PROVABLE ISOMORPHISMS AND DOMAIN EQUATIONS IN MODELS OF TYPED LANGUAGES[1]

(Preliminary version)

by

Kim B. Bruce
Department of Mathematical Sciences
Williams College
Williamstown, Ma. 01267

Giuseppe Longo
Dipartimento di Informatica
Universita' di Pisa
Pisa, Italy

INTRODUCTION

The interest in isomorphisms in models for typed languages mainly arises from work done in the study of denotational semantics. This work has led to the notion of solving domain equations as a tool for giving mathematical meaning to common constructs in computer science, e.g. recursive definitions of data types.In this paper we investigate the question of which domain equations must hold in all models of a paradigmatic functional language, the typed lambda calculus, and of second order (polymorphic) extensions of it.

In order to do this we will have to specify what we mean by the solution to a domain equation. Here we will deal with one reasonable definition for this notion. With respect to this definition, we give a complete account of the domain equations which must hold in all models of both the first and second order typed lambda calculus. Our main interest is in the second order lambda calculus, but our methods work equally well for first order typed lambda calculus. For simplicity, we present the results in more detail for the first order case and simply sketch the results for the more interesting second order case.

If E_1 and E_2 are expressions involving type variables $t_1,t_2,...,t_n$ then a solution to the domain equation $E_1=E_2$ typically involves an assignment of types (or domains) to the type variables $t_1,t_2,...,t_n$, and a pair of maps establishing an "isomorphism" between the domains denoted by E_1 and E_2 with the given assignment of types to the type variables. (Note that the "=" in the equation thus really represents an isomorphism, not usually equality or identity.) For example Scott's original solution to the domain equation $t=[t\to t]$ involved constructing the domain D_∞ (a lattice) and structure preserving maps:

$$\phi : D_\infty \to [D_\infty \to D_\infty],$$

and $$\psi : [D_\infty \to D_\infty] \to D_\infty$$

such that $\phi\cdot\psi$ and $\psi\cdot\phi$ are the identity maps on $[D_\infty\to D_\infty]$ and D_∞ respectively. Thus $[D_\infty\to D_\infty]$ and

[1]Research supported by N.S.F. grant MCS-8402700 and Italian C.N.R. contract 8300031.01, under the auspices of the U.S.-Italy Cooperative Science Program.

D_∞ are isomorphic as lattices. What is the appropriate definition of such an isomorphism in a general (not necessarily lattice or c.p.o.) model? It seems clear that this should depend on the category of domains we are considering as solutions to the equations. Thus when we are modelling the semantics of a language using lattices to interpret types, the isomrphisms should be lattice isomorphisms. Similarly if we are modelling types with c.p.o.'s, we should only be considering c.p.o. isomorphisms. But what if we are looking at term models? What then?

The problem thus is to somehow reflect this varying definition of isomorphism when discussion the general question of when a domain equation has a solution in some type frame (involving for example, lattices, consistently complete algebraic c.p.o.'s). In particular the question we wish to address in this paper is: "Which domain equations must hold in *all* possible type frames (e.g. in all models of the typed lambda calculus)?" Our definition of isomorphism thus must make sense for all possible type frames (including those arising from term models). Happily each type frame contains classes of objects which may be considered as candidate functions to demonstrate the isomorphism of domains. Let $\mathcal{D}^\mathcal{T}=\{D_i: i\in I\}$ be the set of types for a model $\mathcal{T}$ of the typed lambda calculus. Then $\mathcal{D}^\mathcal{T}$ is closed under creation of function spaces. I.e. if $D_i, D_j \in \mathcal{D}^\mathcal{T}$ then $[D_i\to D_j]$, representing the type of functions from D_i to D_j, must also be in $\mathcal{D}^\mathcal{T}$. Since $[D_i\to D_j]$ need not contain all functions from D_i to D_j (e.g. in lattice models, $[D_i\to D_j]$ typically contains only continuous functions from D_i to D_j), we will take these to represent our "structure-preserving" maps. Thus we will say D_i is isomorphic to D_j (with respect to $\mathcal{T}$) if and only if there are $\phi \in [D_i\to D_j] \in \mathcal{D}^\mathcal{T}$ and $\psi \in [D_j\to D_i] \in \mathcal{D}^\mathcal{T}$ such that $\phi\cdot\psi$ and $\psi\cdot\psi$ are the identity functions on D_j and D_i. This corresponds exactly to what has traditionally been used in the case of models which are lattices or c.p.o.'s whose function spaces consist of all continuous functions.

