Preface

A collection of papers and memoirs celebrating the contribution of Rod Burstall to advances in Computer Science

These issues of *Formal Aspects of Computing* are dedicated to Professor Rod Burstall, and, as a collection of papers, memoirs and incidental pieces, form a Festschrift for Rod. The contributions are made by some of the many who know Rod and have been influenced by him. The research papers included here represent some of the areas in which Rod has been active, and the Editors thank their colleagues for agreeing to contribute to this Festschrift.

In this Preface, we attempt to summarize Rod Burstall's scientific achievements. In addition, we describe the personal style and enthusiasm that Rod has brought to the subject.

Introduction

In the development of our understanding of computing, Rod's work has been truly pioneering and seminal. Not only has Rod contributed to the early development of a wide range of topics, but he has also branched out into new and often quite unorthodox territories with ideas that are striking in their originality. He has never ploughed a single furrow for long, rather taking a new technical direction every few years and leaving the further development to others. Some of these forays developed into what are now large areas of research. There are perhaps other pioneering ideas in his work which are yet to develop into scientific maturity.

Rod's research is not only founded in practice — in the actual practice of writing programs, designing languages, doing proofs etc. — but is at a point where theory and practice often cannot and should not be distinguished. This close interplay has been a constant characteristic of his work and often results in a very distinctive approach to research topics.

Rod's idea of "doing research" is a delight: a very human occupation — real fun and a form of intellectual play. No-one who has been closely involved in research with Rod can forget the experience, and the enthusiasm and pleasure that he brings to the activity. Also important is the sense that research is a community activity, shown in his fostering (sometimes almost father-like) of his research group, and in his concern about the wider community of researchers.

Trying to summarise Rod's scientific contribution has not been easy. His interests range widely, and yet the interplay of constant themes and aims is always apparent. Indeed, Rod himself may well not appreciate the notion of trying to "classify" his work into areas when it is so obviously an organic whole. However, in order to describe the range of his activity we will necessarily have to lapse into describing topics and areas of development.

In the following summary, numerical references refer to the bibliography at the end of this preface; name references refer to papers and articles included in the Festschrift.

Scientific contribution

Rod's early work and his Ph.D. topic was in Operational Research, in particular developing heuristics for solving specific problems, and implementing these as computer programs. These early ideas are reflected in his two major areas of research: Artificial Intelligence and what one could broadly describe as Software Development.

Much of Rod's early work at Edinburgh was in Artificial Intelligence in a group headed by Donald Michie who had worked with Turing during World War II at Bletchley Park. This was an exciting era in the subject with a host of ideas, systems, and languages being developed, and a fruitful interplay with areas such as linguistics and psychology. One of Rod's major contributions during this period was leading the team that programmed Freddy, the first hand-eye assembly robot. As part of this he developed an early vision recognition program which computed partial isomorphisms between relational structures, and also contributed to natural language processing. The style of his work at this time is expressed well by the following quote from his paper [19] in 1970:

> "we are coming to a better understanding of the algebraic and logical principles underlying the problems which have been attempted so far. I am thinking here of the use of first order logic in question answering and problem solving programs, of a more rigorous set-theoretic approach in describing puzzle solving and game playing programs, [and] of studying inductive generalisation in the context of logic and automata theory"

Much of his effort during this period was devoted to the development of the POP-2 programming language (see below), which was directly based on emerging ideas about underlying principles of programming languages and was used for all AI programming in Edinburgh at that time.

During this period, Rod became interested in a mathematical approach to software development. In those days (the mid-1960s) this was a radical idea and the links between mathematics and program design were recognised by very few. Amongst them were Peter Landin and Christopher Strachey. Rod met Peter Landin through chance acquaintance in a London bookshop, and was later introduced to Christopher Strachey who was Landin's employer at the time. Their informal and almost clandestine meetings and their subsequent influence on Rod's work are engagingly described in [72] (and see [Landin] for Peter Landin's own comments). Through these meetings, Rod was introduced to new and exciting ideas, including universal algebra and early ideas about functional programming, which led him to a series of fundamental innovations in the development and application of mathematics to computer programming. If one were to summarise the main theme of this work, it lies in the dream that we ought to be able to say in a mathematical language what a program is supposed to do and then prove that it does indeed do this. Behind this simple aim, however, lies a formidable requirement: to develop new mathematics — especially in mathematical logic, abstract algebra and topology — and to use this to invent new approaches to programming and to understanding programming languages.

