April 1988

Extensional Models for Polymorphism

Val Tannen
University of Pennsylvania, val@cis.upenn.edu

Thierry Coquand
INRIA

Extensional Models for Polymorphism

Abstract
We present a general method for constructing extensional models for the Girard-Reynolds polymorphic lambda calculus - the *polymorphic extensional collapse*. The method yields models that satisfy additional, computationally motivated constraints like having only two polymorphic booleans and having only the numerals as polymorphic integers. Moreover, the method can be used to show that any simply typed lambda model can be fully and faithfully embedded into a model of the polymorphic lambda calculus.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-25.

EXTENSIONAL MODELS FOR POLYMORPHISM

Val Breazu-Tannen
Thierry Coquand

MS-CIS-88-25
LINC LAB 109

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

April 1988

This is a slightly revised version of MS-CIS-87-75/LINC LAB 81

To appear in the special issue of THEORETICAL COMPUTER SCIENCE
dedicated to TAPSOFT'87 Conference

Acknowledgements: This research was supported in part by NSF-CER grant MCS-8219196, U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Extensional Models for Polymorphism

Val Breazu-Tannen [1]

Department of Computer and Information Science
University of Pennsylvania

Thierry Coquand

INRIA
Domaine de Voluceau, 78150 Rocquencourt, France

April 8, 1988

This is a slightly revised version of MS-CIS-87-75/LINC LAB 81

Abstract. We present a general method for constructing extensional models for the Girard-Reynolds polymorphic lambda calculus—the *polymorphic extensional collapse*. The method yields models that satisfy additional, computationally motivated constraints like having only two polymorphic booleans and having only the numerals as polymorphic integers. Moreover, the method can be used to show that any simply typed lambda model can be fully and faithfully embedded into a model of the polymorphic lambda calculus.

To appear in the special issue of THEORETICAL COMPUTER SCIENCE
dedicated to the TAPSOFT'87 conference

[1] Supported in part by U.S. Army Research Office Grant DAAG29–84–K–0061, and, while at MIT, in part by an IBM Graduate Fellowship and in part by NSF Grant DCR–8511190

1 Introduction

The design of functional and object-oriented programming languages has recently witnessed the widespread adoption of polymorphic type systems. A list of examples that is by no means exhaustive includes, in addition to the archetype ML [GMW79], such languages as Miranda [Tur85], Poly [Mat85], Amber [Car85], polymorphic FQL [Nik84], Ponder [Fai82], and Hope [BMS80], while an excellent survey of the field is provided by [CW85].

To study properties of such languages, we will adopt as a formal setting the Girard-Reynolds polymorphic lambda calculus [Gir72, Rey74] (henceforth denoted $\lambda^\forall$).

Our concern here will be with constructing models of $\lambda^\forall$ that satisfy certain special constraints. In fact, the paper is built around the presentation of a general method for constructing such models, which we call the *polymorphic extensional collapse*. The direct motivation for these constructions is that they imply the consistency and/or conservativity of certain extensions of $\lambda^\forall$. At their turn, these extensions are motivated by the study of the interaction between the computational mechanism of the type discipline of $\lambda^\forall$ and the specification of the data types with which we compute, *e.g.,* integers, booleans, *etc.,* We will explain this primary motivation in what follows.

Several researchers [Rey83, Lei83, BB85] have shown how to represent interesting data types inside the pure polymorphic lambda calculus. This unusual programming style is illustrated by Reynolds in [Rey85]. To make our point we will review here the representation of the integers (the reader unfamiliar with the syntax of $\lambda^\forall$ should take a detour through the next section before continuing).

The numerals are taken to be the closed terms of type

$$polyint \stackrel{\text{def}}{=} \forall t.\,(t \to t) \to t \to t\ .$$

The numeral corresponding to the integer n is

$$\tilde{n} \stackrel{\text{def}}{=} \lambda t.\,\lambda f{:}t \to t.\,\lambda x{:}t.\,f^n x\ .$$

One can define, for example,

$$Add \stackrel{\text{def}}{=} \lambda u{:}polyint.\lambda v{:}polyint.\lambda t.\lambda f{:}t \to t.\lambda x{:}t.utf(vtfx) \quad : polyint \to polyint \to polyint$$

and verify that $\lambda^\forall$ proves

(1) $$Add\,\tilde{m}\,\tilde{n} = \widetilde{m+n}\ .$$

The arithmetic functions that are numeralwise representable in the same way addition is represented above are exactly the recursive functions which are provably total in second-order Peano arithmetic [Gir72] (see also [Sta81, FLO83]). To date, no "natural" examples of total recursive functions that are not in this class are known and one can argue that such computational power is adequate for most purposes [Lei83], [Rey85]. Therefore it

appears that $\lambda^\forall$ can be regarded as a programming language already equipped with a type of integers and, as it turns out, also with one of booleans:

$$\mathit{polybool} \stackrel{\text{def}}{=} \forall t.\, t \to t \to t$$

$$\mathit{True} \stackrel{\text{def}}{=} \lambda t.\, \lambda x{:}t.\, \lambda y{:}t.\, x$$

$$\mathit{False} \stackrel{\text{def}}{=} \lambda t.\, \lambda x{:}t.\, \lambda y{:}t.\, y$$

as well as many other familiar data types [Rey85].

Now, if we were to adopt this paradigm, we would expect that the formal reasoning (in the calculus) would allow us to treat *any* terms of type *polyint* as if they actually *were* integers. However, the pure $\lambda^\forall$ is not sufficient for that, as it cannot even prove, for example, that the operation of addition is commutative:

$$\mathit{Add}\, u\, v = \mathit{Add}\, v\, u \tag{2}$$

with arbitrary u, v : *polyint* is not provable in $\lambda^\forall$ (by a simple Church-Rosser argument).

A possible remedy to this shortcoming would be to add to the pure $\lambda^\forall$, as further axioms, equations such as (2). But are such extensions *consistent*?

Notice that when u and v are *numerals* the equation (2) follows from (1). Thus, the consistency of (2) follows from the existence of a non-trivial model in which the *only* elements of type *polyint*, *i.e.,* the only "polymorphic integers", are the (denotations of the) numerals.

Such a consistency question, (actually for an equation involving the conditional operation on *polybool*) was first asked by Meyer [Mey86], and was one of the main sources of motivation for the model constructions presented here. Positive answers to Meyer's question, namely constructions of models with exactly two "polymorphic booleans", were given by Moggi [Mog86a] and the second author [Coq86] (for an account of Meyer's question, see [MMMS87], to which our presentation owes). Both constructions used partial equivalence relations to interpret types, suggesting that there might be some relationship between them and, indeed, a common generalization was found by the first author [Bre86]. All these model constructions are particular instances of the polymorphic extensional collapse method.

We now turn to our other main source of motivation, illustrated in [BM87a, BM87b, Bre87a]. The consistency question we asked above, or that asked in [Mey86], can be seen as a particular form of a more general, albeit more vague, question: is it possible to have data types with "classical" specifications, say, algebraic axiomatizations, live in a computational framework? In [BM87a] it is remarked that unrestricted recursion is not consistent with arbitrary algebraic data type specifications and computation done within the framework of the type discipline of $\lambda^\forall$ is then offered as an alternative. The approach differs from the one we have discussed above, as one does not use the built-in representations of the integers, booleans, *etc.,* but instead one *adds* such data type specifications to $\lambda^\forall$ as algebraic or simply typed lambda theories. The advantage is that we can postulate for these added data types whatever equations we wish, so that problems like the unprovability of the equation (2) do

not arise. The consistency question is therefore replaced by one of *conservative extension*: is the theory of the resulting language ($\lambda^\forall$ plus data type specification) conservative over that of the data type specification? A positive answer would assure us that we can continue to reason about data type expressions "classically", *i.e.,* using the data type specification, even when these expressions occur in the computational framework provided by $\lambda^\forall$.

We use our general method to show that arbitrary simply typed lambda models, and therefore arbitrary algebras, can be fully and faithfully embedded in models of $\lambda^\forall$. The full and faithful embeddings easily imply the desired conservative extension results [BM87a, BM87b, Bre87a].

There are a number of differences between this paper and the preliminary version of this work, presented at the conference [BC87]. We omit here the full and faithful embedding of arbitrary algebras into models of $\lambda^\forall$ in which there is also a one-to-one correspondence between the polymorphic integers and an arbitrary sequence of observables in the algebra. We do so because we feel now that the result can be well motivated only in the context of the conservative extension theorem that called for it, as explained in [BM87a]. On the other hand, we include the full and faithful embedding of models of the simply typed lambda calculus into models of $\lambda^\forall$, which appeared in [BM87b]. Finally, we further generalize here the polymorphic extensional collapse construction to use arbitrary *logical p.e.r. collections*, while the construction in [BC87] was using only the logical p.e.r. collection consisting, at each type, of *all* partial equivalence relations. (*Warning:* the name "polymorphic extensional collapse" is used both here, for the general construction, and in [BC87], for the less general version.) This makes the construction more flexible, allowing us to present a new result (proving a conjecture from [BM87b]): full and faithful embedding of simply typed lambda models with all type domains non-empty into models of $\lambda^\forall$ which also have all type domains non-empty. In this more general form, the polymorphic extensional collapse also covers Mitchell's class of *PER models* [Mit86b].

In Section 2 we review the syntax and semantics of the polymorphic lambda calculus, introducing polymorphic lambda theories and the concept of polymorphic lambda interpretation and revisiting polymorphic logical relations. In Section 3 we show how to construct, out of closed type expressions and closed terms, a polymorphic lambda interpretation for any polymorphic lambda theory. In Section 4 we present the polymorphic extensional collapse construction. In Section 5 we present several applications of the construction: models with exactly two polymorphic booleans and other "minimality properties", "erase-types" models and full and faithful embeddings. Some proofs that would have broken the stride of the presentation, as well as a historical note, have been relegated to appendices.

2 The polymorphic lambda calculus

2.1 Syntax

Let K be a set of *(ground* or *base) type constants.* The *K-polymorphic type expressions* are defined by

$$\sigma ::= k \mid t \mid \sigma \to \sigma \mid \forall t.\, \sigma$$

where k ranges over K and t ranges over an infinite set of *type variables.* The construction $\forall t.\, \sigma$ *binds* the occurrences of the type variable t in σ. The set of free type variables of a type expression τ will be denoted $fv(\tau)$.

Let C be a set of *constants.* By definition, each constant $c \in C$ comes equipped with its type, $Type(c)$ which must be a *closed* K-type expression. We also assume a separate infinite set of *(ordinary) variables.* The (K, C)*-raw terms* are defined by

$$M ::= c \mid x \mid MM \mid \lambda x{:}\sigma.\, M \mid M\sigma \mid \lambda t.\, M$$

where c ranges over C and x ranges over ordinary variables. The set of free variables of M will be denoted $FV(M)$ while the set of free *type* variables of M will be denoted $fv(M)$.

As in [Bar84] we identify the terms or type expressions that differ only in the names of bound variables. We use the notation

$$\sigma[t_1 := \tau_1, \ldots, t_n := \tau_n]$$

$$M[t_1 := \tau_1, \ldots, t_n := \tau_n,\, x_1 := N_1, \ldots, x_m := N_m]$$

for *simultaneous substitution.*

Not all raw terms are acceptable, only those that *type-check.* Type assumptions about the free variables are needed to type-check an open term. Such assumptions are provided by type assignments. A *type assignment* is a partial function with *finite* domain that maps variables to type expressions. Alternatively, we will also regard type assignments as finite sets of pairs $x{:}\sigma$ such that no x occurs twice. We will use Δ to range over type assignments. We write $\Delta, x{:}\sigma$ for $\Delta \cup \{x{:}\sigma\}$ and, by convention, the use of this notation implies that $x \notin dom\Delta$. The *empty* type assignment is usually omitted in formulas.

