Lambda Calculus
Type Theory
and
Natural Language

\( \lambda c_{Tt}.n_l \)

Edited by
Maribel Fernández
Chris Fox
Shalom Lappin

King’s College London
Strand, London, U.K.
12th September 2005
The Second Workshop on
Lambda Calculus, Type Theory, and Natural Language

King’s College London, Strand, London, U.K., 12th September 2005

Contributors:
Mario Fadda       Universitat Politcnica de Catalunya, Spain
Elias Ponvert    University of Texas at Austin, USA
Carl Pollard      Ohio State University, USA
Paula Severi      University of Leicester, UK
Glyn Morrill     Universitat Politcnica de Catalunya, Spain
Reinhard Muskens  Tilburg University, Netherlands
Fer-Jan de Vries  University of Leicester, UK

Invited Speakers:
J Roger Hindley  University of Swansea, UK
Raymond Turner  University of Essex, UK

Local Organizers:
Maribel Fernández King’s College London, UK
⟨maribel@dcs.kcl.ac.uk⟩
Shalom Lappin    King’s College London, UK
⟨lappin@dcs.kcl.ac.uk⟩

Programme Committee:
Jan van Eijck     CWI, Netherlands
Maribel Fernandez King’s College London, UK
Chris Fox         University of Essex, UK
Nissim Francez    Technion, Israel
Dov Gabbay       King’s College London, UK
Chris Hankin     Imperial College London, UK
Shalom Lappin    King’s College London, UK
Glyn Morrill     Universitat Politècnica de Catalunya, Spain
Ian Pratt        University of Manchester, UK
Aarne Ranta     Chalmers University of Technology, and Gteneborg University, Sweden
Phil Scott       University of Ottawa, Canada
Jacqueline Vauzeilles University of Paris Nord, France

Programme Chair:
Chris Fox        University of Essex, UK
⟨foxcj@essex.ac.uk⟩
Programme

Monday, 12th September.

9:15  Registration & Coffee — Welcome
10:00  Opening
10:10  Roger Hindley
    Invited talk: “Types in early combinatory logic”
10:55  Coffee Break (25 mins)
11:20  Paula Severi & Fer-Jan de Vries
    “Separability of infinite terms”
12:00  Elias Ponvert
    “Polymorphism in English logical grammar”
12:40  Lunch (60 mins)
13:40  Carl Pollard
    “Hyperintensional Semantics in a Higher-Order Logic with
    Definable Subtypes”
14:20  Reinhard Muskens
    “Intensional models for the theory of types”
15:00  Coffee Break (25 mins)
15:25  Glyn Morrill & Mario Fadda
    “Proof Nets for Basic Discontinuity”
16:05  Ray Turner
    Invited talk: “Computationalism”
16:55  Closing
19:00  Dinner
Preface

This is the second Workshop on Lambda Calculus, Type Theory, and Natural Language (LCTTNL). The first workshop was held in London in December 2003, and selected papers were published in a special issue of the Journal of Logic and Computation, Volume 15 Number 2, April 2005. The workshop was established with the goal of bringing together researchers interested in functional programming, type theory, and the application of the lambda calculus to the analysis of natural language. The communities that have developed around each of these areas share many formal and computational interests, but typically have little contact with each other. LCTTNL is intended to provide a forum in which people working on the lambda calculus from a variety of distinct perspectives will share their research and come to appreciate new domains of application. The success of the first workshop prompted us to hold this second workshop. We hope that this will continue as a regular series of workshops and further cooperation and collaboration between disciplines.

We would like to express our thanks to the programme committee, for their hard work in providing detailed and helpful feedback to the authors of all the submitted papers, to the speakers for their participation, and to our invited speakers, Roger Hindley and Ray Turner, for agreeing to present their lectures.

Maribel Fernández  
Chris Fox  
Shalom Lappin

Contents

Proof Nets for Basic Discontinuity 1
   Glyn Morrill and Mario Fadda

Intensional Models for the Theory of Types 17
   Reinhard Muskens

Hyperintensional Semantics in a Higher-Order Logic with Definable Subtypes 32
   Carl Pollard

Polymorphism in English Logical Grammar 47
   Elias Ponvert

Separability of Infinite Lambda Terms 61
   Paula Severi and Fer-Jan de Vries

Computationalism 76
   Raymond Turner
Proof Nets for Basic Discontinuity

Glyn Morrill
Universitat Politècnica de Catalunya
morrill@lsi.upc.edu
www.lsi.upc.edu/~morrill/

Mario Fadda
Universitat Politècnica de Catalunya
mfadda@lsi.upc.edu

Abstract

The theory of continuity based on the Lambek Calculus is well-developed, but we need a compatible extension to include discontinuity. Earlier work set out ingredients: hypersequent calculus and proof nets expanded with parameter edges. This paper completes a preliminary line by finalising proof nets for basic discontinuity (that with one point of discontinuity) and proving correctness with respect to hypersequent calculus.

1 Motivation

Since Pentus [15] proved the completeness of the calculus of Lambek [7] with respect to free semigroups, the Lambek calculus can lay a seemingly unassailable claim to be the logic of concatenation. And where syntactic structure is taken to be the concrete geometrical representation of the essential structure wherein an expression is deemed to be grammatical, Girard’s proof nets [4] for the Lambek calculus (Roorda [17]), for their parsimony and economy, can lay a seemingly unassailable claim to be the syntactic structures of the Lambek calculus. Furthermore, Morrill’s [12] processing model of parsing as the incremental construction of proof nets provides a complexity metric concordant with a range of performance phenomena (see also Johnson [6]). Thus the questions of the logic, structure and processing of concatenation in categorial grammar appear to be essentially resolved.

However, natural grammar is not purely concatenative: it includes discontinuous phenomena. Starting with Moortgat [8], the search has been on to find

*Thanks to Bob Carpenter and Oriol Valentín for comments and suggestions. All errors are our own. Work supported by CICYT project TIC2002–04019–C03–01.
for discontinuity what the Lambek calculus provides for continuity. One possible solution comes in the form of the *generalised discontinuity* of Morrill [13]. This extends the *basic discontinuity* of [14] by lifting the upper bound on the number of points of discontinuity. The question arises as to how to formulate proof nets for discontinuity; in particular, not proof nets based on procedural graph rewriting [16], which do not sit well with the incremental performance model, but rather proof nets based on a declarative, intrinsic, correctness criterion, for which the incremental performance model applies; the present proposal appears to succeed in this respect.

The concatenative Lambek proof nets are planar; proof nets for discontinuity must be partially non-planar, but it is quite unclear how to tune degrees of non-planarity. Morrill [11] formulates proof nets expanded with parameter edges for discontinuity in which planarity is not a correctness condition, but is meant to be entailed in the concatenative case; that proposal was conjectural and correctness was not proved. In this paper we modify the proposal of proof nets with parameter edges for basic discontinuity and prove correctness with respect to a sequent calculus.

## 2 Basic discontinuity

Let there be a *vocabulary* $V$ which is a set, and a *separator* $1$, $1 \not\in V$. The *basic discontinuity* prosodic structure induced by the vocabulary and the separator is the two-sorted algebra $(L_0, L_1, +, W)$ where $L_0 = V^*$, $L_1 = V^*1V^*$, $+$ is the operation of concatenation of functionality $L_0, L_0 \rightarrow L_0$ and $W$ is the operation of wrapping of functionality $L_1, L_0 \rightarrow L_0$ such that $(s_1s_2s_3)Ws_2 = s_1s_2s_3$.