This then sets the scene for Rod's major contributions. Rod was among the first to study the problem of proving properties of programs. He was the earliest to recognize the central role of structural induction in establishing properties of data and programs, giving a clear and appealing exposition of this fundamental technique in [15]. He developed the so-called "intermittent assertion" method of program proving in [27], in which an assertion attached to a point in a program means that control will *sometime* pass that point and satisfy the assertion, rather than satisfying the assertion *whenever* it (perhaps never) passes that point, as advocated by Floyd, Hoare and Naur; this allows total correctness to be established with a single proof. This paper was the first to point out the connection between program proof and modal logic, an idea that was developed by Amir Pnueli and many others into what is now a major research area, with particular applications to specifying and reasoning about concurrent systems.

To reason about what programs do, it is necessary to formalise the semantics of the programming language. The mathematical foundations of this area were being developed from the mid-1960s, in particular through the work of Dana Scott and Christopher Strachey. Rod was active in the area, investigating the semantics of languages with a notion of "state" in [8] and showing how to define programming language semantics in first-order logic in [18]. Rod was responsible for the idea, used by Strachey and thereafter, that the store is a mapping from locations ("L-values") to their contents. His student Mike Gordon wrote a denotational semantics for LISP, and Rod encouraged his post-doc Gordon Plotkin [Plotkin] in his early work on denotational semantics.

As part of his quest for correctness of programs, Rod, together with his student John Darlington, undertook the first major work on program transformation, whereby programs whose correctness has been established are transformed into equivalent and possibly more efficient programs through correctness preserving transformations. They isolated a range of these transformations and implemented a partly automated system. This inspired a fruitful worldwide research movement, being viewed as a possible way forward in the development of methods for creating more reliable, provably correct, programs (an example of current research in the area is [Pettorossi and Proietti]).

One of the major themes supporting his work on program correctness is Rod's long-standing interest in novel programming languages. This began with the POP-2 language developed with Robin Popplestone and others in the 1960s, which was a blend of LISP and Algol strongly influenced by Strachey's CPL and early functional programming ideas of Landin (see [Popplestone] for a review of this language). Later, Rod devised an experimental functional language, called Hope (nothing to do with faith and charity: at the time, his research group was located in a warren of small Edinburgh houses in Hope Park Square, named after Thomas Hope, a 18th-century Scottish agricultural reformer). Hope was a small language adapted to correctness proofs and transformations, but it had some useful and novel features, including the elegant and now-familiar combination of algebraic datatypes

```
data tree(alpha) == empty ++ node(tree(alpha),alpha,tree(alpha))
```

with pattern-matching clausal function definitions

```
dec flatten: tree(alpha) -> list(alpha)
--- flatten(empty) <= []
--- flatten(node(t1,n,t2)) <= flatten(t1)<>(n::flatten(t2))
```

as well as a simple module system. The language Standard ML and its close relative CAML — which have become the most widely used of the so-called "strict" or "eager" functional languages — combine features from Hope with those of their precursor, ML, devised by Robin Milner [Leifer and Milner]; Rod was an active member of the Standard ML design team. Another theme in Rod's work is the role of abstraction in programming and related issues around modularity of code. Standard ML incorporates a sophisticated system for describing program modules, devised by Dave MacQueen [MacQueen], who worked on Hope with Rod as a post-doc and was also influenced by Rod's work on modular specifications. Another language that Rod devised, with Butler Lampson, was Pebble ("small and hard"!) that moved programming languages towards type theories, using dependent types for the interfaces of modules. Pebble was an experimental language based on the λ-calculus which, using only a few constructs, could express a wide variety of programming language features: modules, interfaces and implementations, abstract data types, generic types, recursive types and unions.

Logic clearly has a role in program correctness, but so too does algebra, and Rod pioneered the use of algebraic techniques in programming. With Peter Landin, he used algebraic ideas — algebras, homomorphisms, commuting diagrams — to structure proofs of correctness, showing how to use this for the correctness of compilers. The role of category theory is important here as a tool for describing structural aspects at the right level of abstraction. Again Rod pioneered this fusion of two such apparently disparate areas. On one hand, he was an early user of categorical ideas in computing, for example in describing programs as free categories, and interpreting recursive program schemes, and later in devising "specification languages"; on the other, he saw that category theory itself could be implemented as programs. The latter led to the development of Computational Category Theory [60] together with Rod's student David Rydeheard where constructions in category theory are expressed as programs, producing code of unusual abstract functionality. Nowadays, category theory is routinely used to formulate ideas in computing, in particular in the mathematical semantics of programming languages (for examples, see [Robinson] and [Gabbay and Pitts]), but in those early days of the 1970s, it was viewed as a very exotic branch of mathematics and its application to computing was a "leap of faith".

