Preface

Computers are gaining more and more control over systems that we use or rely on in our daily lives. Besides the visible appearance of computers, for example in the form of PCs at home or at work, booking terminals in travel agencies, or automatic teller machines in banks, there is a growing demand for computers as invisible parts of systems in order to control their proper functioning. For example, computers inside washing machines or central heating systems should help in saving energy in our homes. Computer-controlled airbags in cars and automatic pilots in airplanes should bring more safety to the passengers. Computerised signalling systems should guarantee safe and efficient operation of railway traffic. Computer-operated telecommunication networks should bring individual services to their customers.

But despite all these successes of computer applications, the question is how much can we really rely on these systems ? Every now and then we read about failures of computerised systems. It may only be annoying if the PC does not work as expected. More serious is a breakdown of a railway signalling system or an unwanted explosion of an airbag. Here the question of *correctness* of the system, of its software and hardware arises. In other words: does the computer system behave according to expectations and requirements ?

This question is a challenge to software engineers and computer scientists: they need to understand the foundations of programming, need to understand how different formal theories are linked together, how compilers correctly translate high-level programs into machine code, why optimisations performed are justifiable. They need the intellectual power to understand all these aspects together in the framework of a suitable methodology for designing correct computerised systems. These are the questions this book is about.

The concern about correctness goes in fact back to the origins of computer programming. Already in 1949 Alan M. Turing posed in a conference paper entitled *"On Checking a Large Routine"* [1] the question: "How can one check a routine in the sense of making sure that it is right ?" and went on to answer that "... the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows." In modern terminology, Turing proposes the "inductive assertion method" to prove program correctness, but the "large routine" considered in the report is in fact quite small: a flow chart program for computing the square root.

Where are we today, 50 years after Turing's pioneering paper ? What is our current understanding of programming methodology, what has been achieved in automatic correctness proofs, how far are we with the production of correct compilers, what application domains of industrial relevance can be handled today ? To answer these questions we invited 15 researchers to help us by explaining their current understanding and contributions to the issue of correct system design.

[1] A.M. Turing. *On Checking a Large Routine.* Report of a Conference on High Speed Automatic Calculating Machines, Univ. Math. Laboratory, Cambridge, pages 67–69, 1949. (See also: F.L. Morris and C.B. Jones, *An Early Program Proof by Alan Turing*, *Annals of the History of Computing 6*, pages 139–143, 1984.)

A Book Dedicated to Hans Langmaack

How did we select the authors ? We took the opportunity of Prof. Dr. Dr. h.c. Hans Langmaack's retirement from his professorship at the University of Kiel on October 1st, 1999, in order to present the area closest to his heart from a personal perspective. The contributors are on the one hand prominent scientists who had or have some close scientific contacts with Langmaack and on the other hand some of his former students who directly contributed to the area of correct system design.

Hans Langmaack studied mathematics, physics, and logic at the Universities of Münster and Freiburg. After receiving his doctorate in mathematics he became the assistant of Klaus Samelson, who together with Friedrich L. Bauer, is one of the German pioneers in the area of computer science in general and compiler construction in particular. The assistantship brought Langmaack to the University of Mainz and the Technical University of Munich where he worked on the topics of programming languages, compiler construction and formal languages. This is where contacts to Gerhard Goos and David Gries were established. Langmaack wrote one of the first compilers for the programming language ALGOL 60. After a visiting assistant professorship at Purdue University, Lafayette, Indiana, he took positions as a full professor of computer science at the Universities of Saarbrücken and Kiel.

Hans Langmaack's scientific interest is motivated by his work on compilers and his pursuit of correctness. Among others, he investigated the procedure concept of ALGOL 60 and ALGOL 68, where procedures are allowed as untyped and typed parameters, and formalised the semantics of procedures in a partly operational style in terms of a copy rule. This formalisation allowed him to attack and solve decidability questions of so-called "formal" run time properties of ALGOL-like programs, i.e. where the data are left uninterpreted. These questions concerned the formal reachability of procedures, their formally correct parameter transmission, and the formal "most-recent" property. The last property is concerned with an error occurring in an implementation of recursive procedures by Edsger W. Dijkstra.[2] A major achievement was the decidability proof of the formal reachability of procedures within ALGOL 68 programs, which are typed in the sense of the Lambda calculus.

Due to Edmund M. Clarke's investigations on obtaining sound and relatively complete Hoare-like proof systems for programming languages with an ALGOL-like procedure concept, Langmaack's interest turned to the issue of program correctness in the sense of Hoare. This is where contacts to Krzysztof R. Apt came about. Very interestingly, it turned out that essentially the same proof techniques that had been used to prove the decidability or undecidability of formal procedure properties could also be applied to show the presence or absence of sound and relatively complete Hoare-like proof systems for programming languages with ALGOL-like procedures. This topic was pursued further by Werner Damm, Ernst-Rüdiger Olderog and Hardi Hungar.