We are also interested in knowing when we can prove in the formal frameword of the typed λ-calculus that these isomorphisms must exist. Thus we are interested in when two types α and β are provably isomorphic, i.e. when there are terms M and N of types $\alpha\to\beta$ and $\beta\to\alpha$, respectively, such that $\lambda\beta\eta \vdash M\cdot N=I_\beta$, $N\cdot M=I_\alpha$. We will see that the characterization of provably isomorphic types leads to the solution of our original problem. We emphasize here that we are concerned with all possible type frames and not just those built up inductively from a fixed set of base types.

Our original motivation for examining this question was in trying to understand the models of an extension of the typed lambda calculus, the second order (polymorphic) lambda calculus. The (typed) λ-calculus provides a mathematically simple, but powerful, setting for relating the syntax and semantics of traditional functional programming languages. As a matter of fact, any functional language is essentially based on (functional) application and abstraction, which are exactly the defining notions in the λ-calculus. More recently Girard[1971], De Bruijn [1980], and Reynolds[1974] have each suggested extending the typed λ-lambda calculus by the

addition of types as parameters. This mechanism has provided a very fruitful extension of the pure language both for the purposes of logic and of computer science. In actual programming, polymorphic types and the possibility of updating types during computations provide powerful and useful tools in developing flexible programs. The second order λ-calculus, defined in similar ways by the various authors above, provides a sufficiently elegant and simple language for the investigation of such a complicated syntactic construct.

One of the major open problems in the semantics of second order λ-calculus is the construction of natural mathematical models. Aside from trivial models and term models, the only published models of the second order λ-calculus are constructed using universal domains. While there are three variants of these models known (constructed from closures [McCracken 79], finitary retracts [McCracken 85], and finitary projections [Bruce and Meyer 84], all due in varying degrees to Dana Scott), no other interesting models are known. After first conjecturing that natural set-theoretic models were possible, Reynolds[1984] later showed that in fact there is no way of interpreting types as sets when modelling polymorphism and parametric types. This is actually proved by exhibiting a syntactically derived isomorphism between types which must hold of all models which satisfy certain properties. This isomorphism, however, cannot hold between sets. While this isomorphism depended on particular properties of sets, it seemed possible that other strange isomorphisms might have to exist in all models of the second order λ-calculus. This was the original motivation for this paper. We asked which isomorphisms must hold in all second order models, and, more generally, in all typed models. Thus we find the answer for the second order λ-calculus to be the more interesting (and perhaps more unexpected).

The question as to which isomorphisms must hold in all models of the typed λβη- calculus has quite a simple answer, as we see below (the answer for the typed λβ-calculus is even easier). In section 1 we characterize the valid isomorphisms in all functional type structures by a simple decidable theory of type equality. This characterization (for both the first order and second order cases) was conjectured by Albert Meyer. Our proof is based on non-trivial combinatorial facts about the *type-free* λ-calculus (the "invertibility theorem" of Dezani-Ciancaglini [1976]). We also mention briefly a recent alternative proof by Meyer and Statman[1984]. In section 2 we give a characterization for the more intriguing case of second order models. This proof uses the ideas and results from our earlier proof for first order type models. We currently see no way of extending the Meyer, Statman proof to this more complex situation. Retractions between types (a retraction is an invertible embedding) may be even more important and interesting than isomorphisms. For example, Meyer and Wand [1985] show that the natural connection between standard and continuation semantics in the typed λ-calculus can be explained using retractions. That is, the meaning of a closed term in the standard semantics is a retraction of the meaning of the corresponding "continuized" term. Section 3 discusses retractions between types , where we provide some completeness results for the λβ- calculus and some conjectures for the λβη- calculus. Finally we remark that, by the tidy

correspondence between Cartesian Closed Categories and typed $\lambda\beta\eta$- calculus, the results in section 1 below characterize the isomorphisms between objects which are valid in all Cartesian Closed Categories.