A *typing judgment* has the form

$$\Delta \vdash M : \sigma\ .$$

The proof system for deriving typing judgments (*i.e.,* type-checking) is the following:

(*constants*) $\qquad \Delta \vdash c : Type(c) \qquad c \in C$

(*projection*) $\qquad \Delta \vdash x : \Delta(x) \qquad x \in dom\Delta$

($\rightarrow$ *elimination*) $$\frac{\Delta \vdash M : \sigma \rightarrow \tau \quad \Delta \vdash N : \sigma}{\Delta \vdash MN : \tau}$$

($\rightarrow$ *introduction*) $$\frac{\Delta, x{:}\sigma \vdash M : \tau}{\Delta \vdash \lambda x{:}\sigma.\, M : \sigma \rightarrow \tau}$$

($\forall$ *elimination*) $$\frac{\Delta \vdash M : \forall t.\, \sigma}{\Delta \vdash M\tau : \sigma[t := \tau]}$$

($\forall$ *introduction*) $$\frac{\Delta \vdash M : \sigma}{\Delta \vdash \lambda t.\, M : \forall t.\, \sigma} \qquad t \notin fv(ran\Delta)$$

Usually, instead of "$\Delta \vdash M : \sigma$ is derivable (in the proof system)" we will write simply "$\Delta \vdash M : \sigma$".

If $\Delta \vdash M : \sigma$, we say that M *type-checks with type* σ *under* Δ. We say that the raw term M *type-checks under* Δ if there exists σ such that $\Delta \vdash M : \sigma$. Note that, when it exists, σ is uniquely determined by M and Δ. Note also that if M type-checks under Δ then it type-checks with the same type under any type assignment that coincides with Δ on the free variables of M. Finally, we say that the raw term M *type-checks* if there exist Δ and σ such that $\Delta \vdash M : \sigma$. Thus, if M is a closed term that type-checks then there exists a unique closed ω such that $\vdash M : \omega$ in which case we say that M *is of type* ω.

Note the *inherent impredicativity* allowed by the type discipline: polymorphic functions can be applied to any type, in particular their own type. For example:

$$\vdash (\lambda t.\, \lambda x{:}t.\, x)(\forall t.\, t \rightarrow t) : (\forall t.\, t \rightarrow t) \rightarrow (\forall t.\, t \rightarrow t)\ .$$

In order to define equational reasoning that is *type-correct* the usual equations $M = N$ will be *tagged* with a type assignment Δ and and a type expression σ with the intention that both M and N should type-check with type σ under Δ. Moreover, it turns out that tagging equations with lists of variables (that include the free variables of the equation) allows us to isolate the proof rules that are sound in models in which some type domains may be *empty* [GM82, MMMS87].

Equation judgments (or, simply, *equations*) have the form

$$\Delta\ ;\ M = N\ ;\ \sigma\ .$$

We will use the following proof system for deriving equations (see however the discussion on completeness and other proof systems in subsection 2.2):

(*extend assign*) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \sigma}{\Delta'\ ;\ M\ =\ N\ ;\ \sigma} \qquad \Delta \subseteq \Delta'$$

(*reflexivity*) $$\Delta\ ;\ M\ =\ M\ ;\ \sigma$$

where $\Delta \vdash M : \sigma$

(*symmetry*) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \sigma}{\Delta\ ;\ N\ =\ M\ ;\ \sigma}$$

(*transitivity*) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \sigma \quad \Delta\ ;\ N\ =\ P\ ;\ \sigma}{\Delta\ ;\ M\ =\ P\ ;\ \sigma}$$

(*congruence*) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \sigma \to \tau \quad \Delta\ ;\ P\ =\ Q\ ;\ \sigma}{\Delta\ ;\ MP\ =\ NQ\ ;\ \tau}$$

(ξ) $$\frac{\Delta, x{:}\sigma\ ;\ M\ =\ N\ ;\ \tau}{\Delta\ ;\ \lambda x{:}\sigma.\ M\ =\ \lambda x{:}\sigma.\ N\ ;\ \sigma \to \tau}$$

(β) $$\Delta\ ;\ (\lambda x{:}\sigma.\ M)N\ =\ M[x := N]\ ;\ \tau$$

where $\Delta, x{:}\sigma \vdash M : \tau$, $\Delta \vdash N : \sigma$

(η) $$\Delta\ ;\ \lambda x{:}\sigma.\ Mx\ =\ M\ ;\ \sigma \to \tau$$

where $\Delta \vdash M : \sigma \to \tau$, $x \notin dom\Delta$

(*type congruence*) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \forall t.\ \sigma}{\Delta\ ;\ M\tau\ =\ N\tau\ ;\ \sigma[t := \tau]}$$

(*type* ξ) $$\frac{\Delta\ ;\ M\ =\ N\ ;\ \sigma}{\Delta\ ;\ \lambda t.\ M\ =\ \lambda t.\ N\ ;\ \forall t.\ \sigma} \qquad t \notin fv(ran\Delta)$$

(*type* β) $$\Delta\ ;\ (\lambda t.\ M)\tau\ =\ M[t := \tau]\ ;\ \sigma[t := \tau]$$

where $\Delta \vdash M : \sigma$, $t \notin fv(ran\Delta)$

(*type* η) $\quad \Delta \; ; \; \lambda t.\, Mt \;=\; M \; ; \; \forall t.\, \sigma$

where $\Delta \vdash M : \forall t.\, \sigma$, $t \notin fv(ran\Delta)$, $t \notin fv(M)$

If E is a set of equations and e is a single equation, we write $E \vdash^{\lambda^\forall} e$ when e is derivable in the above proof system using additional premises from E. An equation $\Delta \; ; \; M \;=\; N \; ; \; \sigma$ is *type-correct* if $\Delta \vdash M : \sigma$ and $\Delta \vdash N : \sigma$. It is easy to see that if $E \vdash^{\lambda^\forall} e$ and all the equations in E are type-correct then e is type-correct.

The rule

(*substitutivity*) $\quad \dfrac{\Delta, x{:}\tau \; ; \; M \;=\; N \; ; \; \sigma}{\Delta' \; ; \; M[x := P] \;=\; N[x := P] \; ; \; \sigma}$

where $\Delta' \vdash P : \tau$, $\Delta \subseteq \Delta'$

while not included in the proof system, can be safely used since it is a *derived* rule. Indeed, we have

Lemma 2.1 (substitutivity)
If $\Delta, x{:}\tau \vdash M : \sigma$, $\Delta, x{:}\tau \vdash N : \sigma$, $\Delta' \vdash P : \tau$ *and* $\Delta \subseteq \Delta'$ *then*

$$(\Delta, x{:}\tau \; ; \; M \;=\; N \; ; \; \sigma) \vdash^{\lambda^\forall} (\Delta' \; ; \; M[x := P] \;=\; N[x := P] \; ; \; \sigma) \; .$$

Equational reasoning can be analyzed with a *reduction system.* Our terminology and notation follows [Bar84]. Given a notion of reduction R, we will use $\xrightarrow[R]{}\!\!\!\!\!\rightarrow$ for the multi-step R-reduction relation.

We define $\lambda^\forall$*-reduction* on raw terms as the union of the four basic notions of reduction: β, η, *type* β and *type* η, obtained by orienting from left to right the axiom schemes with the same name. It is easy to show that if $\Delta \vdash M : \sigma$ and $M \xrightarrow[\lambda^\forall]{}\!\!\!\!\!\rightarrow N$ then $\Delta \vdash N : \sigma$. Thus, type-checking is preserved under reduction, which justifies defining reduction on raw terms, in an "untyped" manner.

Theorem 2.2 (Girard)
$\lambda^\forall$ *is Church-Rosser on terms that type-check.*

In fact, η and *type* η are not considered in [Gir72]. However, one can proceed as follows. First prove that β is Church-Rosser using the method of Martin-Löf and Tait (see [Bar84], pp. 59–62). Then, since each of *type* β, *type* η and η is also, by itself, Church-Rosser (trivially), show that each two of the four notions of reduction commute and invoke the Hindley-Rosen Lemma (see [Bar84], pp. 64–66). The restriction to terms that type-check is used only for the commutativity of β and η.

Reduction gives the following alternative characterization of derivable equations:

Proposition 2.3
$\vdash^{\lambda^\forall} \Delta\ ;\ M = N\ ;\ \sigma$ *if and only if* $\Delta \vdash M : \sigma$, $\Delta \vdash N : \sigma$ *and there exists a term* P *such that* $M \xrightarrow{\lambda^\forall}\!\!\!\!\!\rightarrow\ P \xleftarrow{\lambda^\forall}\!\!\!\!\!\leftarrow N$.

The most important technical property enjoyed by the polymorphic type discipline is the following:

Theorem 2.4 (Girard)
$\lambda^\forall$ *is strongly normalizing on terms that type-check.*

Again, η and *type* η are not considered in [Gir72], but, for example, the strong normalization proof in [Mit86b] can be immediately extended to $\lambda^\forall$-reduction. Together with the Church-Rosser property, this result implies that any term M that type-checks has a unique $\lambda^\forall$-normal form, which we will denote by $nf(M)$.

A *polymorphic lambda theory* is completely specified by the following

- a *polymorphic signature* $\Sigma = (K, C)$, where K is a set of *(ground* or *base) type constants* and C is a set of *constants*, each with a closed K-polymorphic type;
- a set E of type-correct equations between Σ-terms to be used as *additional axioms*;

We will symbolize by $\lambda^\forall(\Sigma, E)$ the corresponding polymorphic lambda theory.

2.2 Interpretations and models

Fix a polymorphic signature $\Sigma = (K, C)$.

A *K-algebra of polymorphic types*, $\mathcal{T}$, consists of the following:

- a non-empty set T of *types*;
- a binary operation $\rightarrow$ on T;
- a non-empty set $[T \Rightarrow T]$ of functions from T to T;
- a map $\forall$ from $[T \Rightarrow T]$ to T;
- an interpretation $\mathcal{T}(k) \in T$ for each type constant $k \in K$;

such that the following inductive definition of an assignment of meanings in T to K-type expressions in type environments is possible (we define a *type environment* to be a map from type variables to T and we will use η to range over type environments):

1. $[\![k]\!]\eta = \mathcal{T}(k)$
2. $[\![t]\!]\eta = \eta(t)$
3. $[\![\sigma \to \tau]\!]\eta = [\![\sigma]\!]\eta \to [\![\tau]\!]\eta$
4. $[\![\forall t.\, \sigma]\!]\eta = \forall(\lambda a \in T.\, [\![\sigma]\!]\eta\{t := a\})$

By "the definition is possible" we understand that each inductive application of step 4 is defined, *i.e.*, $\lambda a \in T.\, [\![\sigma]\!]\eta\{t := a\} \in [T \Rightarrow T]$. Here $\eta\{t := a\}$ is the type environment equal to η everywhere except at t where it takes the value a.

Lemma 2.5 *1. If for each* $t \in fv(\sigma)$ *we have* $\eta(t) = \eta'(t)$ *, then* $[\![\sigma]\!]\eta = [\![\sigma]\!]\eta'$.

2. $[\![\sigma[t := \tau]]\!]\eta = [\![\sigma]\!]\eta\{t := [\![\tau]\!]\eta\}$.