The types $\mathcal{F}_0$ of sort $0$ and $\mathcal{F}_1$ of sort $1$ are defined on the basis of a set $A$ of atomic types of sort $0$ by:

\[
\begin{align*}
\mathcal{F}_0 & ::= A \mid \mathcal{F}_0 \cdot \mathcal{F}_0 \mid \mathcal{F}_0 \backslash \mathcal{F}_0 \mid \mathcal{F}_0 / \mathcal{F}_0 \mid \mathcal{F}_1 \cap \mathcal{F}_0 \mid \mathcal{F}_1 \setminus \mathcal{F}_0 \\
\mathcal{F}_1 & ::= \mathcal{F}_0 \upharpoonright \mathcal{F}_0 
\end{align*}
\]

Given a prosodic interpretation function $F$ mapping from $A$ into subsets of $L_0$, the denotation $[A_0] \subseteq L_0$ of types $A_0$ of sort $0$ and $[A_1] \subseteq L_1$ of types $A_1$ of sort $1$ are defined by:

\[
\begin{align*}
[A] &= F(A) & \text{for } A \in A \\
[A\cdot B] &= \{ s_1+s_2 \mid s_1 \in [A] \land s_2 \in [B] \} \\
[A\backslash C] &= \{ s_2 \mid \forall s_1 \in [A], s_1+s_2 \in [C] \} \\
[A/B] &= \{ s_1 \mid \forall s_2 \in [B], s_1+s_2 \in [C] \} \\
[A\cap B] &= \{ s_1Ws_2 \mid s_1 \in [A] \land s_2 \in [B] \} \\
[A\upharpoonright C] &= \{ s_2 \mid \forall s_1 \in [A], s_1Ws_2 \in [C] \} \\
[C\upharpoonright B] &= \{ s_1 \mid \forall s_2 \in [B], s_1Ws_2 \in [C] \} 
\end{align*}
\]
3 Hypersequent calculus

In hypersequent calculus for discontinuity [10] a discontinuous type is represented by punctuated type occurrences at the distinct loci of its discontinuous segments. The configurations \( \mathcal{O}_0 \) of sort 0 and \( \mathcal{O}_\{\} \) of sort 1 are defined as follows, where \( \Lambda \) is the empty configuration:

\[
\begin{align*}
\mathcal{O}_0 & ::= \Lambda \mid A_0 \mid \mathcal{O}_0, \mathcal{O}_0 \mid \sqrt{A_1}, \mathcal{O}_0, \sqrt{A_1} \\
\mathcal{O}_\{\} & ::= [] \mid \mathcal{O}_0, \mathcal{O}_\{\} \mid \mathcal{O}_\{\}, \mathcal{O}_0 \mid \sqrt{\mathcal{A}_1}, \mathcal{O}_\{\}, \sqrt{\mathcal{A}_1}
\end{align*}
\]

In configurations, all occurrences of types of sort 1 are pairwise matched. In a configuration \( \mathcal{O}_\{\} \) of sort 1 there is a metalogical separator \( [] \) marking the point of the discontinuity. Sequents are of the form \( \mathcal{O}_0 \Rightarrow A_0 \) (sort 0) and \( \mathcal{O}_\{\} \Rightarrow A_1 \) (sort 1).

We extend the interpretation of types to include configurations as follows, where \( \emptyset \) is the empty string:

\[
\begin{align*}
[\Lambda] = \{\emptyset\} \\
[[\]] = \{1\} \\
[\mathcal{O}^{(1)}, \mathcal{O}^{(2)}] = \{s_1s_2 \mid s_1 \in [\mathcal{O}^{(1)}] \& s_2 \in [\mathcal{O}^{(2)}]\} \\
[\sqrt{\mathcal{A}_1}, \mathcal{O}, \sqrt{\mathcal{A}_1}] = \{s_1s_2s_3 \mid s_1s_3 \in [A_1] \& s_2 \in [\mathcal{O}]\}
\end{align*}
\]

A sequent \( \Gamma \Rightarrow A \) is valid if and only if \( [\Gamma] \subseteq [A] \) in all interpretations. The hypersequent calculus for basic discontinuity is as given in Figure 1 where \( \Gamma \) and \( \Delta \) range over arbitrary sequences of sort 0 and punctuated sort 1 types, and separators.

\[
(5) \quad \text{Proposition (Soundness of hypersequent calculus for basic discontinuity).} \quad \text{If a sequent is derivable in the hypersequent calculus then it is valid.}
\]

Proof. Induction on the construction of hypersequent proofs. For example, to check the case corresponding to the \( \odot \) rule, one has to show that if \( [\Delta_1, \sqrt{\mathcal{A}}, B, \sqrt{\mathcal{A}}, \Delta_2] \subseteq [D] \) then \( [\Delta_1, A \odot B, \Delta_2] \subseteq [D] \). This is the case,}

\[\text{1}\]The configurations can be generated by an unambiguous grammar:

\[
\begin{align*}
\mathcal{O}_0 & ::= \Lambda \mid A_0 \mid \mathcal{O}_0, \mathcal{O}_0 \mid \sqrt{A_1}, \mathcal{O}_0, \sqrt{A_1} \\
\mathcal{O}_\{\} & ::= \mathcal{O}_0, [], \mathcal{O}_0 \mid \mathcal{O}_0, \sqrt{A_1}, \mathcal{O}_\{\}, \sqrt{A_1}, \mathcal{O}_0
\end{align*}
\]

\[\text{2}\]O. Valentín, p.c. A simple proof by induction on the complexity of configurations shows that the same semantics can be defined on the bases of the given unambiguous grammar:

\[
\begin{align*}
[A_0, \Gamma_0] &= \{s_1s_2 \mid s_1 \in [A_0] \& s_2 \in [\Gamma_0]\} \\
[\sqrt{A_1}, \Gamma_0, \sqrt{A_1}, \Delta_0] &= \{s_1s_2s_3s_4s_5 \mid s_1s_5 \in [A_1] \& s_2 \in [\Gamma_0] \& s_4 \in [\Delta_0]\} \\
[\Gamma_0, [], \Delta_0] &= \{s_1s_2s_3s_4s_5 \mid s_1 \in [\Gamma_0] \& s_2 \in [\Delta_0]\}
\end{align*}
\]

The application of the inductive step in the definition does not depend on the way the configuration is decomposed, because the operation of concatenation is associative.
Figure 1: Hypersequent calculus for basic discontinuity
since a simple proof by induction over the complexity of the configuration \( \Delta_1, \sqrt{A}, B, \sqrt{A}, \Delta_2 \) shows that \( [[\Delta_1, \sqrt{A}, B, \sqrt{A}, \Delta_2]] = [[\Delta_1, A \odot B, \Delta_2]] \). Similarly, for the \( \odot R \)-rule one has to show that \( [[\Gamma_1, \Delta, \Gamma_2]] \subseteq [[A \odot B]] \) follows from \( [[\Gamma_1, [\cdot], \Gamma_2]] \subseteq [[A]] \) and \( [[\Delta]] \subseteq [[B]] \). Again, an induction over the complexity of \( \Gamma_1, [\cdot], \Gamma_2 \) shows that \( [[\Gamma_1, \Delta, \Gamma_2]] = \{ s_1s_3s_2 | s_1s_3 \in [[\Gamma_1, [\cdot], \Gamma_2]], s_2 \in [[\Delta]] \} \), that is then contained in \( \{ s_1s_3s_2 | s_1s_3 \in [[A]], s_2 \in [[B]] \} = [[A \odot B]] \). Q.E.D.

