Symposium for Gordon Plotkin

Edinburgh 07-08/09/2006

From Partial Lambda-calculus to Monads

Eugenio Moggi

moggi@disi.unige.it

DISI, Univ. of Genova

AIM: recall influence of Plotkin (and others) on my PhD research (and beyond)

- Mainly overview of published work + personal comments and opinions

  Please interrupt to correct my account or to add your comments

  Main focus on

- Partial lambda-calculus [Mog88] 1984-1988

  but work placed in broader context:

  - partiality in: Logic, Algebra, Computability
  - lambda-calculi as: PL [Lan66, Plo75], ML [Sco93, GMW79, Plo85]
  - domain theory [FJM^{+}96]: classical, axiomatic, synthetic

- Applications of monads [Mac71, Man76]

  - for computational types (lifting and recursion) [Mog89, Mog91] 1988-...
  - in pure functional languages (Haskell) – Wadler et al.
  - for collection types (in databases) – Buneman et al.

  including recent contributions by Plotkin et al. [HPP02]

Partial Lambda-Calculus: Background

Relevant work (1983-1985)

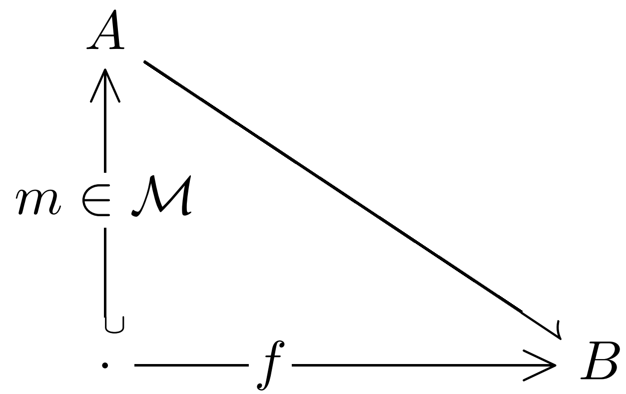
- 1983 categories of partial maps for computability [dPH86, LM84]

  1985 the cleanest exposition on categories of partial maps [RR88]

  $\mathcal{C}$ category (of total maps) $f\colon A \longrightarrow B$

  $\mathcal{M}$ dominion = class of monos (with certain properties)

  $\mathcal{C}_p$ category $\mathcal{P}(\mathcal{C},\mathcal{M})$ of partial maps



  $\top\colon\ 1 \longrightarrow \Sigma$ dominance classifying $\mathcal{M}$ (in topos $\Sigma \subset \Omega$ with certain properties)
  central role in Synthetic Domain Theory [Hyl91, Pho90, Pho91]

Partial Lambda-Calculus: Background

Relevant work (1983-1985)

- 1983 categories of partial maps
- 1984 reformulation of domain theory using partial continuous maps [Plo85]
  Hand-written notes for the TPG course
  - computational lambda-calculus [Mog89] influenced by metalanguage
    $\tau ::= \ldots \mid \tau_\perp \mid \tau_1 \rightharpoonup \tau_2 \quad e ::= \ldots \mid [e] \mid$ let $[x]$ be e in e'
    and its interpretation in $\mathbf{Cpo}_p$
  - Axiomatic Domain Theory [Fio94] influenced by reformulation:
    $\mathbf{Cpo}_p$ paradigmatic example of algebraically compact category [Fre92]

Partial Lambda-Calculus: Background

Relevant work (1983-1985)

- 1983 categories of partial maps
- 1984 reformulation of domain theory using partial continuous maps [Plo85]

Previous relevant work (. . . -1983)

- 1982 more systematic study of categories of partial maps [Obt86, CO87]
- partiality in algebraic specifications: [Bur82]
- partiality in (intuitionistic) logic: LPE [Fou77, Sco79], LPT [Bee85]
- mismatch between lambda-calculus and programming languages [Plo75]

λ_V CBV axioms $(\lambda x.t)v \longrightarrow t[x{:=}v]$ and $(\lambda x.vx) \longrightarrow v$ with $v ::= x \mid \lambda x.t$ values

- λ_p is derived from models like λ_V is derived from operational semantics
- $\lambda_V \subset \lambda_c \subset \lambda_p$ are correct (but incomplete) for CBV (on untyped λ-terms)
- λ_c on (simply typed) λ-terms is inverse image of $\lambda\beta\eta$ w.r.t. CBV CPS [SF93]

