Preface

Proof theory has long been established as a basic discipline of mathematical logic. It has recently become increasingly relevant to computer science. The deductive apparatus provided by proof theory has proved useful for metatheoretical purposes as well as for practical applications. Thus it seemed to us most natural to bring researchers together to assess both the role proof theory already plays in computer science and the role it might play in the future.

The form of a Dagstuhl seminar is most suitable for purposes like this, as Schloß Dagstuhl provides a very convenient and stimulating environment to discuss new ideas and developments. To accompany the conference with a proceedings volume appeared to us equally appropriate. Such a volume not only fixes basic results of the subject and makes them available to a broader audience, but also signals to the scientific community that *Proof Theory in Computer Science* (PTCS) is a major research branch within the wider field of logic in computer science.

Therefore everybody invited to the Dagstuhl seminar was also invited to submit a paper. However, preparation and acceptance of a paper for the volume was not a precondition of participating at the conference, since the idea of a Dagstuhl seminar as a forum for spontaneous and open discussions should be kept. Our idea was that the papers in this volume should be suitable as starting points for such discussions by presenting fundamental results which merit their publication in the Springer LNCS series. The quality and variety of the papers received and accepted rendered this plan fully justified. They are a state-of-the-art sample of proof-theoretic methods and techniques applied within computer science.

In our opinion PTCS focuses on the impact proof theory has or should have on computer science, in particular with respect to programming. Major divisions of PTCS, as represented in this volume, are the following:

1. The *proofs as programs* paradigm in general
2. Typed and untyped systems related to functional programming
3. Proof-theoretic approaches to logic programming
4. Proof-theoretic ways of dealing with computational complexity
5. Proof-theoretic semantics of languages for specification and programming
6. Foundational issues

This list is not intended to be exclusive. For example, there is undoubtedly some overlap between *Automated Deduction* and PTCS. However, since *Automated Deduction* is already a well-established subdiscipline of logic in computer science with its own research programs, many of which are not related to proof theory, we did not include it as a core subject of PTCS.

In the following, we briefly address the topics of PTCS mentioned and indicate how they are exemplified in the contributions to this volume.

1. The most intrinsic relationship between proof theory and computer science, if proof theory is understood as the theory of formal proofs and computer science as the theory of computing, is provided by the fact that in certain formalisms proofs can be evaluated (reduced) to normal forms. This means that proofs can be viewed as representing a (not necessarily deterministic) program for their own evaluation. In particular contexts they allow one to extract valuable information, which may be given, e.g., in the form of particular terms. The idea of considering *proofs as programs*, which in the context of the typed λ-calculus is known as the Curry-Howard-correspondence, is a research program touched upon by most contributions to this volume. The papers by *Baaz & Leitsch* and by *Berger* are directly devoted to it. *Baaz & Leitsch* study the relative complexity of two cut elimination methods and show that they are intrinsically different. *Berger* investigates a proof of transfinite induction given by Gentzen in order to extract algorithms for function hierarchies from it.

2. *Functional programming* has always been at the center of interest of proof theory, as it is based on the λ-calculus. Extensions of the typed λ-calculus, in particular type theories, lead to powerful frameworks suitable for the formalization of large parts of mathematics. The paper by *Alt & Artemov* develops a reflective extension of the typed λ-calculus which internalizes its own derivations as terms. *Dybjer & Setzer* show how indexed forms of inductive-recursive definitions, which would enable a certain kind of generic programming, can be added to Martin-Löf type theory. The main proof-theoretic paradigm competing with type theory is based on type-free applicative theories and extensions thereof within Feferman's general program of explicit mathematics. In his contribution, *Studer* uses this framework in an analysis of a fragment of Java. In particular, he manages to proceed without impredicative assumptions, thus supporting a general conjecture by Feferman.

3. *Logic programming*, which uses the Horn clause fragment of first-order logic as a programming language, is a natural topic of PTCS. Originally it was not developed within a proof-theoretic framework, and its theoretical background is often described in model-theoretic terms. However, it has turned out that a proof-theoretic treatment of logic programming is both nearer to the programmer's way of thinking and conceptually and technically very natural. It also leads to strong extensions, including typed ones which combine features of functional and logic programming. In the present volume, *Elbl* uses proof-theoretic techniques to give "metalogical" operators in logic programming an appropriate rendering.

4. The machine-independent characterization of classes of *computational complexity* not involving explicit bounds has recently gained much attention in proof theory. One such approach, relying on higher-type functionals, is used in *Aehlig et al.*'s paper to characterize the parallel complexity class NC. Another proof-theoretic method based on term rewriting is applied by *Oitavem* in her characterization of PSPACE and is compared and contrasted with other implicit characterizations of this class. *Gordeew* asks the fundamental question of whether functional analysis may serve as an alternative framework in certain subjects of PTCS. He suggests that non-discrete methods may provide powerful tools

for dealing with certain computational problems, in particular those concerning polynomial-time computability.

5. Besides the systematic topics mentioned, the study of *specific languages* is an important aspect of PTCS. In his contribution, *Schmitt* develops and studies a language of iterate logic as the logical basis of certain specification and modeling languages. *Studer* gives a denotational semantics of a fragment of Java. By interpreting Featherweight Java in a proof-theoretically specified language he shows that there is a direct proof-theoretic sense of denotational semantics which differs both from model-theoretic and from domain-theoretic approaches. This shows that the idea of *proof-theoretic semantics* discussed in certain areas of philosophical and mathematical logic, is becoming fruitful for PTCS as well.

6. Finally, two papers concern *foundational aspects* of languages. *Baaz & Fermüller* show that in formalizing identity, it makes a significant difference for uniform provability, whether identity is formulated by means of axioms or by means of a schema. The paper by *Došen & Petrić* presents a coherence result for categories which they call "sesquicartesian", contributing new insights into the equality of arrows (and therefore into the equality of proofs and computations) and its decidability.

We thank the authors and reviewers for their contributions and efforts. We are grateful to the Schloß Dagstuhl conference and research center for acting as our host, and to Springer-Verlag for publishing these proceedings in their LNCS series.

The second and the third editor would like to add that it was Reinhard Kahle's idea to organize a Dagstuhl seminar on PTCS, and that he had the major share in preparing the conference and editing this volume. He would have been the first editor even if his name had not been the first alphabetically.

October 2001

Reinhard Kahle
Peter Schroeder-Heister
Robert Stärk

Program Committee

Arnold Beckmann (Münster)
Roy Dyckhoff (St. Andrews)
Rajeev Goré (Canberra)
Gerhard Jäger (Bern)
Reinhard Kahle (Tübingen)
Dale Miller (Penn State)
Tobias Nipkow (München)
Frank Pfenning (Pittsburgh)
Peter Schroeder-Heister (Tübingen)
Robert Stärk (Zürich)

Additional Reviewers

Steve Bellantoni (Toronto)
Norman Danner (Los Angeles)
Lew Gordeew (Tübingen)
Peter Hancock (Edinburgh)
Jörg Hudelmaier (Tübingen)
Jim Lambek (Montreal)
Daniel Leivant (Bloomington)
Jean-Yves Marion (Nancy)
Ralph Matthes (München)
Hans Jürgen Ohlbach (München)
Christine Paulin-Mohring (Paris)
Thomas Strahm (Bern)