A Σ-*polymorphic lambda interpretation* (p.l.i.), $\mathcal{I}$, consists of the following:

- a K-algebra of polymorphic types, $\mathcal{T}$;
- a set D_a for each type $a \in T$ (the *domain* of a);
- a binary operation $\cdot_{ab} : D_{a \to b} \times D_a \longrightarrow D_b$ for each pair of types $a, b \in T$ (functional application);
- a binary operation $\cdot_\phi : D_{\forall(\phi)} \times T \longrightarrow \bigcup\{D_a\}$ for each function $\phi \in [T \Rightarrow T]$, such that $p \cdot_\phi a \in D_{\phi(a)}$ (polymorphic application);
- an interpretation $\mathcal{I}(c) \in \bigcup\{D_a\}$ for each constant $c \in C$ such that $\mathcal{I}(c) \in D_{[\![Type(c)]\!]}$;

Given a type assignment Δ and a type environment η, we define a $\Delta\eta$-*environment* to be a function ρ that maps $dom\Delta$ to $\bigcup\{D_a\}$ such that $\rho(x) \in D_{[\![\Delta(x)]\!]\eta}$ for each variable $x \in dom\Delta$. As with type assignments, we will regard $\Delta\eta$-environments as finite sets of pairs $x{:}d$, extending to them the notational convention $\rho, x{:}d$. With this, the final component of the p.l.i. is

- a *meaning map* that assigns to every typing judgment $\Delta \vdash M : \sigma$ that is *derivable* and every type environment η a function $[\![\Delta \vdash M : \sigma]\!]\eta$ from $\Delta\eta$-environments to $D_{[\![\sigma]\!]\eta}$ such that (ρ ranges over $\Delta\eta$-environments)

  1. $[\![\Delta \vdash c : Type(c)]\!]\eta\rho = \mathcal{I}(c)$

  2. $[\![\Delta \vdash x : \Delta(x)]\!]\eta\rho = \rho(x)$ where $x \in dom\Delta$

3. $[\![\Delta \vdash MN : \tau]\!]\eta\rho = [\![\Delta \vdash M : \sigma \to \tau]\!]\eta\rho \cdot_{ab} [\![\Delta \vdash N : \sigma]\!]\eta\rho$

   where $a \stackrel{\text{def}}{=} [\![\sigma]\!]\eta$ and $b \stackrel{\text{def}}{=} [\![\tau]\!]\eta$

4. $[\![\Delta \vdash \lambda x{:}\sigma.\, M : \sigma \to \tau]\!]\eta\rho \cdot_{ab} d = [\![\Delta, x{:}\sigma \vdash M : \tau]\!]\eta\rho'$

   where $a \stackrel{\text{def}}{=} [\![\sigma]\!]\eta$, $b \stackrel{\text{def}}{=} [\![\tau]\!]\eta$, $d \in D_a$ and $\rho' \stackrel{\text{def}}{=} \rho, x{:}d$

5. $[\![\Delta \vdash M\tau : \sigma[t{:=}\tau]]\!]\eta\rho = [\![\Delta \vdash M : \forall t.\, \sigma]\!]\eta\rho \cdot_{\phi} [\![\tau]\!]\eta$

   where $\phi \stackrel{\text{def}}{=} \lambda a \in T.\, [\![\sigma]\!]\eta\{t{:=}a\}$

6. $[\![\Delta \vdash \lambda t.\, M : \forall t.\, \sigma]\!]\eta\rho \cdot_{\phi} a = [\![\Delta \vdash M : \sigma]\!]\eta\{t{:=}a\}\,\rho$

   where $a \in T$ and $\phi \stackrel{\text{def}}{=} \lambda a \in T.\, [\![\sigma]\!]\eta\{t{:=}a\}$

7. if for each $x \in FV(M)$ we have $\rho(x) = \rho'(x)$
   then $[\![\Delta \vdash M : \sigma]\!]\eta\rho = [\![\Delta \vdash M : \sigma]\!]\eta\rho'$.
8. if for each $t \in fv(ran\Delta) \cup fv(M)$ we have $\eta(t) = \eta'(t)$
   then $[\![\Delta \vdash M : \sigma]\!]\eta = [\![\Delta \vdash M : \sigma]\!]\eta'$

As opposed to the previous definition of meaning for type expressions, these clauses are *conditions* to be satisfied by an *a priori* given meaning map and do not constitute an inductive definition. An alternative definition of *models* in which meaning is defined inductively is possible [BM84].

Note that the definition allows *empty* type domains. If for some $x \in dom\Delta$, $D_{[\![\Delta(x)]\!]\eta}$ is empty then the set of $\Delta\eta$-environments is empty and there is only one choice for $[\![\Delta \vdash M : \sigma]\!]\eta$—the empty function. This case still fits in our definition since the clauses above then hold *vacuously.*

A type-correct equation $\Delta\ ;\ M = N\ ;\ \sigma$ is *valid* in $\mathcal{I}$, write

$$\mathcal{I} \models \Delta\ ;\ M = N\ ;\ \sigma\ ,$$

iff $[\![\Delta \vdash M : \sigma]\!]\eta = [\![\Delta \vdash N : \sigma]\!]\eta$ for every type environment η. By convention, all the equations we will talk about in the context of validity will be implicitly assumed type-correct.

A p.l.i. is a quite general concept. For example, not even basic axioms like β are necessarily valid in arbitrary p.l.i.'s. However, most of the p.l.i.'s we will consider are instances of the closed type/closed term construction of section 3 and β as well as all the equations provable in $\lambda^{\forall}(\Sigma, \emptyset)$ are valid in any such p.l.i. We chose not to include these validities among the conditions satisfied by p.l.i's because they are not needed for the construction

in section 4 to produce a model. In fact, Proposition 2.6 and the polymorphic extensional collapse construction do not even need the meaning map to satisfy conditions 7 and 8. The absence of these conditions, however, would burden the treatment of validities in p.l.i.'s.

A Σ-*polymorphic lambda model* is a Σ-p.l.i. in which functional and polymorphic application are *extensional*:

$$(\forall d \in D_a, f \cdot_{ab} d = g \cdot_{ab} d) \Longrightarrow f = g \quad f, g \in D_{a \to b}$$

$$(\forall a \in T, p \cdot_\phi a = q \cdot_\phi a) \Longrightarrow p = q \quad p, q \in D_{\forall(\phi)} \ .$$

This definition of model for $\lambda^\forall$ is equivalent and, in fact, very close to the one in [BM84]. (In the presence of extensionality, conditions 7 and 8 are consequences of the other conditions satisfied by the meaning map.)

A model is *trivial* when all its domains have at most one element. It is easy to see that a model is trivial if and only if it equates *True* and *False* (*i.e.,* $; True = False ;$ *polybool* is valid).

It is easy to see that the axioms and rules of the proof system are *sound* for this notion of model. As was explained in [MMMS87], *completeness* is more complicated. One is, of course, interested in the strong kind of completeness, *i.e.,* completeness of reasoning from additional premises. In [BM84] such a result is stated, but it amounts to completeness of the proof system we gave here (call it "*core*"), extended with the rule:

(*discharge var*) $$\frac{\Delta, x{:}\sigma \ ; \ M = N \ ; \ \tau}{\Delta \ ; \ M = N \ ; \ \tau} \qquad x \notin FV(M) \cup FV(N)$$

for the subclass of models with all type domains non-empty. (The rule *discharge var* is *not sound*, in general, in models that can have empty type domains.)

In [MMMS87], Meyer *et al.,* argue that models with empty types come naturally into consideration. For example, the models that satisfy constraints like having exactly two polymorphic booleans or having exactly the numerals as polymorphic integers *must* have empty types. Furthermore, it is stated that the "*core*" proof system is not complete neither for deriving semantic consequences over the class of all models (and thus) nor for deriving semantic consequences over the class of all models with all types non-empty. However, as an aside, we note that "*core*" is complete for the hyperdoctrine models of Seely [See86b, See86a] and the topos models of Pitts [Pit87], which do not seem to fit into the framework of [BM84], and that Mitchell and Moggi [MM87] have shown that the proof system for the simply typed lambda calculus that corresponds to "*core*" is complete for their simply typed Kripke-style models. Meyer *et al.,* give an extension of the "*core*" proof system that is sound and complete for all models [MMMS87]. This extension involves modifying the syntax of the equations to allow "type emptiness" assertions to be added to the type assignments as well as new axioms and inference rules.

However, as far as the *model constructions* described in the present paper are concerned, we note that it does not matter which of the three proof systems we use to construct our

closed type/closed term interpretations. Indeed, by Church-Rosser arguments, *discharge var* is a derived rule in the pure $\lambda^\forall$ theory or in other theories axiomatized by additional equations that can be analyzed, together with $\lambda^\forall$, by Church-Rosser reduction. Thus, the closed type/closed term constructions of subsections 5.1 and 5.3 are the same as the ones that would be obtained using the extended proof system of [MMMS87] or the proof system with *discharge var*.

2.3 Logical relations

Second-order logical relations were introduced in [MM85]. Here we will review only a particular case of this concept, the case that we need for the polymorphic extensional collapse.

Fix a polymorphic signature, $\Sigma = (K, C)$, and a Σ-p.l.i., $\mathcal{I}$.

A *logical relation* on $\mathcal{I}$ is a family $\mathcal{R} = \{R_a\}_{a \in T}$ of binary relations on the type domains, $R_a \subseteq D_a \times D_a$ such that

$$f\ R_{a \to b}\ g \text{ iff } \forall d \forall e\ \ d\ R_a\ e \implies f \cdot_{ab} d\ \ R_b\ \ g \cdot_{ab} e$$

and

$$p\ R_{\forall(\phi)}\ q \text{ iff } \forall a\ \ p \cdot_\phi a\ \ R_{\phi(a)}\ \ q \cdot_\phi a\ .$$

We say that $\mathcal{R}$ *relates the constants in* C if

$$\forall c \in C,\quad \mathcal{I}(c)\ R_{[\![Type(c)]\!]}\ \mathcal{I}(c)\ .$$

Proposition 2.6 (Fundamental property of logical relations)
If $\mathcal{R}$ is a logical relation on $\mathcal{I}$ which relates the constants in C then, for any derivable typing judgment $\Delta \vdash M : \sigma$, any type environment η and any two $\Delta\eta$-environments ρ_1 and ρ_2, if

$$\forall x \in dom\Delta,\quad \rho_1(x)\ R_{[\![\Delta(x)]\!]\eta}\ \rho_2(x)$$

then

$$[\![\Delta \vdash M : \sigma]\!]\eta\rho_1\ R_{[\![\sigma]\!]\eta}\ [\![\Delta \vdash M : \sigma]\!]\eta\rho_2\ .$$

The proof is by a routine induction on $\Delta \vdash M : \sigma$.

We will make essential use of this property in the polymorphic extensional collapse and, in fact, the definition of polymorphic lambda interpretations was engineered to consist of the "minimum necessary" to make the proof of Proposition 2.6 work (almost; see the remark about conditions 7 and 8 in the previous subsection).

3 The closed type/closed term construction

For any polymorphic theory, it is possible to construct a p.l.i. out of closed type expressions and classes of closed terms equated in the theory. This p.l.i. is closely related to the theory and, as we will see in the applications, the additional axioms of the theory can be used to determine the content of its type domains. In this section we present the construction in general.

Fix a polymorphic theory, $\lambda^\forall(\Sigma, E)$, and construct a p.l.i., $\mathcal{I}$, as follows:

We start with the observation that the closed type expressions form an algebra of polymorphic types. Indeed, if we take T to be the set of closed K-polymorphic type expressions, we can take

$$[T \Rightarrow T] \stackrel{\text{def}}{=} \{\lambda\omega \in T.\, \sigma[t := \omega] \mid \forall t.\, \sigma \text{ closed}\} .$$

The rest of the definition of the algebra of types of $\mathcal{I}$ is straightforward. The meaning map satisfies

$$[\![\sigma]\!]\eta = \sigma[t_1 := \eta(t_1), \ldots, t_k := \eta(t_k)]$$

where $\{t_1, \ldots, t_k\} \stackrel{\text{def}}{=} fv(\sigma)$.

Then, for each closed type expression ω we define a relation G_ω on the set of closed Σ-terms of type ω:

$$P\, G_\omega\, Q \quad \text{iff} \quad E \vdash^{\lambda^\forall} ;\ P = Q\ ;\ \omega\ .$$

G_ω is an equivalence relation and even a congruence w.r.t. functional and polymorphic application. Therefore, we take the domain of ω in $\mathcal{I}$ to consist of the congruence classes of closed terms of type ω, modulo G_ω, and we define application via representatives, as usual. We denote by $G_\omega[P]$ the congruence class of P modulo G_ω. Take $\mathcal{I}(c) \stackrel{\text{def}}{=} G_{Type(c)}[c]$.

The meaning map is defined via substitution:

$$[\![x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \vdash M : \sigma]\!]^{\mathcal{I}}\eta\rho \stackrel{\text{def}}{=} G_{[\![\sigma]\!]\eta}[Q]$$

where

$$Q \stackrel{\text{def}}{=} M[t_1 := \eta(t_1), \ldots, t_m := \eta(t_m), x_1 := P_1, \ldots, x_n := P_n]$$

where $\{t_1, \ldots, t_m\} \stackrel{\text{def}}{=} fv(M) \cup fv(\sigma_1) \cup \cdots \cup fv(\sigma_n)$ and

$$\rho(x_1) = G_{[\![\sigma_1]\!]\eta}[P_1], \ldots, \rho(x_n) = G_{[\![\sigma_n]\!]\eta}[P_n] .$$

By Lemma 2.1, the choice of the P_i's does not matter so meaning is well-defined. It is easy to check that $\mathcal{I}$ is a Σ-p.l.i. and that all the equations in E are valid in $\mathcal{I}$. In fact, more is true:

Lemma 3.1
If $E \vdash^{\lambda^\forall} e$ *then* $\mathcal{I} \models e$.