Partial Lambda-Calculus: Approach and Results

- Systematic and *unbiased* investigation of partiality
  - in the setting of both intuitionistic and classical logic
  - in set-theoretic models and order-theoretic models ($\mathbf{PoSet}$ or $\mathbf{Cpo}$)

Axiomatization in Logic of Partial Terms (LPT) [Bee85]

  - variables x range on *existing elements*, but terms e could be undefined (in LPE free x range on *partial elements*, and e denote partial elements)
  - $e\downarrow$ means "e defined" (or "evaluation of e terminates")
    $e_1 = e_2$ means "e_1 and e_2 defined and equal" (similarly $e_1 \leq e_2$)
    $e_1 \simeq e_2 \overset{\Delta}{\iff} (e_1\downarrow \vee e_2\downarrow) \supset e_1 = e_2$ (but $e_1 \lesssim e_2 \overset{\Delta}{\iff} e_1\downarrow \supset e_1 \leq e_2$)

Partial Lambda-Calculus: Approach and Results

- Systematic and *unbiased* investigation of partiality
  $\lambda_p\beta\eta$-models as partial Combinatory Algebras (pCA) + extensionality
  For $\lambda_p\beta$-models adapt FOL axiomatization of $\lambda\beta$-models in [Mey82]

pCA $Kxy = x \quad (Sxy)\downarrow \quad Sxyz \simeq xz(yz)$
abstraction $[x]e$ s.t. $([x]e)\downarrow$ and $([x]e)x \simeq e$ definable by induction [Bar84]

ext.$\simeq$ $(\forall z.xz \simeq yz) \supset x = y$

ext.$\lesssim$ $(\forall z.xz \lesssim yz) \supset x \leq y$ (and monotonicity $x_1 \leq x_2 \wedge y_1 \leq y_2 \supset x_1y_1 \lesssim x_2y_2$)

tot $(xy)\downarrow$ (i.e. application always defined) – $\lambda\beta\eta$ = $\lambda_p\beta\eta$ + tot ($\perp$ not built-in)

	intuitionistic	classical
set-theoretic	$J\lambda_p\beta\eta$	$K\lambda_p\beta\eta$
order-theoretic	$J\lambda_p^{\leq}\beta\eta$	$K\lambda_p^{\leq}\beta\eta$

Partial Lambda-Calculus: Approach and Results

- Systematic and *unbiased* investigation of partiality
- Comparison among λ_p-calculi (and also λ- and λ_V-calculi)
  - on equations $e_1 = e_2$ between pure λ-terms – quite complex

$$\begin{array}{ccccccc} \lambda_V\beta\eta & \subset & J\lambda_p\beta\eta & \subset & J\lambda_p^{\leq}\beta\eta & & \\ & & \cap & & \cap & & \\ & & K\lambda_p\beta\eta & \subset & K\lambda_p^{\leq}\beta\eta & \subset & \lambda\beta\eta(= J\lambda\beta\eta = K\lambda^{\leq}\beta\eta) \end{array}$$

  - on definedness assertions $e \downarrow$ for pure λ-terms

$$\lambda_V\beta\eta \quad = \quad J\lambda_p\beta\eta \quad = \quad K\lambda_p^{\leq}\beta\eta \quad \subset \quad \lambda\beta\eta$$

Corollary of computational adequacy result in [Plo85]

Partial Lambda-Calculus: Approach and Results

- Systematic and *unbiased* investigation of partiality
- Analogies between λ_p-calculi and λ-calculus (results and proof techniques)

  Completeness results for type hierarchies (simply typed fragment)
  - $\lambda\beta\eta$ complete for $\mathbf{Set}(N)$ [Fri75] and $\mathbf{Cpo}(N_\perp)$ [Plo82]
  - $K\lambda_p\beta\eta$ complete for $\mathbf{Set}_p(N)$ – partial functions
  - $K\lambda_p^{\leq}\beta\eta$ complete (on inequalities) for $\mathbf{PoSet}_p(P(N))$

    Unknown if similar result holds in $\mathbf{Cpo}_p$

  Results proved using *logical relations* for λ_p-calculus

  $fR_{\tau\rightharpoonup\tau'}g \stackrel{\Delta}{\Longleftrightarrow} \forall x,y.\, xR_\tau y \supset fx\widetilde{R}_{\tau'}gy$ where $e_1\widetilde{R}e_2 \stackrel{\Delta}{\Longleftrightarrow} (e_1\downarrow \vee e_2\downarrow) \supset e_1Re_2$

