Samson Abramsky (Ed.)

LNCS 2044

Typed Lambda Calculi and Applications

5th International Conference, TLCA 2001
Kraków, Poland, May 2001
Proceedings

Springer

Samson Abramsky (Ed.)

Typed Lambda Calculi and Applications

5th International Conference, TLCA 2001
Kraków, Poland, May 2-5, 2001
Proceedings

Springer

Volume Editor

Samson Abramsky
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD
E-mail: Samson.Abramsky@comlab.ox.ac.uk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Typed lambda calculi and applications : 5th international conference ; proceedings / TLCA 2001, Kraków, Poland, May 2 - 5, 2001. Samson Abramsky (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2044)
ISBN 3-540-41960-8

CR Subject Classification (1998): F.4.1, F.3, D.1.1, D.3

ISSN 0302-9743
ISBN 3-540-41960-8 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10782565 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the Fifth International Conference on Typed Lambda Calculi and Applications, held in Kraków, Poland on May 2–5, 2001. It contains the abstracts of the four invited lectures, plus 28 contributed papers. These were selected from a total of 55 submissions. The standard was high, and selection was difficult.

The conference programme also featured an evening lecture by Roger Hindley, on "The early days of combinators and lambda".

I would like to express my gratitude to the members of the Program Committee and the Organizing Committee for all their dedication and hard work. I would also like to thank the many referees who assisted in the selection process. Finally, the support of Jagiellonian University, Warsaw University, and the U.S. Office of Naval Research is gratefully acknowledged.

The study of typed lambda calculi continues to expand and develop, and touches on many of the key foundational issues in computer science. This volume bears witness to its continuing vitality.

February 2001 Samson Abramsky

Program Committee

S. Abramsky (Oxford) (Chair)
P.-L. Curien (Paris)
P. Dybjer (Gothenburg)
T. Ehrhard (Marseille)
M. Hasegawa (Kyoto)
F. Honsell (Udine)
D. Leivant (Bloomington)
S. Ronchi della Rocca (Turin)
H. Schwichtenberg (Munich)
P. Scott (Ottawa)
J. Tiuryn (Warsaw)

Organizing Committee

M. Zaionc (Kraków)
P. Urzyczyn (Warsaw)
J. Wielgut-Walczak (Kraków)

Referees

Peter Aczel
Klaus Aehlig
Fabio Allesi
Thorsten Altenkirch
Andrea Asperti
Patrick Baillot
Franco Barbanera
Gilles Barthe
Nick Benton
Stefano Berardi
Ulrich Berger
Gavin Bierman
Rick Blute
Viviana Bono
Wilfried Buchholz
Juliusz Chroboczek
Alberto Ciaffaglione
Robin Cockett
Loic Colson
Adriana Compagnoni
Mario Coppo
Thierry Coquand
Ferrucio Damiani
Vincent Danos
Ewen Denney
Mariangola Dezani
Roberto Di Cosmo
Pietro Di Gianantonio
Roy Dyckhoff
Andrzej Filinski
Bernd Finkbeiner
Gianluca Franco
Herman Geuvers
Paola Giannini
Jeremy Gibbons
Bruno Guillaume
Esfandiar Haghverdi
Peter Hancock
Russ Harmer
Ryu Hasegawa
Michael Hedberg
Peter Moller Heergard
Hugo Herbelin
Roger Hindley
Martin Hofmann
Doug Howe
Radha Jagadeesan
Patrik Jansson
Felix Joachimski
Jan Johannsen
Jean-Baptiste Joinet
Thierry Joly
Delia Kesner
Yoshiki Kinoshita
Josva Kleist
Jean-Louis Krivine
Francois Lamarche
Olivier Laurent
Marina Lenisa
Ugo de'Liguoro
Luigi Liquori
John Longley
Zhaohui Luo
Harry Mairson
Jean-Yves Marion
Simone Martini
Ralph Matthes
Paul-Andre Mellies
Marino Miculan
Larry Moss
Koji Nakazawa
Susumu Nishimura
Vincent Padovani
Luca Paolini
Michel Parigot
C. Paulin-Mohring
Dusko Pavlovic
Adolfo Piperno
Jaco van der Pol
Jeff Polakow
Randy Pollack
Laurent Regnier
Eike Ritter
Kristoffer Rose

