Linking Theories in Computing Science

C.A.R. Hoare

September 1997

For the last two years I have been working with my valued colleague Jifeng He on an EPSRC-funded project for "Linking Theories in Computing Science".

The results have exceeded all our original hopes and expectations. We have investigated a wide variety of computational paradigms, procedural and declarative, sequential and parallel, centralised and distributed, synchronised and asynchronous, even hardware and software. This work has fully confirmed a widely held conjecture that all paradigms can be embedded in the single mathematical theory of relations; and a number of projections have been discovered that map the general theory to its more particular instances.

This means that a single model-based specification language like Z or VDM will serve at the highest level of abstraction to specify all systems, without concern for the technology or mixture of technologies in which they will be implemented. The notations of each programming language are definable as extensions to the schema calculus of Z, and powerful algebraic laws are provided to assist in subsequent design. As a result, all paradigms are susceptible of the same design methodology, with proofs based on familiar techniques of calculational refinement, assertions and weakest preconditions. It means that diverse design calculi and languages and support tools can be safely used together at different stages for different parts of a design project. We hope we have provided a framework and inspiration for research into further diversity, by eliminating risks of confusion, overlap, and unnecessary conflict. Prospects of industrial application are enhanced by removal of the excuse that experts disagree.

Our results are heavily indebted to the major achievements of scientists from many schools of research: programming language semantics, the foundations of computing science, and the application of formal methods in software engineering. Differences in approach between these schools can be attributed to differences in level of abstraction, whether appropriate for specification, design or implementation; and to different styles of presentation: denotational, algebraic, or operational. We have found mathematical constructions that will transform any of these approaches to any other, establishing their total mutual consistency. We thereby hope to promote further advances in all these areas, as a result of improved understanding, communication and cross-fertilisation between specialists from complementary schools of research.

The long-term goal has been to contribute to the reliability and cost-effectiveness of software engineering, with early application more likely in critical system design and in the reengineering of legacy code. The next practical steps towards this goal are

1. the extension of the range of unified theories to cover codesign using notations and formalisms which already have commercial penetration as well as commercially available tool support [SDL, Statecharts, UML, VHDL, Verilog, ...]

2. the strengthening of tool support by inclusion of algorithms for deeper semantic analysis, transformation, test case generation and optimisation. Unification of theories offers a

prospect of combining techniques of proof search, term rewriting, model checking and decision procedures.

Proposals have been prepared to support further research in these directions, with the aid of several academically available systems [Isabelle, PVS, FDR, ...]. We should be interested in hearing from anybody who would like to collaborate in this research.

Experience shows that research results in computing science take many decades to reach industrial acceptance. In the shorter term, their educational significance may be far greater. Our hope is that a unification of theories, even only at an intellectual level, may help to alleviate a major problem that afflicts the software engineering profession today. It is the difficulty of transferring laboriously acquired knowledge, experience and skills between computational paradigms and languages, and particularly the difficulty of using them in combination. A well educated engineer is one whose understanding of a more general theory justifies the confidence to welcome the challenges of the new.

Our joint work has been reported at conferences, and will appear next year as a book entitled *"Unifying Theories of Programming"*. Meanwhile, we have written a five-page summary, with a selection of technical details.