SECTION 1

Definition 1.1 The set Tp of type-symbols is the least set containing atomic or base type-symbols $\varphi,\psi,\ldots$ and such that, if σ,τ are in Tp, then also $(\sigma\to\tau)$ is in Tp.

Typed and type-free terms of λ- calculus (λ-terms) are defined as usual. We write $M : \sigma$ when the λ- term M has type σ. Provability in $\lambda\beta(\eta)$, typed or type-free according to the context, is expressed by $\lambda\beta(\eta) \vdash \ldots$.

Definition 1.2 Let $\sigma,\tau \in$ Tp. Then σ and τ are **provably isomorphic** ($\sigma \simeq_p \tau$) iff there exist λ- terms $M : \sigma\to\tau$ and $N : \tau\to\sigma$ such that $\lambda\beta\eta \vdash M\cdot N = I_\tau$ and $\lambda\beta\eta \vdash N\cdot M = I_\sigma$, where I_σ and I_τ are the identities of type σ and τ.

Suppose that types σ and τ are provably isomorphic and consistently substitute the common base types by arbitrary types, then the isomorphism still holds: just use the corresponding terms with updated types. By borrowing from Statman[1983] we may say that the notion of provable isomorphism is *typically ambiguous*. Or, more generally, for the reader acquainted with Hindley[1969], consider the principal type schemes $\alpha\to\beta$ and $\gamma\to\delta$ of M and N in definition 1.2, then for all their instances $\alpha'\to\beta'$ and $\gamma'\to\delta'$ such that $\alpha'=\delta'$ and $\beta'=\gamma'$ one has $\alpha' \simeq_p \beta'$.

We introduce the following axioms and rules for type expressions to deduce provable isomorphisms of types. Let A, B, C be any type expressions:

AXIOMS: (A1) $A = A$,

(A2) $A \to (B \to C) = B \to (A \to C)$,

RULES:

$$(R1)\quad \frac{A = A',\ B = B'}{A \to B = A' \to B'}$$

$$(R2)\quad \frac{A = B}{B = A}$$

$$(R3)\quad \frac{A = B,\ B = C}{A = C}$$

We write $\vdash A = B$ when $A = B$ is derivable in the above theory. Let M be a typed λ- term, write then e(M) for the **erasure** of M, i.e. for M with all type labels on variables erased.

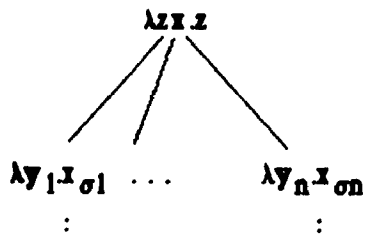
Definition 1.3 Let M be an (untyped) term. Write N for a finite sequence of terms. Then M is a **finite hereditary permutation** (f.h.p.) iff either

(i) $\lambda\beta\eta \vdash M = \lambda x.x$, or

(ii) $\lambda\beta\eta \vdash M = \lambda z.\lambda x.zN_\sigma$, where if $|x| = n$ then σ is a permutation of $(1,\ldots,n)$ and $zN_\sigma = zN_{\sigma 1}N_{\sigma 2}\ldots N_{\sigma n}$, where for $1 \le i \le n$, $\lambda x_i.N_i$ is a finite hereditary

permutation.

Thus $\lambda z.\lambda x_1.\lambda x_2.z x_2 x_1$ and $\lambda z.\lambda x_1.\lambda x_2.z x_2 (\lambda x_3.\lambda x_4.x_1 x_4 x_3)$ are f.h.p.'s. In terms of the Böhm-trees in Barendregt[1984], one may look at f.h.p.'s as follows:



and so on, up to a finite depth. Note that f.h.p.'s possess (are in) $\beta\eta$- normal form and they (their normal forms) are closed terms. In particular, exactly the abstracted variables at level $n+1$ appear at level $n+2$, modulo some permutation of the order (note the special case of z at level 0).

Theorem 1.4 (Dezani-Ciancaglini[1976]) Let M be an untyped term possessing normal form. Then M is $\lambda\beta\eta$- invertible (i.e. there is an N such that $\lambda\beta\eta \vdash M \cdot N = N \cdot M = I$) iff M is an f.h.p.

