Lecture Notes in Computer Science 1706
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

John Hatcliff Torben Æ. Mogensen
Peter Thiemann (Eds.)

Partial Evaluation

Practice and Theory

DIKU 1998 International Summer School
Copenhagen, Denmark, June 29 – July 10, 1998

Springer

Volume Editors

John Hatcliff
Department of Computing and Information Sciences
Kansas State University
234 Nichols Hall, Manhattan, KS 66506, USA
E-mail: hatcliff@cis.ksu.edu

Torben Æ. Mogensen
DIKU, Københavns Universitet
Universitetsparken 1, DK-2100 København Ø, Denmark
E-mail: torbenm@diku.dk

Peter Thiemann
Institut für Informatik, Universität Freiburg
Universitätsgelände Flugplatz, D-79110 Freiburg i.Br., Germany
E-mail: thiemann@informatik.uni-freiburg.de

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Partial evaluation : practice and theory ; DIKU 1998 international summer school, Copenhagen, Denmark, 1998 / John Hatcliff ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1706)
ISBN 3-540-66710-5

CR Subject Classification (1998): D.3.4, D.1.2, D.3.1, F.3, D.2

ISSN 0302-9743
ISBN 3-540-66710-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10704931 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

As the complexity of software increases, researchers and practitioners continue to seek better techniques for engineering the construction and evolution of software. Partial evaluation is an attractive technology for modern software construction for several reasons.

- It is an automatic tool for software specialization. Therefore, at a time when software requirements are evolving rapidly and when systems must operate in heterogeneous environments, it provides an avenue for easily adapting software components to particular requirements and to different environments.
- It is based on rigorous semantic foundations. Modern applications increasingly demand high-confidence software solutions. At the same time, traditional methods of validation and testing are failing to keep pace with the inherent complexity found in today's applications. Thus, partial evaluation and the mathematically justified principles that underly it are promising tools in the construction of robust software with high levels of assurance.
- It can be used to resolve the tension between the often conflicting goals of generality and efficiency. In most cases, software is best engineered by first constructing simple and general code rather than immediately implementing highly optimized programs that are organized into many special cases. It is much easier to check that the general code satisfies specifications, but the general code is usually much less efficient and may not be suitable as a final implementation. Partial evaluation can be used to automatically generate efficient specialized instances from general solutions. Moreover, since the transformation is performed automatically (based on rigorous semantic foundations), one can arrive at methods for more easily showing that the specialized code also satisfies the software specifications.

Partial evaluation technology continues to grow and mature. ACM SIGPLAN-sponsored conferences and workshops have provided a forum for researchers to share current results and directions of work. Partial evaluation techniques are being used in commercially available compilers (for example the *Chez Scheme* system). They are also being used in industrial scheduling systems (see Augustsson's article in this volume), they have been incorporated into popular commercial products (see Singh's article in this volume), and they are the basis of methodologies for implementing domain-specific languages.

Due to the growing interest (both inside and outside the programming languages community) in applying partial evaluation, the DIKU International Summer School on Partial Evaluation was organized to present lectures of leading researchers in the area to graduate students and researchers from other communities.

The objectives of the summer school were to

- present the foundations of partial evaluation in a clear and rigorous manner,
- offer a practical introduction to several existing partial evaluators, including the opportunity for guided hands-on experience,

- present more sophisticated theory, systems, and applications, and
- highlight open problems and challenges that remain.

The summer school had 45 participants (15 lecturers and 30 students) from 24 departments and industrial sites in Europe, the United States, and Japan.

This volume

All lecturers were invited to submit an article presenting the contents of their lectures for this collection. Each article was reviewed among the lecturers of the summer school.

Here is a brief summary of the articles appearing in this volume in order of presentation at the summer school.

Part I: Practice and experience using partial evaluators

- Torben Mogensen. *Partial Evaluation: Concepts and Applications.* Introduces the basic idea of partial evaluation: specialization of a program by exploiting partial knowledge of its input. Some small examples are shown and the basic theory, concepts, and applications of partial evaluation are described, including "the partial evaluation equation", generating extensions, and self-application.
- John Hatcliff. *An Introduction to Online and Offline Partial Evaluation Using a Simple Flowchart Language.* Presents basic principles of partial evaluation using the simple imperative language FCL (a language of flowcharts introduced by Jones and Gomard). Formal semantics and examples are given for online and offline partial evaluators.
- Jesper Jørgensen. *Similix: A Self-Applicable Partial Evaluator for Scheme.* Presents specialization of functional languages as it is performed by Similix. The architecture and basic algorithms of Similix are explained, and application of the system is illustrated with examples of specialization, self-application, and compiler generation.
- Jens Peter Secher (with Arne John Glenstrup and Henning Makholm). *C-Mix II: Specialization of C Programs.* Describes the internals of C-Mix – a generating extension generator for ANSI C. The role and functionality of the main components (pointer analysis, in-use analysis, binding-time analysis, code generation, etc.) are explained.
- Michael Leuschel. *Logic Program Specialization.* Presents the basic theory for specializing logic programs based upon partial deduction techniques. The fundamental correctness criteria are presented, and subtle differences with specialization of functional and imperative languages are highlighted.