The development of algebraic specification techniques owes much to Rod's long-standing collaboration with Joseph Goguen who had done some of the first work in this area as a member of the so-called ADJ group at IBM. They proposed the first algebraic specification language, Clear, which focussed on the provision of mechanisms for combining specifications of program behaviour. (Many colleagues found the semantics of Clear, expressed in unfamiliar categorical language, complicated and difficult to understand. In reaction, Jacques Loeckx in Saarbrücken later invented a specification language called Obscure.) Then, in trying to describe the semantics at the right level of abstraction, they introduced a notion of abstract logical system which they called "institutions" [49] and [66] (see [Goguen and Roşu]). The aim was to describe how specification modules could be combined or parametrised independently of the nature of the individual modules. This work, and their ideas in [40] about "vertical composability" of refinement steps (i.e., the correctness of *stepwise* refinement) and "horizontal composability" (i.e., compatibility of refinement with specification structure), inspired by two-dimensional categories, have had great influence. This general line of work has been intensively developed by a worldwide community of researchers including Rod's student Don Sannella and post-doc Andrzej Tarlecki [Bidoit, Sannella and Tarlecki].

The interplay between the activities of programming and proving programs correct, and actual code, is again evident in Rod's contributions to the development of automated proof support systems. His work on this began in the late 1960s with [15] and [18] and continued in the early and mid 1970s with theses on the topic by his students Rodney Topor and Raymond Aubin. Later, he developed IPE ("Interactive Proof Editor") in the mid-1980s, and in the 1990s led Randy Pollack [Pollack], Zhaohui Luo and a team of other students and post-docs in the development and use of the Lego proof assistant. Lego implements a number of related type systems including the Edinburgh Logical Framework, Coquand's Calculus of Constructions, and Luo's Unified Theory of Dependent Types, supporting interactive proof development in the natural deduction style and generating explicit proof objects. In this framework, Rod with his student James McKinna investigated

notions of programs packaged with proofs of their correctness, introducing the idea of "deliverables", a precursor to the currently hot topic of "proof-carrying code". Recently he has returned to the topic of making proof fun and easy for novices with his ProveEasy system which he used in teaching logic to undergraduates in the late 1990s.

Much of Rod's work is collaborative with other research workers and with his research students and post-docs, often with long-standing collaborations. Unfortunately, in order to keep this summary to a reasonable length, we have not been able to mention all of the many collaborators in his work. The list of co-authors in the bibliography below documents some of these collaborations.

Influenced by his Buddhism, Rod became interested in aspects of consciousness and our perception of reality in relation to our experience of computing (see [Barendregt] for reflections in this area). We end this summary of Rod's work with a quote from one of his papers on this subject [65]:

> "In working with a computer we interact with a small world, partly of our own creation, in which we have a special role. We are the agent who makes things happen. In this world we play the role traditionally assigned to God. We have complete power and our aim is to control everything that happens in this world. The better we are at programming the more nearly we approach total control over what happens ... Contrast this with an attitude of respect for other inhabitants of our world, other people, animals or forests, a view of the world in which we do not have some distinguished role. In such an attitude we are open to the richness of phenomena over which we have no dominion. Think of sitting on a hillside watching clouds move through the sky, as opposed to sitting at your terminal ...
>
> Computing carries a great deal of energy in our current culture, and fuels our curiosity and inventiveness. But in order to fully enjoy its possibilities we need to appreciate the way it can subtly influence our frame of thought. The recognition of this influence does not free us; but it may provide a starting point for us to look at ways of working with computers without being entrapped by a limited perspective based on our desire for control and exclusive reliance on conceptual thought."

Biographical details

Rodney Martineau Burstall was born in 1934, the son of a draftsman and a housewife from Liverpool. He attended King George V Grammar School at Southport, moved on to King's College, Cambridge, reading Natural Sciences, and then took a Masters in Operational Research at Birmingham University. There followed a period in industry, working on operational research at Bureau Gombert, Brussels and then the Reed Paper Group in Kent. After the first year at Reed, he moved to the programming group having written his first program while working in Brussels.