Partial Lambda-Calculus: Approach and Results

- Systematic and *unbiased* investigation of partiality
- Analogies between λ_p-calculi and λ-calculus (results and proof techniques)

  Characterization of equality (on untyped pλ-terms) by confluent reduction

  Very complex in comparison with characterization of equality in $\lambda\beta\eta$

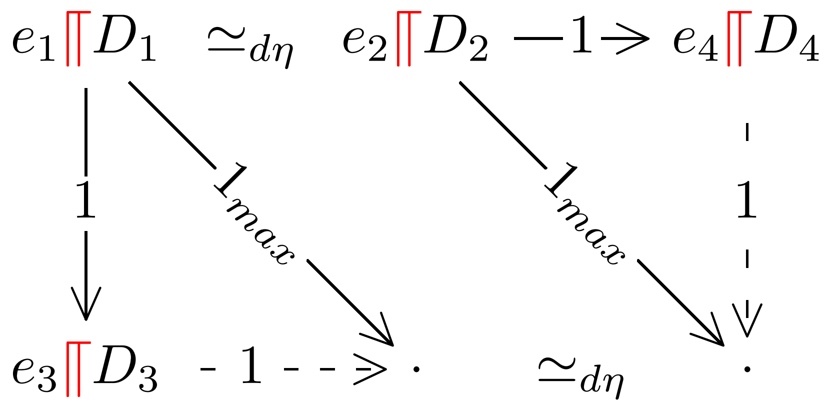
$$e \| D \qquad \frac{e_1 \| D_1 \qquad e_2 \| D_2}{e_1 e_2 \| \underline{\{e_1 e_2\} \cup D_1 \cup D_2}} \qquad \frac{e \| \underline{D_1 \cup D_2}}{(\lambda x.e \| D_1) \| D_2}\; x \notin \mathrm{FV}(D_2) \quad \ldots \text{ [Obt86]}$$

$$\beta_p \quad (\lambda x.e_1 \| D_1) e_2 \| \underline{\{(\lambda x.e_1 \| D_1) e_2\} \cup D_2} \longrightarrow e_1[x{:}{=}e_2] \| \underline{(D_1[x{:}{=}e_2]) \cup D_2}$$

On pλ-terms one should use one-step parallel β_p-reduction $\;\xrightarrow{}_1\!\!>$

- $\simeq$ in $J\lambda_p\beta\eta$ characterized by $\;-_1\!\!\!>\;$ and decidable equivalence $\simeq_{d\eta}$ [Per88]
- $\lesssim$ in $J\lambda_p^{\leq}\beta\eta$ characterized by $\;-_1\!\!\!>\;$ and decidable preorder $\lesssim_{d\eta}$

diamond property of $\;-_1\!\!\!>$
up to $\simeq_{d\eta}$ (or $\lesssim_{d\eta}$)



Partial Lambda-Calculus: Concluding Remarks

- Too much focus on partiality!
  - λ_p-calculus is not sound for reasoning on CBV languages (e.g. SML) with other computational effects, while λ_V-calculus is still sound

  Generalize $\Longrightarrow$ Notions of computations as Monads

  Work in this direction started after PhD submission, and was discussed during the PhD exam (Hyland, Milner, Plotkin). Essential contributions in early stages:
  - Plotkin $\Longrightarrow$ lots of examples inspired by Denotational Semantics
    Essential to get reassurance that we were on the right track
  - Hyland, Kock,... $\Longrightarrow$ pointers to the relevant Category Theory literature
    Basically all the necessary mathematics was already there [Koc72, Man76]

Partial Lambda-Calculus: Concluding Remarks

- Too much focus on partiality!
  - λ_p-calculus is not sound for reasoning on CBV languages (e.g. SML) with other computational effects, while λ_V-calculus is still sound

  Generalize $\Longrightarrow$ Notions of computations as Monads

- Too generic on what kind of partiality!
  - λ_p-calculus does not adequately capture partiality in the setting of computability and domain theory

  Specialize $\Longrightarrow$ Axiomatic and Synthetic Domain Theory

ADT Axioms for a dominance Σ in a (order-enriched) category [FP94, Fio94] to interpret a metalanguage (e.g. FPC) with recursive definitions and recursive types (in the category of partial maps [Fre90, Fre92])