The converse is not true, in general. Counterexample: $K = \{k\}$, $C = \{c : k\ ,\ f, g : k \rightarrow k\}$, $E = \{\ ;\ f\,c\ =\ g\,c\ ;\ k\}$ and $e = (x{:}k\ ;\ f\,x\ =\ g\,x\ ;\ k)$.

We will call $\mathcal{I}$ the *closed type/closed term* polymorphic lambda interpretation of the given theory.

In general, closed type/closed term p.l.i.'s are not extensional (see subsection 5.1 for a counterexample) and, therefore, are not models. The following section will describe a general method for achieving extensionality.

4 Polymorphic extensional collapse

4.1 Preliminaries

The extensional collapse construction starts from an arbitrary p.l.i. and produces a model, thus achieving extensionality of functional and polymorphic application. In doing so, the construction makes heavy use of *partial equivalence relations* (p.e.r.'s), *i.e.,* relations that are symmetric and transitive but not necessarily reflexive.

If D is a set, let $per(D)$ denote the set of partial equivalence relations on D. Note that the empty relation is also a p.e.r. Let $R \in per(D)$. We denote by $R[d]$ the p.e.r. class of d mod R. Note that $R[d] \neq \emptyset$ iff $d\ R\ d$. The *quotient set* D/R is the set of all non-empty p.e.r. classes mod R. Clearly R is empty iff D/R is empty. Another way of looking at p.e.r.'s is as pairs consisting of a subset of D and an equivalence relation on that subset. For this reason, D/R is sometimes called a *subquotient.*

For the remainder of the section, fix a polymorphic signature, $\Sigma = (K, C)$.

4.2 Factoring by a logical partial equivalence relation

Suppose that we have a Σ-p.l.i., $\mathcal{I}$, and a logical relation $\mathcal{R}$ on it, which is such that *each* R_a *is a partial equivalence relation.* We will call such an $\mathcal{R}$ a *logical p.e.r.* Suppose, moreover, that $\mathcal{R}$ relates the constants in C. We construct a new Σ-p.l.i., denoted $\mathcal{I}/\mathcal{R}$ and called the *quotient* of $\mathcal{I}$ by $\mathcal{R}$ as follows:

The algebra of types will be the same. As domains we take the quotient sets D_a/R_a. Since $\mathcal{R}$ is logical, it is also a congruence w.r.t. functional and polymorphic application therefore we can define functional and polymorphic application on the quotient sets straightforwardly, via representatives. Take

$$\mathcal{I}/\mathcal{R}(c) \stackrel{\text{def}}{=} R_{[\![Type(c)]\!]}[\mathcal{I}(c)]$$

(which is non-empty because $\mathcal{R}$ relates the constants).

To show how to define the meaning function let us fix a derivable typing statement $\Delta \vdash M : \sigma$ and a type environment η. For any $\mathcal{I}/\mathcal{R}$-$\Delta\eta$-environment $\hat{\rho}$, choose an $\mathcal{I}$-$\Delta\eta$-environment ρ such that

$$\forall x \in dom\Delta,\ \ \hat{\rho}(x)\ =\ R_{[\![\Delta(x)]\!]\eta}[\rho(x)]$$

and then define

$$[\![\Delta \vdash M : \sigma]\!]^{\mathcal{I}/\mathcal{R}}\eta\hat{\rho} \stackrel{\text{def}}{=} R_{[\![\sigma]\!]\eta}[[\![\Delta \vdash M : \sigma]\!]^{\mathcal{I}}\eta\rho] .$$

Indeed, by Proposition 2.6 the definition does not depend on the choice of ρ and also $[\![\Delta \vdash M : \sigma]\!]^{\mathcal{I}}\eta\rho$ is related to itself hence its p.e.r. class is nonempty. It is easy to check that the meaning map defined this way satisfies the required conditions, thus $\mathcal{I}/\mathcal{R}$ is a p.l.i.

Lemma 4.1
$\mathcal{I}/\mathcal{R}$ is actually a model, that is, functional and polymorphic application are extensional.

Indeed, for polymorphic application extensionality is immediate while for functional application it can be seen that

$$\text{if } f\ R_{a\to b}\ f \text{ and } g\ R_{a\to b}\ g \text{ and } (\forall d\ \ f\cdot_{ab} d\ \ R_b\ \ g\cdot_{ab} d) \text{ then } f\ R_{a\to b}\ g\ .$$

Moreover, from the way meaning is defined, it follows that any equation valid in $\mathcal{I}$ is also valid in $\mathcal{I}/\mathcal{R}$. The converse is in general not true since, for example, there can be pairs of closed terms whose meanings in $\mathcal{I}$ are distinct but *related* by $\mathcal{R}$ and therefore whose meanings in $\mathcal{I}/\mathcal{R}$ are the same p.e.r. class.

4.3 Tagging the types with partial equivalence relations

Suppose that we have a p.l.i. and we would like to construct a model out of it, by factoring through a logical p.e.r., as explained in the previous subsection. How do we obtain such a logical p.e.r.?

The difficulty comes from the inherent impredicativity of $\lambda^{\forall}$. This prevents the condition

$$p\ R_{\forall(\phi)}\ q \text{ iff } \forall a\ \ p\cdot_\phi a\ \ R_{\phi(a)}\ \ q\cdot_\phi a\ .$$

from being used as an inductive definition clause since it is possible that $\forall(\phi) = \phi(a)$ (*e.g.*, take ϕ to be the identity map and a to be $\forall(\phi)$). Thus, constructing a logical p.e.r. is like solving a system of equations. It is therefore natural to enlarge the space in which we search for solutions, in the hope of increasing the chances of finding one. Instead of logical p.e.r.'s, which have one p.e.r. for each type, we will search for collections of *several* p.e.r.'s at each type, satisfying "logicality" conditions. In extremis, taking *all* p.e.r.'s at each type will always work, but smaller collections will also be useful. Then, such a collection will give an actual logical p.e.r., not on the original p.l.i., but on a modified one in which each original type is split in as many copies as there are p.e.r.'s on it in the collection.

Fix a Σ-p.l.i., $\mathcal{I}$ (with algebra of types $\mathcal{T}$). A *logical p.e.r. collection*, $\mathcal{P}$, on $\mathcal{I}$ has two components. The first component is a family $\{P_a\}_{a\in T}$, of non-empty *sets* of p.e.r.'s, $\emptyset \neq P_a \subseteq per(D_a)$, satisfying the following closure conditions:

- for any $a, b \in T$, any $R \in P_a$ and any $S \in P_b$, the binary relation $R \to S$ on $D_{a \to b}$ defined by

$$f \; R \to S \; g \text{ iff } \forall d \forall e \;\; d \; R \; e \implies f \cdot_{ab} d \; S \; g \cdot_{ab} e$$

(which is easily shown to be a p.e.r. too) is actually in $P_{a \to b}$;

- for any $\phi \in [T \Rightarrow T]$ and any family, $\mathcal{H} = \{H_a\}_{a \in T}$, of maps $H_a : P_a \longrightarrow P_{\phi(a)}$, the binary relation $\forall(\mathcal{H})$ on $D_{\forall(\phi)}$ defined by

$$p \; \forall(\mathcal{H}) \; q \text{ iff } \forall a, \forall R \;\; p \cdot_\phi a \; H_a(R) \; q \cdot_\phi a \; .$$

(which is, trivially, also a p.e.r.) is actually in $P_{\forall(\phi)}$;

The second component is a p.e.r.-interpretation $\mathcal{P}(k) \in P_{\mathcal{T}(k)}$ for each type constant $k \in K$.

Any logical p.e.r. $\mathcal{R}$ trivially determines a logical p.e.r. collection, $\mathcal{P}$, namely take $P_a \stackrel{\text{def}}{=} \{R_a\}$ and $\mathcal{P}(k) \stackrel{\text{def}}{=} R_{\mathcal{T}(k)}$. At the other extreme, the collection that consists of *all* p.e.r.'s at each type is also logical (since $R \to S$ and $\forall(\mathcal{H})$ are always p.e.r.'s). See the proof of Theorem 5.7 for an intermediate example.

Given such a logical p.e.r. collection, $\mathcal{P}$, on $\mathcal{I}$, we construct a new Σ-p.l.i., $\mathcal{I}(\mathcal{P})$. First, from the algebra of types $\mathcal{T}$, construct $\mathcal{T}(\mathcal{P})$ as follows:

- the new set of types is $T(\mathcal{P}) \stackrel{\text{def}}{=} \{<a, R> \mid a \in T \, , \, R \in P_a\}$;
- $<a, R> \to <b, S> \stackrel{\text{def}}{=} <a \to b, R \to S>$;
- $[T(\mathcal{P}) \Rightarrow T(\mathcal{P})]$ consists of all the functions determined (one-to-one) by pairs $<\phi, \mathcal{H}>$ where $\phi \in [T \Rightarrow T]$ and $\mathcal{H} = \{H_a\}_{a \in T}$ is a family of maps $H_a : P_a \longrightarrow P_{\phi(a)}$;
- $\forall(<\phi, \mathcal{H}>) \stackrel{\text{def}}{=} <\forall(\phi), \forall(\mathcal{H})>$;
- $\mathcal{T}(\mathcal{P})(k) \stackrel{\text{def}}{=} <\mathcal{T}(k), \mathcal{P}(k)>$.

Lemma 4.2
$\mathcal{T}(\mathcal{P})$ is an algebra of polymorphic types. Moreover

$$proj_1([\![\sigma]\!]^{\mathcal{T}(\mathcal{P})}\eta) = [\![\sigma]\!]^{\mathcal{T}}(proj_1 \circ \eta) \; .$$

The proof shows by simultaneous induction on σ that the inductive definition of $[\![\sigma]\!]^{\mathcal{T}(\mathcal{P})}$ is possible and that, when defined that way, $[\![\sigma]\!]^{\mathcal{T}(\mathcal{P})}$ satisfies, for all η, the property stated in the lemma.

The rest of the definition of $\mathcal{I}(\mathcal{P})$ is by "taking the first projection of the types":

- $D_{<a,R>} \stackrel{\text{def}}{=} D_a$;

- $f \cdot_{<a,R><b,S>} d \stackrel{\text{def}}{=} f \cdot_{ab} d$;
- $p \cdot_{<\phi,\mathcal{H}>} a \stackrel{\text{def}}{=} p \cdot_{\phi} a$;
- $\mathcal{I}(\mathcal{P})(c) \stackrel{\text{def}}{=} \mathcal{I}(c)$;
- $[\![\Delta \vdash M : \sigma]\!]^{\mathcal{I}(\mathcal{P})}\eta \stackrel{\text{def}}{=} [\![\Delta \vdash M : \sigma]\!]^{\mathcal{I}}(proj_1 \circ \eta)$.

Clearly, any equation valid in $\mathcal{I}$ is valid in $\mathcal{I}(\mathcal{P})$. The converse is also true since any T-type environment is the first projection of some $T(\mathcal{P})$-type environment.

The benefit of all this is that now we have a family $\mathcal{R}(\mathcal{P}) = \{R(\mathcal{P})_{<a,S>}\}_{<a,S>\in T(\mathcal{P})}$ of p.e.r.'s, one at each new type, namely $R(\mathcal{P})_{<a,S>} \stackrel{\text{def}}{=} S$ such that

Lemma 4.3
$\mathcal{R}(\mathcal{P})$ is a logical p.e.r. on $\mathcal{I}(\mathcal{P})$.

As usual with logical relations, it does not follow in general that $\mathcal{R}(\mathcal{P})$ relates the constants in C. This has to be checked for each particular construction.

4.4 Properties of the construction

Suppose we have a Σ-p.l.i., $\mathcal{I}$, and a logical p.e.r. collection, $\mathcal{P}$, on it such that the resulting logical p.e.r. $\mathcal{R}(\mathcal{P})$ relates the constants in C. Then, by the lemmas in the previous subsections, $\mathcal{I}(\mathcal{P})/\mathcal{R}(\mathcal{P})$ is a Σ-model. We will call this model the *polymorphic extensional collapse of $\mathcal{I}$ and $\mathcal{P}$* and denote it by $coll(\mathcal{I},\mathcal{P})$.

By the observations in the previous subsections, any equation valid in $\mathcal{I}$ is also valid in $coll(\mathcal{I},\mathcal{P})$, while the converse fails, in general. On the other hand, it is important in applications to ensure that $coll(\mathcal{I},\mathcal{P})$ is *non-trivial*. The following result gives useful sufficient conditions for this.

