Formal Aspects of Computing (1989) 1: 293-301

Formal Aspects of Computing

Book Reviews

Temporal Logics and Their Applications, *Anthony Galton (editor). Academic Press, 1987. 244 pages. Price £22.00. ISBN 0-12-274060*

This volume arose out of a conference on temporal logic and its applications run by the Centre for Theoretical Computer Science at Leeds University in January 1986. It contains three papers based on presentations given at the conference supplemented by three additional papers containing closely related material.

Temporal logics (there is more than one such logic) are now recognised as powerful tools for reasoning about dynamic systems such as distributed and reactive systems. In essence temporal logic deals with statements such as:

eventually . . . ,

. . . after . . . ,

henceforth . . . etc.

Linguistically this means that we can argue formally about tenses other than the present:

- it will be true that Jack was formally an undergraduate,
- it used to be his ambition to be a snooker champion,
- he will practice until he succeeds.

In his introduction to the book, John Tucker cites the origins of temporal logic as being in philosophy. (Indeed he argues that this is further evidence why philosophy as an academic discipline should not be discarded whimsically - Here Here!!) However it is not noteworthy that, despite their backgrounds, all the contributing authors are currently working in Computer Science departments, not in Philosophy or Mathematics, and this serves to reinforce the impression that temporal logic is given purpose by, and owes much of its development to, activity in Computer Science and, in a less rigorous style, AI.

Essentially the book is in two parts, the first (written by the editor) is a comprehensive overview of the basics of the subject as it stood circa 1986/7. Like most branches of Computer Science, as one might justifiably regard temporal logic, advances in - and variants of - temporal logic continue to evolve apace. Nevertheless, for those who are new to the subject it is a readable and thought provoking introduction. Readers not accustomed to logical arguments above the level of "symbol juggling" in first order predicate calculus may encounter difficulties and perhaps not appreciate the subtleties; but this is not a reflection on the author. The reader familiar with current verification proof methods for sequential programs should have relatively few problems and hopefully will be interested

by the potential descriptive and analytical possibilities that the temporal operators promise. Intrinsic to the introduction is the characterisation of time. At the very least we have discrete, continuous, linear and branching models of time. There is more to this than just counting!

The rest of the book comprises self-contained papers that address either the application of temporal logics to certain areas of Computer Science or discuss more specialised variants of the logic. These papers not only serve to underline the diversity of the subject but hint at yet more links with Computer Science.

Chapter 2, written by Howard Barringer, is a tutorial on the use of temporal logic in the compositional specification of concurrent systems and addresses the usual problems such as fairness and liveness. The exposition is centred an action language, its semantics and an associated proof system. This is not just a very readable account of work which is of interest and relevance to any one involved with the serious study of disturbed systems, but serves also as an introduction to his more extensive collaborative work with Kuiper and Pneuli.

The next chapter, by Roger Hale, addresses temporal logic programming. Based strongly on the work of Moszkowski, Interval Temporal Logic is introduced and developed into the Tempura programming language. The classical "Towers of Hanoi" is discussed and solutions presented in various programming paradigms. The appropriateness of this example to illustrate a system which allows/supports/encourages concurrency is questionable, and the author seems to agree, but use of temporal logic makes it easier to reason about the solution.

Fariba Sadri surveys three different approaches to temporal reasoning: Kowalski-Sergot, Lee-Coelho-Cotta, and Allen. The paper describes and compares the formalisations and goes on to argue that each could be modified to incorporate the features of the other two.

Chapter 5 is written by Anthony Galton and concerns the Logic of Occurrence. The subject matter includes a formal treatment of tenses and on-going or complete events and, as to be expected, links to other work on so-called Event Logic. The paper then goes on to present a tentative development of event logic and its use to formalise temporal reference.

The final paper is by Dov Gabbay and returns to the subject of programming in temporal logic. The approach taken is to incorporate modal and temporal connectivities into Prolog. The basic ideas in moving towards Temporal Prolog are discussed and illustrated by considering a selection of the new operators that are required; a fuller exposition of the extended language is promised elsewhere.

The growing importance of temporal logic, in describing distributed systems and reasoning about them, means that the subject should be brought to the attention of a much wider audience. The discipline is still in its infancy and it will be some considerable time before it is mature enough to spawn a proper text which is suitable for teaching purposes. In the mean time this volume provides an introduction to the field which is accessible to the mathematically aware software engineer - but he will have to work hard to master the new concepts.

