressoud provides us with snippets from verbal and written accounts from some of the key players in the proof of the ASM conjecture, and even a few photographs. The book's title was inspired by Imre Lakatos' *Proofs and Refutations*, which, in turn, borrows from Karl Popper's *Conjectures and Refutations*. In adapting Popper's analysis of the sciences to the history of mathematics, Lakatos' book created a stir when it was published in 1976; Lakatos aimed to show that formal models of mathematics based on the axiomatic method do not do justice to the "logic of discovery," i.e. the heuristic development of the subject.[4]

Although Bressoud tries to fill out the picture of mathematical discovery in ways that complement Lakatos', *Proofs and Confirmations* is a very different book. *Proofs and Refutations* was intended as a philosophical work that uses mathematical and historical examples to support its claims. In contrast, *Proofs and Confirmations* does not try to provide a sustained historical or philosophical analysis; the primary emphasis is on the mathematics, with philosophical remarks confined mainly to the book's introduction and conclusion. The latter draw on the usual metaphors of mathematical inquiry—exploration, discovery, and creation—and supplement them with some

[4]Lakatos' rhetoric is less damning when one recognizes that the axiomatic method was not designed to play this role. But Lakatos' writings do contain important insights, and are still widely discussed today. I am partial to a critique of Lakatos' work by Solomon Feferman, "The logic of mathematical discovery versus the logical structure of mathematics," in P. D. Asquith and I. Hacking, eds., *PSA 1978: Proceedings of the 1978 Biennial Meeting of the Philosophy of Science Association, vol. 2*, 1981, pages 309–327, recently republished in *In the Light of Logic*, Oxford University, 1998, pages 77–93.

new ones. For example:

> The doing of mathematics is more akin to being dropped on a distant and unknown mountain peak and then seeking to find one's way home.... Research in mathematics almost never begins with careful definitions and lemmas on which we build until something interesting is discovered. It starts with the discovery, and proof is the process of tying that discovery back to what is already known. (p. 258)

In the same discussion, mathematics is also compared to archeology, where objects are found in isolation and then related to their social, cultural, and historical context:

> This is the role of proof, to enrich the entire web of context that leads to understanding. The mathematician does not dig for lost artifacts of a vanished civilization but for the fundamental patterns that undergird our universe, and like the archeologist we usually only find small fragments. As archeology attempts to reconstruct the society in which this object was used, so mathematics is the reconstruction of these patterns into terms we can comprehend. (Ibid.)

Mathematics tends to resist simplifying characterizations, and one should be wary of drawing sweeping conclusions on the basis of one case study. Moreover, as a branch of mathematics, algebraic combinatorics has some atypical features. For example, the accessibility of its basic notions to a bright high school student, and the amenability of its conjectures to computational verification, give the subject a flavor that is distinct from, say, that of algebraic geometry or ergodic theory. Bressoud's remarks do, however, contain interesting insights into the attitudes and motives that drive mathematical inquiry, and they make it clear that there is much more to be said about the process of mathematical discovery than can be couched in terms of formal axiomatic proof.

3 Opinions

Proofs and Confirmations is a lovely book, a pleasure to read and to learn from. The exposition is clear and well organized, and anything but stodgy. As an example, consider the following colorful passage:

> Let us pause for the moment to appreciate the audacity of what we are proposing... Equation (5.4) is a system of $r + 1$ equations in r unknowns. There is no guarantee that it has a solution. The only reason to proceed is the observation that we seem to have completed a missing symmetry. To a mathematician, there could be no better reason. (page 158)

This breathless exuberance may strike the reader as a little campy, but comments like these serve a useful purpose, providing structure to a long proof, and highlighting key ideas and intuitions.

It would be nice if mathematical narratives like Bressoud's were more common. This is not to deny the importance of traditional textbooks and references, where style and content is dictated by the need to lay out the fundamentals of a subject, rather than carry the reader in pursuit of a single thread. After all, setting forth the basics in an organized manner is certainly important. But so is the joy of buckling down and following a line of inquiry just for the fun of it, and few books manage to share that enjoyment as well as this one does.

Review of **Proofs and Refutations**[5]
Author of Book: Imre Lakatos
Published by Cambridge University Press
174 pages, $27.95
Author of Review: William Gasarch (gasarch@cs.umd.edu)

4 What is this book about?

The following questions have been raised by some philosophers of mathematics.

1. When is a statement considered true?
2. What are the standards of rigor? Are they absolute?
3. Is certainty possible?
4. When is a statement considered interesting?

The *formalists* believed that rigorous proof from axioms constituted mathematical truth and that certainty is possible. Moreover, they *identified* math with formal proof. Lakatos disagrees and his book is a counterargument.

The book consists mainly of a dialogue between a teacher and several students about the following statement

"For all polyhedra $V - E + F = 2$ where V is the number of vertices, E is the number of edges, and F is the number of faces."

The dialogue is actually a presentation of various thoughts on this statement from many mathematicians over the last 200 years. There are many footnotes that say who is begin paraphrased; however, the dialogue format allows the arguments to flow smoothly. In the course of the dialogue proofs are offered, counterexamples are found, and statements are modified. The dialogue vividly shows how standards of rigor have changed and how elusive certainty is. It also (implicitly) calls into question the current confidence that we now have it right.

After the dialogue Lakatos claims that mathematical discovery follows the following simple pattern.

1. Primitive conjecture.
2. Informal proof (perhaps a thought experiment).
3. Counterexamples are found (perhaps counterexamples to the proof, perhaps counterexamples to the conjecture).
4. Proofs are re-examined and a new primitive conjecture is found.