Jiri Rosicky
Luca Roversi
Martin Ruckert
Don Sanella
Ivan Scagnetto
Alan Schmitt
Robert Seely
Jonathan Seldin
Harold Simmons
Thomas Streicher
Eijiro Sumii
Makoto Takeyama
Robert Tennent
Wolfgang Thomas
Christian Urban
Paweł Urzyczyn
Tarmo Uustalu
Betti Venneri
Roel de Vrijer
Stan Wainer
Benjamin Werner

Table of Contents

Invited Lectures

Contributed Papers

Many Happy Returns

Olivier Danvy

BRICS*
Department of Computer Science
University of Aarhus
Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark
E-mail: danvy@brics.dk
Home page: http://www.brics.dk/~danvy

Abstract. Continuations occur in many areas of computer science: logic, proof theory, formal semantics, programming-language design and implementation, and programming. Like the wheel, continuations have been discovered and rediscovered many times, independently. In programming languages, they represent of "the rest of a computation" as a function, and proved particularly convenient to formalize control structures (sequence, gotos, exceptions, coroutines, backtracking, resumptions, etc.) and to reason about them. In the lambda-calculus, terms can be transformed into "continuation-passing style" (CPS), and the corresponding transformation over types can be interpreted as a double-negation translation via the Curry-Howard isomorphism. In the computational lambda-calculus, they can simulate monads. In programming, they provide functional accumulators.

Yet continuations are remarkably elusive. They can be explained in five minutes, but grasping them seems to require a lifetime. Consequently one often reacts to them to an extreme, either loving them ("to a man with a hammer, the world looks like a nail") or hating them ("too many lambdas").

In this talk, we will first review basic results about continuations, starting with Plotkin's Indifference and Simulation theorems (evaluating a CPS-transformed program yields the same result independently of the evaluation order). Thus equipped, we will identify where continuations arose and how they contributed to solving various problems in computer science. We will conclude with the state of the art today, and present a number of examples, including an illustration of how applying the continuation of a procedure several times makes this procedure return several times—hence the title of the talk.

* Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, p. 1, 2001.

From Bounded Arithmetic to Memory Management: Use of Type Theory to Capture Complexity Classes and Space Behaviour

Martin Hofmann

Laboratory for the Foundations of Computer Science
Division of Informatics, University of Edinburgh

Bounded arithmetic [3] is a subsystem of Peano arithmetic defining exactly the polynomial time functions. As Gödel's system T corresponds to Peano arithmetic Cook and Urquhart's system PV_ω [4] corresponds to bounded arithmetic. It is a type system with the property that all definable functions are polynomial time computable.

PV_ω as a programming language for polynomial time is, however, unsatisfactory in several ways. Firstly, it requires to maintain explicit size bounds on intermediate results and secondly, many obviously polynomial time algorithms do not fit into the type system. The attempt to alleviate these restrictions has lead to a sequence of new type systems capturing various complexity classes (PTIME, PSPACE, EXPTIME, LINSPACE) without explicit reference to bounds. Among them are Cook-Bellantoni's [2] and Bellantoni-Niggl-Schwichtenberg's systems of safe recursion [1], tiered systems by Leivant and Marion [12,11], subsystems of Girard's linear logic [6,5], and various systems by myself [9,7,8].

The most recent work [10] has shown that one of these systems can be adapted to allow for explicit memory management including in-place update while still maintaining a functional semantics.

The talk will give a bird's eye overview of the above-mentioned calculi and then discuss in some more detail the recent applications to memory management. This will include recent yet unpublished results about the expressive power of higher-order linear functions and general recursion in the context of [10]. These results suggests that the expressive power equals $\bigcup_c \mathrm{DTIME}(2^{n^c})$.

References

1. S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Ramification, Modality, and Linearity in Higher Type Recursion. *Annals of Pure and Applied Logic*, 2000. to appear.
2. Stephen Bellantoni and Stephen Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.
3. Samuel R. Buss. *Bounded Arithmetic*. Bibliopolis, 1986.
4. S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63:103–200, 1993.
5. J.-Y. Girard. Light Linear Logic. *Information and Computation*, 143, 1998.
6. J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic. *Theoretical Computer Science*, 97(1):1–66, 1992.

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, pp. 2–3, 2001.