In 1962 he returned to Birmingham, working as a research fellow at the Department of Engineering Production and taking a Ph.D. on heuristic programming for operational research. During this period — the early 1960s — his interests began to broaden out into various aspects of artificial intelligence and programming. He had met Peter Landin and then Christopher Strachey who introduced him to the then very new mathematical ideas of programming language semantics; this later became a major theme of his work at Edinburgh.

Rod's career at Edinburgh University, where he remained until his retirement, began when Donald Michie invited him to join his Experimental Programming Unit as a research fellow in 1964. He became a lecturer in the Department of Machine Intelligence and Perception in 1967, a Reader in the School of Artificial Intelligence in 1970, and was promoted to a personal chair in Artificial Intelligence in the department of the same name in 1977. In 1979 he moved from the Department of Artificial Intelligence to the Department of Computer Science, joining with Robin Milner, Gordon Plotkin and others to form a very strong group in theoretical computer science. This happened at a time when the theory subdiscipline was becoming more clearly part of computer science and somewhat detached from artificial intelligence. In 1980 his personal chair was retitled "Computer Science".

Rod maintained an impressive group of post-docs, Ph.D. students and visitors, partly supported by a continuous stream of funded projects. He has had visiting positions in the USA, France, Belgium and Japan, and, among others, industrial consultancies at Xerox Parc, IBM Laboratories UK, ICL Kidsgrove and Digital Equipment Corporation's System Research Center. He has been invited speaker at a range of conferences

in Europe, the United States and Japan. He was elected to the Academia Europea in 1989 and the Royal Society of Edinburgh in 1995.

Much of Rod's most important contribution has been informal; he has always been concerned with building up community, whether among his research team, or more widely within the Department of Computer Science and, latterly, the Division of Informatics. One manifestation of this commitment was his role in the formation of the Laboratory for the Foundation of Computer Science at Edinburgh in 1986, serving as its director for three years.

Rod was sustained and supported by his Finnish wife Sissi, whom he married in 1959. They had three daughters. Together they created a warm and welcoming family atmosphere from which generations of his graduate students and post-docs benefited. Sadly, Sissi died in 1990, and his daughter Kaija died in 1994.

As well as academic life, Rod has a long-standing interest in Buddhism as a student and practitioner, taking temporary vows at one stage, and happily combining the life of a university professor with wearing Buddhist robes.

Rod was supported throughout his professional life by his personal secretary, Eleanor Kerse. Eleanor worked with Rod from the mid-1960s until his retirement, striving to keep him organized (a futile enterprise, her superhuman efforts notwithstanding!) and protecting him from the demands of the outside world. Eleanor has kindly contributed her personal reflections to this Festschrift in rhyme [Kerse].

Rod retired from the University of Edinburgh in April 2000. He now lives mostly in France, near Limoges, travelling widely and visiting Edinburgh University from time to time.

Acknowledgements: Our thanks to Eleanor Kerse and Gordon Plotkin for help. Most of the biographical information is extracted from an article by Gordon.

David E. Rydeheard
Donald T. Sannella

Full bibliography

1. R.M. Burstall, R.A. Leaver and J.E. Sussams. Evaluation of transport costs for alternative factory sites — a case study. *Operational Research Quarterly* 13:345–354 (1962).
2. R.M. Burstall and G.R. Kiss. Information processing language V for the KDF-9 computer. *Automatic Programming Information* 18 (1963).
3. R.M. Burstall. Heuristic and decision tree methods on computers: some operational research applications. Ph.D. thesis, University of Birmingham (1966).
4. R.M. Burstall. Computer design of electricity supply networks by a heuristic method. *Computer Journal* 9(3):263–274 (1966).
5. R.M. Burstall and J.V. Oldfield. A software stack system for the 340/347 display. *Decuscope* 5(3):1–3 (1966).
6. R.M. Burstall. A heuristic method for a job-scheduling problem. *Operational Research Quarterly* 17:291–304 (1966).
7. R.M. Burstall. Tree searching methods with an application to a network design problem. *Machine Intelligence 1* (eds. N.L. Collins and D. Michie). Edinburgh: Oliver and Boyd, 65–85 (1967).
8. R.M. Burstall. Semantics of assignment. *Machine Intelligence 2* (eds. E. Dale and D. Michie). Edinburgh: Oliver and Boyd, 3–20 (1968).
9. R.M. Burstall, J.S. Collins and R.J. Popplestone. *POP-2 Papers.* Edinburgh: Oliver and Boyd (1968).
10. R.M. Burstall and R.J. Popplestone. POP-2 reference manual. *Machine Intelligence 2* (eds. E. Dale and D. Michie). Edinburgh: Oliver and Boyd, 205–246 (1968). Also in [9].
11. D. Michie, A. Ortony and R.M. Burstall. *Computer programming for schools: first steps in Algol.* Edinburgh: Oliver and Boyd (1968).
12. R.M. Burstall and J.S. Collins. An introduction to the POP-2 programming language. In [9] (1968).
13. J.S. Collins, A.P. Ambler, R.M. Burstall, R.D. Dunn, D. Michie, D.J.S. Pullin and R.J. Popplestone. Multi-POP/4120: a cheap on-line system for numerical and non-numerical computing. *Computer Bulletin* 12(5):186–189 (1968).