Lakatos then shows this pattern in the history of Cauchy's theorem that the limit of a converging series of continuous functions is continuous (this statement is actually false, you need to assume uniform convergence). This process of mathematical discovery illustrates that "final" proof from axioms is neither the heart of the matter nor the last word.

[5]

5 Further Reading

Here is a list of other material on the issues raised by Lakatos.

1. *The Logic of Mathematical Discovery versus the Logical Structure of Mathematics.* Included in the book *In the Light of Logic* by Solomon Fefferman. (The book was published in 1998, but this article originally appeared in the Proceedings of the 1978 Biennial Meeting of the Philosophers of Science Association, vol 2, 309-327.) This is a fine criticism of some of Lakatos' points. To name two such points: (a) most mathematicians work in domains where certainty is possible and common, and (b) the history presented is not really correct but is arranged to make Lakatos' points.

2. *Lakatos' Philosophy of Mathematics: A historical approach* by Koetsier. (Published in 1991.) This book looks at some episodes in the history of mathematics and sees how well they fit Lakatos' framework.

3. *Social Constructivism as a Philosophy of Mathematics* Edited by Paul Ernest. This is a collection of essays about mathematics. The authors all think that math is more a social construct then an absolute science. There is a chapter about Lakatos' work. The book has some valid points to make, though I think it is too extreme.

4. *Conjectures and Refutations: The Growth of Scientific Knowledge* by Karl Popper. (Published in 1963, and known to Lakatos.) Popper was concerned with Philosophy of Science. He believed that certainty was elusive in science and that scientists should propose ways to refute their own theories- if they survive those refutations then they are accepted (for now). Lakatos would agree that certainty is elusive.

5. *The Structure of Scientific Revolution* by Thomas Kuhn. (Published in 1962. Not referenced in Lakatos's book which was written at about the same time but not published until much later. It is not clear if Lakatos knew of Kuhn's ideas.) Kuhn claims that *in practice* when scientists do normal science, they do not run experiments to test their theories. They adapt a paradigm (way of looking at things) and all the experiments they do are within that paradigm. It takes drastic evidence and time to change paradigms. When this happens the paradigm gets radically overthrown (a revolution) and a new paradigm sets in. Kuhn views on science can be seen as analogous to Lakatos in that certainty is elusive and part of what is accepted depends on the prevailing attitudes. (Over time Kuhn's views have mutated into the point of view that there is no objective truth and that all of science depends on social surroundings. See *The Sokal Hoax: The sham that shook the academy* by the editors of Lingua Franca.)

6. *Social process and Proofs of Theorems and Programs* by DeMillo, Lipton, and Perlis. This paper's goal was to show that program verification is not a realistic goal; however, it makes the (more interesting) point that in mathematics a theorem's proof is not the end point for knowing that it is true- it is only the beginning. The work needs to be passed around and the acceptance of its truth is more of a social process among mathematicians.

7. The following papers illustrate that the questions of what criteria we use to establish truth is not a dry topic occupying philosophers far removed from the practice of math, but is actually being debated within the math community.

   (a) *"Theoretical Mathematics": Toward a cultural synthesis of mathematics and theoretical physics* by Jaffe and Quinn, Bulletin of the AMS, Vol 29, No 1, July 1993, pages 1-13.

(b) *On Proof and Progress in Mathematics* by Thurston, Bulletin of the AMS, Vol 30, No 2, April 1994, pages 161-177.

(c) *Responses to "Theoretical Mathematics": Toward a cultural synthesis of mathematics and theoretical physics* by Atiyah, Borel, Chaitin, Fridean, Glimm, Gray, Hirsch, MacLane, Mandelbrot, Ruelle, Schwarz, Uhlenbeck, Thom, Witten, Zeeman. Bulletin of the AMS, Vol 30, No 2, April 1994, pages 178-207.

(d) *Response to Comments on "Theoretical Mathematics"* by Jaffe and Quinn Bulletin of the AMS, Vol 30, No 2, April 1994, pages 208-211.

6 Opinion

The introduction to *Lakatos' Philosophy of Mathematics: A historical approach* contains the following quote:

"This study is a contribution to the philosophy of mathematics and like most philosophical treatises it neither addresses itself to any practical problem, nor does it solve them."

This quote raises the question "Is this type of material worth reading?" The answer is yes. Just as people in (say) AI should have a nodding acquaintance with NP-completeness, Computer scientist theorists should have a nodding acquaintance with Philosophy of Mathematics. We are in a young evolving field that is still defining new terms and concepts. It is helpful to realize that we have underlying assumptions and what they are. Consider the following examples.

1. A set A is NP-complete if $A \in NP$ and $(\forall B \in NP)[B \leq_m^p A]$. Cook originally defined this concept as A set A is NP-complete if $A \in NP$ and $(\forall B \in NP)[B \leq_T^p A]$. Other similar definitions are also possible. It is not clear which one is "correct" or if any are.

2. There were several variants of ZK (zero-knowledge) and even some proofs that used the term before it was formally defined (*The Knowledge Complexity of Interactive Proof-Systems* by Goldrich, Goldwasser, and Rackoff, SICOMP Volume 18, 1989). Under the formal definition of ZK, the proof that Graph Isomorphism is in ZK has to be done sequentially and not in parallel. But the parallel protocol seems to not leak any knowledge. Does the parallel protocol leak knowledge or not? If it does not, then perhaps the definition of ZK should be changed?