(6) **Theorem** (*Cut-elimination for the hypersequent calculus for basic discontinuity*). If a sequent is derivable in the hypersequent calculus then it has a Cut-free derivation.

**Proof.** Cut-elimination follows from the embedding result presented below, because the embedding can be used to translate derivations as well as sequents. A proof \( \pi \) of a hypersequent of the calculus for basic discontinuity can be translated into a proof of MILL1. Any occurrence of a Cut in the translation of \( \pi \) can be removed, since MILL1 enjoys the Cut-elimination property. The Cut-free proof can be translated back into a proof of the hypersequent calculus for basic discontinuity. Q.E.D.

(7) **Corollary** (*Decidability of basic discontinuity*). It is decidable whether a hypersequent of basic discontinuity is a theorem.

**Proof.** By backward-chaining in the finite Cut-free hypersequent search space. Q.E.D.

4 **Embeddings in MILL1**

The hypersequent calculus for basic discontinuity can be embedded into MILL1, the multiplicative and exponential free fragment of intuitionistic linear logic with first order quantification, along the lines of the embedding of the Lambek Calculus into MILL1 of Moot and Piazza [9]. In the translation, a type of sort 0 is turned into a binary predicate, and a type of sort 1 into a quaternary predicate. The arguments encode the start and end positions of segments of a type in a hypersequent.

Let us call the *length* of a configuration \( \mathcal{O} \) the number \( l(\mathcal{O}) \) of type occurrences and separators that form it. To a configuration \( \mathcal{O}|\cdot | \) of sort 1 we furthermore associate the number \( p\mathcal{O} \) of type occurrences that precede the separator. A sequent \( \mathcal{O} \Rightarrow B \) is translated as \( ||\mathcal{O}||^\varphi \Rightarrow ||B||^\varphi \) where for distinct constants \( e_0, \ldots, e_{l(\mathcal{O})} \), \( \varphi = \langle e_0, e_{l(\mathcal{O})} \rangle \) if the sequent is of sort 0, and \( \varphi = \langle e_0, e_{p\mathcal{O}}, e_{p\mathcal{O}+1}, e_{l(\mathcal{O})} \rangle \) if it is of sort 1. The translation of the configurations and of the types are defined by induction in Tables 1 and 2.
A sequent \( \Gamma \Rightarrow B \) is a theorem of the hypersequent calculus for basic discontinuity if and only if its translation \( ||\Gamma||^\varphi \Rightarrow ||B||^\varphi \) is a theorem of MILL1, \( \varphi = (\epsilon_0, \epsilon_l(\Gamma)) \) if the sequent is of sort 0, and \( \varphi = (\epsilon_0, \epsilon_p^{l+1}, \epsilon_p^{l+1}, \epsilon_l(\Gamma)) \) if it is of sort 1.

**Proof.** This is a straightforward generalization of the analogous result for the embedding of the Lambek Calculus into MILL1 of [9]. On the one hand, the translation of the conclusion of any hypersequent calculus rule can be derived in MILL1 from the translation of its premise(s). On the other hand, without loss of generality, in proving that the translation of a hypersequent is a theorem of MILL1, the order of application of rules can be modified so as to view

\[
\begin{array}{|c|c|}
\hline
\text{configuration } \mathcal{O} & \text{translation } ||\mathcal{O}||^{(\epsilon_i, \epsilon_j)} \\
\hline
\Lambda & \Lambda \\
\hline
\mathcal{A}_0 & ||\mathcal{A}_0||^{(\epsilon_i, \epsilon_j)} \\
\hline
\mathcal{O}_0^{(1)} , \mathcal{O}_0^{(2)} & ||\mathcal{O}_0^{(1)}||^{(\epsilon_i, \epsilon_h)}, ||\mathcal{O}_0^{(2)}||^{(\epsilon_h, \epsilon_j)} \\
\hline
\sqrt{\mathcal{A}_1}  , \mathcal{O}_1 & ||\mathcal{A}_1||^{(\epsilon_i, \epsilon_{i+1} , \epsilon_{j-1} , \epsilon_{j-1} )}, ||\mathcal{O}_1||^{(\epsilon_{i+1} , \epsilon_{j-1} , \epsilon_{j-1} )} \\
\hline
\end{array}
\]

Table 1: Translation of hypersequent configurations into MILL1

\[
\begin{array}{|c|c|}
\hline
\text{Continuous Types} & \text{Discontinuous Types} \\
\hline
||A \bullet B||^{<u,v>} & = \exists x( ||A||^{<u,x>} \otimes ||B||^{<x,v>}) \\
||A / B||^{<u,v>} & = \forall x( ||B||^{<x,v>} \circ -o ||A||^{<u,x>}) \\
||A \backslash B||^{<u,v>} & = \forall x( ||A||^{<x,u,>} -o ||B||^{<x,v>}) \\
\hline
\end{array}
\]

Table 2: Translation of types into MILL1
the derivation in MILL1 as a translation of a derivation in the hypersequent calculus for basic discontinuity. Q.E.D.

5 Proof nets

There are the standard notions of polarity, logical link and proof frame as the arranged formula trees of a sequent, proof structure as a result of connecting complementary literals in a proof frame by axiom links, and proof net as a proof structure which corresponds to a sequent proof. Keeping the usual Danos-Regnier condition [3] for multiplicative linear validity, we use the proof nets expanded with parameter edges of [11] to encode the sublinear structure. We replace the two “resolution criteria” with a single new geometric condition of unicity, for which we prove correctness.

The proofs are based on the theory for LL1, i.e. multiplicative linear logic with first order quantification, expounded in [1], and enriched with the following result:

\[\text{(9) Definition. Let } \Pi \text{ be an LL1 proof structure and } V \text{ a set of variables. An } V\text{-path in } \Pi \text{ is a path that goes exclusively through nodes labelled by types containing free occurrences of all variables that appear in } V.\]

\[\text{(10) Theorem. Let } \Pi \text{ be a proof structure of LL1 all of the Danos-Regnier } \wp\text{-switchings of which are connected and acyclic. Then the following conditions are equivalent:}\]

1. \(\Pi\) is correct with respect to \(\wp\)- and \(\forall\)-switchings in the sense of [1].

2. There are no \(\forall\)-links with conclusions \(\forall x A\) and \(\forall y B\) the premisses of which can be joined by an \(\{x, y\}\)-path.

\[\text{Proof. By induction over the size of the proof net. The two mentioned properties are preserved by removal of final } \wp\text{- and } \forall\text{-links. Moreover, if there are no such final links, the Splitting Lemma can be applied to } \Pi \text{ thought of as a proof structure of multiplicative linear logic. Again, the mentioned properties are preserved in each substructure. Q.E.D.}\]

Consider, among all LL1 proof structures, only those that can be built unfolding the \(\|\|-\)-translation of a Lambek sequent. The unfolding of, say, the translation \(\|A\bullet B^*\|^\x\eta\) of an input type \(A\bullet B^*\) will create a tree rooted by a \(\forall\)-link with conclusion \(\forall x(\|A^*\|^\x\eta \wp\|B^*\|^\x\eta)\) and with branches leading to the nodes \(\|A^*\|^\x\eta\) and \(\|B^*\|^\x\eta\). The latter types are joined by an \(\{x\}\)-path (that does not go through the premise of the \(\wp\)-link immediately below them). A simple application of the previous theorem will show that any such
path does not go through the premise of another $\forall$-link. Furthermore, this condition, together with acyclicity and connectedness of switchings, is sufficient for the correctness of the proof structure. This gives rise to an alternative characterization of sequentializable Lambek proof structures. In addition to the usual (solid) edges (which we refer to as predicate edges), we decorate links with (nonsolid) parameter edges, see Figures 2 and 3, which are in essence trip instructions on the proof structure. To obtain a proof frame, we add moreover parameter edges joining pairwise the starts and ends of the roots as illustrated in Figure 4 for sort 0 types (these edges stand for the parameters that freeze the order of the types in the sequent).
Figure 3: Logical proof links of the Lambek calculus and their expansions, II

Figure 4: Base parameter edges for a sequent of sort 0 types $A_1, A_2, \ldots, A_n \Rightarrow A$. 
**Definition.** A proof structure expanded with parameter edges is *correct* if and only if the following conditions are satisfied:

1. *Danos-Regnier acyclicity.* Every predicate edge cycle crosses both edges of some \(\otimes\)-link.