Proposition 4.4
Assume that $\mathcal{I} \models \{\beta\,,\ type\,\beta\}$ and that $\mathcal{P}$ contains, at each type, the identity p.e.r. Then, if $\mathcal{I}$ does not equate (the meanings of) True and False, $coll(\mathcal{I},\mathcal{P})$ does not equate them either.

Proof. Let $tt \stackrel{\text{def}}{=} [\![True]\!]^{\mathcal{I}}$ and $ff \stackrel{\text{def}}{=} [\![False]\!]^{\mathcal{I}}$. Suppose $coll(\mathcal{I},\mathcal{P})$ equates *True* and *False*. Then,

$$tt\ R(\mathcal{P})_{[\![polybool]\!]^{\mathcal{I}(\mathcal{P})}}\ ff\ .$$

A straightforward computation gives

$$[\![polybool]\!]^{\mathcal{I}(\mathcal{P})} = < \forall(\lambda a \in T.\, a \to a \to a)\,,\ \forall(\{\lambda R \in P_a.\, R \to R \to R\}_{a\in T}) > \ .$$

Thus, for any $a \in T$ and any $R \in P_a$ we have

$$(tt\ a)\ \ R \to R \to R\ \ (ff\ a)\ .$$

Let $bb \stackrel{\text{def}}{=} [\![polybool]\!]^{\mathcal{T}}$. Take $a = bb$ and $R = ID$, the identity on D_{bb}. Since $tt\ ID\ tt$ and $ff\ ID\ ff$ we obtain

$$(tt\ bb\ tt\ ff)\ ID\ (ff\ bb\ tt\ ff)\ .$$

Using the fact that $\mathcal{I} \models \{\beta\ ,\ type\ \beta\}$ we obtain $tt\ bb\ tt\ ff\ =\ tt$ and $ff\ bb\ tt\ ff\ =\ ff$, hence $\mathcal{I}$ equates *True* and *False* too. **End of Proof.**

If $\mathcal{I}$ is a model to begin with, then, because of extensionality, the relation $\mathcal{ID}$ consisting of the identity on each type domain is actually logical and thus a logical p.e.r. on $\mathcal{I}$. Moreover, as we have noted before, $\mathcal{ID}$ trivially determines a logical p.e.r. collection on $\mathcal{I}$ (call it $\mathcal{ID}$ too) such that $\mathcal{R}(\mathcal{ID})$ turns out to be the identity logical p.e.r. on $\mathcal{I}(\mathcal{ID})$ (and thus to relate the constants). Clearly, we have the isomorphisms $coll(\mathcal{I},\mathcal{ID}) \simeq \mathcal{I}(\mathcal{ID}) \simeq \mathcal{I}$. Therefore, *the class of models obtained by polymorphic extensional collapse is as general as the class of all models.*

Finally, for any p.l.i., $\mathcal{I}$, and any p.e.r.-interpretation, π, for the type constants (*i.e.*, π maps each type constant k to a p.e.r. $\pi(k) \in per(D_{\mathcal{T}(k)})$ consider the logical p.e.r. collection $\mathcal{ALL}_\pi$ consisting, at each type, of all p.e.r.'s (*i.e.*, $(ALL_\pi)_a \stackrel{\text{def}}{=} per(D_a)$), and such that $\mathcal{ALL}_\pi(k) \stackrel{\text{def}}{=} \pi(k)$. Then, $coll(\mathcal{I},\mathcal{ALL}_\pi)$ is the construction that we called "polymorphic extensional collapse" in [BC87]. Since the empty relation is a p.e.r., we have immediately

Proposition 4.5
The domain of the (meaning of the) type $\forall t.\, t$ *in* $coll(\mathcal{I},\mathcal{ALL}_\pi)$ *is empty.*

Consequently, the models obtained as $coll(\mathcal{I},\mathcal{ALL}_\pi)$ from arbitrary p.l.i.'s, $\mathcal{I}$, are not arbitrary at all. For example, Proposition 4.5 implies that the non-trivial equation

$$;\ True\ (\forall t.\, t)\ =\ False\ (\forall t.\, t)\ ;\ (\forall t.\, t) \to (\forall t.\, t) \to (\forall t.\, t)$$

is valid in all such models. It is an open question to characterize the theory of this class of models.

5 Applications

5.1 A minimal(?) model

We will present here the model, call it $\mathcal{C}$, with exactly two polymorphic booleans that was given in [Coq86] (see the discussion in the introduction). It turns out that the model has also exactly the numerals as polymorphic integers as well as an entire class of such "minimality" properties.

Consider $\mathcal{I}_0$, the closed type/closed term p.l.i. of the pure polymorphic lambda calculus (the polymorphic theory with $K = C = E = \emptyset$). In $\mathcal{I}_0$, we can think of the domain of each closed type ω as consisting of all closed normal forms of type ω. In particular, the interpretation does not equate *True* and *False*. Moreover, by Lemma 3.1, all the equations that are provable in the pure $\lambda^\forall$ are valid in $\mathcal{I}_0$. However, this p.l.i. is not yet a model because extensionality fails. For example, there are two *distinct* elements of type $(\forall t.\, t) \to (\forall t.\, t) \to (\forall t.\, t)$ namely (the equivalence classes of) $\lambda x{:}\forall t.\, t.\, \lambda y{:}\forall t.\, t.\, x$ and $\lambda x{:}\forall t.\, t.\, \lambda y{:}\forall t.\, t.\, y$ while by extensionality there should be at most one since there are no elements of type $\forall t.\, t$.

Hence the idea of [Coq86] of combining the closed type/closed term construction with factoring by p.e.r.'s and thus achieving extensionality. This amounts to constructing the model

$$\mathcal{C} \stackrel{\text{def}}{=} coll(\mathcal{I}_0, \mathcal{ALL}_\emptyset) \,.$$

(since $K = \emptyset$, the p.e.r.-interpretation can only be the empty map). By Proposition 4.4, $\mathcal{C}$ is non-trivial.

The model $\mathcal{C}$ has a very interesting property: by construction, *all the elements of $\mathcal{C}$ are denotable by pure closed terms.* Therefore, since $\mathcal{C}$ is non-trivial, it must have exactly two polymorphic booleans. The idea used in the proof of Proposition 4.4 can be adapted (see [BC87]) to show that $\mathcal{C}$ does not equate distinct numerals either, thus has only the numerals as polymorphic integers. However, it has been pointed out to us by Meyer that this, and more, should already follow from the non-triviality of $\mathcal{C}$ and a general result of Statman [Sta83]. Notice that *polybool* and *polyint* are quite special types. More precisely, let us define an *ML-polymorphic* type to be a closed type expression of the form $\forall t_1 \ldots \forall t_n.\, \sigma$ where σ is $\forall$-free. (These types correspond to the limited kind of polymorphism allowed in the language ML [GMW79].) As expected, the "combinatorics" of pure terms of ML-polymorphic type is essentially that of simply typed terms. The following is a result about simply typed lambda terms from [Sta83], rephrased for ML-polymorphic terms.

Theorem 5.1 (Statman's Typical Ambiguity Theorem)
Let ω be an ML-polymorphic type and M, N two pure closed terms of type ω. If

$$\not\vdash^{\lambda^\forall} \; ; \; M = N \; ; \; \omega$$

then there exists a pure closed term $L : \omega \to polybool$ such that

$$\vdash^{\lambda^\forall} \; ; \; L\,M = \mathit{True} \; ; \; \mathit{polybool} \quad \text{and} \quad \vdash^{\lambda^\forall} \; ; \; L\,N = \mathit{False} \; ; \; \mathit{polybool} \,.$$

Therefore, while a lot of additional equations are expected to hold in $\mathcal{C}$, no such closed equations will hold at ML-polymorphic types. To summarize, we introduce the following definition. Let ω be a closed type. A model of $\lambda^\forall$ is said to be *canonical* at ω if the meaning map establishes a bijection between the closed normal forms of type ω and the elements of the domain of ω.

Proposition 5.2
The model $\mathcal{C}$ is canonical at all ML-polymorphic types.

This proves, and strengthens, the first half of Conjecture 1.3 of [MMMS87], which that for each ML-polymorphic type ω, there exists a non-trivial model (with empty types) that is canonical at ω. In [Bre87b] it is shown that the second half of this conjecture, which states that a non-trivial model that has all types non-empty cannot be canonical at *any* ML-polymorphic type, also holds.

A very interesting open problem is to characterize the theory of $\mathcal{C}$. A possibility, suggested by Statman [Mog86b], is that the theory of $\mathcal{C}$ is the *maximal consistent polymorphic theory* whose existence was proved by Moggi [Mog86b]. This would completely justify calling $\mathcal{C}$ a *minimal* model.

5.2 "Erase-types" models

These models are obtained by applying the polymorphic extensional collapse to "erase-types" p.l.i.'s in which the terms are interpreted by first erasing the type information and then interpreting the resulting untyped terms in, say, some combinatory algebra (via the usual translation into combinatory terms; see [Bar84]).

Fix a polymorphic signature $\Sigma = (K, C)$. We consider untyped lambda terms built from the same variables used for the polymorphic lambda terms and from constants in C. Again, we identify terms that differ only in the name of the bound variables.

A *C-untyped lambda interpretation* (called *pseudo-λ-structure* in [HL80]), $\mathcal{U}$, consists of the following:

- a non-empty set D;
- a binary operation $\cdot$ on D (application);
- an interpretation $\mathcal{U}(c) \in D$ for each constant $c \in C$;
- a "meaning" map that assigns to every untyped lambda term M and every D-environment π an element $[\![M]\!]\pi$ of D (we define a *D-environment* to be a map from variables to D and we will use π to range over D-environments) such that:

  1. $[\![c]\!]\pi = \mathcal{U}(c)$
  2. $[\![x]\!]\pi = \pi(x)$
  3. $[\![MN]\!]\pi = [\![M]\!]\pi \cdot [\![N]\!]\pi$

4. $[\![\lambda x.\, M]\!]\pi \cdot d = [\![M]\!]\pi\{x := d\}$ where $d \in D$

5. if $\forall x \in FV(M),\ \pi_1(x) = \pi_2(x)$ then $[\![M]\!]\pi_1 = [\![M]\!]\pi_2$

If application is also extensional we get the usual concept of *model of the untyped $\lambda\beta\eta$-calculus* [HL80, Mey82, Bar84].

Any combinatory algebra yields an untyped lambda interpretation. Namely, the meaning map $[\![M]\!]\pi$ is defined by first translating M into a combinatory term [Bar84] and then interpreting the result in the algebra.

Now, starting from an arbitrary K-algebra of polymorphic types $\mathcal{T}$ and an arbitrary C-untyped lambda interpretation $\mathcal{U}$, we construct an *erase-types* Σ-p.l.i., $erase(\mathcal{T},\mathcal{U})$, as follows.

The algebra of types is, of course, $\mathcal{T}$. The domain of each type is a copy of D, the domain of $\mathcal{U}$. Functional application is given by the application in $\mathcal{U}$. Polymorphic application simply erases the type:

$$p \cdot_\phi a \stackrel{\text{def}}{=} p\ .$$

Finally, the meaning map is defined by

$$[\![\Delta \vdash M : \sigma]\!]^{erase(\mathcal{T},\mathcal{U})}\eta\rho \stackrel{\text{def}}{=} [\![Erase(M)]\!]^{\mathcal{U}}\pi$$

where π is some D-environment that takes the same values as ρ on $FV(M)$ (by (4) above, only these values matter) and where $Erase(\lambda x{:}\sigma.\, M) = \lambda x.\, Erase(M)$, $Erase(M\sigma) = Erase(M)$, $Erase(\lambda t.\, M) = Erase(M)$, *etc.*, .

An *erase-types model* is a model obtained by polymorphic extensional collapse from an erase-types p.l.i., *i.e.*, has the form $coll(erase(\mathcal{T},\mathcal{U}),\mathcal{P})$.