SDT Axioms for a dominance $\Sigma \subset \Omega$ in a topos to ensure existence of full sub-category of domains (Scott's slogan "domains as sets") [Hyl91, Pho90, Pho91] (some axioms incompatible with classical logic)

Monads and Computational Types

- $\mathcal{C}$ category of *sets and maps*, τ set of *values* and $M\tau$ set of *programs* of type τ
- M should have (at least) the structure of a monad

$$\frac{e\colon\tau}{\textit{ret}\ e\colon M\tau} \qquad \frac{e_1\colon M\tau_1 \quad x\colon\tau_1 \vdash e_2\colon M\tau_2}{\textit{do}\ x \leftarrow e_1;\ e_2\colon M\tau_2}$$

$$\begin{aligned}
\textit{do}\ x \leftarrow (\textit{ret}\ e_1);\ e_2 &= e_2[x\colon=e_1] \\
\textit{do}\ x_2 \leftarrow (\textit{do}\ x_1 \leftarrow e_1;\ e_2);\ e_3 &= \textit{do}\ x_1 \leftarrow e_1;\ (\textit{do}\ x_2 \leftarrow e_2;\ e_3) \quad x_1 \notin \mathrm{FV}(e_3) \\
\textit{do}\ x \leftarrow e;\ (\textit{ret}\ x) &= e
\end{aligned}$$

PL CBV programming language interpreted in Kleisli category $\mathcal{C}_M$ for a given M

Like interpretation of Plotkin's metalanguage in $\mathbf{Cpo}_p$ [Plo85]

ML_M Monadic metalanguage interpreted in $\mathcal{C}$ (possibly with several monads)

extends *conservatively* a typed λ-calculus for $\mathcal{C}$ with computational types

Semantics of programming languages via translation in ML_M followed by interpretation of M

Monads and Computational Types

- $\mathcal{C}$ category of *sets and maps*, τ set of *values* and $M\tau$ set of *programs* of type τ
- M should have (at least) the structure of a monad
- Relation to partiality: partial map classifier $\mathcal{C}(A, B_\perp) \cong \mathcal{C}_p(A, B)$ is a monad
  the category $\mathcal{C}_p$ of partial maps isomorphic to the Kleisli category $\mathcal{C}_\perp$
  monad morphism $(-)_\perp \stackrel{\cdot}{\rightarrow} M$ in model for Axiomatic/Synthetic Domain Theory
  $\Longrightarrow$ recursive definitions of elements in $M\tau$ (is a EM-algebra for $(-)_\perp$)
  M usable in recursive domain equations (as M extends to $\mathcal{C}_\perp$)
- Relation to λ_V: CBV translation $(-)^v$ in ML_M with $V = (MV)^V$ – $x_i\colon V \vdash e^v\colon MV$
  - $\lambda_V \subset \lambda_c =$ calculus on (untyped) λ-terms induced by $(-)^v$

[SF93] $\lambda_c =$ inverse image of $\lambda\beta\eta$ w.r.t. CBV CPS translation $\overline{(-)}$ of [Plo75]
Continuations as worst case instance of computational types
Corollary: completeness results for simply typed fragment
ML_M complete for $\mathbf{Set}$ and $\mathbf{Cpo}$ with monad $M- = A^{(A^-)}$
In $\mathbf{Set}$ interpret A and base type with N, in $\mathbf{Cpo}$ with $N_\perp$

Monads in Functional Programming

- *Extension* of Haskell with computational types [Wad90, Wad92a, Wad92b]

  Motivation: to *mimic* impure features in a pure functional language
  and also hide how much impure features to mimic

  Analogy with monadic metalanguage extending *conservatively* λ-calculus

- Further ideas originally developed within Haskell
  - Monadic encapsulation of effects *run*: $(\forall\alpha.M_\alpha\tau) \to \tau$ [LP94, LS97, MS01]
    No need to change Haskell's type system with a type-and-effect system [LG88, TJ94]
  - *Refinement* of computational types with effects $M_\epsilon\tau$ [Wad98, BHM02]
    Refined type *semantically* $(A \in \mathbf{Set}, P \subseteq A)$, *syntactically* $|M_\epsilon\tau| = M|\tau|$
  - Monadic *value recursion* $\dfrac{x\colon M\tau \vdash e\colon M\tau}{\textit{Mfix}\ x.e\colon M\tau}$ [EL00, Erk02, MS04]