2. *Unicity.* Every parameter edge cycle contains exactly one \(\forall\).

Consider an expanded proof structure \(\Pi\) that can be sequentialized as the derivation of a sequent \(\Phi\), and let \(\Pi'\) be the proof structure associated with a derivation of the translation \(||\Phi||\). The parameter edges in Figures 2 and 3 can be joined to form parameter paths (cycles). The name is motivated by the fact that parameter paths in \(\Pi\) correspond to a way of travelling along \(\{x\}\)-paths in \(\Pi'\), where \(x\) stands for a variable occurring in \(\Pi'\).

In expanded proof nets for the Lambek Calculus, polar type trees reflect the binary relational interpretation clauses of [2] and the translation above. Each node labelled by a polar type has two incident dashed edges referred to as its start and its end parameter edges. The start comes on the left and the end comes on the right; for an output type this is reversed:

\[
\text{(12) start } A^* \text{ end } A^o \text{ start end} \]

These parameter edges are connected to quantifiers in the expanded proof structures which bind the parameters of types regarded as binary predicates. Extending to discontinuity, while types of sort 0 have two incident parameter edges, types of sort 1 have four incident parameter edges, corresponding to a quaternary relational predication, notated in expanded proof nets as in (13):

\[
\text{(13) start}_1 \text{ end}_2 A^* \text{ start}_2 \text{ end}_1 \text{ end}_1 \text{ start}_2 A^o \text{ end}_2 \text{ start}_1 \]

The subscripts refer to the first (left) and second (right) segments of a string containing a separator; note that as for types of sort 0 the input and output orderings are mirror-images, which promotes visual symmetry. The expanded links for the discontinuity operators are given in Figure 5.

**Proposition** (*Completeness of expanded proof nets with respect to hypersequent calculus for basic discontinuity*). If a hypersequent is derivable, there is a proof net for it.

**Proof.** Induction over the length of a derivation. Q.E.D.

**Theorem** (*Soundness of expanded proof nets with respect to hypersequent calculus for basic discontinuity*). Every proof net corresponds to a derivable hypersequent.

**Proof.** Consider a proof net \(\Pi\) of the discontinuity calculus. \(\Pi\) can be turned into a proof structure of MIL1 thus:
Figure 5: Expanded proof links for the basic discontinuity operators
1. Associate to each base parameter edge a distinct constant.

2. If $X$ is a root type of sort 0, replace $X$ with $||X||_{(e_i,e_j)}$ where $e_i$ ($e_j$) is the constant associated to the start (end) of $X$.

3. If $X$ is a root type of sort 1, replace $X$ with $||X||_{(e_{i_1},e_{j_1},e_{i_2},e_{j_2})}$ where $e_{i_h}$ ($e_{j_h}$) is the constant associated with the start (end) of the $h$-th segment of $X$.

4. In logical links, replace parameter edges by quantified variables.

This yields an MILL1 proof structure $\Pi'$ the roots of which form a sequent $\Sigma'$ which is the translation of a sequent $\Sigma$ of the hypersequent calculus. $\Pi'$ inherits correctness from $\Pi$ because of Theorem (10). Thus $\Pi'$ corresponds to a MILL1 derivation of $\Sigma'$, and, because of Theorem (8) $\Pi$ can be sequentialized as a derivation of $\Sigma$. Q.E.D.

6 Examples

Figure 6 shows an example of a proof net with a discontinuous functor for the following type assignments:

$\begin{align*}
\text{(16)} & \quad \text{gave}+1+\text{the}+\text{cold}+\text{shoulder} - \text{shun}: (N\setminus S)\uparrow N \\
& \quad \text{John} - j: N \\
& \quad \text{Mary} - m: N
\end{align*}$

The semantic reading of the proof net according to the semantic trip (see [5, 12]) is $((\text{shun } m) \cdot j)$.

Figures 7 and 8 show subject wide scope and object wide scope analyses respectively for ‘everyone loves someone’, where the quantifier words are assigned type $(S\uparrow N)\downarrow S$. The former includes parameter edges; in the latter these are omitted. The type assignments are as follows:

$\begin{align*}
\text{(17)} & \quad \text{everyone} - \forall: (S\uparrow N)\downarrow S \\
& \quad \text{loves} - \heartsuit: (N\setminus S)/N \\
& \quad \text{someone} - \exists: (S\uparrow N)\downarrow S
\end{align*}$

The results of the semantic trips are respectively $(\forall \lambda x (\exists y ((\heartsuit y) \cdot x)))$ and $(\exists y (\forall x ((\heartsuit y) \cdot x)))$. 

12
Figure 6: Proof net for the discontinuous idiom example ‘John gave Mary the cold shoulder’ via a wrapping functor
Figure 7: Proof net for ‘everyone loves someone’, subject wide scope
everyone loves someone

References


Figure 8: Proof net for ‘everyone loves someone’, object wide scope


Intensional Models for the Theory of Types

Reinhard Muskens
Tilburg University
r.a.muskens@uvt.nl

Abstract

In this paper we define intensional models for the classical theory of types, thus arriving at an intensional type logic ITL. Intensional models generalize Henkin’s general models and have a natural definition. As a class they do not validate the axiom of Extensionality. We give a cut-free sequent calculus for type theory and show completeness of this calculus with respect to the class of intensional models via a model existence theorem. After this we turn our attention to an application. It is argued that, since ITL is truly intensional, it can be used to model ascriptions of propositional attitude without predicting logical omniscience. In order to illustrate this a small fragment of English is defined and provided with a semantics.

1 Introduction

The axiom scheme of Extensionality states that whenever two predicates or relations are coextensive they must have the same properties:

$$\forall X Y (\forall \bar{x}(X \bar{x} \leftrightarrow Y \bar{x}) \rightarrow \forall Z (Z X \rightarrow Z Y))$$

Historically Extensionality has always been problematic, the main problem being that in many areas of application, though not perhaps in the foundations of mathematics, the statement is simply false. This was recognized by Whitehead and Russell in Principia Mathematica [31], where intensional functions such as ‘A believes that p’ or ‘it is a strange coincidence that p’ are discussed at length. However, in the introduction to the second edition (1927) of the Principia Whitehead and Russell (influenced by Wittgenstein’s Tractatus) already entertain the possibility that “all functions of functions are extensional”. Thirteen years later, in Church’s [5] canonical formulation of the Theory of Types, it is observed that axioms of Extensionality should be adopted “[i]n order to obtain classical real number theory (analysis)”, a wording that does not seem to rule out the option of not adopting them. Church’s formulation of type theory was completely syntactic, and axioms could be adopted or dropped at will,
but in Henkin's [12] classical proof of generalized completeness, the models that are considered, both the “standard” models and the “generalized” ones, simply validate Extensionality. Intensional predicates and functions are now ruled out.