7 Parting Thoughts

Lakatos' book (and others) are excellent to make you think about what math is and how much it is affected by the times it is developed. These books are less good at reaching a final conclusion. Past a point the never ending debate and lack of resolution can get tiring. It is good to be acquainted with the issues raised; however, after a while one longs to read a proof of a theorem.

Review of **Dynamic Logic (Foundations of Computing)** [6]
Authors of Book: D. Harel, D. Kozen and J. Tiuryn
Published in 2000 by MIT press, 450 pages, $50.00
Review by Riccardo Pucella, Dept of CS, Cornell Univ

Introduction

In the 1960s, as programming languages were being used to write larger programs, those programs became harder to understand, and people began to worry about issues such as correctness, that is, determining whether a program computed what it was supposed to compute. As a consequence, researchers started to look into the pragmatics of programming, leading among others to a criticism of the GOTO statement [2] and the development of structured programming by Knuth and Wirth. These greatly helped writing programs that were easier to understand, but the issue of showing program correct remained. Undoubtedly helped by the fact that programs had a cleaner structure, researchers began investigating formal approaches to proving programs correct, or, in general, proving that programs satisfied properties of interest. Most approaches involved deriving the proof of a property as one was writing the program, taking advantage of the structured way these programs were written [3, 8]. These approaches were formalized, leading to the total or partial correctness assertions of Hoare [10] or the weakest-precondition calculus of Dijkstra [3]. Essentially, the logic of Hoare dealt with assertions $\{A\}P\{B\}$ around a program P, indicating that if A were true, executing program P would result in B being true. Inference rules indicated how to transform assertions about programs into assertions about larger programs. For instance, if $\{A\}P_1\{B\}$ and $\{B\}P_2\{C\}$ were true, then one could infer that $\{A\}P_1; P_2\{B\}$ was true, with the intuitive reading of $P_1; P_2$ as the sequential composition of P_1 and P_2. Many more formal systems along such lines were devised, and they collectively acquired the name *logics of programs*, or *program logics*.

In a 1976 landmark paper, Pratt recognized that many such program logics could best be understood as modal logics, by essentially associating with every program a modal operator [13]. His idea was developed and refined by Fischer and Ladner [6] and others, culminating into a particular form of program logic called *Dynamic Logic*. Basically, the recognition of the relationship between program logic and modal logic allowed researchers to make use of the vast array of results on modal logics.

The book "Dynamic Logic", by Harel, Kozen, and Tiuryn, offers a self-contained introduction to the subject. The earlier treatments on the subject are either dated, such as the survey by Harel [9] giving the state of the field in 1984, or study Dynamic Logic as a non-trivial extension of modal logic [7]. The latter approach is much more abstract and technically involved. In this review, I hope to give a taste for the subject by looking at its simplest incarnation, the propositional variant of Dynamic Logic, called PDL. (The book describes both the propositional and the first-order variants.) Following which, I will return to the structure of the book, and some personal opinions.

Propositional Dynamic Logic

Let's start from the basics. Recall (classical) propositional logic: start with a set Φ_0 of primitive propositions $\{p_1, p_2, \ldots\}$, where a primitive proposition can be understood as a basic fact about which we want to reason, such as "it is raining in Ithaca", or "it is sunny in Ithaca". We define the set of formulas of the logic by induction: a primitive proposition is a formula, and if φ and ψ

are formula, so are $\neg\varphi$ and $\varphi \wedge \psi$. We define $\varphi \vee \psi$ as an abbreviation for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \Rightarrow \psi$ as an abbreviation for $\neg\varphi \vee \psi$, and $\varphi \Leftrightarrow \psi$ as an abbreviation for $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$. To contrast with the logics we will soon introduce, we refer to this logic (and related ones) as *static logics*. Intuitively, such logics are used to reason about "unchanging" conditions. Propositional modal logic is an extension of propositional logic that permits reasoning about modalities. Typical modalities historically studied by philosophers include *necessity*, namely, that a formula φ is necessarily true (written $\Box\varphi$). One can discuss provability in such a framework, interpreting $\Box\varphi$ to mean "φ is provable". Temporal logic is a logic to reason about time, where $\Box\varphi$ is interpreted as "it is always the case that φ" [4]. Epistemic logic is a modal logic to reason about knowledge, where $\Box\varphi$ (often written $K\varphi$) is interpreted as "the agent knows φ" [5]. Modal logic is a general framework for reasoning about such features, and it was Pratt's insight that one could reason about programs in such a setting. The intuition is to associate with every program α a modal operator $[\alpha]$, and to interpret the formula $[\alpha]\varphi$ to mean "All halting executions of program α result in a state satisfying φ". (The program α may have many possible executions if it is nondeterministic.)

As we will be reasoning about programs, it is a good idea to focus on what the programs look like. We start with an extremely simple language in which to express our programs. We take a set $\mathcal{A} = \{a_1, \ldots\}$ of *primitive programs*. These are abstract operations we want our programs to perform. *Regular programs* are formed by sequencing other programs $\alpha_1; \alpha_2$, by taking nondeterministic choices of programs $\alpha_1 \cup \alpha_2$, or by looping α^* (meaning repeating program a 0 or more times, nondeterministically).[7] We define a logic called PDL (for Propositional Dynamic Logic) to reason about such programs. The syntax of the logic distinguishes between programs and formulas, both sets defined by mutual induction. We define the set of programs essentially as above: a primitive program is a program; if α and β are programs, so are $\alpha;\beta$, $\alpha \cup \beta$, and α^*; moreover, if φ is a formula, then $\varphi?$ is a program. We define the set of formulas as follows: a primitive proposition is a formula; if φ and ψ are formulas, so are $\neg\varphi$ and $\varphi \wedge \psi$; if α is a program and φ is a formula, then $[\alpha]\varphi$ is a formula. Intuitively, the meaning of programs and formulas should be clear. We have seen how programs are interpreted. The program $\varphi?$ checks whether formula φ holds at the current state of the program. If it does, the program terminates. Otherwise, it blocks. Formulas keep the interpretation they have in propositional logic, with the added understanding that $[\alpha]\varphi$ means to execute program α and check if φ holds whenever the program halts. We write $\langle\alpha\rangle\varphi$ as an abbreviation for $\neg[\alpha]\neg\varphi$; one reading of $\langle\alpha\rangle\varphi$ is "at least one halting execution of α results in a state satisfying φ."