7. Martin Hofmann. Linear types and non size-increasing polynomial time computation. To appear in Theoretical Computer Science. See www.dcs.ed.ac.uk/home/papers/icc.ps.gz for a draft. An extended abstract has appeared under the same title in Proc. Symp. Logic in Comp. Sci. (LICS) 1999, Trento, 2000.
8. Martin Hofmann. Programming languages capturing complexity classes. *SIGACT News Logic Column*, 9, 2000. 12 pp.
9. Martin Hofmann. Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 104:113–166, 2000.
10. Martin Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 2001. To appear, see www.dcs.ed.ac.uk/home/mxh/papers/nordic.ps.gz for a draft. An extended abstract has appeared in *Programming Languages and Systems*, G. Smolka, ed., Springer LNCS, 2000.
11. D. Leivant and J.-Y. Marion. Predicative Functional Recurrence and Poly-Space. In *Springer LNCS 1214: Proc. CAAP*, 1997.
12. Daniel Leivant. Stratified Functional Programs and Computational Complexity. In *Proc. 20th IEEE Symp. on Principles of Programming Languages*, 1993.

Definability of Total Objects in *PCF* and Related Calculi

Dag Normann

Department of Mathematics
University of Oslo

We let *PCF* be Plotkin's [8] calculus based on Scott's [10,11] *LCF*, and we consider the standard case with base types for the natural numbers and for the Booleans. We consider the standard interpretation using algebraic domains. Plotkin [8] showed that a finite object in general will not be definable, and isolated two nondeterministic constants *PAR* and $\exists_\omega$ such that each computable object is definable in $PCF + PAR + \exists_\omega$.

The first result to be discussed is

Theorem 1. *If Φ is computable and hereditarily total, then there is a PCF definable $\Psi \sqsubseteq \Phi$ that is also total.*

For details, see [4,5]

Escardó [1,2] extended *PCF* to $R - PCF$, adding base types for the reals and the unit interval I, using continuous domains for the interpretation.
We investigate the hereditarily total objects and obtain

Theorem 2. *The hereditarily total objects in the semantics for $R-PCF$ posess a natural equivalence relation, and the typed structure of equivalence classes can be characterized in the category of limit spaces.*

For details, see [6]

PAR is definable in $R-PCF$, but $\exists_\omega$ is not. It is an open problem if Theorem 1 can be generalized to $R - PCF$.
We will discuss a partial solution of the problem in

Theorem 3. *$\exists_\omega$ is not uniformly $R-PCF$-definable from any hereditarily total object.*

Uniformly definable will mean that the object is definable by one term from each element of the equivalence class.
For details, see [7]

The final result to be discussed is joint with Christian Rørdam [9].
We will compare *PCF* with Kleene's classical approach from 1959, and see that when we restrict ourselves to μ-recursion in higher types of continuous functionals, the differences are only cosmetical. Niggl [3] devised a calculus $\mathcal{M}^\omega$ that essentially is

$$(PCF \text{ - Fixpoints}) + PAR + \mu\text{-operator}.$$

Theorem 4. *$\mathcal{M}^\omega$ is strictly weaker than $PCF + PAR$.*

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, pp. 4–5, 2001.

References

1. Escardó, M. H., *PCF extended with real numbers: a domain-theoretic approach to higher-order exact number computation*, Thesis, University of London, Imperial College of Science, Technology and medicine (1996).
2. Escardó, M. H., *PCF extended with real numbers*, Theoretical Computer Science 162 (1) pp. 79 - 115 (1996).
3. Niggl, K.-H., $\mathcal{M}^\omega$ *considered as a programming language*, Annals of Pure and Applied Logic 99, pp. 73-92 (1999)
4. Normann, D., *Computability over the partial continuous functionals*, Journal of Symbolic Logic 65, pp. 1133 - 1142, (2000)
5. Normann, D.,*The Cook-Berger Problem. A Guide to the solution*,In Spreen, D. (ed.): Electronic Notes in Theoretical Computer Science. 2000-10; 35 : 9
6. Normann, D., *The continuous functionals of finite types over the reals*, To appear in Keimel, Zhang, Liu and Chen (eds.) *Domains and Processes* Proceedings of the 1st International Symposium on Domain Theory, Luwer Academic Publishers
7. Normann, D., *Exact real number computations relative to hereditarily total functionals*, To appear in Theoretical Computer Science.
8. Plotkin, G.,*LCF considered as a programming language*, Theoretical Computer Science 5 (1977) pp. 223 - 255.
9. Rørdam, C., *A comparison of the simply typed lambda calculi* $\mathcal{M}^\omega$ *and* $\mathcal{L}_{PA}$, Cand. Scient. Thesis, Oslo (2000)
10. Scott, D. S., *A theory of computable functionals of higher type*, Unpublished notes, University of Oxford, Oxford (1969).
11. Scott, D. S., *A type-theoretical alternative to ISWIM, CUCH, OWHY*, Theoretical Computer Science 121 pp. 411 - 440 (1993).

