JFP **14** (1): 125–128, January 2004.
DOI: 10.1017/S0956796803004830 Printed in the United Kingdom

Book reviews

Proof, Language and Interaction, Essays in Honour of Robin Milner eds Gordon Plotkin, Colin Sterling, Mads Tofte, Foundations of Computing Series, MIT Press 2000

This Festschrift in honour of the scientific life and achievements of Robin Milner consists of 24 papers by various authors plus a biography of Robin Milner by the editors. The volume is well suited to the aim of the series in Foundations of Computing which is to provide a forum in which important research topics can be presented and placed in perspective for researchers, students and practitioners alike.

For anyone not familiar with the extent of Robin Milner's contributions to theoretical computer science, the biography by Plotkin, Stirling and Tofte provides an illuminating summary of the remarkable influence Milner has had.

It discusses his work with semantics and domain theory in the Scott-Strachey paradigm, his work with computer assisted theorem proving, his work on polymorphic type systems and the development of Standard ML, and his work with concurrency (CCS) leading to calculi to model mobile processes. Throughout, his work has had a semantic approach to studying, analysing and synthesising phenomena arising from computing systems.

The biography points out that there has been an "entirely natural, if miraculous, progression" to the calculi/theories produced by Milner – knitting together in remarkable ways to give a rich body of theory with many applications and deep insights.

Despite this potential underlying coherence, the book of essays is quite an eclectic mix touching on many different topics in the foundations of computer science. This variety is not so surprising as contributors have either been colleagues of Milner or have worked in areas related to work by Milner.

The styles of the 'essays' are also an eclectic mix. The majority are theoretical papers which include detailed proofs with significant results. However, there are also papers which give an overview of results in an area with reference to proofs elsewhere and there are essays with an historical account of development of a topic. A few of the papers are not explicit about the link to Milner's work, but most are. The papers are organised under five headings: Semantic Foundations, Programming Logic, Programming Languages, Concurrency, and Mobility, although there are several papers which would fit with more than one of these headings. I give a very brief summary of each paper under these headings (numbered as in the book).

Semantic Foundations

In this section, the first two papers develop theory which arose from papers by Milner and Plotkin in 1977 introducing the notion of fully abstract semantics for a programming language PCF. This gave rise both to much study of the notion of sequentiality which proves hard to capture semantically and to the search for a syntax-independent, fully abstract model.

Curien, Plotkin and Wynskel (1) show some links between semantic catogories. This is work towards models for sequentality (in light of an undecidability result by Loader in 1996). It summarizes some important steps and connections in the search for a direct extensional

and 'mathematically natural' account of sequentiality (and of Milner's fully abstract model of PCF).

Abramsky (2) analyses key features of his game-theoretic fully abstract model of PCF to extract an axiomatic account (thus identifying the essential features of the construction).

The remaining three papers in this section are concerned in some way with operational semantics. This is linked with Milner's work with CCS which gave rise to much study of operational semantics and relationships to algebra and logic (including bisimulation and Hennessy-Milner Logic).

Hoare, Jifeng and Sampaio (3) develop a link from algebraic semantics back to operational semantics with specific reference to a language based on Dijkstra's non-deterministic language. de Bakker and van Breugel (4) take a second-order concurrent imperative language (with statements sent across channels) and give it an intermediate semantics combining (syntactic) operational semantics with some denotational semantic values. This allows a comparison of the operational and denotational and relationship to second-order bisimulation.

Gadducci and Montanari (5) introduce a generalised transition system with '3 dimensional tiles'. These tiles can be composed sequentially, in parallel or coordinated (hence 3 dimensions) and are able to capture rewrite systems and structural operational semantics (S.O.S.) as entailments in a logic (where abstract sequents correspond to computations). The tiles are illustrated using CCS as an example and the paper shows an equivalent formulation of abstract sequents as double categories.

Programming Logic

Robin Milner has made major contributions to the area of computer assisted theorem proving (automated reasoning) starting with the LCF proof checker that he developed whilst in Stanford in 1972. This later lead to the development of the functional programming language ML with a polymorphic type system to support theorem proving and to the development of many systems derived in some way from LCF. Gordon's paper (6) is an historical account of many of these developments culminating in his widely used HOL system and variations. Paulson's paper (7) takes a specific development in theorem proving support which arose from data structures (inductive definitions) and develops a fixed-point package for datatypes and the dual notion of co-datatypes (co-inductive definitions). This is implemented in Paulson's Isabelle (another theorem proving system with roots in LCF).

Two further papers describe the use of theorem proving systems based on type theories to formalize some important parts of mathematics. Constable *et al.* (8) report on a specific formalization of Automata Theory using Nuprl (Nuprl makes use of constructive type theory as a functional language to program proofs).

Huet and Saibi (9) present a formalization of Category Theory using the Coq system which implements the Calculus of Inductive Constructions.