Note that we can write some fairly involved formulas using this setup. If we do not put any restrictions on the φs that can appear in programs of the form $\varphi?$, we can write formulas such as $[[\alpha]\psi?;\beta]\varphi$, which says that if all halting executions of α result in a state where ψ holds, then all halting executions of β result in a state where φ holds. It seems counterintuitive for our programs to be able to perform speculative execution in that way, especially since such properties have a tendency to be undecidable for any reasonable programming language. If we restrict PDL to only allow as test formulas those in which no modal operator appears (in other words, a formula in a test can only be a boolean combination of primitive propositions), we call the logic *poor test PDL*. In contrast, the unrestricted version is called *rich test PDL*.

At this point, our logic is just a syntax for writing formulas with an intuitive understanding of what they mean. We can formalize our intuitions about the logic by writing down axioms and inference rules—essentially the properties of the logic. Given our intuitive understanding of the meaning of formulas, we can come up with the following axioms for the logic:

[7] Hence the name regular programs: consider a program as a regular expression and let L be the language (over $\mathcal{A}$) generated by the regular expression; each sentence in L is a possible trace or execution of the program.

1. axioms for propositional logic
2. $[\alpha](\varphi \Rightarrow \psi) \Rightarrow ([\alpha]\varphi \Rightarrow [\alpha]\psi)$
3. $[\alpha](\varphi \wedge \psi) \Leftrightarrow [\alpha]\varphi \wedge [\alpha]\psi$
4. $[\alpha \cup \beta]\varphi \Leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
5. $[\alpha;\beta]\varphi \Leftrightarrow [\alpha][\beta]\varphi$
6. $[\psi?]\varphi \Leftrightarrow (\psi \Rightarrow \varphi)$
7. $\varphi \wedge [\alpha][\alpha^*]\varphi \Leftrightarrow [\alpha^*]\varphi$
8. $\varphi \wedge [\alpha^*](\varphi \Rightarrow [\alpha]\varphi) \Rightarrow [\alpha^*]\varphi$

The following inference rules are also used:

$$\frac{\varphi \qquad \varphi \Rightarrow \psi}{\psi} \qquad\qquad \frac{\varphi}{[\alpha]\varphi}.$$

We say φ is provable (written $\vdash \varphi$) if φ is derivable from the axioms and the inference rules given above.

At this point, note that all we have done is playing with syntax. We have no way of saying whether a given formula is true or not. It was Tarski's great contribution in the 1930s to point out that semantics can be used to discuss truth of formulas in a logic, independently of any axiom system. One gives a semantics for a logic by exhibiting a *model* for the formulas in the logic, telling us how to assign truth values to formulas. Models for PDL are derived from models for general modal logics due to Kripke [12]. (Indeed, this was one reason for the interest in casting program logics in a modal framework.) Essentially, a model is a set of states; think of all the states a program could be in. Programs take us from one state to another, and at every state we have an interpretation function telling us what primitive propositions are true at that state. General formulas will express properties of moving through that state space. Formally, a model M is a tuple (S, π, σ) where S is a set of states, π is an interpretation function assigning a truth value to each primitive proposition p at each state s, i.e. $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$, and σ associates to every primitive program a binary relation on the set of states. Intuitively, $(s_1, s_2) \in \sigma(a)$ if executing the primitive program a in state s_1 leads to state s_2. Our first step is to extend σ to all programs by posing:

$$\sigma(\alpha;\beta) = \sigma(\alpha) \circ \sigma(\beta),$$

$$\sigma(\alpha \cup \beta) = \sigma(\alpha) \cup \sigma(\beta),$$

$$\sigma(\alpha^*) = \textstyle\bigcup_{n \geq 0} \sigma(\alpha)^n.$$

For R and S binary relations, we write $R \circ S$ for the relation $\{(u, v) \ : \ \exists w.(u, w) \in R, (w, v) \in S\}$, and R^n is defined inductively with R^0 the identity relation, and $R^{n+1} = R^n \circ R$.

Using these definitions, we define what it means for a formula φ to be true (or *satisfiable*) in state s of a model M, written $(M, s) \models \varphi$, by induction on the structure of φ. Note that my notation is slightly different from the one in the book, but more in keeping with traditional modal logic presentations.

$(M, s) \models p$ for a primitive proposition p if $\pi(s)(p) = \mathbf{true}$,

$(M, s) \models \neg\varphi$ if $(M, s) \not\models \varphi$,

$(M, s) \models \varphi \wedge \psi$ if $(M, s) \models \varphi$ and $(M, s) \models \psi$,

$(M, s) \models [\alpha]\varphi$ for a program α if for all s' such that $(s, s') \in \sigma(\alpha)$, $(M, s') \models \varphi$.

A formula $[\alpha]\varphi$ is true at a state s if for all states s' that can be reached by executing the program α, at state s, φ holds. We can verify that $\langle\alpha\rangle\varphi$ holds at a state s if and only if there is at least one state that is reachable by program α from state s such that φ holds in the state, hence justifying our intuitive reading of $\langle\alpha\rangle\varphi$.