Categorical Semantics of Control

Peter Selinger

Department of Computer Science
Stanford University

In this talk, I will describe the categorical semantics of Parigot's $\lambda\mu$-calculus [7]. The $\lambda\mu$-calculus is a proof-term calculus for classical logic, and at the same time a functional programming language with control operators. It is equal in power to Felleisen's $\mathcal{C}$ operator [2,1], except that it allows both a call-by-name and call-by-value semantics. The connection between classical logic and continuation-like control operators was first observed by Griffin [4].

The categorical semantics of the $\lambda\mu$-calculus has been studied by various authors in the last few years [6,5,10]. Here, we give a semantics in terms of *control categories*, which combine a cartesian-closed structure with a premonoidal structure in the sense of Power and Robinson [8]. The call-by-name $\lambda\mu$-calculus (with disjunctions) is an internal language for control categories, in much the same way the simply-typed lambda calculus is an internal language for cartesian-closed categories. Moreover, the call-by-value $\lambda\mu$-calculus is an internal language for the dual class of co-control categories. As a corollary, one obtains a syntactic duality result in the style of Filinski [3]: there exist syntactic translations between call-by-name and call-by-value which are mutually inverse and which preserve the operational semantics.

References

1. P. De Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. Springer LNCS 822, 1994.
2. M. Felleisen. *The calculi of λ_v-conversion: A syntactic theory of control and state in imperative higher order programming languages.* PhD thesis, Indiana University, 1986.
3. A. Filinski. Declarative continuations and categorical duality. Master's thesis, DIKU, Computer Science Department, University of Copenhagen, Aug. 1989. DIKU Report 89/11.
4. T. G. Griffin. A formulae-as-types notion of control. In *POPL '90: Proceedings of the 17th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1990.
5. M. Hofmann and T. Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 387–397, 1997.
6. C.-H. L. Ong. A semantic view of classical proofs: Type-theoretic, categorical, and denotational characterizations. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 230–241, 1996.
7. M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning, St. Petersburg*, Springer LNCS 624, pages 190–201, 1992.

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, pp. 6–7, 2001.

8. J. Power and E. Robinson. Premonoidal categories and notions of computation. *Math. Struct. in Computer Science*, 7(5):445–452, 1997.
9. P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Struct. in Computer Science*, 11(2), 2001. To appear.
10. H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.

Representations of First Order Function Types as Terminal Coalgebras

Thorsten Altenkirch

School of Computer Science and Information Technology
University of Nottingham, UK
txa@cs.nott.ac.uk

Abstract. We show that function types which have only initial algebras for regular functors in the domains, i.e. first order function types, can be represented by terminal coalgebras for certain nested functors. The representation exploits properties of ω^{op}-limits and local ω-colimits.

1 Introduction

The work presented here is inspired by discussions the author had some years ago with Healfdene Goguen in Edinburgh on the question *Can function types be represented inductively?* or maybe more appropriately: *Can function types be represented algebraically?*.

In programming and type theory the universe of types can be divided as follows:

- function types (cartesian closure)
- algebraic types
  - inductive types (initial algebras)
  - coinductive types (terminal coalgebras)

In programming the difference between inductive and coinductive types is often obliterated because one is mainly interested in the collection of partial objects of a certain type. Inspired by Occam's razor it would be interesting if we could explain one class of types by another. Here we try to reduce function types to algebraic types.

The first simple observation is that function spaces can be eliminated using products if the domain is finite. Here we show that function spaces $A \to B$ can be eliminated using coinductive types if the domain A is defined inductively. It is interesting to note that ordinary coinductive types are sufficient only for functions over linear inductive types (i.e. where the signature functor has the form $T(X) = A_1 \times X + A_0$) but in general we need to construct functors defined by terminal coalgebras in categories of endofunctors. Those correspond to nested or nested datatypes which have been the subject of recent work [BM98,AR99, Bla00].

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, pp. 8–21, 2001.