Recall now that all typed terms possess a (unique) $\lambda\beta\eta$- normal form (see Barendregt[1984]).

Lemma 1.5 Let M and N be terms of the typed λ- calculus:

(i) If $M : \sigma$ then the $\beta\eta$- normal form of M also has type σ.

(ii) If $\lambda\beta\eta \vdash M = N$ then $\lambda\beta\eta \vdash e(M) = e(N)$.

Proof: (i) is a special case of the Subject Reduction Theorem in Curry et al. [1972]. (ii) is immediate. □

Theorem 1.6 If $M : \sigma\to\tau$ and $N : \tau\to\sigma$ are terms of the typed λ-calculus such that $\lambda\beta\eta \vdash M \cdot N = I_\tau$ and $\lambda\beta\eta \vdash N \cdot M = I_\sigma$ then $e(M)$ and $e(N)$ are f.h.p.'s.

Proof: By $e(N \cdot M) = e(N) \cdot e(M)$, 1.5(ii) and 1.4. □

We are now in the position to state our main characterization theorem.

Theorem 1.7 Let σ,τ be type expressions. Then

$$\sigma =_p \tau \text{ iff } \vdash \sigma = \tau.$$

(Proof hint: ($\leftarrow$) $\lambda x.x : \sigma\to\sigma$ proves, in $\lambda\beta\eta$, the isomorphism given by axiom (A1). As for (A2), consider $\lambda z.\lambda x_1.\lambda x_2.z x_2 x_1 : \sigma\to(\tau\to\rho)\to(\tau\to(\sigma\to\rho))$.

An easy induction on rules completes the proof. ($\rightarrow$) goes by induction on the depth of the Böhm-trees of the terms which prove the isomorphism (just observe that f.h.p.'s hereditarily "switch types" as described by axiom (A2)).

A recent alternative proof of ($\rightarrow$) (due to Meyer and Statman [1984]) proceeds by regarding each type as an exponential monomial in the base types ($\sigma \to \tau$ becomes τ^σ). The proof follows from the fact that the given axioms and rules are complete for equality of these exponential monomials over the positive integers and that types may be interpreted in models consisting of the full (finite) type structure built from finite base sets.)

We note here that the alternative proof sketched above relies heavily on the fact that the finite full type structures over finite ground domains are models

of the typed λ-calculus. We know of no way to extend this proof to the second order λ-calculus described in the next section.

The reader may be wondering what happens if we simply take the typed λβ-calculus and dispense with extensionality. Böhm and Dezani-Ciancaglini[1974] showed that the only invertible term of the typed λβ-calculus is the identity. Hence A=A is the only provable isomorphism for this theory.

By definition, provable isomorphisms between types hold in all models of typed λβη -calculus, and thus in all Cartesian Closed Categories (CCC's). Of course other isomorphisms will hold in particular models, especially when a specific interpretation of the base types is given.

Definition 1.8 Let $\mathcal{M}$ be a model of the typed λβη -calculus. Define then

(i) $\mathcal{M} \models \sigma \simeq_d \tau$ (σ,τ are **definably isomorphic** in $\mathcal{M}$) iff there exist closed λ-terms M, N such that $\mathcal{M} \models M \cdot N = I_\sigma$ and $\mathcal{M} \models N \cdot M = I_\tau$.

(ii) $\mathcal{M} \models \sigma \simeq \tau$ (σ,τ are **isomorphic in** $\mathcal{M}$) iff there is an isomorphism in $\mathcal{M}$ (not necessarily definable by closed λ-terms) between the interpretations of σ and τ .

Note: Given $\mathcal{M}$, the notion of definable isomorphism is not typically ambiguous, i.e. it may hold for a specific type but not for all substitution instances of its base types. Similarly for isomorphisms. This is in contrast with the typical ambiguity of provable isomorphisms.

Theorem 1.9 (Completeness) The following are equivalent:

(1) $\vdash \sigma = \tau$,

(2) For all $\mathcal{M}$, $\mathcal{M} \models \sigma \simeq_d \tau$,

(3) For all $\mathcal{M}$, $\mathcal{M} \models \sigma \simeq \tau$.