Moggi's ingenious construction [Mog86a] having exactly two elements of type *polybool* amounts to the erase-types model

$$\mathcal{M} \stackrel{\text{def}}{=} coll(erase(\mathbf{1}, \mathcal{T}erm), \mathcal{ALL}_\emptyset)$$

where $\mathbf{1}$ is the algebra of polymorphic types consisting of just one type (the rest of the definition is forced) and $\mathcal{T}erm$ is the $\beta\eta$ open term model of the untyped lambda calculus (see [Bar84] p. 96). In what follows we generalize the argument about $\mathcal{M}$ that Moggi gave for *polybool* to a certain class of ML-polymorphic types. Define the *rank* of an ML-polymorphic type by induction on the $\forall$-free part: $rank(t) = 0$ for any type variable t and $rank(\sigma \to \tau) = \max\{rank(\sigma) + 1, rank(\tau)\}$. Thus, $\forall t.\, t$ has rank 0, *polybool* has rank 1 and *polyint* has rank 2.

Proposition 5.3 (Moggi)
The model $\mathcal{M}$ is canonical at all ML-polymorphic types of rank ≤ 2.

Proof. We will give the proof for a complicated enough type of rank 2, say

$$\omega \;=\; \forall r.\, \forall s.\, r \to (s \to r) \to r \to (r \to s) \to s \to r \;.$$

From this, it should be clear to the reader how to proceed in general. The closed normal forms of type ω are

$$\lambda r.\, \lambda s.\, \lambda x{:}\, r.\, \lambda g{:}\, s \to r.\, \lambda y{:}\, r.\, \lambda f{:}\, r \to s.\, \lambda z{:}\, s.\, X$$

where X is one of the terms in

$$\begin{aligned}\Phi \stackrel{\text{def}}{=}\; & \{x, g(fx), g(f(g(fx))), \ldots\} \cup \\ & \cup \{y, g(fy), g(f(g(fy))), \ldots\} \cup \\ & \cup \{gz, g(f(gz)), g(f(g(f(gz)))), \ldots\} \;.\end{aligned}$$

Define also the set

$$\begin{aligned}\Psi \stackrel{\text{def}}{=}\; & \{z, f(gz), f(g(f(gz))), \ldots\} \cup \\ & \cup \{fy, f(g(fy)), f(g(f(g(fy)))), \ldots\} \cup \\ & \cup \{fx, f(g(fx)), f(g(f(g(fx)))), \ldots\} \;.\end{aligned}$$

Let $[M]$ denote the equivalence class of the untyped term M mod $\beta\eta$-conversion. Let $\mathcal{A} \stackrel{\text{def}}{=} \mathbf{1}(\mathcal{ALL}_\emptyset)$, *i.e.,* the logical p.e.r. constructed on $\mathit{erase}(\mathbf{1}, \mathcal{T}\mathit{erm})(\mathcal{ALL}_\emptyset)$. We claim that, for any untyped terms M, N, if

$$[M]\; A_{[\omega]}\; [N] \tag{3}$$

then there exists a closed normal form of type ω, Q, such that both M and N $\beta\eta$-convert to $\mathit{Erase}(Q)$. It is easy to see that the claim implies that $\mathcal{M}$ is canonical at ω.

To prove the claim, note that (3) implies that for any R, S, p.e.r.'s on the set of equivalence classes of untyped lambda terms

$$[M]\; (R \to (S \to R) \to R \to (R \to S) \to S \to R)\; [N] \;.$$

Let x, y, z, f, g be fresh variables. Take

$$R \stackrel{\text{def}}{=} \{< [X], [X] > \,|\, X \in \Phi\} \quad \text{and} \quad S \stackrel{\text{def}}{=} \{< [Z], [Z] > \,|\, Z \in \Psi\} \;,$$

with the notations defined above. Clearly, $[x]R[x]$, $[y]R[y]$ and $[z]S[z]$, and, it is easy to see that $[f](R \to S)[f]$ and $[g](S \to R)[g]$. It follows that $[Mxgyfz]R[Nxgyfz]$ therefore there exists an $X \in \Phi$ such that both $Mxgyfz$ and $Nxgyfz$ $\beta\eta$-convert to it, thus both M and N convert to $\lambda xgyfz.\, X$. **End of Proof.**

We don't know what happens at ML-polymorphic types of rank 3 or more.

Note that if $\mathcal{U}$ is actually an untyped lambda model (untyped application is already extensional) then $\mathit{erase}(\mathcal{T},\mathcal{U})$ is *already* a model: functional application is extensional because

the untyped application is while polymorphic application is always (trivially) extensional! However, such a model is much too coarse: all elements have all types! The point of continuing with a polymorphic extensional collapse—as suggested by Moggi's idea—seems therefore to be the "pruning" of the model, while preserving extensionality.

Finally, we show that the class of erase-types models is (up to isomorphism) the same as Mitchell's class of PER models [Mit86b]. Such models are determined by three parameters: an algebra of polymorphic types $\mathcal{T}$, an untyped lambda interpretation $\mathcal{U}$ (actually, Mitchell requires $\mathcal{U}$ to be an untyped lambda model but the construction goes through for interpretations) and (what amounts to) a logical p.e.r. that relates the constants, $\mathcal{R}$, on $erase(\mathcal{T},\mathcal{U})$. The model is then defined as the quotient $erase(\mathcal{T},\mathcal{U})/\mathcal{R}$.

Given a PER model determined by $\mathcal{T}$, $\mathcal{U}$ and $\mathcal{R}$, as we have remarked before, $\mathcal{R}$ trivially determines a logical p.e.r. collection, call it $\mathcal{R}$ too, and $erase(\mathcal{T},\mathcal{U})/\mathcal{R}$ is isomorphic to the erase-types model $coll(erase(\mathcal{T},\mathcal{U}),\mathcal{R})$. Conversely, any erase-types model, $coll(erase(\mathcal{T},\mathcal{U}),\mathcal{P})$, is isomorphic to the PER model determined by $\mathcal{T}(\mathcal{P})$, $\mathcal{U}$ and $\mathcal{R}(\mathcal{P})$.

5.3 Full and faithful embedding of simply typed models

We begin by reviewing the simply typed lambda calculus, seen as a fragment of $\lambda^\forall$. Let K be a set of (base) type constants. The *K-simple type expressions* are built out of base types from K and the $\rightarrow$ constructor, thus are always closed. Let C be a set of constants of simple type. The *(K,C)-simple raw terms* are defined by:

$$M ::= c \mid x \mid MN \mid \lambda x{:}\sigma.\, M \ .$$

From what we described for the polymorphic calculus, it is straightforward to define simple type-checking and simply typed equational reasoning. *Simply typed models* are like in [Fri75]. The definition, as well as that of *simply typed lambda interpretations*, can also be extracted from our definitions of polymorphic interpretations and models, by ignoring the algebra of polymorphic types, taking the meanings of simple types to be the types themselves, ignoring type environments and polymorphic application and keeping only the relevant meaning clauses.

Fix a simply typed signature $\Sigma = (K,C)$, *i.e.,* a polymorphic signature in which the types of the constants are simple. Let $\mathcal{E}$ be a *Σ-polymorphic* model. There is an obvious way to extract a *Σ-simply typed* model, $s(\mathcal{E})$, out of $\mathcal{E}$ by selecting only the domains (of the meanings) of the simple types. Given a Σ-simply typed model $\mathcal{D}$ and a polymorphic model $\mathcal{E}$ we say that $\mathcal{D}$ is *fully and faithfully embedded* in $\mathcal{E}$ if $\mathcal{D}$ is isomorphic to $s(\mathcal{E})$. (Here, an *isomorphism* is a bijection that preserves types and meaning; hence, in particular, it will also preserve application and the interpretation of constants.) The crucial property of full and faithful embeddings, used in proofs of conservative extension, is that for any simple equation e, $\mathcal{D} \models e$ iff $\mathcal{E} \models e$.

Theorem 5.4 ([BM87b])
Any simply typed lambda model can be fully and faithfully embedded in a polymorphic lambda model.

Proof. Let $\mathcal{D}$ be a (K,C)-simply typed model. For each element d of $\mathcal{D}$ we introduce a new constant q_d of the same type. Let $C' \stackrel{\text{def}}{=} C \cup \{q_d \mid d \in \mathcal{D}\}$ and $\Sigma' \stackrel{\text{def}}{=} (K,C')$. $\mathcal{D}$ is also a Σ'-model, in an obvious manner. For each simple type σ and for each closed Σ'-simply typed term M of type σ that is *not* a q_d, we introduce an equation

$$; \ M \ = \ q_{\lceil M \rceil} \ ; \ \sigma \ .$$

We will denote by H the set formed by these equations.

The first step is to construct the closed type/closed term Σ'-p.l.i., $\mathcal{I}$, of the theory $\lambda^{\forall}(\Sigma', H)$. We claim that, as a Σ'- simply typed interpretation, $\mathcal{D}$ is fully and faithfully embedded in $\mathcal{I}$ via the map that takes $d \in D_\sigma$ to the congruence class $G_\sigma[q_d]$. In particular, in $\mathcal{I}$, the restriction of functional application to simple types is *extensional* (because application in $\mathcal{D}$ is extensional).

Indeed, note that for any closed Σ'-polymorphic term M of simple type, $nf(M)$ is simple (see Lemma A.2 in Appendix A) hence M is provably equal in $\lambda^{\forall}(\Sigma', H)$ to $q_{\lceil nf(M) \rceil}$. This proves the fullness of the embedding. For faithfulness, we show

Lemma 5.5
If P and Q are closed polymorphic terms of simple type σ and $; \ P = Q \ ; \ \sigma$ is provable in $\lambda^{\forall}(\Sigma', H)$, then $[\![nf(P)]\!]^{\mathcal{D}} = [\![nf(Q)]\!]^{\mathcal{D}}$.

Indeed, this implies that if $H \vdash^{\lambda^{\forall}} q_d = q_{d'}$ then $d \equiv d'$.

Proof of Lemma. In view of the well-known equivalence between provability using an equational proof system like the one we defined, and provability by chains of replacements of equals by equals (see, for example, [BM87b]), we can prove the result by induction on the length of such chains of replacements.

The induction is trivial except for the base case of a replacement using one of the axioms in H. Suppose Q is obtained from P by replacing an occurrence of a closed and simply typed subterm M of P by $q_{\lceil M \rceil}$. Let τ be the type of M and z a fresh variable of type τ. Then, there exists a polymorphic term Z with $FV(Z) = \{z\}$ and with *exactly one* occurrence of z, such that $P \equiv Z[z := M]$ and $Q \equiv Z[z := q_{\lceil M \rceil}]$. We have

$$\vdash^{\lambda^{\forall}} \ ; \ P \ = \ (\lambda z{:}\tau.\ nf(Z))M \ ; \ \sigma \quad \text{and} \quad \vdash^{\lambda^{\forall}} \ ; \ Q \ = \ (\lambda z{:}\tau.\ nf(Z))q_{\lceil M \rceil} \ ; \ \sigma \ .$$

But, by Lemma A.2, $nf(Z)$ is a simply typed term too, so we calculate

$$\begin{aligned} [\![nf(P)]\!] = [\![nf((\lambda z{:}\tau.\ nf(Z))M)]\!] = \\ [\![(\lambda z{:}\tau.\ nf(Z))M]\!] = [\![\lambda z{:}\tau.\ nf(Z)]\!][\![M]\!] = \\ [\![\lambda z{:}\tau.\ nf(Z)]\!][\![q_{\lceil M \rceil}]\!] = \cdots = [\![nf(Q)]\!] \end{aligned}$$

where all the meanings are in $\mathcal{D}$ and where we use the fact that $\lambda^{\forall}$-reductions from a simply typed term are in fact simply typed reductions, hence sound in $\mathcal{D}$. **End of Proof.**

While it is true that $\mathcal{D}$ is fully and faithfully embedded in $\mathcal{I}$, $\mathcal{I}$ is not extensional and hence not a model. We will therefore continue with a polymorphic extensional collapse, and show that the full and faithful embedding "survives" it. More precisely, we construct $\mathcal{E} \stackrel{\text{def}}{=} coll(\mathcal{I}, \mathcal{ALL}_\pi)$ where, for each k, $\pi(k)$ is the identity p.e.r. on the domain of (the meaning of) k in $\mathcal{I}$.