    Operational semantics of *Mfix* , only *loose axiomatization*
    Mfix $x.$*ret* $e =$ *fix* $x.$*ret* e *Mfix* $(x_2.$*do* $x_1 \leftarrow c; e) =$ *do* $x_1 \leftarrow c;$ (*Mfix* $x_2.e)$

Monads and Collection Types [BNTW95]

- From list comprehension to monad comprehension [Wad92a]

$$\frac{x_1{:}M\tau_1,\ldots,x_{j-1}{:}M\tau_{j-1}\vdash e_j{:}M\tau_j \quad 1\leq j\leq n \qquad x_1{:}M\tau_1,\ldots,x_n{:}M\tau_n\vdash e{:}\tau}{\{e|x_1\leftarrow e_1,\ldots,x_n\leftarrow e_n\}{:}M\tau \quad \text{comprehension notation} \qquad \textit{do}\ x_1\leftarrow e_1;\ldots;x_n\leftarrow e_n;\ \textit{ret}\ e{:}M\tau \quad \text{do-notation}}$$

  - $c\in M\tau$ collection of elements in τ – $M\tau$ collection type
  - $c\in M\tau$ computes values in τ – $M\tau$ computational type
- A collection M should have (at least) the structure of a monad, but also
  - a collection $c\in M\tau$ should have a finite numbers of elements
  - empty collection $0{:}M\tau$ and union $c_1+c_2{:}M\tau$ of two collections $c_1,c_2{:}M\tau$

  [Man98] defines and studies collection monads in $\mathbf{Set}$

  M collection monad $\Longleftrightarrow$ induced by a *balanced* algebraic theory (Σ,E)
  i.e. $\mathrm{FV}(e_1)=\mathrm{FV}(e_2)$ for all equations $(e_1,e_2)\in E$
- M collection monad $\Longrightarrow mem_X{:}MX\longrightarrow\mathcal{P}_{fin}(X)$ monad morphism

Combining Monads

- Motivation: modular approach to semantics of programming languages

  ideally semantics as easy to extend as syntax of PL [Mos90]
- Approach: build monad for a complex language from simple monads
  Ideally from monads capturing *one* computational effect

  Early proposals, e.g. monad transformers [BHM02]:
  methodologically and mathematically unsatisfactory

Combining Monads

- Motivation: modular approach to semantics of programming languages
- New proposal [PP02, PP03, HPP02]:
  revisits correspondence between *algebraic theories* and monads
  Originally the correspondence was used to establish computational adequacy between denotational and operational semantics for a monadic language with *algebraic* effects.
- May identify better monads, e.g. monad MX induced by read/write operations $lkp\colon L \longrightarrow MV$ and $upd\colon L, V \longrightarrow M1$ on set L of global variables is strictly included in naive state monad, when L is infinite
  - $MX \subset (X \times S)^S$ with $S = V^L$
    $c \in MX \iff \forall s_1.\exists R, W \subseteq_{fin} L.\forall s_2.s_1 =_R s_2 \supset cs_2 = (x_1, s'_2)$
    where $(x_1, s'_1) = cs_1$ and s'_2 s.t. $s'_2 =_W s'_1$ and $s'_2 =_{\overline{W}} s_2$
- Main limitation: continuation monads don't fit in algebraic framework
- Technical complications: need to go beyond plain algebraic theories
  Enriched setting (e.g. **Cpo**-enrichment), and arities not necessarily finite (e.g. countable).

Combining Monads

- Motivation: modular approach to semantics of programming languages
- New proposal [PP02, PP03, HPP02]:
  revisits correspondence between *algebraic theories* and monads
- Mathematically clean: use few natural combinations for algebraic theories
  - $+$ disjoint union of operations and equations of two theories T_1 and T_2
  - $\otimes$ disjoint union of operations and equations + equations for commutativity

$$op_1(op_2(x_{i,j}|j \in n)|i \in m) = op_2(op_1(x_{i,j}|i \in m)|j \in n)$$

  operation op_1 of theory T_1 commutes with op_2 of theory T_2

  Most monad transformers defined using (conterpart of) $+$ and $\otimes$ (on monads).