This poses problems for those areas of application of the logic where it is important to distinguish between predicates that are coextensive and where propositions that determine the same set of possible worlds should be kept apart nevertheless. Linguistic semantics and Artificial Intelligence are such applications and the problem has been dubbed one of “logical omniscience” there, for it is with propositional attitudes like knowledge and belief that predicates of predicates and predicates of propositions most naturally arise. Is there a deep foundational difficulty with type theory that makes the theory adequate for one area of application (mathematics) but not for others? Or is it possible to come up with a revised and generalized semantics for the logic, in which intensional predicates of predicates (or intensional functions of functions) are allowed? In the latter case Extensionality becomes a non-logical axiom that can be added to the theory for the purposes of one area of application while in other areas of application it is not added.

Even if one is interested in mathematical applications of type theory only there are good reasons to consider a generalization of its models in which Extensionality fails. This was realized by Takahashi [26] and Prawitz [23] in their (independent, but closely related) proofs of Cut-elimination. These proofs make use of what Andrews [1] calls “V-complexes”, structures whose typed domains consist of elements \( \langle A, e \rangle \), where \( A \) is a term and \( e \) is a possible extension of \( A \). Clearly, two objects \( \langle A, e \rangle \) and \( \langle B, e' \rangle \) can be distinct even if \( e = e' \). Andrews [1] uses V-complexes to show that a certain resolution system \( R \) corresponds to the first six axioms of Church [5] (not comprising Extensionality).

V-complexes in themselves cannot be used as independent models for an intensional type theory, as their definition depends on Schütte’s [24] “semi-valuations”, essentially sets of sentences (the “V” in “V-complex” ranges over semi-valuations). Is it possible to define a stand-alone notion of general intensional model that has V-complexes as a special case? I know of two proposals for such general models, both recent. The first is found in Fitting [9], the second in Benzmüller et al. [2]. In Fitting’s “generalized Henkin models” abstraction may receive a non-standard interpretation, while in the “\( \Sigma \)-models” of Benzmüller et al. it is application that may be interpreted in a non-standard way. Such non-standard evaluations seem unnecessary, however, and in this paper, I will propose a simple definition of intensional model that generalizes Henkin models for type theory but gives all logical operations their usual semantics. The possession of such a simplified and abstract notion of intensional model will hopefully contribute to a deeper understanding of the distinction between intension and extension. The key idea will be that domains of complex type are inhabited by intensional objects, that a relatively unconstrained function \( I \) sends terms (and assignments) to these intensions, and that a function \( E \) (borrowed from Fitting, on whose shoulders I stand) sends intensions to extensions.
The composition of $E$ and $I$ must be so constrained that terms will obtain their usual semantics. The system of type theory interpreted with the help of the intensional models thus defined will be called ITL (‘Intensional Type Logic’).

The remainder of the paper is organized as follows. The following section will define the types and terms of a simple type theory in the spirit of Church [5] (but framed as a relational theory, as in Orey [22] and Schütte [24]). In section 3 our notion of intensional model will be defined, with a corresponding notion of entailment. Section 4 introduces a cut-free Gentzen calculus for ITL while section 5 proves a Model Existence theorem. The proofs in that section all employ familiar techniques but are given as a sanity check on the definition of the basic modeltheoretic notions. The paper ends with a section on linguistic applications, followed by a conclusion.

## 2 Terms

In this section the types and terms of ITL will be defined and some notation will be adopted. Assuming that some finite set $B$ of basic types is given, the following definition gives the set of (relational) types.

**Definition 1.** The set $\mathcal{T}$ of types is the smallest set of strings such that

1. $B \subseteq \mathcal{T}$
2. If $\alpha_1, \ldots, \alpha_n \in \mathcal{T}$ ($n \geq 0$) then $\langle \alpha_1 \ldots \alpha_n \rangle \in \mathcal{T}$

Types formed with the second clause of this definition will be called complex. Note that, as a limiting case, $\langle \rangle$ is defined to be a (complex) type; this will be the type of propositions and truth values. A language will be a countable set of typed non-logical constants. If $\mathcal{L}$ is a language, the set of constants from $\mathcal{L}$ that have type $\alpha$ is denoted $L_{\alpha}$. For each $\alpha \in \mathcal{T}$ we moreover assume the existence of a denumerably infinite set $V_{\alpha}$ of variables of type $\alpha$. We let $V = \bigcup_{\alpha \in \mathcal{T}} V_{\alpha}$.

The following definition gives us terms in all types. Apart from variables and non-logical vocabulary there will be a sentence $\bot$ that is always false, and there will be application and abstraction. Furthermore, a symbol $\sqsubseteq$ will denote inclusion of extensions, so that $A \sqsubseteq B$ is true if the extension of $A$ is a subset of that of $B$.

**Definition 2.** Let $\mathcal{L}$ be a language. Define sets $T_{\alpha}^{\mathcal{L}}$ of terms of $\mathcal{L}$ of type $\alpha$, for each $\alpha \in \mathcal{T}$, as follows.

1. $L_{\alpha} \subseteq T_{\alpha}^{\mathcal{L}}$ and $V_{\alpha} \subseteq T_{\alpha}^{\mathcal{L}}$ for each $\alpha \in \mathcal{T}$
2. $\bot \in T_{0}^{\mathcal{L}}$
3. If $A \in T_{\langle\alpha_1\alpha_2\ldots\alpha_n\rangle}^{\mathcal{L}}$ and $B \in T_{\alpha_1}^{\mathcal{L}}$, then $(AB) \in T_{\langle\alpha_2\ldots\alpha_n\rangle}^{\mathcal{L}}$
4. If $A \in T^c_{(\alpha_2...\alpha_n)}$ and $x \in \mathcal{V}_{\alpha_1}$, then $(\lambda x.A) \in T^c_{(\alpha_1\alpha_2...\alpha_n)}$

5. If $A \in T^c_\alpha$ and $B \in T^c_\alpha$ then $(A \sqsubseteq B) \in T^c_{\emptyset}$, if $\alpha$ is complex

We will write $T^c_\alpha$ for the set of all terms of the language $\mathcal{L}$, i.e. for the union $\bigcup_{\alpha \in \mathcal{T}} T^c_{\alpha}$. If $A$ is a term of type $\alpha$, we may indicate this by writing $A_\alpha$, and we will use $\varphi$, $\psi$, $\chi$ for terms of type $\langle \rangle$, which we call formulas. The notions free and bound occurrence of a variable and the notion $B$ is free for $x$ in $A$ are defined as usual. As are closed terms and sentences. Substitutions are functions $\sigma$ from variables to terms such that $\sigma(x)$ has the same type as $x$. If $\sigma$ is a substitution then the substitution $\sigma'$ such that $\sigma'(x) = A$ and $\sigma'(y) = \sigma(y)$ for all $y \neq x$ is denoted as $\sigma[x := A]$. If $A$ is a term and $\sigma$ is a substitution, $A_\sigma$, the extension of $\sigma$ to $A$, is defined in the usual way. The substitution $\sigma$ such that $\sigma(x_i) = A_i$ and $\sigma(y) = y$ if $y \notin \{x_1, \ldots, x_n\}$ is written as $\{x_1 := A_1, \ldots, x_n := A_n\}$. Parentheses in terms will often be dropped on the understanding that $ABC$ is $((AB)C)$, i.e. association is to the left.

Our stock of operators may seem somewhat spartan, but is rich enough to let the usual connectives and quantifiers be defined. In particular, $\forall$, $\rightarrow$ and $=$ are easily obtained.