John Cooke
Loughborough

Rewriting Techniques and Applications, *J.-P. Jouannaud (editor). Academic Press, 1988. 216 pages. Price £12.50. ISBN 012 3909 600*

This is the book of the journal [Jou85] of the proceedings [Jou87]. To be more

precise: selected papers from an international conference entitled "Rewriting Techniques and Applications", held at Dijon in May 1985, were revised, reworked, strictly refereed and assembled into a special issue of the *Journal of Symbolic Computation.* This volume is a reprint of that issue.

It comprises eight papers. The first three (based on invited lectures at the original conference) are surveys/overviews of specific aspects of Term Rewriting and as such would be an excellent springboard for newcomers to the subject, especially research students. Some of the material is rather condensed but it is all very readable and extensive sets of references are included. The topics covered are: critical pairs and completion (by B. Buchberger), Thue Systems viewed as rewriting systems (by R. V. Book) and termination (by N. Dershowitz).

These three papers account for half of the book. The remaining papers are shorter and more specialised. M. Rusinowitch examines the interrelationship between various simplification orderings. J. Hsiang demonstrates the application of term rewriting to the proofs of first order theorems - underlining the potential importance of term rewriting outside its currently perceived boundaries. The paper by K. A. Yelick is perhaps the most noteworthy from the technical standpoint. In it she presents a method for combining equational unification algorithms for certain classes of equational theories. The last two papers (by E. Tiden and S. Arnborg, and D. Benanav et al. respectively) deal with termination and NP-completeness of term rewriting in certain algebraic systems.

Although the material is of a very high quality, there is nothing new here for the term rewriting community - but this is not the publisher's intention. The book is a worthwhile venture of the kind that should be encouraged; promoted adequately it will be of significant benefit to newcomers and as such it will increase interest in term rewriting. Academic and technical libraries ought to have a copy.

References

[Jou85] Jouannaud, J.-P. (ed.): *Rewriting Techniques and Applications,* LNCS 202, Springer-Verlag 1985.

[Jou87] Jouannaud, J.-P. (ed.): *Rewriting Techniques and Applications. J. Symbolic Computation,* 3 (1/2) (1987).

John Cooke
Loughborough

Algorithmic Information Theory, *G. J. Chaitin. Cambridge Tracts in Theoretical Computer Science 1. Cambridge University Press, 1987. 175 pages. Price £20.00. ISBN 0521 343 062*

This book is for various reasons *unusual.* First of all, it is essentially a proof of a single theorem (i.e. Theorem D in Chapter 8). Secondly, the proof itself is a mixed one, i.e. the pure mathematical theory, developed to a large extent by the author himself (see [Cha66, Cha74, Cha87]) is used as a corpus of abstract ideas leading to a program written in a pure version of LISP which produces a huge exponential diophantine equation (of about 900,000 characters and 17,000 variables), the "behaviour" of which *mimics coin tosses.* As a whole, the book is a fascinating report of the author's adventure trying to answer some deep questions such as "what is a random real?" or "is number theory an experimental science?".

The main conclusion (to put it into the author's words) is that there is in number theory "a region in which mathematical truth has no discernible structure or pattern and appears to be completely random.... This doesn't mean that the universe or mathematics are lawless, it means that some times laws of different kind apply: statistical laws".

Gödel and Turing's seminal work concerning the existence of undecidable propositions essentially deals with the question whether an algorithm ever halts or, equivalently, whether an algorithm ever produces any output. Chaitin's idea is to ask a more complex question, namely *whether an algorithm produces an infinite amount of output or not.* Using some clever ideas from [JoM84] and an old theorem of E. Lucas one obtains, in an algorithmic way, an exponential diophantine equation with one parameter p which has an infinity of solutions iff the pth bit of Chaitin's number Ω is 1; here Ω is the halting probability of a universal Turing machine, if an n-bit program has measure 2^{-n}. One proves that Ω is an algorithmically random real in the sense that the first N bits of the base-two expansion of it cannot be compressed into a program shorter than N bits, from which it follows that the initial bits of Ω mimic the result of independent tosses of a fair coin. Moreover, an N-bit program cannot compute the positions and the values of more than N scattered bits of Ω, not just the first N bits. Encoding the halting probability Ω into an exponential diophantine equation with a parameter p one deduces that no formal axiomatic theory is capable to settling whether the number of solutions of the equation is finite or infinite for more than a finite number of values of p.