If a formula φ is true at all the states of a model M, we say that φ is valid in M and write $M \models \varphi$. If a formula φ is valid in all models, we say φ is valid, and write simply $\models \varphi$.

We now have two distinct ways of reasoning in the logic: syntactically, by using the provability relation, and semantically, by reasoning about the truth of formulas. Ideally, we would like these two approaches to yield equivalent results. The properties relating the $\vdash$ and $\models$ relations are called soundness and completeness. An axiomatization for a logic is *sound* if anything provable is valid: formally, if $\vdash \varphi$ implies $\models \varphi$. This property allows us to safely reason syntactically: anything we can prove will be true. An axiomatization is *complete* if anything valid is in fact provable: formally, if $\models \varphi$ implies $\vdash \varphi$. Completeness guarantees us that if there is anything interesting we want to say, we can in fact derive it syntactically. One of the core foundational results related to PDL is that the axiomatization above is sound and complete for the Kripke models introduced above.

As we noted, the logic we described above treats programs as sequences of abstract primitive programs. One can say much more interesting and precise things by considering actual operations on an actual state. The typical approach is to consider a set of variables and take a state to be an assignment of values to those variables (also known as a *valuation*), where the values are taken from a fixed *domain of computation*. The primitive programs in such a setting are simply assignments of values to variables. Intuitively, the primitive program $x := 3$ will make a transition from any given state to a state which has the same valuation but for the fact that variable x is associated with the value 3. To reason about such programs, we can extend our logic to a first-order logic, simply called Dynamic Logic. On the logic side, we replace the primitive propositions by a first-order vocabulary of predicate and function symbols. A predicate takes the form $r(t_1, \ldots, t_n)$ for terms $t_1, \ldots, t_n$. A term is either a variable or a function symbol applied to other terms. The interpretation of predicate and function symbols is as in first-order predicate logic: they correspond respectively to relations and functions over the domain of computation. We further allow quantification over variables, so that $\forall x.\varphi$ is an allowed formula. (As usual, we write $\exists x.\varphi$ for $\neg\forall x.\neg\varphi$.) For example, if we assume a standard interpretation for the equality predicate, the formula $\forall y.\langle x := 3\rangle y = x$ will be true in any state with a valuation assigning the value 3 to all variables except possibly x. For a full formalization of the logic, its semantics, and its properties, I will at this point refer to the book.

The book

The book is divided in three parts. The first part is meant to make the book essentially self-contained, by providing the necessary background material needed for the presentation of Dynamic Logic. At a full one hundred and fifty pages, it makes up almost half of the book. The second part focuses on the propositional variant of Dynamic Logic, while the third part focuses on the first-order variant.

Chapter 1, **Mathematical Preliminaries**, is the obligatory review of basic mathematical ideas from discrete mathematics, such as sets, relations, graphs, lattices, transfinite ordinals, and set operators.

Chapter 2, **Computability and Complexity**, reviews the relevant topics from the theory of computation. Among others, it looks at computational models including deterministic, nondeterministic, and alternating Turing machines, the characterization of undecidable problems, and reviews the basic complexity classes, discusses the arithmetic and analytic hierarchies, and first-order inductive definability. It also reviews the basic notions of problem reducibility and completeness, and presents a family of tiling problems, shown complete for various complexity classes.

Chapter 3, **Logic**, thoroughly reviews the basic notions of logic. It introduces several classical logical systems: propositional logic, equational logic (the logic of equality), first-order predicate logic, infinitary logic (a variant of predicate logic that allows some infinite expressions), and modal logic. For each system, syntax and semantics are discussed, as well as axiomatizations and elementary results.

Chapter 4, **Reasoning About Programs**, defines the programming framework assumed throughout the book. It defines the kind of programs studied, namely state-based imperative programs, with a focus on the input/output relation of a program. It discusses the program constructs appearing in the study of Dynamic Logic: while programs, regular programs, recursion, r.e. programs, nondeterminism. It defines the notion of partial and total correctness (partial correctness does not stipulate that a program halts, total correctness does), and introduces Hoare Logic.

Chapter 5, **Propositional Dynamic Logic**, starts the study of PDL, the propositional variant of dynamic logic. It gives the syntax and semantics of PDL, discusses the axiomatization given above, proves it is sound with respect to the semantics, and proves various properties of the logic, with a special focus on the iteration operator * that makes the whole logic nontrivial. It also shows how to faithfully encode Hoare Logic in PDL: intuitively, the Hoare Logic assertion $\{\varphi\}\alpha\{\psi\}$ corresponds to the PDL formula $\varphi \Rightarrow [\alpha]\psi$.

Chapter 6, **Filtration and Decidability**, establishes one of the most significant and surprising results for PDL, the so-called *Small Model Theorem*: if a formula φ is satisfiable, then it is satisfiable in a finite model with a small number of states, exponential in the size of φ. A standard technique from modal logic, *filtration*, is used to establish this result—although the technique has to be adapted because of the special nature of the iteration operator. The Small Model Theorem is surprising because PDL is not compact: an infinite set of formulas can be finitely satisfiable while not being satisfiable, because of the iteration operator.[8] The classical example of such a set of formulas is the set $\{\langle\alpha^*\rangle\varphi\} \cup \{\neg\varphi, \neg\langle\alpha\rangle\varphi, \neg\langle\alpha^2\rangle\varphi, \ldots\}$. However, it turns out that the iteration operator, albeit infinitary in nature, is still uniform enough in its effect to give a Small Model Theorem. This theorem yield as a consequence that it is decidable to determine if a formula is satisfiable: just enumerate all the finite models containing up to $2^{|\varphi|}$ states, and check if φ holds in any of them.