Definition 3. Write

\[
\begin{align*}
\varphi \rightarrow \psi & \quad \text{for} \quad \varphi \sqsubseteq \psi, \\
\top & \quad \text{for} \quad \bot \rightarrow \bot, \\
\forall x \varphi & \quad \text{for} \quad (\lambda x. \top) \sqsubseteq (\lambda x. \varphi), \text{ and} \\
A_\alpha = B_\alpha & \quad \text{for} \quad \forall x_{(\alpha)} (xA \rightarrow xB).
\end{align*}
\]

The operators $\neg$, $\land$, $\lor$ and $\exists$ are obtained as usual.

Our presentation of the logic will revolve around sequents. A signed sentence of $\mathcal{L}$ will be a pair $\langle L, \varphi \rangle$ (written $L: \varphi$) or a pair $\langle R, \varphi \rangle$ (written $R: \varphi$), such that $\varphi$ is a sentence of $\mathcal{L}$ ($L$ indicates ‘left’ and $R$ indicates ‘right’). A sequent of $\mathcal{L}$ is a set of signed sentences of $\mathcal{L}$. Letting sequents be sets has some advantages, but we may also want to use a more conventional form and write $\Pi \Rightarrow \Sigma$ for $\{\langle L, \varphi \rangle \mid \varphi \in \Pi\} \cup \{\langle R, \varphi \mid \varphi \in \Sigma\}$ if $\Pi$ and $\Sigma$ are sets of sentences.

3 Intensional Models

Let us turn to the semantics of ITL. A collection of domains will be a set $\{D_\alpha \mid \alpha \in \mathcal{T}\}$, each of whose elements is non-empty. An assignment $a$ for a collection of domains $D = \{D_\alpha \mid \alpha \in \mathcal{T}\}$ is a function which has the set of variables $\mathcal{V}$ as domain and has the property that $a(x) \in D_\alpha$ if $x \in \mathcal{V}_\alpha$. The set of all assignments for $D$ is denoted $\mathcal{A}_D$. If $a$ is an assignment, $d \in D_\alpha$, and $x$ is a variable of type $\alpha$, $a[d/x]$ is defined by letting $a[d/x](x) = d$ and $a[d/x](y) = a(y)$, if $y$ is not equal to $x$. 

20
We need to tease apart the intensions and extensions of terms of complex
type. While extensions of such terms will be certain relations, with their iden-
tity criteria therefore given by set membership, the intension functions
defined below send terms to almost arbitrary domain elements. There are a few restric-
tions on these functions but they are rather liberal.

**Definition 4.** An intension function for a collection of domains \( D = \{ D_\alpha \mid \alpha \in T \} \) and a language \( \mathcal{L} \) is a function \( I : A_D \times T^L \to D \) such that

1. \( I(a, A) \in D_\alpha \), if \( A \) is of type \( \alpha \)
2. \( I(a, x) = a(x) \), if \( x \) is a variable
3. \( I(a, A) = I(a', A) \), if \( a \) and \( a' \) agree on all variables free in \( A \)
4. \( I(a, A\{x := B\}) = I(a[I(a, B)/x], A) \), if \( B \) is free for \( x \) in \( A \)

The following definition provides extension functions sending objects of complex
types to certain relations over the relevant domains. We first give very general
constraints and will put more requirements on extension functions in definition
7 below.

**Definition 5.** Let \( D = \{ D_\alpha \mid \alpha \in T \} \) be a collection of domains and let \( \alpha = \langle \alpha_1 \ldots \alpha_n \rangle \) be a complex type. An extension function of type \( \alpha \) for \( D \) is a
function \( E_\alpha : D_\alpha \to \mathcal{P}(D_{\alpha_1} \times \cdots \times D_{\alpha_n}) \). A collection of extension functions for \( D \) is a set \( \{ E_\alpha \mid \alpha \in T \setminus \mathcal{B} \text{ and } E_\alpha \text{ is an extension function of type } \alpha \text{ for } D \} \).

In the limiting case that \( n = 0 \) the product \( D_{\alpha_1} \times \cdots \times D_{\alpha_n} \) equals \( \{ \langle \rangle \} \). We identify \( \langle \rangle \) with \( \emptyset \), \( \emptyset \) with 0, and \( \{ \emptyset \} \) with 1, so that \( E_{\emptyset} : D_{\emptyset} \to \{0, 1\} \) if \( E_{\emptyset} \)
is an extension function of type \( \langle \rangle \) for \( D \).

**Definition 6.** A generalized frame for the language \( \mathcal{L} \) is a triple \( \langle D, I, E \rangle \) such
that \( D \) is a collection of domains, \( I \) is an intension function for \( D \) and \( \mathcal{L} \), and
\( E \) is a collection of extension functions for \( D \).

We are interested in the extensions \( E_\alpha(I(a, A_\alpha)) \) of terms \( A \) of complex type \( \alpha \). Let \( V_\alpha \) be the composition of \( E_\alpha \) and \( I \), so that, in the interest of readability, we can write \( V_\alpha(a, A) \), or even \( V(a, A) \), for \( E_\alpha(I(a, A)) \). The following definition,
which gives the central notion of this paper, puts constraints on intension and
extension functions that cause terms to get their usual semantic values.

**Definition 7.** A generalized frame \( \langle D, I, E \rangle \) for \( \mathcal{L} \) is an intensional model for \( \mathcal{L} \) if

1. \( V(a, \bot) = 0 \)
2. \( V(a, AB) = \{ \langle \tilde{d} \rangle \mid \langle I(a, B), \tilde{d} \rangle \in V(a, A) \} \)
3. \( V(a, \lambda x\beta. A) = \{ \langle d, \tilde{d} \rangle \mid d \in D_\beta \text{ and } \langle \tilde{d} \rangle \in V(a[d/x], A) \} \)
4. $V(a, A \sqsubseteq B) = 1 \iff V(a, A) \subseteq V(a, B)$

To better understand the motivation behind the second and third clauses of this definition, it may help to consider that any $n + 1$ place relation $R$ can be thought of as a unary function $F$ such that $F(d) = \{ \langle \vec{d} \rangle \mid \langle d, \vec{d} \rangle \in R \}$. Thus $V(a, AB) = F(I(a, B))$, where $F$ is the function corresponding to $V(a, A)$ and $V(a, \lambda x_\beta.A)$ corresponds to the function $F$ such that $F(d) = V(a[d/x], A)$ for each $d \in D_\beta$. For further discussion of this little trick in an extensional setting see Muskens [18, 20].

If $M = \langle D, I, E \rangle$ is an intensional model, $a$ is an assignment for $D$, and $\varphi$ is a formula, we may alternatively write $M \models \varphi[a]$ for $V(a, \varphi) = 1$. In case $\varphi$ is a sentence it makes sense to write $M \models \varphi$ if $M \models \varphi[a]$ for some $a$. The following proposition lists some unsurprising but useful facts.

**Proposition 1.** Let $M = \langle D, I, E \rangle$ be an intensional model, and let $a$ be an assignment for $D$. Then, for all $\varphi, \psi, A, B$ and $B'$ of appropriate types,

1. $V(a, \varphi \rightarrow \psi) = 0$ iff $V(a, \varphi) = 1$ and $V(a, \psi) = 0$;
2. $V(a, \forall x_\alpha \varphi) = 1$ iff $V(a[d/x], \varphi) = 1$ for all $d \in D_\alpha$;
3. $V(a, (\lambda x.A)B) = V(a, A\{x := B\})$, if $B$ is free for $x$ in $A$;
4. If $V(a, A = B) = 1$ then $V(a, A \sqsubseteq B) = 1$;
5. $V(a, A = A) = 1$;
6. If $V(a, B = B') = 1$ then $V(a, A\{x := B\} = A\{x := B'\}) = 1$, provided $B$ and $B'$ are free for $x$ in $A$.