Recall that $\mathcal{I}(\mathcal{ALL}_\pi)$ is the notation for the p.l.i. obtained by tagging the types of $\mathcal{I}$ with p.e.r.'s from $\mathcal{ALL}_\pi$, that $\mathcal{T}(\mathcal{ALL}_\pi)$ is the notation for the algebra of polymorphic types of $\mathcal{I}(\mathcal{ALL}_\pi)$, that $\mathcal{R}(\mathcal{ALL}_\pi)$ is the notation for the resulting logical p.e.r. on $\mathcal{I}(\mathcal{ALL}_\pi)$, and, finally, that $coll(\mathcal{I}, \mathcal{ALL}_\pi) \stackrel{\text{def}}{=} \mathcal{I}(\mathcal{ALL}_\pi)/\mathcal{R}(\mathcal{ALL}_\pi)$.

Lemma 5.6
For any simple type σ, the component of $\mathcal{R}(\mathcal{ALL}_\pi)$ which corresponds to the meaning of σ in $\mathcal{T}(\mathcal{ALL}_\pi)$ is also the identity p.e.r.

This follows by induction on simple types using the fact that functional application in $\mathcal{I}$ is extensional at simple types.

One consequence of the lemma is that $\mathcal{R}(\mathcal{ALL}_\pi)$ relates the constants in C' since they are all of simple type. Thus, $\mathcal{E}$ is a Σ'-polymorphic model. Another consequence is that the extensional collapse leaves the domains of simple types unchanged, which was exactly the part of $\mathcal{I}$ onto which $\mathcal{D}$ was fully and faithfully embedded. Thus, $\mathcal{D}$ is also fully and faithfully embedded in $\mathcal{E}$. The desired Σ-model is obtained by "forgetting" about the interpretation of the q_d's. **End of Proof.**

As we mentioned in the introduction, this result serves to prove that simply typed theories are *conservatively* extended by polymorphic constructs and axioms [Bre87a, BM87b]. Of course, a conservative extension result depends on the proof sytems that are considered. As explained in [MMMS87] (see also our discussion in subsection 2.2), for both $\lambda^\forall$ and the simply typed lambda calculus, there are at least two proof systems of interest: one that is complete for deriving semantic consequences over all models, call it "*all*", and one that is complete for deriving semantic consequences over all models with all types non-empty, call it "*non-empty*". By Proposition 4.5, the previous full and faithful embedding construction always produces polymorphic models that have some empty types, even if the embedded simple model had all types non-empty. Therefore, Theorem 5.4, will imply only conservative extension for the "*all*" proof systems. For "*non-empty*", we need our next result (there is also a purely syntactic proof [BM87b]).

Theorem 5.7
Any simply typed lambda model with all type domains non-empty can be fully and faithfully embedded in a polymorphic lambda model with all type domains non-empty.

Proof. The proof uses and extends both ideas from the previous embedding and an idea of Mitchell and Moggi on how to fully and faithfully embed arbitrary non-empty one-sorted algebras in erase-types models with all types non-empty [Mit86a]. Remember that a polymorphic model has all types non-empty iff the domain of $[\![\forall t.\, t]\!]$ is non-empty.

Let $\mathcal{D}$ be a (K,C)-simply typed model with all types non-empty. For each base type $k \in K$, choose an element $d_k^0 \in D_k$. For each element d of $\mathcal{D}$ we introduce a new constant q_d of the same type. Moreover, we introduce another new constant ε of type $\forall t.\, t$. Let $C' \stackrel{\text{def}}{=} C \cup \{q_d \mid d \in \mathcal{D}\}$ and $\Sigma' \stackrel{\text{def}}{=} (K,C')$. Let $C'' \stackrel{\text{def}}{=} C' \cup \{\varepsilon\}$ and $\Sigma'' \stackrel{\text{def}}{=} (K,C'')$. For each *simple* type σ and for each closed Σ'-simply typed term M of type σ that is *not* a q_d, we introduce an equation

$$; \; M \; = \; q_{\lceil M \rceil} \; ; \; \sigma \; .$$

We will denote by H the set formed by these equations. We also introduce the equations

$$; \; \varepsilon\,(\sigma \to \tau) \; = \; \lambda x{:}\sigma.\, \varepsilon\,\tau \; ; \; \sigma \to \tau$$

where σ, τ range over closed *polymorphic* type expressions,

$$; \; \varepsilon\,(\forall t.\, \sigma) \; = \; \lambda t.\, \varepsilon\,\sigma \; ; \; \forall t.\, \sigma$$

where σ ranges over *polymorphic* type expressions such that $\forall t.\, \sigma$ is closed and

$$; \; \varepsilon\, k \; = \; q_{d_k^0} \; ; \; k$$

where k ranges over K. We will denote by E_ε the set formed by these three kinds of equations.

We construct again the closed type/closed term Σ''-p.l.i., $\mathcal{I}$, of the theory $\lambda^{\forall}(\Sigma'', E_\varepsilon \cup H)$ and, again, we claim that, as a Σ'- simply typed interpretation, $\mathcal{D}$ is fully and faithfully embedded in $\mathcal{I}$ via the map that takes $d \in D_\sigma$ to the congruence class $G_\sigma[q_d]$. To see this, consider the reduction system $\lambda^{\forall}\varepsilon$ consisting of the usual reductions of $\lambda^{\forall}$ plus the notion of reduction ε defined as the union of all the equations of E_ε oriented from left to right. It is easy to show that $\lambda^{\forall}\varepsilon$ is Church-Rosser on terms that type-check (use the Hindley-Rosen Lemma; see our comments to the proof of Theorem 2.2). Moreover, in Lemma A.1 (Appendix A), we show that any polymorphic term of simple type and with free variables of simple type has an $\lambda^{\forall}\varepsilon$-normal form which is actually a simply typed term. With this, the fullness and faithfulness of the embedding are shown just like in the proof of the previous theorem.

By abuse of notation, let us denote the meaning of the constant ε in $\mathcal{I}$ also by ε . Let $\mathcal{T}$ be the algebra of polymorphic types of $\mathcal{I}$ and $\{I_a\}_{a \in T}$ be the type domains of $\mathcal{I}$. For any $a \in T$ there is an element $\varepsilon\, a \in I_a$ (we omit the dot for polymorphic application) and we have

$$(\varepsilon\,(a \to b)) \cdot_{ab} i = \varepsilon\, b \quad \text{where } a, b \in T \; , \; i \in I_a$$

$$(\varepsilon\, \forall(\phi)) \cdot_\phi a = \varepsilon\, \phi(a); \quad \text{where } \phi \in [T \Rightarrow T] \; , \; a \in T \; .$$

This will insure that the p.e.r. collection $\mathcal{P}$, on $\mathcal{I}$, defined by

$$P_a \stackrel{\text{def}}{=} \{R | R \in per(I_a) \text{ and } (\varepsilon\, a)\; R\; (\varepsilon\, a)\}$$

and by taking $\mathcal{P}(k)$ to be the identity p.e.r. on the domain of (the meaning of) k, is *logical.* Thus, we can construct the model $\mathcal{E} \stackrel{\text{def}}{=} coll(\mathcal{I}, \mathcal{P})$. It is easy to see that

$$\varepsilon \; \mathcal{R}(\mathcal{P})_{[\forall t.\, t]^{\mathcal{T}(\mathcal{P})}} \; \varepsilon$$

thus ε will survive the polymorphic extensional collapse. It follows that $\mathcal{E}$ has all types non-empty. The rest of the proof, which checks that $\mathcal{D}$ is also fully and faithfully embedded in $\mathcal{E}$ is just like in the proof of the previous theorem. **End of Proof.**

5.4 Full and faithful embedding of algebras

It is well-known that any many-sorted algebra can be fully and faithfully embedded into a simply typed lambda model. Indeed, the simply typed model will have as base type domains the carriers of the algebra and, at higher types, the domain of $\sigma \to \tau$ will consist of *all* functions from the domain of σ to the domain of τ. Moreover, if the carriers of the algebra are all non-empty then the type domains of the simply typed model are all non-empty too. From this and Theorems 5.4 and 5.7 we deduce:

Theorem 5.8
Any many-sorted algebra can be fully and faithfully embedded in a polymorphic lambda model. Any many-sorted algebra with all sort carriers non-empty can be fully and faithfully embedded in a polymorphic lambda model with all type domains non-empty.

This will imply some of the desired conservative extension results about adding $\lambda^{\forall}$ to arbitrary algebraic theories [BM87a, Bre87a].

It can be argued that that the full and faithful embedding of algebras into simply typed models described above, while conceptually very simple, is too "lavish", that is, the resulting simply typed model has much "more" elements than what is needed for the embedding. Indeed, a better construction uses the simply typed version of the extensional collapse (see [Tro73] or [Bar84], p. 565). It is also possible to give a direct and "lean" constructions for the proof of Theorem 5.8. The constructions are similar to those of Theorem 5.4 Theorem 5.7 and the first one is sketched in [BC87]. Most likely, this direct constructions produces models that are isomorphic to the ones produced by the simply typed extensional collapse embeddings followed by the embeddings of Theorem 5.4 and Theorem 5.7.

In [Mit86b] it is stated that PER models can be used to obtain faithful but not full embeddings of algebras into models of the polymorphic lambda calculus. Next, we obtained full and faithful embeddings of many-sorted algebras into models of $\lambda^{\forall}$, but the resulting models had always empty types (work described in [BC87]). Subsequently, Mitchell and Moggi independently discovered how to do faithful *and full* embeddings of many-sorted algebras into PER models that have empty types and full and faithful embeddings of non-empty one-sorted algebras into PER models with all types non-empty [Mit86a]. Theorem 5.8 finally provides a completely satisfactory answer to the question.

Acknowledgments

We are grateful to John Mitchell for reading an earlier version of the conference paper on which this paper is based, and for suggesting some corrections and improvements. Any remaining errors are, of course, our responsibility. A portion of this work grew out of the first author's PhD thesis, done under the supervision of Albert Meyer at the Massachusetts Institute of Technology. This paper has directly benefited from Albert Meyer's suggestions.

Appendix A

Refer to the notation of the proof of Theorem 5.7.

Lemma A.1 *Any Σ''-polymorphic term of simple type and with free variables of simple type only has a $\lambda^\forall\varepsilon$-normal form, which, moreover is a simply typed term (in particular, ε-free).*

Proof. It is sufficient to prove the lemma for $\lambda^\forall$-normal forms. We proceed by induction on the length of the n.f.

$\lambda^\forall$-normal forms of length 1, of simple type and with free variable of simple type must be variables or constants of simple type (thus distinct from ε) and therefore are already $\lambda^\forall\varepsilon$-normal forms and simply typed.

If the lenghth is greater than 1, the $\lambda^\forall$-n.f. must have the form

$$\lambda x_1{:}\sigma_1 \ldots \lambda x_m{:}\sigma_m.\, h\, \alpha_1 \cdots \alpha_n$$

where $m, n \geq 0$, $\sigma_1, \ldots, \sigma_m$ are simple types, h is a variable or a constant, $\alpha_1, \ldots, \alpha_m$ are either type expressions or terms in $\lambda^\forall$-normal form and $h\,\alpha_1 \cdots \alpha_n$ is of simple type.

If h is not ε then it must be a variable or constant of simple type, thus $\alpha_1, \ldots, \alpha_m$ are all terms of simple type. Then, by induction hypothesis, each α_j has a $\lambda^\forall\varepsilon$-normal form N_j which is also a simply typed term. Hence, the whole term has the simply typed $\lambda^\forall\varepsilon$-normal form

$$\lambda x_1{:}\sigma_1 \ldots \lambda x_m{:}\sigma_m.\, h\, N_1 \cdots N_n\ .$$

If h is ε, let τ be the (simple) type of $\varepsilon\,\alpha_1 \cdots \alpha_m$. An easy induction on m shows that

$$\varepsilon\,\alpha_1 \cdots \alpha_m \overset{\lambda^\forall\varepsilon}{\longrightarrow\!\!\!\!\!\rightarrow} \varepsilon\,\tau\ .$$

But τ must be of the form $\tau_1 \to \cdots \to \tau_{m'} \to k$ so

$$\varepsilon\,\tau \overset{\varepsilon}{\longrightarrow\!\!\!\!\!\rightarrow} \lambda y_1{:}\tau_1 \ldots \lambda y_{m'}{:}\tau_{m'}.\, q_{d_k^0}\ .$$

Thus, the whole term has the simply typed $\lambda^\forall\varepsilon$-normal form

$$\lambda x_1{:}\sigma_1 \ldots \lambda x_m{:}\sigma_m.\, \lambda y_1{:}\tau_1 \ldots \lambda y_{m'}{:}\tau_{m'}.\, q_{d_k^0}\ .$$

End of Proof.