[2] A minimal program violating the "most recent" property is shown on the cover.

Besides his theoretical investigations Hans Langmaack has always been involved in practical compiler projects. These projects were conducted in cooperation with industrial partners. A major concern of these projects was to lift the level of specification of the compiling function. Due to contact with and visits by Dines Bjørner, VDM was used as a specification technique. Work on optimisation techniques and their scientific foundation also started from these practical projects. This topic, comprising abstract interpretation and data flow analysis, was pursued further by Bernhard Steffen, Jens Knoop and Oliver Rüthing, and, focussing on functional and parallel languages, by Flemming Nielson, who, on the initiative of Hans Langmaack, was guest professor at the University of Kiel in 1992.

A major activity was the European basic research project ProCoS (Provably Correct Systems) founded by Tony Hoare, Dines Bjørner and Hans Langmaack. The project was conceived by Hoare in reaction to the so-called "CLInc stack", a multi-level verification task solved by Robert S. Boyer and J S. Moore. It defined the so-called "ProCoS tower" of language levels and was concerned with the correct links between these levels. In this project Hans Langmaack's group was responsible for correct compilers from an OCCAM-like programming language to transputer machine code. Martin Fränzle and Markus Müller-Olm earned their first scientific merits in this project. Anders P. Ravn and Hans Rischel at Lyngby, and Ernst-Rüdiger Olderog at Oldenburg also participated in ProCoS. The work of Langmaack's group on compiler correctness in the ProCoS project is currently continued in the project Verifix conducted by Gerhard Goos, Friedrich von Henke and Hans Langmaack.

Links to local companies in Kiel established contacts to Jan Peleska, who was rigorously applying formal methods in industry. Amir Pnueli brought "Turing power" to international meetings in Schleswig-Holstein.

Hans Langmaack has also been successful as a teacher and advisor to many students. About 170 students graduated with a Diplom (MSc) from his reseach group, 22 completed a doctoral dissertation (PhD) under his supervison and 5 finished the habilitation procedure to qualify for a professorship with his guidance. Being among his students, we have closely experienced his continuous engagement for the topic of "Correct System Design", in research, education and unforgettable grill parties. Here, Bengt Jonsson, who regularly visited Kiel, revealed unexpected talents, as an interpreter for both a Russian visitor and for Chopin at the piano.

Structure of this Book

This book consists of 17 chapters describing recent insights and advances in *Correct System Design*. They are grouped together under the five topics of methodology, programming, automation, compilation and application.

Methodology. Tony Hoare discusses in his paper *"Theories of Programming: Top-Down and Bottom-Up and Meeting in the Middle"* complementary approaches to describing the behaviour of programs. The top-down approach starts from a specification of the desired behaviour; the bottom-up approach from a collection of realisable components. A complete theory of programming will combine both approaches. Throughout the presentation Hoare stresses the advantages of algebraic laws in conveying the essence of both approaches.

Dines Bjørner illustrates in his chapter *"A Triptych Software Development Paradigm: Domain, Requirements and Software"* his view of software engineering by describing a three-step approach to rigorous software development. The approach comprises descriptions of the application domain, the requirements, and the software architecture. It is exemplified in terms of a decision support software for sustainable development.

Anders P. Ravn and Hans Rischel summarise in their chapter *"Real-Time Constraints Through the ProCoS Layers"* the results of the European research project ProCoS in which the Universities of Oxford, Lyngby, Kiel and Oldenburg collaborated. They concentrate on the main achievements of ProCoS in the area of real-time systems: the correct link between different layers of formal description ranging from requirements capture, through design and programming, down to machine code generation.

David Gries looks in his chapter *"Monotonicity in Calculational Proofs"* at logic formulae as a basic formalism for specifying systems. He is interested in a specific aspect that is of importance in the calculational manipulation of logic formulae: the monotonicity of positions where substitutions of formulae for variables are applied, and he presents a suitable metatheorem concerning monotonicity.

Programming. Krzysztof R. Apt and Andrea Schaerf explain in their chapter *"The Alma Project, or How First-Order Logic Can Help Us in Imperative Programming"* how a given imperative programming language like Modula-2 can be extended by concepts from the logic programming paradigm in order to raise the level of abstraction. As a result executable specification statements allowing bounded quantifiers can be formulated in the extended language. This leads to surprisingly clear and short programs.

Flemming Nielson and Hanne Riis Nielson show in their chapter *"Type and Effect Systems"* how static inference techniques can be used to ensure that the dynamic behaviour of a program satisfies the specification. To this end, the basic type information is extended by suitable annotations to express further intensional or extensional properties of the semantics of the program.

Automation. J S. Moore describes in his chapter *"Proving Theorems about Java-Like Byte Code"* how correctness theorems about Java programs compiled into code for a toy version of the Java Virtual Machine can be proved mechanically. The basis for this work is the mechanized logic ACL2 (A Computational Logic for Applicative Common Lisp) developed by Boyer and Moore.