Combining Monads

- Motivation: modular approach to semantics of programming languages
- New proposal [PP02, PP03, HPP02]:
  revisits correspondence between *algebraic theories* and monads
- Mathematically clean: use few natural combinations for algebraic theories
  - $+$ disjoint union of operations and equations of two theories T_1 and T_2
  - $\otimes$ disjoint union of operations and equations + equations for commutativity

$$op_1(op_2(x_{i,j}|j \in n)|i \in m) = op_2(op_1(x_{i,j}|i \in m)|j \in n)$$

  operation op_1 of theory T_1 commutes with op_2 of theory T_2
- The approach can be used for a modular approach to collection types too!
  in this case plain algebraic theories (finitary monads on $\mathbf{Set}$) suffice [Man98]
  $+$ and $\otimes$ preserve balanced algebraic theories
  problematic monads (like continuations) are ruled out by application domain.

Conclusions

- I would like to conclude by mentioning a very influential *unpublished* work by

  Rod Burstall
  Robin Milner
  Gordon Plotkin et al.
- well-known worldwide, and
- particularly appreciated by CS researchers that had the opportunity to be or visit Edinburgh in the last 20 years

Conclusions

- I would like to conclude by mentioning a very influential *unpublished* work by

  Rod Burstall
  Robin Milner
  Gordon Plotkin et al.
- well-known worldwide, and
- particularly appreciated by CS researchers that had the opportunity to be or visit Edinburgh in the last 20 years

The Laboratory for Foundations of Computer Science

Many Thanks, Gordon!

Technical Details: CBV

- λ_V-calculus terms $t{::=}\,x \mid \lambda x.t \mid t_1 t_2$ values $v{::=}\,x \mid \lambda x.t$ and axioms

$$\beta_V \;\; (\lambda x.t)v = t[x{:=}v] \qquad \eta_V \;\; (\lambda x.vx) = v \text{ if } x \notin \mathrm{FV}(v)$$

- CBV translation $(-)^v$ in ML_M and CBV CPS translation $\overline{(-)}$ in λ-calculus

t	x	$\lambda x.t$	$t_1 t_2$
t^v	*ret* x	*ret* $(\lambda x{:}V.t^v)$	*do* $x_1 \leftarrow t_1^v;\ x_2 \leftarrow t_2^v; x_1 x_2$
$\overline{t}$	$\lambda k.kx$	$\lambda k.k(\lambda x.\overline{t})$	$\lambda k.\overline{t}_1(\lambda x_1.\overline{t}_2(\lambda x_2.x_1 x_2 k))$

Monad of continuations $M\tau = A^{(A^\tau)}$ with answers in A

$$\textit{ret}\ e = \lambda k.ke \qquad \textit{do}\ x \leftarrow e_1; e_2 = \lambda k.e_1(\lambda x.e_2 k)$$

Technical Details: CBV

- λ_V-calculus terms $t ::= x \mid \lambda x.t \mid t_1 t_2$ values $v ::= x \mid \lambda x.t$ and axioms

$$\beta_V \quad (\lambda x.t)v = t[x := v] \qquad \eta_V \quad (\lambda x.vx) = v \text{ if } x \notin \mathrm{FV}(v)$$

- inverse image of $\lambda\beta\eta$ w.r.t. CBV CPS translation [SF93] $\beta_V + \eta_V +$

$$\begin{array}{rcll} (\lambda x.x)t &=& t & \\ E[(\lambda x.t_1)t_2] &=& (\lambda x.E[t_1])t_2 & x \notin \mathrm{FV}(E) \\ E[t_1 t_2 t_3] &=& (\lambda x.E[x t_3])(t_1 t_2) & x \notin \mathrm{FV}(E, t_3) \\ (\lambda x.E[x' x])t &=& E[x' t] & x \notin \mathrm{FV}(E, x') \end{array}$$

where $E ::= [] \mid vE \mid Et$ CBV evaluation contexts, or alternatively

$$\begin{array}{rcll} (\lambda x.x)t &=& t & \\ (\lambda x_2.t_3)((\lambda x_1.t_2)t_1) &=& (\lambda x_1.(\lambda x_2.t_3)t_2)t_1 & x_1 \notin \mathrm{FV}(t_3) \\ t_1 t_2 t_3 &=& (\lambda x.x t_3)(t_1 t_2) & x \notin \mathrm{FV}(t_3) \\ v t_1 t_2 &=& (\lambda x.vx)(t_1 t_2) & x \notin \mathrm{FV}(v) \end{array}$$

References

[Bar81] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1981.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984. revised edition.