Collette and Jones' paper (10) concerns formal methods and uses a case study (representing equivalence classes as trees) to illustrate the use of, and exploration of, rely/guarantee conditions with potentially concurrent operations on shared state.

The last paper in this section is by Berezin *et al.* (11) and discusses model checking for the propositional μ-calculus (an extension of Hennessy-Milner logic). The paper presents important algorithms (translated from graph-based algorithms) as well as translations to OBDDs. This work shows that (amongst other things) bisimulation equivalences can be calculated efficiently for many practical cases.

Programming Languages

This section contains two papers involving analysis of ML programs (using Standard ML as an example language), and two papers describing very different languages for concurrency.

https://doi.org/10.1017/S0956796803004830 Published online by Cambridge University Press

Harper and Stone (12) show how a fully-featured language (Standard ML with its module system) can be given a typed dynamic semantics (as opposed to the type-erased semantics in the formal definition of the language).

Tofte and Birkedal (13) describe the mechanisms of region inference – a compile-time program analysis which supports run-time memory management (with stacks of regions but without traditional garbage collection). This paper emphasises the close links between region analysis and polymorphic type checking algorithms such as those concerning type and effect systems.

Berry (14) discusses the foundations of the language Esterel. The paper explains why synchronous models and languages might be more appropriate than the more widely studied asynchronous models and languages (such as CCS, pi calculus, CSP, petri nets) for studying reactive systems and hardware design in particular. The paper gathers some key results which clarify semantics and equivalences and justify translations, giving illustrative examples rather than details of proofs.

Pierce and Turner's (15) Pict language is based on the pi calculus. This paper discusses the design decisions in developing the programming language and prospects for its practical use for programming systems with mobility.

Concurrency

The four papers in the concurrency section are all linked to Milner's CCS in some way. One is a technical result about regular behaviours, one concerns language design and the other two are concerned with extending process calculi with timing and probability respectively. Hirshfeld and Moller's (16) paper confirms a conjecture of Milner relating to unary regular behaviours and star heights. Regular behaviours are an interpretation of regular expressions but, unlike regular languages, they take account of possible transition steps and bisimilarity of expressions.

Ferreira, Hennessy and Jeffrey (17) describe one of several ways in which a programming language might combine a process calculus (CCS) and the lambda calculus. In this paper expressions denote threads of computation and are distinguished from processes (choice is an operator on processes but not expressions). This is similar to the style of CML but distinct from the approach taken in Facile. Baeten, Bergstra and Reniers (18) study the combination of discrete time process algebras (process algebras equipped with an enumeration of time slices) and the silent action which Milner introduced in CCS. The paper also introduces notions of absolute and parametric time process algebras.

Stark and Smolka (19) provide a complete axiom system for finite state probabilistic processes. Probabilistic processes add probabilities to the choice operator of CCS-like process expressions. The system described keeps a clear distinction between transitions (given by familiar rules) and a least fixed point calculation of probabilities over transitions.

Mobility

The remaining papers are concerned with the pi calculus in some way. The pi calculus was first introduced by Milner, Parrow and Walker in 1989 and could be considered as a completion of CCS to provide a fundamental calculus for mobile processes.

The first paper in this section by Engberg and Nielsen (20) gives an historical account of the transition from CCS to the pi calculus and the role of their own 1986 report acknowledged by Milner. They show that the culmination of such an elegant calculus was not easily arrived at starting from CCS even though Milner had the initial ideas in 1979 before CCS was even published.

Parrow (21) shows how the pi calculus can be embedded within a very restricted fragment ('concerts of trios'), demonstrating the full expressive power of this fragment (with associated undecidabilities). Liu and Walker (22) use a variation on the pi calculus as a target semantics for a language of concurrent objects with simple types. The translation has a continuation passing style where continuations are agent abstractions of a particular form.

Milner's translation of the lazy lambda calculus into the pi calculus shows that mobile processes are an extension of purely functional programs. The last two papers consider the semantics imposed on lambda terms by the translation. Thus they study the behaviour of purely functional programs in the context of mobile processes. Boudol and Leneve (23) use a lambda calculus with resource accounting ('multiplicities') to relate may testing equivalence (via the pi calculus translation) to the Longo-Levy preorder. This extends a result from the final paper by Sangiorgi.

Sangiorgi (24) characterises an equivalence of translated terms by extending Abramsky's applicative bisimulation to 'open applicative bisimulation'.

Conclusion

An alternative way to have honoured Robin Milner might have been to republish some of his key papers in a single volume with a commentary. However, the present volume manages to convey the dynamics of the theoretical developments from the point of view of many other researchers who pick up on ideas and contribute to a fascinating evolution of theory linked by the common thread of the work of Robin Milner.

Chris Reade
Kingston Business School