Armin Biere, Edmund M. Clarke and Yunshan Zhu present in their chapter *"Multiple State and Single State Tableaux for Combining Local and Global Model Checking"* a new algorithm for the automatic verification of reactive systems specified in linear temporal logic. It combines the advantages of local and explicit state model checking with those of global and symbolic model checking.

Parosh A. Abdulla and Bengt Jonsson consider in their chapter *"On the Existence of Network Invariants for Verifying Parametrized Systems"* infinite classes of finite state systems consisting of an unbounded number of similar processes specified by a parametrised system. The aim is an inductive correctness proof of such systems using the notion of a network invariant. The paper presents sufficient conditions under which a finite-state network invariant exists.

Compilation. Gerhard Goos and Wolf Zimmermann report in their chapter *"Verification of Compilers"* on the results of a joint research project of the Universities of Karlsruhe, Kiel and Ulm. They discuss a suitable notion of correctness for compilers and how it can be verified exploiting the traditional compiler architectures involving certain intermediate languages. A main achievement is the use of program checking for replacing large parts of compiler verification by the simpler task of verifying program checkers. As semantic basis for the programming language abstract state machines are used.

Amir Pnueli, Ofer Shtrichman and Michael Siegel present in their chapter *"Translation Validation: From* SIGNAL *to C"* an alternative approach to compiler verification where each run of a compiler is considered individually and followed by a validation phase. This phase verifies that the target code produced on this run correctly implements the submitted source program. The authors address the practicality of this approach for an optimising, industrial code generator from SIGNAL to C.

Martin Fränzle and Markus Müller-Olm provide in their chapter *"Compilation and Synthesis for Real-Time Embedded Controllers"* an overview of two constructive approaches for the generation of hard real-time code from abstract specifications. The first approach starts from a real-time imperative programming language and pursues an incremental code generation. The second approach starts from formulae in a real-time logic and pursues a synthesis approach.

Jens Knoop and Oliver Rüthing investigate in their chapter *"Optimization Under the Perspective of Soundness, Completeness, and Reusability"* the code optimization technique PRE (partical redundancy elimination) in various programming paradigms: imperative, parallel, and object-oriented. For each of these paradigms the authors analyse whether PRE is sound (i.e. admissible) and complete (i.e. optimal).

Application. Tom Bienmüller, Jürgen Bohn, Henning Brinkmann, Udo Brockmeyer, Werner Damm, Hardi Hungar and Peter Jansen describe in their chapter *"Verification of Automotive Control Units"* the application of automatic verification (model-checking) tools to specification models of electronic control units for automotive applications. The approach is based on the use of the design tool STATEMATE for dealing with the discrete part of the models. For dealing with values ranging over continuous domains, the authors also present a new technique of first-order model-checking. The approach has been successfully applied in cooperation with the car manufacturer BMW.

Ernst-Rüdiger Olderog shows in his chapter *"Correct Real-Time Software for Programmable Logic Controllers"* how ideas from the theory of real-time systems can be applied to an area from industrial practice: the design of railway signalling systems to be implemented on PLCs (programmable logic controllers). The proposed approach comprises the levels of requirements, design specifications and programs for PLCs. Correctness between these levels is achieved on the basis of a real-time logic.

Jan Peleska and Bettina Buth summarise in their chapter *"Formal Methods for the International Space Station ISS"* the results and experiences obtained in a project in collaboration with DaimlerChrysler Aerospace. The aim of this project was to check a number of correctness requirements for a fault-tolerant computer to be used in the International Space Station ISS. To this end, a combination of formal verification, simulation and testing was applied. The formal verifcation relied on the careful use of Hoare's theory CSP (communicating sequential processes) and its associated model checker FDR.

Bernhard Steffen and Tiziana Margaria explain in their chapter *"MetaFrame in Practice: Design of Intelligent Network Services"* how concepts from the theory of reactive systems, temporal logics, and model-checking can be applied to the area of intelligent networks. The problem considered is how to assist programmers in the correct design of new telecommunication services for customers. The approach, on the basis of the MetaFrame environment, led to a product that has been adopted, bought, and marketed by Siemens Nixdorf Informationssysteme AG.

Acknowledgements

In the first stages of the book project Krzysztof R. Apt and David Gries were helpful. Alfred Hofmann from Springer-Verlag was supportive from the very first moment. Annemarie Langmaack kindly helped us to obtain a photograph of Hans Langmaack. We are particularly grateful to Claudia Herbers for her devoted support in the production of this book. Last but not least all contacted authors were very easy to motivate to contribute and in the end kept their promise. They also helped in the mutual refereeing process of the contributed papers.

Oldenburg and Dortmund E.-R. Olderog and B. Steffen
July 1999

Prof. Dr. Dr. h.c. Hans Langmaack
Foto: Foto-Renard, Kiel