14. R.M. Burstall. Writing search algorithms in functional form. *Machine Intelligence 3* (ed. D. Michie). Edinburgh University Press, 373–385 (1968).
15. R.M. Burstall. Proving properties of programs by structural induction. *Computer Journal* 12(1):41–48 (1969).
16. R.M. Burstall. A program for solving word sum puzzles. *Computer Journal* 12(1):48–51 (1969).
17. R.M. Burstall and P.J. Landin. Programs and their proofs: an algebraic approach. *Machine Intelligence 4* (eds. B. Meltzer and D. Michie). Edinburgh University Press, 17–43 (1969).
18. R.M. Burstall. Formal description of program structure and semantics in first order logic. *Machine Intelligence 5* (eds. B. Meltzer and D. Michie). Edinburgh University Press, 79–98 (1969).
19. R.M. Burstall. Machine intelligence research at Edinburgh University. *Proc. ACM Intl. Computing Symposium*, Bonn, 696–708 (1970).
20. R.M. Burstall, J.S. Collins and R.J. Popplestone. *Programming in POP-2.* Edinburgh University Press (1971). A revision of [9] incorporating much new material.
21. H.G. Barrow, A.P. Ambler and R.M. Burstall. Some techniques for recognising structures in pictures. *Proc. Intl. Conf. on Frontiers of Pattern Recognition*, Honolulu. Academic Press, 1–29 (1972).
22. R.M. Burstall. An algebraic description of programs with assertions, verification and simulation. *Proc. ACM Conf. on Proving Assertions About Programs*, Las Cruces. *SIGPLAN Notices* 7(1):7–14 (1972).
23. R.M. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence 7* (eds. B. Meltzer and D. Michie). Edinburgh University Press, 23–50 (1972).
24. D. Michie, A.P. Ambler, H.G. Barrow, R.M. Burstall, R.J. Popplestone and K.J. Turner. Vision and manipulation as a programming problem. *Proc. 1st Conf. on Industrial Robot Technology*, Nottingham, 185–190 (1973).
25. A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall and R.J. Popplestone. A versatile computer-controlled assembly system. *Proc. 3rd Intl. Joint Conf. on Artificial Intelligence*, Stanford, 298–307 (1973).
26. J. Darlington and R.M. Burstall. A system which automatically improves programs. *Proc. 3rd Intl. Joint Conf. on Artificial Intelligence*, Stanford, 479–485 (1973).
27. R.M. Burstall. Program proving as hand simulation with a little induction. Invited paper, *Proc. IFIP Congress '74*, Stockholm, 308–312 (1974).
28. R.M. Burstall and J.W. Thatcher. The algebraic theory of recursive program schemes. *Proc. 1st Intl. Symp. on Category Theory Applied to Computation and Control*, San Francisco, 1974. Springer Lecture Notes in Computer Science, Vol. 25, 126–131 (1975).
29. R.M. Burstall and J. Darlington. Some transformations for developing recursive programs. Invited paper, *Proc. Intl. Conf. on Reliable Software*, Los Angeles, 465–472 (1975).
30. A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall and R.J. Popplestone. A versatile system for computer-controlled assembly. *Artificial Intelligence* 6(2):129–156 (1975). Based on [25].
31. J. Darlington and R.M. Burstall. A system which automatically improves programs. *Acta Informatica* 6:41–60 (1976). Revised version of [26].
32. H.G. Barrow and R.M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters* 4(4):83–84 (1976).
33. R.M. Burstall. Program proof, program transformation, program synthesis for recursive programs. Lecture notes for course on "Data and program structures: syntax and semantics". International School on Theory and Application of Computers, Erice, Sicily (1976).
34. R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery* 24(1):44–67 (1977). Based on [29].
35. R. Burstall and J. Goguen. Putting theories together to make specifications. Invited paper, *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Massachusetts, 1045–1058 (1977).
36. R.M. Burstall. Design considerations for a functional programming language. Invited paper, *Proc. Infotech State of the Art Conf. "The Software Revolution"*, Copenhagen, 45–57 (1977).
37. R.M. Burstall and M. Feather. Program development by transformations: an overview. *Les fondements de la programmation. Proc. Toulouse CREST Course on Programming* (eds. M. Amirchahy and D. Neel). Le Chesnay: IRIA-SEFI (1978).