**Proof.** Left to the reader. □

Note that $\beta$-conversion preserves extensional identity, but that it does not necessarily preserve intensional identity, i.e. $(\lambda x_\alpha.A)B = A\{x := B\}$ is not necessarily true given the usual side condition. Similar remarks can be made about $\eta$-conversion and even about $\alpha$-conversion. Since it is not necessary to hardwire these principles into the logic, we have chosen not to do so. However, the principles can clearly be added to the logic by means of an axiomatic extension.

The last two statements in proposition 1 above show that $=$ is the usual congruence, but intensional models may still have the undesirable property that $=$ does not denote true identity of intension. This is an anomaly we want to get rid of. Intensional models are called normal just in case they have the desired property.

**Definition 8.** An intensional model $M = \langle D, I, E \rangle$ is normal if, for any type $\alpha$, any $d, d' \in D_\alpha$, and any $a$, $\langle d, d' \rangle \in V(a, \lambda x_\alpha \lambda x'_\alpha.x = x')$ implies $d = d'$. 22
That a restriction to normal intensional models does not buy us any new truths is shown by the next proposition. Its proof uses the Axiom of Choice unless $M$ is countable.

**Proposition 2.** Let $M$ be an intensional model. There is a normal intensional model $M'$ such that $M \models \varphi \iff M' \models \varphi$ for each sentence $\varphi$.

**Proof.** The proof is suppressed in the interest of space requirements. \qed

We can now define our semantic notion of consequence.

**Definition 9.** An intensional model $M$ for $L$ refutes a sequent $\Pi \Rightarrow \Sigma$ of $L$ if $M \models \varphi$ for all $\varphi \in \Pi$ and $M \not\models \varphi$ for all $\varphi \in \Sigma$. A sequent $\Gamma$ is i-valid if no intensional model for $L$ refutes $\Gamma$. $\Pi$ i-entails $\Sigma$, $\Pi \models_i \Sigma$, if $\Pi \Rightarrow \Sigma$ is i-valid.

### 4 Proof Theory

The following rules constitute a Gentzen sequent calculus for ITL. The usual notational conventions apply.

\[
\frac{\Pi \Rightarrow \Sigma \quad \Pi \subseteq \Pi', \Sigma \subseteq \Sigma'}{\Pi, \varphi \Rightarrow \Sigma, \varphi} \quad \text{[R]} \\
\frac{\Pi \Rightarrow \Sigma}{\Pi, \bot \Rightarrow \Sigma} \quad \text{[L]} \\
\frac{\Pi, A\{x := B\} \tilde{c} \Rightarrow \Sigma}{\Pi, (\lambda x. A)B\tilde{c} \Rightarrow \Sigma} \quad \text{[L]} \\
\frac{\Pi \Rightarrow \Sigma, A\{x := B\} \tilde{c}}{\Pi \Rightarrow \Sigma, (\lambda x. A)B\tilde{c}} \quad \text{[R]} \\
\text{if } B \text{ is free for } x \text{ in } A \\
\frac{\Pi, B\tilde{c} \Rightarrow \Sigma \quad \Pi \Rightarrow \Sigma, A\tilde{c}}{\Pi, A \subseteq B \Rightarrow \Sigma} \quad \text{[L]} \\
\frac{\Pi, A\tilde{c} \Rightarrow \Sigma, B\tilde{c}}{\Pi \Rightarrow \Sigma, A \subseteq B} \quad \text{[R]} \\
\text{if the constants } \tilde{c} \text{ are fresh}
\]

If $\Pi \Rightarrow \Sigma$ is a (finite or infinite) sequent then we say that $\Pi \Rightarrow \Sigma$ is provable, $\Pi \vdash \Sigma$, if there are finite $\Pi_0 \subseteq \Pi$ and $\Sigma_0 \subseteq \Sigma$ such that $\Pi_0 \Rightarrow \Sigma_0$ can be proved in this calculus. Clearly, the calculus should be sound:

**Theorem 3 (Soundness).** If a sequent $\Gamma$ is provable, $\Gamma$ is i-valid. Hence $\Pi \vdash \Sigma \implies \Pi \models_i \Sigma$

**Proof.** Left to the reader. (The proof involves some observations about the behaviour of intension functions when the language is extended.) \qed
5 Model Existence

In this section some elementary model theory will be developed, as a check on the definitions in section 3. The following definition is close to that of a “Hintikka set” in Smullyan [25] and Fitting [8, 9], but is also analogous to Schütte’s [24] semi-valuations.

**Definition 10.** A sequent $\Gamma$ of $\mathcal{L}$ is called a Hintikka sequent of $\mathcal{L}$ if the following hold:

1. $\{L: \varphi, R: \varphi\} \not\subseteq \Gamma$ for any sentence $\varphi$;
2. $L: \bot \notin \Gamma$;
3. $L: (\lambda x. A)BC \in \Gamma \implies L: A\{x := B\}C \in \Gamma$, if $\lambda x. A$, $B$, and the sequence of terms $C$ are closed and of appropriate type;
4. $R: (\lambda x. A)BC \in \Gamma \implies R: A\{x := B\}C \in \Gamma$, if $\lambda x. A$, $B$, and the sequence of terms $C$ are closed and of appropriate type;
5. $L: A \sqsubseteq B \in \Gamma \implies L: B\tilde{C} \in \Gamma$ or $R: A\tilde{C} \in \Gamma$, for all closed $A$, $B$ and sequences of closed $\tilde{C}$ of appropriate types;
6. $R: A \sqsubseteq B \in \Gamma \implies$ there are constants $\tilde{c}$ of appropriate types such that $\{L: A\tilde{c}, R: B\tilde{c}\} \subseteq \Gamma$.

A Hintikka sequent $\Gamma$ of $\mathcal{L}$ is said to be complete if $L: \varphi \in \Gamma$ or $R: \varphi \in \Gamma$, for each sentence $\varphi$ of $\mathcal{L}$.

Hintikka sequents are refuted by intensional models, as the following lemma shows. The intensional models constructed in its proof are closely akin to Andrews’ $V$-complexes.

**Lemma 4** (Hintikka Lemma). Each Hintikka sequent $\Gamma$ is refuted by an intensional model. If $\Gamma$ is complete, then $\Gamma$ is refuted by a normal countable intensional model.

**Proof.** The proof is suppressed in this version of the paper due to limitations of space. It follows the pattern pioneered by Takahashi and Prawitz.

In order to state the model existence theorem below, we need the notion of a provability property, a close relative of the concept of abstract consistency property (Smullyan [25]).

**Definition 11.** Let $\mathcal{P}$ be a set of sequents in the language $\mathcal{L}$. $\mathcal{P}$ is a provability property in $\mathcal{L}$ if $\mathcal{P}$ is closed under sequent rules, i.e. if $\Gamma \in \mathcal{P}$ whenever $\{\Gamma_1, \ldots, \Gamma_n\} \subseteq \mathcal{P}$ and $\Gamma_1, \ldots, \Gamma_n/\Gamma$ is a sequent rule.

A provability property $\mathcal{P}$ in $\mathcal{L}$ is sound if no $\Gamma \in \mathcal{P}$ is refuted by an intensional model for $\mathcal{L}$. 