Chapter 7, **Deductive Completeness**, establishes that the axiomatization given above is complete with respect to the semantics of PDL. This is achieved by showing that a consistent formula φ is satisfiable. The proof follows the canonical model structure of completeness proofs for modal logic, but with a twist. We first build a canonical *nonstandard* model for φ. A nonstandard model is one where the relation corresponding to the iteration operator, i.e. $\sigma(\alpha^*)$, is not the transitive reflexive closure of $\sigma(\alpha)$; rather, we simply require that $\cup_{n\geq 0}\sigma(\alpha)^n \subseteq \sigma(\alpha^*)$. We then collapse the nonstandard model into a standard model by filtration.

[8] A set of formulas F is satisfiable if there exists a model M and a state s such that $(M, s) \models \varphi$ for every $\varphi \in F$; a set of formulas F if finitely satisfiable if every finite subset of F is satisfiable.

Chapter 8, **Complexity of PDL**, revisits the decidability of the satisfiability problem, showing that the problem has a more efficient algorithm than the naive one presented in Chapter 6, by giving an EXPTIME algorithm for satisfiability. This is basically as good as it gets, as satisfiability for PDL is shown to be EXPTIME-complete.

Chapter 9, **Nonregular PDL**, explores what happens when we allow nonregular operators in programs. An easy result is that any nonregular operator yields a logic strictly more expressive than PDL. Unfortunately, it does not take much for satisfiability to become undecidable. The bulk of the chapter is devoted to establishing a "threshold" between decidable and undecidable extensions. For example, PDL over context-free programs is undecidable.

Chapter 10, **Other Variants of PDL**, examines variants of PDL studied in the literature, including deterministic PDL (where we disallow various forms of nondeterminism, either syntactically or semantically) and automata PDL (where programs are finite-state machines), programs with various restrictions on allowable tests, programs with complementation or intersections, programs with a converse operator, logics with well-foundedness and halting predicates, and extensions to model concurrency.

Chapter 11, **First-Order Dynamic Logic**, starts the study of the first-order variant of Dynamic Logic. As we noted above, first-order Dynamic Logic adds a domain of computation to the logic. States are no longer abstract, but are taken as a valuation for variables. Primitive programs are no longer abstract, but consist of assignment to variables. Variables also appear at the level of formulas, where they can be quantified over as in first-order predicate logic. Also as in first-order predicate logic, we have a first-order vocabulary consisting of predicate and function symbols, interpreted over the domain of computation. The chapter first discusses the classes of programs considered. This requires more care than in PDL because of the added level of details in the syntax of programs. Regular programs are defined as in PDL, but extensions include arrays, stacks, and wildcard assignment (i.e. an assignment of the form $x :=?$). The semantics of first-order Dynamic Logic are given, necessarily more involved than for its propositional variant. Essentially, one defines a first-order structure for the vocabulary, specifying the interpretation of the predicate and function symbols. This structure is turned into a Kripke structure by considering as the states the valuations of the variables. As in PDL, programs are binary relations between states, and formulas are given a meaning at particular state using the interpretation of the predicate and function symbols, as well as the valuation for the variables.

Chapter 12, **Relationships with Static Logics**, investigates an aspect of Dynamic Logic peculiar to its first-order variant. Reasoning can take two forms: uninterpreted (involving properties independent of the domain of computation), and interpreted (where one focuses on a particular domain or class of domains). This dichotomy permeates the rest of the book. Some basic properties of first-order predicate logic are shown to fail to hold: the Löwenhein-Skolem theorem, completeness, and compactness. Comparisons are then made between uninterpreted first-order Dynamic Logic and infinitary logics. To study Dynamic Logic at the interpreted level, the chapter focuses on a particular domain of computation, the natural numbers with the usual arithmetic operations. Comparisons with infinitary logics are also made.

Chapter 13, **Complexity**, addresses the complexity of first-order Dynamic Logic. More precisely, the difficulty of determining the validity of formulas at either the uninterpreted or the interpreted level is studied, for a variety of Dynamic Logics over the programming languages of Chapter 11. The expressiveness results of Chapter 12 can be used to give rough bounds, by looking at the difficulty of determining validity for various infinitary logics. (Clearly, because Dynamic Logic subsumes first-order predicate logic, validity is undecidable; the question is: how undecidable?) The chapter also introduces the spectral complexity of a programming language. Roughly

speaking, this notion provides a measure of the complexity of the halting problem for a programming language. The spectral complexity of the languages introduced in Chapter 11 is investigated.

Chapter 14, **Axiomatization**, studies axiomatizations of first-order Dynamic Logic. By the results of Chapter 13, since the validity problem for both the uninterpreted and interpreted levels of Dynamic Logic are highly undecidable, one cannot hope to find a nice finitary axiomatization. (This is a basic result from mathematical logic.) This does not prevent one to derive an infinitary axiomatization, including, for instance, inference rules with infinitely many premises. Such a complete axiomatization is given for uninterpreted first-order Dynamic Logic. Similar axiomatizations are given for the interpreted level, although in this case completeness is taken to be relative to an arithmetical structure.