(Proof hint: Clearly (1) ⇒ (2) ⇒ (3). As for (3) ⇒ (1), consider the typed term model $\mathcal{M}_\mathcal{O}$, i.e. the type structure whose objects or types are the collections of terms, one for each type, modulo provable equalities. Then $\mathcal{M}_\mathcal{O} \models \sigma \simeq \tau$ implies $\sigma \simeq_p \tau$ and, hence, by 1.7, we are done).

Corollary 1.10 Given types σ and τ , it is decidable whether they are isomorphic in all models of λβη .

(Proof hint: The above axioms and rules do not change the length of type expressions).

The "extended completeness" theorem in Friedman[1975] suggests the following completeness property of the CCC of infinite sets.

Theorem 1.11 Let $A = (\alpha_1, \alpha_2, \ldots)$ be a collection of infinite sets and Set_A be the type structure (the subCCC) generated by A . Then one has $\vdash \sigma = \tau$ iff $\mathrm{Set}_A \models \sigma \simeq_d \tau$.

(Proof hint: Friedman's result gives $\lambda\beta\eta \vdash M = N$ iff $\mathrm{Set}_\alpha \models M = N$, for any infinite α. Now, definable isomorphisms are special cases of valid equalities, then use (a generalized version of) Friedman's result and 1.7).

SECTION 2

As mentioned in the introduction, the use of types as parameters in functional laguages has been formalized by various authors. We refer to Bruce & Meyer[1984] for notation and the basic concepts. We only note here that $\Lambda t.M$ represents abstraction over M via a type variable t, where if σ is the type of M then $\Delta t.\sigma$ is the type of $\Lambda t.M$. Similarly $M[\sigma]$ represents applying M to a type expression σ . Because of the extension of the language, our theory of type equality has to be extended accordingly. Thus we write $(II) \vdash \sigma = \tau$ when $\sigma = \tau$ is derivable in the following second order extension of the theory defined in section 1.

AXIOMS: (A1-A2) and

(A3) $\Delta t.\Delta u.A = \Delta u.\Delta t.A$,

(A4) $\Delta t.(A \to B) = A \to \Delta t.B$ if t is not free in A,

(A5) $\Delta t.A = \Delta u.(u/t)A$ where u is free for t in A and u is not free in A.

RULES: (R1-R3) and

$$(R4)\ \frac{A = A'}{\Delta t.A = \Delta t.A'}$$

Provable isomorphisms between second order types ($\sigma \approx_p \tau$) are defined as in 1.2, by using second order terms (note again that we are assuming η throughout).

Definition 2.1 : We define the erasure of second order λ-terms, e : (second order λ-terms $\cup$ type expressions) $\to$ untyped λ-terms, as follows:

For typed expressions: $e(t) = t$, $e(\alpha \to \beta) = K_{\to}e(\alpha)e(\beta)$, $e(\Delta t.\gamma) = K_{\Delta}(\lambda t.e(\gamma))$. ($K_{\to}$ and K_{Δ} are new constant symbols).

For second order λ-terms: $e(x) = x$, $e(MN) = e(M)e(N)$, $e(M[\gamma]) = e(M)e(\gamma)$, $e(\lambda x : \gamma.M) = \lambda x.e(M)$, $e(\Lambda t.M) = \lambda t.e(M)$.

Note: e(M) and M have the same free and bound variables, as do $e(\gamma)$ and γ.

Lemma 2.2: Let P be either a type expression or a second order λ-expression. Then

(i) $e((P/u)\gamma) \equiv (e(P)/u)e(\gamma)$ and

(ii) $e((P/u)M) \equiv (e(P)/u)e(M)$.