It is worth mentioning the great difference between asking, for an arbitrary diophantine equation, whether or not a solution exists rather than whether there are infinitely many solutions. Both questions are undecidable, but the latter is "more undecidable" than the former. The Algorithmic Information Theory offers a brilliant explanation of this fact. Indeed, if one considers an arbitrary diophantine equation with one parameter p and asks whether or not there is a solution for $p = 0, 1, \ldots, N-1$, then the N answers to these N questions *are not independent* (due to the fact that the sets of solutions and of solvable diophantine equations are recursively enumerable it follows that we can determine - in an algorithmic way - which equations have a solution when p ranges from 0 to $N-1$ in case we know *how many* of them are solvable), so these answers constitute only $\log_2 N$ bits of information. In contrast, considering an appropriate diophantine equation and asking whether it has an infinite or finite number of solutions *we get N independent answers.*

From a Computer Science point of view the monograph is especially interesting in the innovations pertaining to Chaitin's version of LISP (only atoms are allowed to be one character long, EVAL (LISP universal function) must not lose control by entering into an infinite loop and the only way a syntactically valid LISP expression can fail to have a value is if it loops forever), the way in which register machines are simulated and the lower bound (of 127/128 ths) obtained for the particular construction of Ω. The detailed study of binary random sequences (including a complexity-theoretic characterisation) as well as various information theoretic forms of Gödel's incompleteness theorem represent the mathematical side of the book, to a large extent the author's own contributions.

Chaitin's monograph is a great success and we cannot help admiring it and highly recommending it to experts in the field. (This reviewer has worked hard to present Chaitin's basic results to undergraduate students in Computer Science at the Faculty of Mathematics of the University of Bucharest, and this work is

still in progress during the writing of this review.) Let us add that Chaitin's seminal results are discussed in some monographs [Fin73, Sch71, Cal88] or in articles for a larger audience [Dar78, Cha88, Del88, Ste88, CaM89]. At the same time the reader must be warned about the style of Chaitin's book (which is close to the style of his research paper [Cha74, Cha87]), many proofs are incomplete and little effort is paid to clarify the details. For example, the key Theorem I2 deserves a more accurate statement and a formal proof. Theorems I4b and I8 have unsatisfactory proofs (note that the set of canonical programs is immune). In a sense, his own statement concerning pure LISP is valid for the entire book. "This chapter can be quite difficult to understand, especially if one has never programmed in LISP before... Initially the material will seem completely incomprehensible, but all of a sudden the pieces will snap together into a coherent whole". In spite of the fact that we don't agree with [Gacnd] we think that the omission of some basic papers in the field (see [Kol65, Kol68, Mar66, Fin73, Sch71]) is a shortcoming of the book. Finally, the absence of Chaitin's famous equation from the book (due to its huge dimensions) begs for further investigations.

References

[Cal88] Calude, C.: *Theories of Computational Complexity*, Annals of Discrete Mathematics 35, North-Holland, 1988.

[CaM89] Calude, C. and Malitza, M.: The Impact of NIT's on Higher Education. In: Calude C. and D. Chitoran and M. Malitza (eds), *The Introduction of New Information Technologies in Higher Education*, pp. UNESCO, CEPES, Bucharest, 1989.

[Cha66] Chaitin, G. J.: On the Length of Programs for Computing Finite Binary Sequences. *J. ACM*, 13, 547-569 (1966).

[Cha74] Chaitin, G. J.: Information-Theoretic Limitations of Formal Systems. *J. ACM*, 21, 403-424 (1974).

[Cha87] Chaitin, G. J.: Incompleteness theorems for random reals. *Adv. Appl. Math.*, 8, 119-146 (1987).

[Cha88] Chaitin, G. J.: Randomness in arithmetic. *Scientific American*, 2567, 860-862 (1988).

[Dav78] Davis, M.: What is a computation? In: L. A. Stein (ed.), *Mathematics Today*, Springer-Verlag, New York, 1978.

[Del88] Delahaye, J. P.: Un problème d'arithmétique élémentaire à jamais insoluble. *La Recherche*, 200, 860-862 (1988).

[Fin73] Fine, T. L.: *Theories of Probability: an Examination of Foundations*, Academic Press, New York, London, 1973.

[Gacnd] Gacs, P.: Review of [Cha87], *Mathematical Reviews* 88h: 68038.

[JoM84] Jones, J. P. and Matijasevich, Yu. I.: Register Machine Proof of the Theorem on Exponential Diophantine Representation of Enumerable sets. *J. Symbolic Logic*, 49, 818-829 (1984).

[Kol65] Kolmogorov, A. N.: Three Approaches for Defining the Concept of "Information Quantity". *Problems of Information Transmission*, 1, 3-11 (1965).

[Kol68] Kolmogorov, A. N.: Logical Basic for Information Theory and Probability Theory. *IEEE Trans.*, IT14, 662-664 (1968).