Refer now to the notation of the proof of Theorem 5.4.

Lemma A.2 *The $\lambda^\forall$-normal form of any Σ'-polymorphic term of simple type and with free variables of simple type only is actually a simply typed term.*

The proof, by induction on the length of $\lambda^\forall$-normal forms, can be extracted from the proof of the previous lemma by ignoring the complications caused by ε.

Appendix B: Historical note

The connection between extensionality and p.e.r.'s has a long history. For the case of simple (finite) types, the idea that (what amounts to) simply typed logical p.e.r.'s give extensionality is attributed in [Tro73], p. 124, to Zucker. Barendregt ([Bar84], p. 565), gives the name "extensional collapse" to Zucker's construction. The construction was extensively used by Statman [Sta82]. However, in [Sta80], Statman, while still refering (via [Sta82]) to [Tro73], calls essentially the same construction, the Gandy "hull", presumably in reference to [Gan56]. The extensional collapse was used to compare Kreisel's extensional model HEO [Kre59] with Kreisel-Troelstra's non-extensional "model" HRO [Tro73]. P.e.r.'s play a crucial role in the construction of HEO, but the extensional collapse of HRO, HRO^E, is set-theoretically distinct from HEO ([Tro73], p. 127). However, it was shown by Bezem (announcement in [Bar84], p. 566) that HRO^E and HEO are isomorphic.

Troelstra has extended HRO to Girard's second-order types (which include the types of $\lambda^\forall$), constructing the non-extensional "model" HRO^2 [Tro73] while Girard extended the HEO construction to HEO^2 [Gir72]. As mentioned in [Mit86b], Plotkin and Moggi have (independently from Girard's construction) constructed a model of $\lambda^\forall$, that seems to be intimately related to HEO^2. The Plotkin-Moggi model construction starts from the *partial* combinatory algebra of natural numbers and Kleene brackets application. It then uses a generalization to partial application of (what amounts to) the polymorphic extensional collapse for erase-types polymorphic lambda interpretations and the logical p.e.r. family (see subsection 4.3) consisting of all p.e.r.'s. The most obvious (but perhaps minor) difference between this model and HEO^2 is that there are empty types here but all the types of HEO^2 are non-empty, due to the addition of a canonical element, 0, at each type.

The categorical correspondent of the Plotkin-Moggi model is the Moggi-Hyland interpretation of $\lambda^\forall$ in the *modest sets* (a name given by D. Scott) which form an internally complete subcategory of the *realizability universe* or *effective topos*, [Hyl87]. This interpretation has the remarkable property that the polymorphic types are interpreted as products of modest-set objects, which can be seen as *intuitionistic*-set-theoretic products. Freyd and Scedrov exploit this interpretation in [FS87], based on a reconstruction of the modest sets described in more detail in [CFS87]. For a related, but technically different construction, see [HRR]. We believe that a minor modification of the HRO^2 construction can be seen as an erase-types polymorphic lambda interpretation (see subsection 5.2, and that the polymorphic

extensional collapse $coll(\mathrm{HRO}^2, \mathcal{ALC})$ (again see subsection 5.2) is isomorphic to a minor modification of the HEO^2 construction. As was pointed out to us by Scedrov, this seems to be supported by the fact that the construction of the modest sets by Carboni, Freyd and Scedrov [CFS87] (see also [FS87]) uses, among other things, the "splitting of all the symmetric idempotents" of a certain category. Indeed, in a category whose morphisms are relations the symmetric idempotents are exactly the p.e.r.'s and splitting corresponds to taking quotients. However, to the best of our knowledge, the *exact* connections between all these constructions, and especially the connections between the categorical constructions and those in the style of this paper, are still unclarified.

The reader can consult [Mit86b] for a few more examples of using p.e.r.'s to interpret types.

References

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, Amsterdam, second edition, 1984.

[BB85] C. Böhm and A. Berarducci. Automatic synthesis of typed Λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

[BC87] V. Breazu-Tannen and T. Coquand. Extensional models for polymorphism. In *Proceedings of TAPSOFT - Colloquium on Functional and Logic Programming and Specifications, Pisa, March 1987*, pages 291–307, *Lecture Notes in Computer Science*, Vol. 250, Springer-Verlag, 1987. An expanded version will appear in the special issue of *Theoretical Computer Science* dedicated to the colloquium.

[BM84] K. B. Bruce and A. R. Meyer. The semantics of second order polymorphic lambda calculus. In G. Kahn, D. B. MacQueen, and G. Plotkin, editors, *Proceedings of the Conference on Semantics of Data Types, Sophia-Antipolis, June 1984*, pages 131–144, *Lecture Notes in Computer Science*, Vol. 173, Springer-Verlag, 1984.

[BM87a] V. Breazu-Tannen and A. R. Meyer. Computable values can be classical. In *Proceedings of the 14th Symposium on Principles of Programming Languages*, pages 238–245, ACM, January 1987.

[BM87b] V. Breazu-Tannen and A. R. Meyer. Polymorphism is conservative over simple types. In *Proceedings of the Symposium on Logic in Computer Science*, pages 7–17, IEEE, June 1987.

[BMS80] R. M. Burstall, D.B. MacQueen, and D.T. Sanella. Hope: an experimental applicative language. In *LISP Conference*, pages 136–143, Stanford University Computer Science Department, 1980.

[Bre86] V. Breazu-Tannen. Communication in the TYPES electronic forum (types-@theory.lcs.mit.edu). July 29, 1986.

[Bre87a] V. Breazu-Tannen. *Conservative extensions of type theories.* PhD thesis, Massachusetts Institute of Technology, February 1987. Supervised by A. R. Meyer.

[Bre87b] V. Breazu-Tannen. Proof of a conjecture on polymorphic lambda models with all types non-empty. manuscript, univ. of pennsylvania. July 1987.

[Car85] L. Cardelli. Amber. In *Combinators and functional programming languages, Proceedings of the 13th Summer School of the LITP*, Le Val D'Ajol, Vosges, France, May 1985.

[CFS87] A. Carboni, P. J. Freyd, and A. Scedrov. A categorical approach to realizability and polymorphic types. In *Proceedings of the 3rd ACM Workshop on the Mathematical Foundations of Programming Language Semantics, New Orleans, April 1987, Lecture Notes in Computer Science*, Springer-Verlag, 1987. To appear.

[Coq86] T. Coquand. Communication in the TYPES electronic forum (types@theory-.lcs.mit.edu). April 14, 1986.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[Fai82] J. Fairbairn. *Ponder and its type system.* Tech. Rep. 31, Computer Laboratory, Univ. of Cambridge, Cambridge, England, November 1982.

[FLO83] S. Fortune, D. Leivant, and M. O'Donnell. The expressiveness of simple and second-order type structures. *Journal of the ACM*, 30(1):151–185, January 1983.

[Fri75] H. Friedman. Equality between functionals. In R. Parikh, editor, *Proceedings of the Logic Colloqium '73*, pages 22–37, *Lecture Notes in Mathematics*, Vol. 453, Springer-Verlag, 1975.

[FS87] P. Freyd and A. Scedrov. Some semantic aspects of polymorphic lambda calculus. In *Proceedings of the Symposium on Logic in Computer Science*, pages 315–319, IEEE, June 1987.

[Gan56] R. O. Gandy. On the axiom of extensionality—Part I. *Journal of Symbolic Logic*, 21:36–48, 1956.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieure.* PhD thesis, Université Paris VII, 1972.

[GM82] J. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *SIGPLAN Notes*, 17:9–17, 1982.

[GMW79] M. J. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF.* Volume 78 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1979.

[HL80] R. Hindley and G. Longo. Lambda-calculus models and extensionality. *Zeitschrift für Mathematische Logic und Grundlagen der Mathematik*, 26:289–310, 1980.

[HRR87] J. M. E. Hyland, E. P. Robinson, and G. Rosolini. The discrete objects in the effective topos. manuscript, univ. of cambridge. 1987.

[Hyl87] J. M. E. Hyland. A small complete category. 1987. Manuscript, to appear in the proceedings of the Conference on Church's Thesis: Fifty Years Later.

[Kre59] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in mathematics*, pages 101–128, North-Holland, 1959.

[Lei83] D. Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Symposium on Foundations of Computer Science*, pages 460–469, IEEE, 1983.

[Mat85] D. C. J. Matthews. *Poly manual.* Tech. Rep. 63, Computer Laboratory, Univ. of Cambridge, Cambridge, England, 1985.

[Mey82] A. R. Meyer. What is a model of the lambda calculus? *Information and Control*, 52(1):87–122, January 1982.

[Mey86] A. R. Meyer. Communication in the TYPES electronic forum (types@theory-.lcs.mit.edu). February 7, 1986.

[Mit86a] J. C. Mitchell. Personal communication. August 1986.

[Mit86b] J. C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *Proceedings of the LISP and Functional Programming Conference*, pages 308–319, ACM, New York, August 1986.

[MM85] J. C. Mitchell and A. R. Meyer. Second-order logical relations (extended abstract). In R. Parikh, editor, *Proceedings of the Conference on Logics of Programs, Brooklyn, June 1985*, pages 225–236, *Lecture Notes in Computer Science*, Vol. 193, Springer-Verlag, 1985.

[MM87] J. C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. In *Proceedings of the Symposium on Logic in Computer Science*, IEEE, June 1987.

[MMMS87] A. R. Meyer, J. C. Mitchell, E. Moggi, and R. Statman. Empty types in polymorphic λ-calculus. In *Proceedings of the 14th Symposium on Principles of Programming Languages*, pages 253–262, ACM, January 1987.

[Mog86a] E. Moggi. Communication in the TYPES electronic forum (types@theory.lcs-.mit.edu). February 10, 1986.

[Mog86b] E. Moggi. Communication in the TYPES electronic forum (types@theory.lcs-.mit.edu). July 23, 1986.

[Nik84] R. S. Nikhil. *An incremental, strongly typed database query language.* PhD thesis, Univ. of Pennsylvania, Philadelphia, August 1984. Available as tech. rep. MS-CIS-85-02.

[Pit87] A. M. Pitts. *Polymorphism is set theoretic, constructively.* Report 2/87, University of Susssex, School of Mathematical and Physical Sciences/Computer Science, July 1987. To appear in the proceedings of the Summer Conference on Category Theory and Computer Science, Edinburgh, Sept. 1987, Springer Lecture Notes in Computer Science.

[Rey74] J. C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, pages 408–425, *Springer Lecture Notes in Computer Science*, Vol. 19, Springer-Verlag, 1974.

[Rey83] J. C. Reynolds. Types, abstraction, and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing '83*, pages 513–523, North-Holland, 1983.

[Rey85] J. C. Reynolds. Three approaches to type structure. In *Mathematical foundations of software development*, pages 97–138, *Lecture Notes in Computer Science*, Vol. 185, Springer-Verlag, 1985.

[See86a] R. A. G. Seely. Categorical semantics for higher order polymorphic lambda calculus. 1986. Manuscript, to appear in Journal of Symbolic Logic.

[See86b] R. A. G. Seely. Higher order polymorphic lambda calculus and categories. *Mathematical Reports, Academy of Science (Canada)*, VIII(2):135–139, 1986.

[Sta80] R. Statman. On the existence of closed terms in the typed λ-calculus. In J. P. Seldin and R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 511–534, Academic Press, New York, 1980.

[Sta81] R. Statman. Number theoretic functions computable by polymorphic programs. In *22nd Symposium on Foundations of Computer Science*, pages 279–282, IEEE, 1981.

[Sta82] R. Statman. Completeness, invariance and λ-definability. *Journal of Symbolic Logic*, 47:17–26, 1982.

[Sta83] R. Statman. λ-definable functionals and $\beta\eta$ conversion. *Arch. Math. Logik*, 23:21–26, 1983.

[Tro73] A. S. Troelstra (ed.). *Metamathematical investigation of intuitionistic arithmetic and analysis.* Volume 344 of *Lecture Notes in Mathematics*, Springer-Verlag, 1973.

[Tur85] D. A. Turner. Miranda: a non-strict functional language with polymorphic types. In J.-P. Jouannaud, editor, *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, pages 1–16, *Springer Lecture Notes in Computer Science*, Vol. 201, Springer-Verlag, 1985.