Chapter 15, **Expressive Power**, studies the relative expressive power of languages. This can be done for the uninterpreted level, by comparing Dynamic Logics over both languages in terms of logical expressibility. By comparing the expressive power of logics, as opposed to the computational power of programs, one can compare for example deterministic and nondeterministic languages. The fundamental connection between expressive power of logics and spectral complexity is explored. The impact of nondeterminism, bounded or unbounded memory, various kinds of stacks, and wildcard assignments is investigated.

Chapter 16, **Variants of DL**, considers restrictions and extensions of Dynamic Logic. The interest is mainly in questions of expressive on the uninterpreted level. Discussed are Algorithmic Logic (a predecessor of Dynamic Logic), Nonstandard Dynamic Logic (allowing nonstandard models of time by referring only to first-order properties of time when measuring the length of a computation), an extension of Dynamic Logic with well-foundedness and halting assertions, Dynamic Algebra (an abstract algebraic framework corresponding to PDL), probabilistic variants of Dynamic Logic, and extensions to handle concurrency and communication.

Chapter 17, **Other Approaches**, explores topics closely related to Dynamic Logic. Dynamic Logic is related to Temporal Logic; the main differences being that in Temporal Logic, programs are not explicit in the language, but one can reason about intermediate states of a computation. Process Logic is introduced, essentially a combination of both Dynamic Logic and Temporal Logic. Process Logic uses explicit programs like Dynamic Logic, but moreover provides a way to reason about the intermediate states reached by a program through its temporal operators. The μ-calculus is introduced, which uses as its central expressive feature an operator to compute least fixpoints. The propositional variant, known as the modal μ-calculus, subsumes all known variants of PDL, and various forms of Temporal Logics, while having a simpler syntax. The modal μ-calculus, has become popular for the specification and verification of properties of transition systems. Finally, Kleene algebras (the algebra of regular expressions) and its extension with tests, can be used to carry out simple program manipulations. The advantage is that Kleene algebras form a purely equational subsystem, apparently less complex than PDL (given all known complexity theory results).

Overall, the book is well-paced. The one hundred and fifty pages of background material on computability theory and logic may seem daunting, but in the long run will be appreciated. You can skim over many of the background sections if you are familiar with the material, once you have figured out the particular notation and terminology of the authors.

The description of PDL in part II of the book is technical (it is after all, a book on logic), but easy to follow, with every detail put down on paper. Part III on first-order Dynamic Logic requires a notably more careful reading, both because the material is more complex, and because the treatment is denser. It is especially in that part that following the bibliographical references at the end of every chapter becomes useful, even necessary, for a thorough understanding.

The book should be of interest mainly to students and researchers interested in program ver-

ification. It offers interesting features for logicians and philosophers as well, as Dynamic Logic is a nontrivial extension of modal logic. As the "programs" studied in Dynamic Logic need not be typical programs, but any formalized notion of action, Dynamic Logic is well suited for reasoning about actions and their effects, with obvious applications to artificial intelligence, and to normative system specification (where it can be taken as a basis for deontic logic).

The book has been used as the basis for graduate courses; depending on the mathematical maturity of the students, a fair amount of material can be covered. The main determining factor being their previous exposure to mathematical logic. As PDL (resp. Dynamic Logic) extends propositional (resp. first-order predicate) logic, both syntactically and semantically, previous exposure greatly helps, as does exposure to modal logic.

As I stated in the introduction, this book has the decisive advantage of focusing exclusively on Dynamic Logic. Other treatments are often studied after a thorough exploration of modal logic, including the temporal variants. This leads to deep, but difficult to follow treatments for the beginner.

Especially interesting is that the book, while being an entry-level introduction to the subject, does point to active areas of research. Particularly relevant are the application of Kleene algebra as a more tractable theory of program transformations; it is currently being used to verify and reason about compiler optimizations [11]. Process Logic is needed if one is interested in reasoning about nonterminating programs, but it has not received much attention lately. Finally, the μ-calculus is currently *en vogue* in the model checking community, where one is interested in verifying transition systems [1], leading to practical considerations driving research. This book can be seen as an introduction to the underlying ideas needed for a thorough understanding of the μ-calculus as a specification language.

References

[1] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[2] E. W. Dijkstra. Go To statement considered harmful. *Communications of the ACM*, 11(3):147–148, March 1968.

[3] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[4] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 995–1072. The MIT Press / Elsevier, 1990.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.

[6] M. J. Fisher and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[7] R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes, No. 7. CSLI, 1992.

[8] D. Gries. *The Science of Programming*. Springer-Verlag, 1981.

[9] D. Harel. Dynamic logic. In Gabbay and Guenthner, editors, *Handbook of Philosophical Logic. Volume II: Extensions of Classical Logic*, pages 497–604. Reidel, 1984.

[10] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 583, 1969.

[11] D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artifical Intelligence*, pages 568–582. Springer-Verlag, 2000.

[12] S. Kripke. A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[13] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th Symposium on the Foundations of Computer Science*, pages 109–121. IEEE Computer Society Press, 1976.

Review of **Analysis of Algorithms:An Active Learning Approach**[9]
Author of Book Jeffrey J. McConnell
Publisher: Jones and Bartlett Computer Science
2001, ISBN 0-7637-1634-0, $65.95
Author of Review: Christopher G. Jennings,McMaster University,cjennings@acm.org

Students don't learn from reading a textbook — or from lectures, for that matter. That's the basic idea behind active learning. Well, okay... I admit it: that definition oversimplifies things. Still, the active learning philosophy does stress that if you want people to acquire a skill rather than just spit point form facts back at you, they have to practice using that skill. A lot. Skills can't be absorbed by reading or listening to other people apply them — a fact anyone who has taught an introductory programming course will have stressed repeatedly to students. Seems obvious enough. So how could a *book* be bold enough to claim that it will help teach students the *skill* of algorithmic analysis using an active learning approach? A good question, and one to which this book is, for the most part, a good answer.