[Mar66] Martin-Löf, P.: The definition of random sequences, *Inform and Control*, 9, 602-619 (1966).

[Sch71] Schnorr, C. P.: *Zufälligküeit und Wahrscheinlichkeit. Eine algorithmische Begrundung der Wahrscheinlichkeitstheorie*, Springer-Verlag, Berlin, Heidelberg, New York, 1971.

[Ste88] Stewart, I.: The ultimate in undecidability. *Nature*, 232, 115-116 (1988).

Crustian Calude
Bucharest

Logic and Computation: Interactive Theorem Proving with Cambridge LCF, *L. C. Paulson. Cambridge Tracts in Theoretical Computer Science 2. Cambridge University Press, 1988. Price £27.50. ISBN 0521 346 320*

LCF is the name of a logic, the Logic of Computable Functions, developed by Vance Scott in 1969. This logic is a logic for a polymorphic lambda calculus and allows the treatment of higher order computable functions, non-termination and lazy computation. In order to perform proofs in this logic, Robin Milner created the Stanford LCF system (an interactive theorem prover). Although the system permitted the development of proofs the system had no capabilities for recording/coding proof strategies; each proof had to be performed by the user at the basic level supported by Stanford LCF. Milner decided to produce a theorem prover for LCF in which user strategies could be programmed. The result was Edinburgh LCF.

Edinburgh LCF may be seen as a watershed in terms of directions in theorem proving support. Its most radical departure was the notion that proofs are developed with a machine not as simply a proof tree, or as a series of lemmas and theorems but as strategies or tactics for constructing the desired proof. These tactics can be thought of as the inverse of proof rules. Whereas proof rules take lists of theorems (the antecedents) to a theorem (the conclusion), tactics take a goal to a list of goals and a validation, where a goal is a formula that we desire to prove to be a theorem and a validation is a (forward) proof rule. Goals are broken down into simpler goals until we find that the goals correspond to axioms or already proved theorems. The validations are then composed to produce the desired proof of the goal.

The second major innovation in Edinburgh LCF was the programming language ML that was developed for the purpose of programming the tactics. ML has become an object of development in its own right in the revised form, SML, a rather clean and very powerful general purpose programming language. ML, like LCF, permitted the introduction of new types and the use of higher order functions, and these became a theme in the development of Edinburgh LCF and even more so in Cambridge LCF.

Cambridge LCF is an update of Edinburgh LCF. The tactical programming language has changed from ML to SML and the logic LCF has been rounded out to include logical connectives missing (but essentially definable) in Edinburgh LCF.

Larry Paulson's book is about the basis and use of the Cambridge LCF system. It starts with a brief survey and history of LCF like systems and much of the foregoing can be found therein (with, I hasten to add, a much more thorough and detailed explanation). It then moves on to three chapters on the basics of the LCF approach to theorem proving. The first of these, chapter 2, is an introduction to formal proofs in first order logic. Chapter 3 extends Chapter 2 by introducing the lambda calculus and the logic of computable functions including fixed point induction and Chapter 4 introduces structural induction as a derived concept from fixed point induction. Both Chapters 3 and 4 introduce sufficient domain theory to justify the inference system development there.

This first part of the book should answer anyone's curiosity about what LCF as a logic is all about and what pencil and paper proofs in LCF could be like. Indeed had this part alone been published the book would still be a valuable reference on the logical system.

The second part (Chapters 5–10) of the book introduces Cambridge LCF as a system. This part is suitably detailed and contains sufficient exercises to allow someone to start using LCF to gain a feel for the system. There is little that can be said here about the book in detail for the general reader of this review, except that this section does describe the paradigm of tactical reasoning in detail and is well worth examination by any person interested in theorem proving.

For the expert I must comment on Larry Paulson's Chapter 9 on re-writing and simplification. Re-writing is the basic workhorse of Cambridge LCF and Chapter 9 shows the development of an approach, similar to tactics, for re-writing strategies. This approach allows the construction of new safe rewriting strategies and their combination into "larger" re-writing strategies in a completely analogous way to the production of tactics in the system. This more than anything else must be credited as a major development in the Cambridge LCF system.

If I have any quibbles about the book at all, they are trivial ones of style. Occasionally Larry's English is somewhat terse and sometimes an in-line comment on the connection of a topic with, say, category theory, should perhaps have been relegated to a footnote. However, this cannot detract from the fact that this book will be of great value to anyone interested in theorem proving and indeed is a much needed book if the world is going to learn more about the LCF approach to theorem proving.

