© 2001 Cambridge University Press

Book reviews

Computational Logic by Ulrich Berger and Helmut Schwichtenberg, editors, Springer-Verlag, 1999, 444 pp.

This volume is based on the lectures given at the Marktoberdorf 1997 summer school, with chapters covering a range of topics of which some (but not all) will be of interest to functional programmers.

Abramsky and McCusker begin the book with an introduction to recent work on application of game semantics to programming languages. The essential problem here is that traditional semantic techniques fail to characterise the computational processes that arise in computing functions of a language such as PCF. Game semantics offers solutions, here given for PCF and its extensions with state and control; other aspects such as recursive types have been covered elsewhere, and current work is addressing other features such as concurrency.

Aczel's chapter on the simply typed lambda calculus is a very clear exposition of standard material, covering first the three standard kinds of inference system for implicational logic, then the untyped lambda calculus, and finally, the Curry style version of the typed calculus, i.e. that consisting of rules for typing untyped terms, rather than Church style, for forming terms that bear type annotations. The main novelties here are (i) a precise definition of the notion of 'formal system', (ii) the use of an infinite set **U** that equals the set $\mathbf{U}^{\mathbf{u}}$ of all unary functions on itself, with a conjecture that although this is inconsistent with classical set theory, it is consistent with an intuitionistic theory of non-well-founded sets, and (iii) a general framework for weak and strong normalisation proofs.

Barendregt briefly presents the basic background on pure type systems, including his well-known cube of such systems, i.e. fragments of the calculus of constructions presented in a uniform fashion, followed by an interesting collection of exercises.

Benl and Schwichtenberg discuss the formal correctness of functional programs. Among the various options, (1) of representing an algorithm by a term and proving its correctness w.r.t. a specification, (2) of extracting a program from a constructive existence proof, and (3) of using a non-constructive existence proof, either directly or after constructivisation (e.g. by Friedman's A-translation), the authors choose a combination of (1) and (2). Dijkstra's algorithm for computing shortest paths in a weighted graph is used as a case study.

Buss gives a detailed introduction to the complexity of several propositional proof systems, including those based on cutting planes, and of Craig interpolations. As is well known, these are topics with connections to questions such as $P = NP\,?$ and $NP = co - NP\,?$. The main systems considered are Frege–Hilbert systems (those with schematic axioms and schematic inference rules, such as Modus Ponens) in which proof size and worst-case analysis are taken to be of greater importance than the average cost of proof search.

Constable describes, in the most interesting chapter of the volume, a formalisation of theorems about decidable properties of finite automata, faithful to the classic 1969 textbook by Hopcroft and Ullman. Since the theory is based on a primitive notion of computability, absent from classical mathematics without prior development of automata theory or its equivalent (and arguably not even then), this imposes a requirement for constructive mathematics (e.g. in the style of Bishop or Martin-Löf), in which the primitive notion of *function* allows the use of the notion of *effective procedure*.

Girard, in his stimulating and provocative chapter on the 'Meaning of Logical Rules I:

Syntax versus Semantics', gives a typically personal disquisition, combining philosophical discussion (e.g. on the semantics of "semantics") and technical details, on sequent calculus, phase semantics, coherent spaces, proof nets and logical time. (For an exact exposition of what the author means by 'logical time', one must look elsewhere; at any rate, he expressly states that it has nothing to do with that "bleak bureaucracy known as temporal 'logics'".) The gist of the chapter is that the traditional logical issues of completeness/soundness and syntax/semantics should be completely re-examined, with meaning sought not in the world of denotation (Frege's *Bedeutung*), but in the world of sense (*Sinn*), in dynamics rather than in kinematics, and given by logical systems with cut-elimination and the subformula property, summed up by the slogan that "The meaning of logical rules is to be found in the rules themselves".

Restriction of recursion to primitive recursion is one way to ensure termination of functional programs, but this still leads to non-feasible algorithms. Handley and Wainer give a clear exposition of work by, for example, Leivant, Sieg and Wainer, Simmons and Bellantoni and Cook, showing that in arithmetic certain natural restrictions of primitive recursion lead to logical characterisations of complexity classes such as polynomial time and linear space. Such restrictions (called 'predicative' or 'tiered' recursion) begin with a distinction between 'input' and 'output' variables, allowing the standard recursive definition of addition, requiring a modified definition of multiplication, but disallowing any definition of exponentiation.

Martin, my close colleague and thus exempt from my criticism (and my flattery), examines the impact of computer-aided formal reasoning on mathematical practice, emphasising the slightness of the role of computational logic in the mathematics of the twentieth century, observing the unsympathetic attitude of mathematicians to formal proof development and identifying realistic research topics in computer-aided reasoning, such as the use of theorem-proving techniques to enhance or extend mathematical software systems.

Meseguer presents 'Rewriting Logic', in which formulae correspond to system states and proofs to concurrent computations; thus the rule $t \to t'$ can be interpreted both as the evolution of a part t of a system state into t' and as the derivability of the formula t' from the formula t. Applications are given to (for example) several models of concurrency; Petri nets, the π-calculus, CCS, the chemical abstract machine, dataflow models, artificial neural networks, and so on. There is an extensive overview of languages based on rewriting logic; these include executable specification languages and logical frameworks. Directions for further research, including just about every aspect of computer science, are outlined.