[Bar92] H.P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*. Oxford Univ. Press, 1992.

[Bee85] M.J. Beeson. *Foundations of Constructive Mathematics*. Springer Verlag, 1985.

[BHM02] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In Gilles Barthe, Peter Dybjer, Luís Pinto, and João Saraiva, editors, *Advanced Lectures from Int. Summer School on Applied Semantics, APPSEM 2000 (Caminha, Portugal, 9–15 Sept. 2000)*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer-Verlag, Berlin, 2002.

[BNTW95] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1), 1995.

[Bur82] P. Burmeister. Partial algebras - survey of an unifying approach towards a two-valued model theory for partial algebras. *Algebra Universalis*, 15, 1982.

[CO87] P-L. Curien and A. Obtulowicz. Partiality, cartesian closedness and toposes. Ecole Normale Sup., preprint, 1987.

[dPH86] R. di Paola and A. Heller. Dominical categories. *Journal of Symbolic Logic*, 1986.

[EL00] Levent Erkök and John Launchbury. Recursive monadic bindings. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP'00*, pages 174–185. ACM Press, September 2000.

[Erk02] Levent Erkök. *Value Recursion in Monadic Computations*. PhD thesis, OGI School of Science and Engineering, OHSU, Portland, Oregon, 2002.

[Fio94] Marcelo P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1994.

[FJM+96] M. P. Fiore, A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini, and I. Stark. Domains and denotational semantics: History, accomplishments and open problems. In *Bulletin of EATCS*, volume 59, pages 227–256. 1996.

[Fou77] M.P. Fourman. The logic of topoi. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic*. North Holland, 1977.

[FP94] Marcelo P. Fiore and Gordon D. Plotkin. An axiomatization of computationally adequate domain theoretic models of FPC. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 92–102, Paris, France, 1994. IEEE Computer Society Press.

[Fre90] P. Freyd. Recursive types reduced to inductive types. In J. Mitchell, editor, *Proc. 5th Symposium in Logic in Computer Science*, Philadelphia, 1990. I.E.E.E. Computer Society.

[Fre92] Peter J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proc. of LMS Symp., Durham, UK, 20–30 July 1991*, volume 177 of *London Math. Society Lecture Note Series*, pages 95–106. Cambridge University Press, Cambridge, 1992.

[Fri75] H. Friedman. Equality between functionals. In R. Parikh, editor, *Logic Colloquium '75*, volume 453 of *LNM*. Springer Verlag, 1975.

[GMW79] M.J.C. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *LNCS*. Springer Verlag, 1979.

[HPP02] Martin Hyland, Gordon Plotkin, and John Power. Combining computational effects: Commutativity and sum. In R. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Proc. IFIP Int. Conf. on Theoretical Computer Science*, volume 223 of *IFIP Conf. Proc.* Kluwer Academic Publishers, Dordrecht, 2002.

[Hyl82] J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*. North Holland, 1982.

[Hyl91] J.M.E. Hyland. First steps in synthetic domain theory. In A. Carboni, C. Pedicchio, and G. Rosolini, editors, *Conference on Category Theory '90*, volume 1488 of *LNM*. Springer Verlag, 1991.

[Klo80] J.W. Klop. *Combinatory Reduction Systems*. PhD thesis, Mathematical Center Amsterdam, 1980. Tracts 129.

[Koc72] A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23, 1972.

[Lan66] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3), 1966.

[LG88] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *Conf. Rec. 15th Ann. ACM Symp. Princ. of Prog. Langs.*, pages 47–57, 1988.

[LM84] G. Longo and E. Moggi. Cartesian closed categories of enumerations for effective type-structures. In G. Kahn, D. MacQueen, and G.D. Plotkin, editors, *Symp. on Semantics of Data Types*, volume 173 of *LNCS*. Springer Verlag, 1984.

[LP94] John Launchbury and Simon L. Peyton Jones. Lazy functional state threads. In *Proc. ACM SIGPLAN '94 Conf. Prog. Lang. Design & Impl.*, pages 24–35, 1994.

[LS97] John Launchbury and Amr Sabry. Monadic state: Axiomatization and type safety. In *Proc. 1997 Int'l Conf. Functional Programming*. ACM Press, 1997.

