Editorial – Formal Methods: What? Why? and When?

- What exactly are they?
- Why do we need them?
- When will they deliver?

The original title of this special issue† was simply 'Formal Aspects', and it was intended to bring topics which are of particular interest to the BCS FACS (Formal Aspects of Computing Science) specialist group to a wider audience. Almost any area of 'computing' has a formal aspect, and hence the potential range of subjects and number of contributions could have been huge. Indeed, the wide range of topics addressed by FACS workshops over the years is indicative of this diversity and is now reflected in FACS volumes within the BCS Workshop series. Some narrowing was obviously needed, and hence the topic of *Formal Methods* was selected.

Although often spoken about and written about, there is much variability in what is actually meant by the term 'Formal Methods'. To extend a familiar quip, 'Formal Methods are over sold, under used, and little understood'.

Many readers of *The Computer Journal* (TCJ) work in areas of computing which are not perceived as being formal – although, ultimately, they probably are; indeed, lack of appropriate formality is arguably the cause of many problems that arise in the development of programs and systems. But I'm getting ahead of myself. In an attempt to bring the essence of so-called Formal Methods to TCJ readers who would not normally delve into publications of a more theoretical or formal nature, the authors of both invited and submitted papers were asked to regard 'Formalism for Informalists' as a working title for this issue. The hope being that those readers not already familiar with Formal Methods might gain an appreciation of the topic and find guidance about where to look in the (introductory?) literature should they wish to investigate the subject further. The intention was that papers would then go on to give 'state of the art' information which would be of more relevance and interest to those who are already 'converts' or who are familiar with the kind of notation which is the *lingua franca* of Formal Methods.

Obviously each author has a different perception of what an 'informalist' is and what he may assume of the typical TCJ reader. Hence, although I have made attempts to avoid the extremes that were present in some early drafts, there is considerable variation in the level of explanation to be found in the papers. The inclusion of more discursive material and an attempt to make each paper reasonably self-contained has resulted in papers being somewhat longer† than is normal. In two cases large papers have been divided and each part written for a different target audience, typically the newcomer to a topic and the aficionado.

† The contents of this special issue extend over two physical issues of the Journal (35(5) and 35(6)) – the symbol ‡ indicates papers that appeared in TCJ, 35(5), October 1992. In order to put the complete set of papers in context, essentially the same editorial as appeared in 35(5) is repeated here.

A major problem with 'Formal Methods' and perhaps one which is largely responsible for the widespread disillusionment that they don't provide all the answers – just like that – is that they are *not prescriptive*. In an attempt to clarify what (I think) Formal Methods are, and what they are not, I include a short paper of my own entitled **Formal Methods – Mathematics, Theory, Recipes or what?**‡ To paraphrase that paper here would largely defeat its objective; but using a loose definition that incorporates the precise descriptions of requirements for systems, of the actual systems themselves, and of the means by which we can reason about their interrelationship, the contents proper of this issue follow as set out below.

We start with two papers on logic by Galton. Here the split is fairly obvious. **Classical Logic: a Crash Course for Beginners**‡ is a short but very readable introduction to the logical concepts that underlie formal reasoning. His second paper, **Logic as a Formal Method**‡, deals with more complex and powerful logical systems.

The next two papers address the topic of model-based specifications (i.e. specifications of the style of VDM and Z) and discuss how such specifications can be used to derive programs. The first, **The Rudiments of Algorithm Refinement**‡ by Woodcock, presents a calculus for the derivation of procedural programs, whereas Clement in his paper, **The Role of Data Reification in Program Refinement: Origins, Synthesis and Appraisal**‡, approaches the task by considering the data and the basic operations that manipulate it. Of course any program involves both data and control, and hence these aspects converge and there is common ground.

An alternative approach to specification – and one which is more fundamental, indeed it is in some sense *assumed* by model-based specifications – is via algebraic axioms. In a two-part paper entitled **Introduction to Algebraic Specification**‡, Part 1, by Ehrig, Mahr, Classen and Orejas sets the groundwork in **Formal Methods of Software Development** and then in Part 2 Ehrig, Mahr and Orejas give a comprehensive account of the origins and development of the subject under the title **Introduction to Algebraic Specification, From Classical View to Foundations of System Specifications**. This is one topic within the ambit of Formal Methods that is now very well established and can be regarded as *mature*. This is in no small measure due to two of our contributing authors.