Part II: More sophisticated theory, systems, and applications

- Torben Mogensen. *Inherited Limits.* Studies the evolution of partial evaluators from an insightful perspective: the attempt to prevent the structure of a source program from imposing limits on its residual programs. If the structure of a residual program is limited in this way, it can be seen as a weakness in the partial evaluator.
- Neil Jones (with Carsten K. Gomard and Peter Sestoft). *Partial Evaluation for the Lambda Calculus.* Presents a simple partial evaluator called *Lambda-mix* for the untyped lambda-calculus. Compilation and compiler generation for a language from its denotational semantics are illustrated.
- Satnam Singh (with Nicholas McKay). *Partial Evaluation of Hardware.* Describes run-time specialization of circuits on Field Programmable Gate Arrays (FPGAs). This technique has been used to optimize several embedded systems including DES encoding, graphics systems, and postscript interpreters.
- Lennart Augustsson. *Partial Evaluation for Aircraft Crew Scheduling.* Presents a partial evaluator and program transformation system for a domain-specific language used in automatic scheduling of aircraft crews. The partial evaluator is used daily in production at Lufthansa.
- Robert Glück (with Jesper Jørgensen). *Multi-level Specialization.* Presents a specialization system that can divide programs into multiple stages (instead of just two stages as with conventional partial evaluators). Their approach creates multi-level generating extensions that guarantee fast successive specialization, and is thus far more practical than multiple self-application of specializers.
- Morten Heine Sørensen (with Robert Glück). *Introduction to Supercompilation.* Provides a gentle introduction to Turchin's supercompiler – a program transformer that sometimes achieves more dramatic speed-ups than those seen in partial evaluation. Recent techniques to prove termination and methods to incorporate negative information are also covered.
- Michael Leuschel. *Advanced Logic Program Specialization.* Summarizes some advanced control techniques for specializing logic programs based on characteristic trees and homeomorphic embedding. The article also describes various extensions to partial deduction including conjunctive partial deduction (which can accomplish tupling and deforestation), and a combination of program specialization and abstract interpretation techniques. Illustrations are given using the online specializer ECCE.
- John Hughes. *A Type Specialization Tutorial.* Presents a paradigm for partial evaluation, based not on syntax-driven transformation of terms, but on type reconstruction with unification. This is advantageous because residual programs need not involve the same types as source programs, and thus several desirable properties of specialization fall out naturally and elegantly.
- Julia Lawall. *Faster Fourier Transforms via Automatic Program Specialization.* Investigates the effect of machine architecture and compiler technology on the performance of specialized programs using an implementation of the

Fast Fourier Transform as an example. The article also illustrates the Tempo partial evaluator for C, which was used to carry out the experiments.
- Jens Palsberg. *Eta-Redexes in Partial Evaluation.* Illustrates how adding eta-redexes to functional programs can make a partial evaluator yield better results. The article presents a type-based explanation of what eta-expansion achieves, why it works, and how it can be automated.
- Olivier Danvy. *Type-Directed Specialization.* Presents the basics of type-directed partial evaluation: a specialization technique based on a concise and efficient normalization algorithm for the lambda-calculus, originating in proof theory. The progression from the normalization algorithm as it appears in the proof theory literature to an effective partial evaluator is motivated with some simple and telling examples.
- Peter Thiemann. *Aspects of the PGG System: Specialization for Standard Scheme.* Gives an overview of the PGG system: an offline partial evaluation system for the full Scheme language – including Scheme's reflective operations (eval, apply, and call/cc), and operations that manipulate state. Working from motivating examples (parser generation and programming with message passing), the article outlines the principles underlying the necessarily sophisticated binding-time analyses and their associated specializers.

Acknowledgements

The partial evaluation community owes a debt of gratitude to Morten Heine Sørensen, chair of the summer school organizing committee, and to the other committee members Neil D. Jones, Jesper Jørgensen, and Jens Peter Secher. The organizers worked long hours to ensure that the program ran smoothly and that all participants had a rewarding time.

The secretarial staff at DIKU and especially TOPPS group secretaries Karin Outzen and Karina Sønderholm labored behind the scenes and provided assistance on the myriad of administrative tasks that come with preparing for such an event.

Finally, a special thanks is due to Jens Ulrik Skakkebæk for the use of his laptop computer and for technical assistance with DIKU's digital projector.

John Hatcliff
Torben Mogensen
Peter Thiemann
July 1999

Table of Contents