The first chapter covers fundamental concepts such as best-case, worst-case, and average-case analysis, big-O notation, and so forth, as well as fundamental mathematical skills. The book assumes no mathematical knowledge beyond basic algebra, but this is a two-edged sword. None of the topics are covered in detail, and the coverage of probability theory and recurrence relations is especially thin. While this makes the book more accessible, it also precludes going directly from this text to more advanced studies without first making up the missing background material.

The remaining chapters each cover one problem space or algorithmic technique. It was refreshing to see a few topics that are often skipped in introductory texts, such as algorithms for parallel architectures. Each chapter analyzes the classic algorithms in that area, deriving best-case, worst-case, and average-case complexities. The discussion of the algorithms and the derivations are at just the right level: they don't make things so easy to follow that the reader takes them for granted and skips over them without understanding; but they don't skip so many details that the reader becomes lost and frustrated.

The problem space presentations are all concisely written and easy to follow. There is just enough historical and theoretical background to ensure that the rest of the chapter makes sense. Although the book includes a list of references, the text rarely invokes them directly. As with the condensed mathematical introduction, the intent of this is to make it possible to read each topic in one sitting. The goal is for a student to cover the text material before class, then to attend a lecture on the topic, and then work on group projects or other collaborative efforts where the material is applied and the knowledge reinforced with application. Many professors assign reading

[9]

materials as preparation for a class, but it is generally a minority of the class that actually comes to class properly prepared. There are ways to correct this habit — for example, by requiring the students to come to class prepared with written summaries along with answers to some related questions. McConnell takes another approach by trying to make the reading itself less of a chore. His argument would seem to be that, since algorithmic analysis is largely a collection of techniques to be skillfully applied, one should spend time learning to apply the techniques rather than learning basic facts.

For the most part, this is a sound approach, but McConnell seems to forget that text books are also used as reference works once a class is over. The proper use of a textbook in teaching is to supply basic facts, and the minimalist approach used by this text sometimes leaves out too many of them. For example, the RAM (random access machine) model (with all operations but loops and function calls assumed to take an equal amount of time), is used as the model of computation throughout most of the textbook but not explicitly named until more than halfway through the book. Even when this does happen (in the chapter on parallel algorithms), it is defined as "the general model for the algorithms [we have studied until now]", and no connection is made to the description of the model that was provided in the first chapter.

Depending on your goals, this may or may not be a problem. If you are concentrating on practical applications, the essentials are covered. If you want to prepare students for further studies in this area, you will have to supplement the text material, either in your lectures or with additional reading. Again, though, this has more to do with a nontraditional learning than a flaw with the book. I might have preferred a book with the same introductory material in each section, followed by some additional facts and background to strengthen the theoretical material in the text — but that is a reflection of what my objectives would be for such a course. To be fair to the author, his own learning objectives are clearly stated in the introduction, and he doesn't waver from them.

Aside from those issues, there were the usual sorts of errors and inconsistencies one expects to find in a first edition. Here and there a word is missing. In a problem on radix sorting, the range 0 to 2^{64} is given where 0 to $2^{64} - 1$ appears to have been intended. That sort of thing. I only found this occasionally distracting, as at the start of the book when the author sets out using the term "repetitive" (in contrast with recursive) and then switches over to the more commonly used "iterative" a few paragraphs later for no apparent reason. However, these will no doubt disappear as the book matures.

In summary, this textbook makes one fundamental assumption: the reader wants to learn the skill of algorithmic analysis for the purpose of applying it to everyday problems. If that is your goal, this is the book to use. If your taste turns more to the theoretical rather than the applied, then this book alone may not suffice (nor does it claim to), though it would serve as a useful partner to a more traditional text.

Get e-books with questions from actual IELTS tests along with suggested answers to acquire a good band score! Get e-books with questions from actual IELTS tests with answers to acquire a good band score! See Ebooks. Home.Â Fetch means to collect someone or something from another place and bring it to the place you are now, to your home or to the place you are talking about. We donâ€™t use bring with this meaning of â€˜collectingâ€™: X Could you bring the children from school on Monday? âœ“ Could you fetch the children from school on Monday? (= go to the school and bring them home). Also check : Grammar for IELTS. This book is not meant to be changed in any way. ISBN 978-1-84862-250-0 Acknowledgements Authors' Acknowledgements We would like to thank all the staff at Express Publishing who have contributed their skills to producing this book. Thanks for their support and patience are due in particular to: Megan Lawton (Editor in Chief); Mary Swan and Sean Todd (senior editors); Michael Sadler and Steve Miller (editorial assistants); Richard White (senior production controller); the Express design team; Sweetspot (recording producers); and Kevin Harris, Kimberly Baker, Steven Gibbs and Christine Litt... A second important skill is managing uncertainty. Work experience is now an essential part of a. Barton gives the classic example of attitudes to time. university education. But competition for places in Europe â€˜Americans and the British always want quick decisions.Â Pronunciation and spelling 4 1:11 Put the letters of the alphabet in the correct column. Then listen and check. /eÉª/ /i I hope the summer job is going well. Have you decided where to go on holiday? That skiing break sounds the best, doesnâ€™t it?Â You know that Scotland is famous for its lakes. The best known of them is Loch Ness where according to the Download this letter. Informal letter sample 12.Â It is spring time, and we usually go to the countryside to have a picnic. We play different games, make a fire and enjoy our time. On my last birthday we didnâ€™t go anywhere.