Proof: An easy induction on type and second order λ-expressions. □

Definition 2.3: We say that a second order term is a ***second order finite hereditary permutation*** (2-f.h.p.) iff

(i) $\lambda\beta\eta \vdash M = \lambda x : \gamma.x$,

or (ii) $\lambda\beta\eta \vdash M = \lambda z : \gamma.\lambda u.zN_{\sigma}$ where if $|u| = n$ then σ is a permutation of (1,...,n), for $1 \leq i \leq n$, $\lambda u_i = \lambda x_i : \gamma$ or $\lambda u_i = \Lambda t_i$, and $zN_{\sigma} = zN_{\sigma 1}N_{\sigma 2}...N_{\sigma n}$ such that for $1 \leq i \leq n$,

(a) if $\lambda u_i = \lambda x_i : \gamma$ then $\lambda x_i : \gamma.N_i$ is a 2-f.h.p.,

or (b) if $\lambda u_i = \lambda t_i$ then $N_i = t_i$.

Lemma 2.4: Let M be a second order λ-expression in normal form. Then M is a 2-f.h.p. iff e(M) is a f.h.p.

Theorem 2.5: Let σ,τ be second order types. Then

$$\sigma \simeq_p \tau \quad \text{iff} \quad (\Pi) \vdash \sigma = \tau .$$

(Proof hint: The proof goes similarly as in 1.7, by using 2-f.h.p.'s and Dezani-Ciancaglini's theorem).

An analogue of the completeness theorem in 1.9 may also be given for the extended language in this section and its semantics by using 2.5.

SECTION 3

Besides isomorphisms, other interesting relations may hold between types, such as retractions. Given a category $\mathcal{C}$ and objects (or type-interpretations) A and B, we say that A is a **retract** of B ($A \lhd B$) iff there exists morphisms, $in : A \to B$ and $out : B \to A$, such that $out \cdot in = id_A$. Retractions between objects and their morphisms spaces, say, characterize models of type-free λβ in CCC's (see Barendregt[1984] or Longo & Moggi[1984]; also see Meyer[1982]).

In this section we characterize the retractions which hold in all models of typed λβ-calculus , i.e. the provable retractions. Models of the typed λβ-calculus correspond to a weak form of cartesian closure for categories, which may be tidily formalized.

Definition 3.1 Types σ and τ are **provably retracts** ($\sigma \lhd_p \tau$) iff there exist terms $M : \sigma \to \tau$ and $N : \tau \to \sigma$ such that $\lambda\beta \vdash N \cdot M = I_\sigma$.

A simple theory of type retractions is given by the following axioms and rules.

AXIOMS: (A1) $A \lhd A$

(A2) $A \lhd B \to A$

RULES:

$$(R1)\ \frac{A \lhd B \ , \ B \lhd C}{A \lhd C}$$

$$(R2)\ \frac{A \lhd A' \ , \ B \lhd B'}{A \to B \lhd A' \to B'}$$

We write $(\lhd) \vdash A \lhd B$ when $A \lhd B$ is derivable in the theory above.

Theorem 3.2 (Margaria & Zacchi[1983]) Let M be an untyped term. Then M is λβ-right-invertible (i.e. there is an N such that $\lambda\beta \vdash M \cdot N = I$) iff $\lambda\beta \vdash M = \lambda x.x\,P$, for some sequence P of terms.

Proposition 3.3 (1) Let $M : \sigma \to \tau$ be such that there is an $N : \tau \to \sigma$ with $\lambda\beta \vdash M \cdot N = I_\sigma$. Then $\lambda\beta \vdash e(M) = \lambda x.x\,P$, for some P.

(2) Let $\sigma \lhd_p \tau$. Then $\tau = \rho_1 \to (\rho_2 \to \ldots(\rho_k \to \sigma)\ldots)$, for some ρ , with $k = |P|$.

(Proof hint: (1) follows from 3.2 and 1.5(ii). (2) follows from (1) and by checking the type of terms such as $\lambda x.x\,P$.)

An immediate consequence of 3.3 above is a characterization theorem for retractions valid in all models (3.4 below). Completeness results as in section 1 similarly follow.

Theorem 3.4 Let σ and τ be type expressions. Then

$$\sigma \leq_p \tau \quad \text{iff} \quad (\leq) \vdash \sigma \leq \tau .$$

Corollary 3.5 Let σ and τ be type expressions. Then $\sigma \leq_p \tau$ and $\tau \leq_p \sigma \Rightarrow \sigma \simeq_p \tau$.