From abstract(?) data types we move to an area that may be regarded as 'an application'. In **Protocol Design and Implementation Using Formal Methods**‡ by van Sinderen, Pires and Vissers the language LOTOS (Language for Temporal Ordering Specifications) is used not only as a specification language but also as a design language. That paper describes the use of specialised LOTOS-to-LOTOS transformations and this leads us

CPJ 35

nicely to a more general presentation in **How to Produce Correct Software – An Introduction to Formal Specification and Program Development by Transformation** by Boiten, Partsch, Tuijnman and Volker. The transformational approach is further developed by Harrison in his paper **A High-Order Approach to Parallel Algorithms**, which highlights the close and natural interplay between functional languages and parallelism.

As will already be apparent, specifications play a crucial role in Formal Methods, and hence it follows that the languages used to express specifications must be properly defined. One approach to this topic is addressed by Larsen and Plat in **Standards for Non-Executable Specification Languages**. Having invested heavily in the formal development of a program or system, what happens if changes are required? Correctness considerations seem to dictate that we should re-derive the (revised) system *ab initio*. Common sense suggests that the two systems are likely to be very similar, and hence there ought to be a quicker and more direct way to tackle this problem. This takes us to the area of the reusability and modification of formally derived software. A contribution to this relatively unexplored subject is made by Kuhn in **A Technique for Analysing the Effects of Changes in Formal Specifications**.

Of course, depending on the context in which a piece of software is to be used, the kind of factors included in its specification will vary greatly. The reliability of software within real-time control systems has come under great scrutiny in recent years, and in their extensive paper Barroca and McDermid appraise **Formal Methods: Use and Relevance for the Development of Safety Critical Systems**.

As argued in my position paper, 'Formal Methods' are not necessarily the same as so-called structured methods. Semmens, France and Docker follow up this point. In **Integrating Structured Analysis and Formal Specification Techniques** they highlight the respective advantages and describe attempts to maximise the inherent benefits by marrying the two.

Continuing the theme of bringing formality into 'established' practice Mišić, Velašević and Lazarević, in their paper **Formal Specification of a Data Dictionary for an Extended ER Data Model**, not only present a formalisation, in Z, of the basic Entity Relation model but also report on on-going work associated with the formal development of an extended (XER) model.

The final contribution, **Automatic Translation of VDM Specifications into Standard ML Programs** by O'Neill, outlines how a VDM syntax-directed editor was modified so as to deliver SML code as output; thus producing a valuable tool to assist in the validation of the original specification.

Working to meet deadlines for this special issue has been hard work for all concerned. I am greatly indebted to the authors and to the referees, all of whom strove very hard to deliver the goods within the very tight timescales. Special mention must be made of several research workers who recognised that their commitments prevented them accepting the invitation to contribute, and of one author who eventually found it impossible to produce copy by the required date and had to withdraw. I must also thank Colin Tully and Cliff Jones, who gave advice which greatly assisted in my attempts to cast the net as wide as possible.

I make no claims that my coverage of Formal Methods is exhaustive. Little, if any, explicit mention is made of topics such as formally based CASE tools, concurrency, the various programming paradigms as such (the most notable omission being Object-Oriented 'programming'), real-time systems, and the use of Formal Methods in the creation of 'general' Information Systems. Nevertheless, the hope is that by studying (some of) the articles herein the reader might appreciate that Formal Methods:

- allow us to reason logically about system development,
- are needed to fully demonstrate the adequacy of our programming products,
- will pay ever-increasing dividends as their use is extended,
- are oversold – but only by those who do not fully understand them,
- are underused – because not enough programmers are skilled in their use, and
- are insufficiently understood – because they require an appreciation of abstract concepts and great attention to detail. These 'skills' take a considerable time to assimilate.

If this issue makes the reader more aware of the capabilities of, and the rationale for, Formal Methods it will have fulfilled the editors' expectations.

JOHN COOKE