[Mac71] S. MacLane. *Categories for the Working Mathematician*. Springer Verlag, 1971.

[Man76] E. Manes. *Algebraic Theories*. Graduate Texts in Mathematics. Springer, 1976.

[Man98] E Manes. Implementing collection classes with monads. *Mathematical Structures in Computer Science*, 8(3), 1998.

[McC84] D.C. McCarty. *Realizability and Recursive Mathematics*. PhD thesis, University of Oxford, 1984.

[Mey82] A. Meyer. What is a model of the lambda calculus? *Information and Computation*, 52, 1982.

[Mog88] E. Moggi. *The Partial Lambda-Calculus*. PhD thesis, University of Edinburgh, 1988. available as CST-53-88.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *4th LICS Conf.* IEEE, 1989.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[Mos90] P. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North Holland, 1990.

[MS01] E. Moggi and A. Sabry. Monadic encapsulation of effects: a revised approach (extended version). *Journal of Functional Programming*, 11(6):591–627, 2001.

[MS04] E. Moggi and A. Sabry. An abstract monadic semantics for value recursion. *Theoretical Informatics and Applications*, 38, 2004.

[Obt86] A. Obtulowicz. The logic of categories of partial functions and its applications, 1982 thesis. *Dissertationes Mathematicae*, 241, 1986.

[Per88] R.P. Perez. Decidability of the restriction equational theory in the partial lambda calculus. presented at the workshop on the typed lambda calculus, Turin, Italy, 15-20 May 1988, 1988.

[Pho90] W. Phoa. Effective domains and intrinsic structure. In *5th LICS Conf.* IEEE, 1990.

[Pho91] W. Phoa. *Domain Theory in Realizability Toposes*. PhD thesis, Cambridge University, 1991. revised version 1991 available as CST-82-91 from Edinburgh Univ., Dept. of Comp. Sci.

[Plo75] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1, 1975.

[Plo77] G.D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5, 1977.

[Plo81] G.D. Plotkin. Postgraduate lecture notes in domain theory (incorporating the “pisa notes”). Edinburgh Univ., Dept. of Comp. Sci., 1981.

[Plo82] G.D. Plotkin. Notes on completeness of the full continuous type hierarchy. Manuscript, Lab. of Comp. Sci., M.I.T., 1982.

[Plo85] G.D. Plotkin. Denotational semantics with partial functions. Lecture Notes at C.S.L.I. Summer School, 1985.

[PP02] Gordon Plotkin and John Power. Notions of computation determine monads. In Mogens Nielsen and Uffe Engberg, editors, *Proc. of 5th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS 2002 (Grenoble, France, 8–12 Apr. 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, Berlin, 2002.

[PP03] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[Ros86] G. Rosolini. *Continuity and Effectiveness in Topoi*. PhD thesis, University of Oxford, 1986.

[RR88] E. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, 79(2), 1988.

[Sco70] D.S. Scott. Outline of a mathematical theory of computation. Technical Report PRG-2, Oxford Univ. Computing Lab., 1970.

[Sco79] D.S. Scott. Identity and existence in intuitionistic logic. In M.P. Fourman, C.J. Mulvey, and D.S. Scott, editors, *Applications of Sheaves*, volume 753 of *LNM*. Springer Verlag, 1979.

[Sco80] D.S. Scott. Relating theories of the λ-calculus. In R. Hindley and J. Seldin, editors, *To H.B. Curry: essays in Combinarory Logic, lambda calculus and Formalisms*. Academic Press, 1980.

[Sco93] D.S. Scott. A type-theoretic alternative to CUCH, ISWIM, OWHY. *Theoretical Computer Science*, 121, 1993.

[SF93] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.

[TJ94] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Inform. & Comput.*, 111(2):245–296, 1994.

[Wad90] Philip Wadler. Comprehending monads. In *Proceedings of Symposium on Lisp and Functional Programming*, pages 61–78, Nice, France, June 1990. ACM.

[Wad92a] P. Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2, 1992.

[Wad92b] Philip Wadler. The essence of functional programming. In *the Symposium on Principles of Programming Languages (POPL '92)*, pages 1–14. ACM, January 1992.

[Wad98] P. Wadler. The marriage of effects and monads. In *International Conference on Functional Programming*. ACM Press, 1998.