Will Harwood
Cambridge

Lambda-Calculus, Combinators and Functional Programming, *G. Revesz. Cambridge Tracts in Theoretical Computer Science 4. Cambridge University Press, 1988. Price £20.00. ISBN 0521 34589 8*

This book is about efficient graph reduction machines and along the way takes in the topics given in the title. It is of interest to those looking for a text which gives a theoretical basis for the implementation of functional languages but of less interest to those wishing to obtain an understanding of the topics given in the title. Mathematicians will find it of interest to see how the topics are being applied while functional programmers should read it to get a better understanding of what they are doing.

The prerequisites for the book are a general background in programming and mathematics with no particular knowledge requirements. The subject is developed clearly with discussion, examples and exercises (but no solutions). In the earlier part of the book a few forward references would have helped with motivation. For example, the Church–Rosser theorem is introduced on p. 25 and is vital to the parallel graph reduction algorithm given in Chapter 7; this is only obvious if you know it is. However, given the general clarity of the book, some interesting biographical notes to start discussions, some good exercises and one short but important bibliography, the book could form the basis of a second year undergraduate course. The author has implemented his system on IBM PC; given that the system was available, I am sure a course would be enjoyed by students.

Chapter 1 sets the scene and provides good motivation for the work by relating it to developments in programming languages and mathematics. Chapter 2 introduces the lambda-calculus from a syntactic point of view (but the syntax is slightly non-standard) and introduces the idea of reductions. It is here that the whole flavour of the book is set with computation efficiency being stressed throughout. Chapter 3 extends the pure notation by introducing combinators and *if then else*,

Arithmetic on numerals with simple operators is built to provide further syntactic conveniences. Recursion is discussed and the comforting notions of the Curry paradox and the need for the *eta* rule are introduced. The ideas of computational efficiency and syntactic convenience are furthered in Chapter 4 when lists (as in FP) are introduced. Evaluation of simple and infinite list operations motivates the idea of lazy evaluation. In Chapter 5 semantics are introduced (as reduction to normal form) with examples from FP and Miranda.

At this point (p. 112 of 170 pp.) the theory is developed and we can down to some serious hacking. A sequential implementation of the PC system mentioned above is developed by considering program structure and graph reduction. A machine code is designed and evaluation by normal order reduction is described. Again considerations of efficiency lead the argument. The final chapter develops a shared memory parallel implementation and discusses the allocation of work to processors. A neat new garbage collection algorithm for cyclic structures should be of interest beyond the particular confines of graph reduction.

If you or your students are interested in functional programming, I suggest you look at this book as it may well be suitable. The author does not try to be comprehensive but suceeds in producing a good computationally motivated introduction to the subject.

Dan Simpson
Brighton

Software Productivity, Harlan D. Mills. *Dorset House Publishing Company, 1988, Price £19.95. ISBN 932 633 102.*

This book is a collection of papers produced by Harlan Mills during the years 1968 to 1981. The papers have been simply bound together with no attempt to provide any continuity nor to link them in any way. This is a pity.

The papers were written to explain various important topics of the time and to exhort IBM management in particular, and the world in general, to use the best available ideas to improve the quality of software. In presenting the papers as is done here much of the impact is lost. The same examples appear over and over again and a crisply made message loses its impact on a fourth or fifth reading.

The papers concentrate on two ideas which were current in the 1970s, structured programming and chief programmer teams. Structured and top down programming are given both mathematical and management treatments. The papers on programmers and their organisation discuss how such ideas could help IBM. It is unfortunate that the collection stops before we get to the cleanroom and an evaluation of its effects on productivity.

Throughout the book we find Mills asking questions which are still valid today, and no doubt will also be valid tomorrow. In the earliest paper he points out that we need "a systematic set of properties which defines the value of a program" a search which guides all the work on software quality today. His 1974 paper on "How to buy Quality Software" is still state of the art for most software procurers. His 1980 paper "Software Education Education" should be read by many of those purporting to convey software engineering course. Even a seemingly dated report such as "0S/360 Programming" has good advice for Unix hackers. Many of todays students will ask of the book "What's all the fuss about?"; it is a tribute to Mills' efforts that much of what he proposed is now generally accepted as good practice. But it must be admitted that one has to search through a lot of wood to find the important trees and it is easy to lose motivation in so doing.

The book is an interesting historical document but the information could have been presented in a quarter of the bulk.

Most readers of this journal will be in sympathy with Mills' view that computing will only progress when it builds upon a sound mathematical basis and his insistence that "packaged" methodologies are no substitute for thinking. When read individually the papers make these points well but lose much of their impact when presented in this volume; the rapier has been replaced by a blunt instrument.

Dan Simpson
Brighton