Proof: Since $(\leq) \vdash \tau \leq \sigma$, only axioms preserving the length of type expressions have been used in $(\leq) \vdash \sigma \leq \tau$. Thus $\vdash \sigma \simeq \tau$. □

Remark 3.6. Note that, in a model, an object (a type) may be a retract of an other and conversely, but they do not need to be isomorphic. Thus the interest of the above corollary. A hint for a counterexample is the following. Consider the category AL of algebraic lattices, with continuous functions as morphisms, and take $P\omega$, the powerset of the natural numbers. Then $AL(P\omega,P\omega)$ is a retract of $P\omega$ (see Scott[1976]). Conversely, $P\omega$ is a retract of $AL(P\omega,P\omega)$, by an obvious argument or by axiom (A2). Observe now that the compact (finite) elements of $P\omega$ have only finitely many smaller elements (subsets); in contrast to this, the compact elements of $AL(P\omega,P\omega)$ always have infinitely many smaller elements. By this, there is no way to find an isomorphisms between these two algebaic lattices.

The characterization of provable retactions in $\lambda\beta\eta$- calculus is an open problem, as it seems an hard task to characterize right or left invertibility in the $\lambda\beta\eta$-calculus

Acknowledgements

The authors would like to express their gratitude to Albert Meyer and John Mitchell for several helpful conversations on these matters.

References

Barendregt, H. [1984], *The lambda calculus; its syntax and semantics* Revised edition, North Holland.

Böhm, C., Dezani-Ciancaglini, M.[1974], "Combinatorial problems, combinator equations and normal forms, in : J. Loeckx, ed., *Automata, Languages and Programming*, LNCS 14 (185-199).

Bruce, K., Meyer, A. [1984], "The semantics of second order polymorphic Lambda Calculus," *Symposium on Semantics of Data Types*,(Kahn, MacQueen, Plotkin, eds.), LNCS 173, Springer-Verlag.

Curry, H. B., Hindley, R., Seldin, J. [1972] *Combinatory Logic II*, North-Holland.

De Bruijn, N. [1980], "A survey of the project AUTOMATH," in Hindley & Seldin[1980].

Dezani-Ciancaglini, M. [1976], "Characterization of normal forms possessing an inverse in the $\lambda\beta\eta$ -calculus," Theor. Comp. Sci., 2 (323-337).

Friedman, H. [1975], "Equality between functionals," *Logic Colloquium* (Parikh ed.), LNM 453, Springer-Verlag.

Girard, J. [1971], "Interpretation fonctionelle et elimination des coupure dans l'arithmetic d'ordre superieur," These de Doctorat d'Etat, Paris.

Hindley, R. [1969], "The principal type-scheme of an object in Combinatory Logic," Trans. AMS, 146 (22-60).

Hindley R., Seldin, J. [1980], *To H.B. Curry: Essays in Combinatory Logic, Lambda calculus and formalism.* Academic Press.

Longo, G., Moggi, E. [1984], "Godel numberings, Principal morphisms, Combinatory Algebras," (revised version), Dip. Informatica, Universita' di Pisa.

Margharia, I., Zacchi, M. [1983], "Right and left invertibility in $\lambda\beta$-calculus," R.A.I.R.O., 17, no.1 (71-88).

McCracken, N.J.[1979], "An investigation of a programming language with a polymorphic type structure," Ph.D. dissertation, Syracuse University.

McCracken, N.J.[1985], "A finitary retract model for the polymorphic lambda-calculus," to appear, Information and Control.

Meyer, A.[1982], "What is a model of the lambda calculus?," Information & Control 52 (87-122)

Meyer, A., Statman, R. [1984], Personal communication.

Reynolds, J. [1974], "Towards a theory of type structures," *Colloque sur la Programmation*, LNCS 19 ,Springer -Verlag .

Reynolds, J. [1984], "Polymorphism is not set-theoretic," *Symposium on Semantics of Data Types*, (Kahn, MacQueen,Plotkin, eds.) LNCS 173, Springer-Verlag.

Scott, D.S. [1976], "Data types as lattices," SIAM J. Comp., 5, no. 3 (522-587).

Statman, R. [1983], "λ-definable functionals and $\beta\eta$-conversion," Arch. Math. Logik, 23 (21-26).