38. R.M. Burstall. Concepts versus code in program design. *Proc. Euro IFIP '79*, London. North-Holland (1979).
39. R.M. Burstall and J.A. Goguen. The semantics of CLEAR, a specification language. Proc. 1979 Winter School on Abstract Software Specifications, Copenhagen. Springer Lecture Notes in Computer Science, Vol. 86, 292–332 (1980).
40. J.A. Goguen and R.M. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Technical Report CSL-118, SRI International, Menlo Park (1980).
41. J.L. Weiner and R.M. Burstall. Making programs more readable. *Proc. 4th Intl. Symp. on Programming*, Paris. Springer Lecture Notes in Computer Science, Vol. 83, 372–341 (1980).
42. R.M. Burstall. Electronic category theory. Invited paper, *Proc. 9th Intl. Symp. on Mathematical Foundations of Computer Science*, Rydzyna. Springer Lecture Notes in Computer Science, Vol. 88, 22–39 (1980).
43. R.M. Burstall, D.B. MacQueen and D.T. Sannella. HOPE: an experimental applicative language. *Proc. 1980 LISP Conference*, Stanford, 136–143 (1980).
44. R.M. Burstall and J.A. Goguen. An informal introduction to CLEAR, a specification language. *The Correctness Problem in Computer Science* (eds. R. Boyer and J. Moore). Academic Press 185–213 (1981). Republished in: *Software Specification Techniques* (eds. N. Gehani and A.D. McGettrick). Addison Wesley, 363–389, 1986.
45. R.M. Burstall and J.A. Goguen. Algebras, theories and freeness: an introduction for computer scientists. *Theoretical Foundations of Programming Methodology: Lecture Notes of an International Summer School*, Marktoberdorf, 1981 (eds. M. Broy and G. Schmidt). D. Reidel, 329–348 (1982).
46. A. Pettorossi and R.M. Burstall. Deriving very efficient algorithms for evaluating linear recurrence relations using the program transformation technique. *Acta Informatica* 18:181–206 (1982).
47. R.M. Burstall and N. Suzuki. Sakura: a VLSI modelling language. *Proc. Conf. on Advanced Research in VLSI*, Cambridge, Massachusetts (1982).
48. D.T. Sannella and R.M. Burstall. Structured theories in LCF. *Proc. 8th Colloq. on Trees in Algebra and Programming*, L'Aquila. Springer Lecture Notes in Computer Science, Vol. 159, 377–391 (1983).
49. J.A. Goguen and R.M. Burstall. Introducing institutions. *Proc. of Logics of Programming Workshop*, Pittsburgh. Springer Lecture Notes in Computer Science, Vol. 164, 221–256 (1983).
50. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 1: Comma categories, colimits, signatures and theories. *Theoretical Computer Science* 31(1–2):175–209 (1984). Part 2: Signed and abstract theories. *Theoretical Computer Science* 31(3):263–295 (1984).
51. R.M. Burstall and B. Lampson. A kernel language for abstract data types and modules. Invited paper, *Proc. Intl. Symp. on Semantics of Data Types*, Sophia-Antipolis. Springer Lecture Notes in Computer Science, Vol. 173, 1–50 (1984).
52. R.M. Burstall. Programming with modules as typed functional programming. Invited paper, *Proc. Intl. Conf. on 5th Generation Computer Systems*, Tokyo, 103–112 (1984).
53. D.E. Rydeheard and R.M. Burstall. Monads and theories: a survey for computation. Seminar on the Application of Algebra to Language Definition and Compilation, Fontainebleau, 1982. *Algebraic Methods in Semantics* (eds. M. Nivat and J.C. Reynolds). Cambridge University Press, 575–605 (1985).
54. R.M. Burstall. Inductively defined functions. Invited paper, *Proc. Intl. Joint Conf. on Theory and Practice of Software Development*, Berlin. Springer Lecture Notes in Computer Science, Vol. 185, 92–96 (1985).
55. J.A. Goguen and R.M. Burstall. A study in the foundations of programming methodology: specifications, institutions, charters and parchments. *Proc. Summer Workshop on Category Theory and Computer Programming*, Guildford, 1985. Springer Lecture Notes in Computer Science, Vol. 240, 313–333 (1986).
56. D.E. Rydeheard and R.M. Burstall. A categorical unification algorithm. *Proc. Summer Workshop on Category Theory and Computer Programming*, Guildford, 1985. Springer Lecture Notes in Computer Science, Vol. 240, 493–505 (1986).
57. R.M. Burstall and D.E. Rydeheard. Computing with categories. *Proc. Summer Workshop on Category Theory and Computer Programming*, Guildford, 1985. Springer Lecture Notes in Computer Science, Vol. 240, 506–519 (1986).
58. R.M. Burstall. Inductively defined functions in functional programming languages. *Journal of Computer and System Sciences* 34:409–421 (1987). Based on [54].