Finally, Miller gives an overview of sequent calculi as a basis for logic programming and goal-directed proof search, as exemplified by the languages lambda Prolog, Lolli and Forum based on intuitionistic and linear logic. This paradigm of *computation as deduction* has in common with functional programming the idea that computation is search for normal form proofs, but differs in allowing non-determinism and backtracking. Moreover, it shows that resolution is a poor basis for understanding logic programming. Forum is a special case, being suitable not for program execution but for specification. Incidentally, one might add that this is really computation as *problem analysis* rather than *solution synthesis* (i.e. forward reasoning, as suggested by the word 'deduction') .

The articles are all of high quality and well-presented, with extensive suggestions for further reading.

Roy Dyckhoff

Communicating and Mobile Systems: the π-calculus by Robin Milner, Cambridge University Press, 1999

The Calculus of Communicating Systems is a theory of computation, where the basic concepts are taken to be multiple processes, communicating along shared synchronous channels. Its theory was developed by Milner and others in the 1980s, and was presented in Milner's 1989 textbook (Milner, 1989).

The 1990s saw Milner and others examining a feature missing from CCS: that of *scoping* of channel names. In CCS, scope is very simple, because names cannot escape from their enclosing context. This produces a very clean theory, as an extension of the theory of automata, but is of restricted applicability. By allowing channel names to be communicated between processes, and treating name scope seriously, Milner, Parrow and Walker introduced the π-calculus (Milner *et al*., 1992), which is the subject of this book.

The π-calculus has been enormously influential, both in terms of work on the π-calculus directly, and also in terms of language semantics which uses techniques from the π-calculus. Language features which have been studied using π-like techniques include cryptography, object-oriented languages, distribution, concurrent higher-order languages, pointers, heap regions and exceptions.

This book is not directly related to functional programming, except in one small section on coding the λ-calculus into π. The techniques described have been used to model parts of functional languages such as Concurrent ML or Haskell, and are now part of the toolkit for providing operational models of higher-order languages. As such, the material is of great interest to the more theoretically-minded functional programmer.

This book is divided into two parts: Part I develops the theory of communicating systems, and Part II extends the theory to deal with names escaping from their static scope.

Part I is a condensed presentation of many of the concepts from Milner (1989). The main difference is that communicating processes are presented as a natural extension of the existing theory of automata. Rather than presenting labelled transition systems directly, first the standard theory of automata is presented, and then labelled transition systems are given as automata where all states are accepting states. This makes it much easier to see communicating systems as arising naturally as an extension of sequential computing, rather than as a completely separate entity.

Having presented labelled transitions as a particular form of automata, Milner than presents bisimulation as a refinement of language equivalence. This is motivated by observing that language equivalence is insensitive to deadlock (obviously not a good property for a theory of concurrency).

The other main change from Milner (1989) is the presentation of a reduction semantics for concurrency, in the style of the Chemical Abstract Machine. This is now the standard presentational technique for π-like languages, and its use in Part I makes the presentation in Part II simpler.

Part I is a good introduction to the theory of communicating systems, containing the core of Milner's previous textbook, and making clearer the connections to both automata theory, and to the π-calculus.

The material in Part II has not been published in book form before, although much of it has appeared in journal papers and book chapters (Milner *et al*., 1992; Milner, 1991). Readers familiar with the π-calculus will find few surprises here, but it is very useful to have the material gathered into one place, and given as one consistent presentation.

Part II begins with a discussion of the term 'mobility', and in particular notes that this word now has a different meaning in computer science than it did ten years ago. In particular, the book does not discuss mobile networks, or mobile agents, but instead looks at link mobility, where the connections between processes move. There are distributed extensions of π which model process mobility as well, but these are beyond the scope of this book.

The π-calculus is then presented as an extension to the theory of concurrent systems to include channel-passing. There are a number of examples of the expressibility of π, for example coding datatypes as processes, and coding functional and object-based languages into π. This material will probably be the part of most interest to the *Journal of Functional Programming* audience, although since it is only one chapter, it can only give a taste of the body of work on π-like treatments of higher-order languages.

The remainder of Part II shows how the theories of strong and weak bisimulation carry over to π.

Part II is a good introduction to the world of π, collecting together material into one accessible source. This book is a textbook for an undergraduate course, though, so experienced researchers will want to read more: this gap may be filled by Walker and Sangiorgi's forthcoming π-calculus book, which has a research audience more firmly in mind.

The original audience for this book is slightly unusual, in that it is based on notes for a ten-week undergraduate lecture course at Cambridge. Many people will find the material too advanced for undergraduates, too theoretical for master's students, and not challenging enough for PhD students to use as a sole textbook. However, it would make a fine book for a PhD course, if it was supplemented with research papers, or for a researcher looking for an introduction to the field.

This book collects together some ground-breaking material from the early 1990s, which has since spawned an industry of π-like languages. It will provide a good introduction to the field for advanced undergraduates, PhD students or researchers wishing to dip into the world of the π-calculus.

References

Milner, R. (1989) *Communication and Concurrency*. Prentice-Hall.

Milner, R. (1991) The polyadic π-calculus: a tutorial. *Proc. International Summer School on Logic and Algebra of Specification*, Marktoberdorf.

Milner, R., Parrow, J. and Walker, D. (1992) A calculus of mobile proceses. *Infor. & Comput.* **100**(1), 1–77.

Alan Jeffrey