59. R.M. Burstall. Research in interactive theorem proving at Edinburgh University. Invited paper, *Proc. 20th IBM Computer Science Symposium*, Shizuoka, Japan (1987).
60. D.E. Rydeheard and R.M. Burstall. *Computational Category Theory*. Prentice-Hall (1988).
61. B. Lampson and R.M. Burstall. Pebble, a kernel language for modules and abstract data types. *Information and Computation* 76:278–346 (1988). Based on [51].
62. R.M. Burstall and F. Honsell. A natural deduction treatment of operational semantics. Invited paper, *Proc. 8th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Pune. Springer Lecture Notes in Computer Science, Vol. 338, 250–269 (1988).
63. R.M. Burstall. Computer assisted proof for mathematics: an introduction, using the LEGO proof system. *Proc. IAM Conference on The Revolution in Mathematics Caused by Computing*, Brighton (1990).
64. A. Tarlecki, R.M. Burstall and J.A. Goguen. Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories. *Theoretical Computer Science* 91:239–264 (1991).
65. R.M. Burstall. Computing: yet another reality construction. Invited paper, *Software Development and Reality Construction* (eds. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slawik). Springer Verlag (1992).
66. J.A. Goguen and R.M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the Association for Computing Machinery* 39(1):95–146 (1992).
67. J. McKinna and R.M. Burstall. Deliverables: a categorical approach to program development in type theory. Invited paper, *Proc. 18th Intl. Symp. on Mathematical Foundations of Computer Science*, Gdansk. Springer Lecture Notes in Computer Science, Vol. 711, 32–67 (1993).
68. R.M. Burstall and R. Diaconescu. Hiding and behaviour: an institutional approach. *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice-Hall, 75–92 (1994).
69. R.M. Burstall. Terms, proofs and refinement. Invited paper, *Proc. 9th IEEE Symp. on Logic in Computer Science*, Paris, 2–7 (1994).
70. M. Sato, T. Sakurai and R.M. Burstall. Explicit environments. *Proc. 4th Intl. Conf. on Typed Lambda Calculi and Applications*, L'Aquila. Springer Lecture Notes in Computer Science, Vol. 1581, 340–354 (1999).
71. R.M. Burstall. ProveEasy: helping people learn to do proofs. Invited paper, *Computing: the Australasian Theory Symposium (CATS 2000)*, Canberra. *Electronic Notes in Theoretical Computer Science* 31 (2000).
72. R.M. Burstall. Christopher Strachey — understanding programming languages. *Higher-Order and Symbolic Computation* 13:51–55 (2000).
73. H. Goguen, R. Brooksby and R.M. Burstall. Memory management: an abstract formulation of incremental tracing. *Types for Proofs and Programs: Intl. Workshop TYPES'99 — Selected Papers*. Springer Lecture Notes in Computer Science, Vol. 1956, 148–161 (2000).