24
Theorem 5 (Model Existence). Let \( L \) and \( C \) be languages such that \( L \cap C = \emptyset \) and each \( C_\alpha \) is denumerably infinite. Assume that \( P \) is a sound provability property in \( L \cup C \) and that \( \Gamma \) is a sequent in the language \( L \). If \( \Gamma \notin P \) then \( \Gamma \) is refuted by a countable normal intensional model.

Proof. The proof is skipped in the interest of space.

From model existence we can derive some nice corollaries. In the following \( \Gamma \) will always be a sequent in some language \( L \) while \( \Delta \) ranges over sequents in \( L \cup C \), where \( L \) and \( C \) are as in the formulation of Theorem 5.

Corollary 6 (Generalized Compactness). If \( \Gamma \) is i-valid then some finite \( \Gamma_0 \subseteq \Gamma \) is i-valid.

Proof. \( \{ \Delta \mid \text{some finite } \Delta_0 \subseteq \Delta \text{ is i-valid} \} \) is a sound provability property.

Corollary 7 (Generalized Löwenheim–Skolem). If \( \Gamma \) is not i-valid then \( \Gamma \) is refutable by a countable normal intensional model.

Proof. \( \{ \Delta \mid \Delta \text{ is i-valid} \} \) is a sound provability property.

Corollary 8 (Generalized Completeness). If \( \Gamma \) is i-valid then \( \Gamma \) is provable. Hence \( \Pi \models_i \Sigma \implies \Pi \vdash \Sigma \).

Proof. \( \{ \Delta \mid \Delta \text{ is provable} \} \) is a sound provability property.

Corollary 9 (Cut elimination). If \( \Pi, \varphi \vdash \Sigma \text{ and } \Pi \vdash \Sigma, \varphi \) then \( \Pi \vdash \Sigma \).

Proof. Use soundness and completeness.

6 A Linguistic Application

We now turn to a linguistic application of ITL and will develop the semantics of a tiny fragment of English containing verbs of propositional attitude. It will be shown that, given the present logic, it is consistent for an agent \( a \) to know that \( \varphi \) without knowing that \( \psi \), even if \( \varphi \) and \( \psi \) are co-entailing.

Before considering our special application, however, let us address the general point of axiomatic extensions of the base logic. In most applications one will like to work with a subclass of the class of intensional models that conform to some set of non-logical axioms \( S \). In that case one can define \( \Pi \models_S \Sigma \) to be \( S \cup \Pi \models_i \Sigma \), while \( \Pi \vdash_S \Sigma \) can be defined as \( S \cup \Pi \vdash \Sigma \). Soundness and generalized completeness immediately give that \( \Pi \models_S \Sigma \iff \Pi \vdash_S \Sigma \). Not all applications will instantiate \( S \) is the same way, but one set of axioms that immediately come to mind, and that we shall adopt here, are the usual principles of \( \lambda \)-conversion. We may add these by assuming that \( S \) contains all universal closures of instantiations of the following schemes.
Table 1: Some words and their translations

<table>
<thead>
<tr>
<th>WORD</th>
<th>TRANSLATION</th>
<th>WORD</th>
<th>TRANSLATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>( \lambda p \langle \rangle \rightarrow q )</td>
<td>man</td>
<td>( \text{man}_{(e)} )</td>
</tr>
<tr>
<td>no</td>
<td>( \lambda p \langle e \rangle \rightarrow q )</td>
<td>unicorn</td>
<td>( \text{unicorn}_{(e)} )</td>
</tr>
<tr>
<td>some</td>
<td>( \lambda p \langle e \rangle \rightarrow q )</td>
<td>runs</td>
<td>( \text{run}_{(e)} )</td>
</tr>
<tr>
<td>every</td>
<td>( \lambda p \langle e \rangle \rightarrow q )</td>
<td>laughs</td>
<td>( \text{laugh}_{(e)} )</td>
</tr>
<tr>
<td>loves</td>
<td>( \lambda Q \langle e \rangle \langle e \rangle \rightarrow q )</td>
<td>Bill</td>
<td>( \text{bill}_{(e)} )</td>
</tr>
<tr>
<td>is</td>
<td>( \lambda Q \langle e \rangle \langle e \rangle \rightarrow q )</td>
<td>Ann</td>
<td>( \text{ann}_{(e)} )</td>
</tr>
<tr>
<td>knows</td>
<td>( \lambda p \langle e \rangle \rightarrow q )</td>
<td>Tully</td>
<td>( \text{tully}_{(e)} )</td>
</tr>
<tr>
<td>believes</td>
<td>( \lambda p \langle e \rangle \rightarrow q )</td>
<td>Cicero</td>
<td>( \text{cicero}_{(e)} )</td>
</tr>
</tbody>
</table>

As soon as these schemes are added to the base logic, the result is full intensional identity of \( \beta \eta \) equivalent terms, i.e. \( \models_{\text{g}} A = B \) will hold if \( A =_{\beta \eta} B \).

For our linguistic application we will proceed along lines pioneered by Montague [16] and define a small fragment of English. The words of this fragment are given in Table 1, along with their translations into type logic. In these translations the terms \( \text{love}, \text{run}, \text{man}, \text{etc.} \) are constants of the types indicated, where \( e \) is the type of \( \text{entities} \). The set of syntactic structures is obtained by stipulating that all words in Table 1 are syntactic structures and that \( [XY] \) is a syntactic structure whenever \( X \) and \( Y \) are syntactic structures. Defining syntactic structures in this way leads to a lot of gibberish along with the structures we are interested in, but this is not important for present purposes. As long as the desired structures are there and get reasonable interpretations our aim is served.

Let us define the relation \( \leadsto \) ("translates as") between syntactic structures and terms as the smallest relation such that 1) \( X \leadsto A \) if \( X \) is a word and \( A \) is its translation in Table 1 and 2) if \( X \leadsto A \) and \( Y \leadsto B \) then \( [XY] \leadsto AB \) if \( AB \) is a well-formed term and \( [XY] \leadsto BA \) if \( BA \) is well-formed. This leaves open the possibility that a syntactic structure does not get a translation and indeed many do not. Structures \( X \) for which there is no \( A \) such that \( X \leadsto A \) are called uninterpretable and we have no interest in them.

Let us turn to some syntactic structures that are interpretable. In (1) below two are given, together with (the \( \beta \) normal forms of) their interpretations. Clearly, (1b), the interpretation of (1a), i-entails and is i-entailed by (1d), which is the interpretation of (1c).

(1) a. \([[[\text{no man}] \text{laughs}][\text{if}[[\text{some unicorn}] \text{runs}]]]\)

b. \( \exists x(\text{unicorn } x \land \text{run } x) \rightarrow \neg \exists x(\text{man } x \land \text{laugh } x) \)
c. \([[\text{no unicorn}\text{runs}][\text{if}][\text{some man}\text{laughs}]]\]

d. \(\exists x (\text{man } x \land \text{laugh } x) \rightarrow \neg \exists x (\text{unicorn } x \land \text{run } x)\)

This does not mean however that (1b) and (1d) are identical in all intensional models, as nothing excludes the possibility that \(I(a, (1b)) \neq I(a, (1d))\) for some intension function \(I\). It follows that the two structures in (2) are not co-entailing.

(2) a. \([[\text{every man}]\text{knows}[[\text{no man}\text{laughs}][\text{if}][\text{some unicorn}\text{runs}]]]]\]

b. \(\forall y (\text{man } y \rightarrow \text{know } y (\exists x (\text{unicorn } x \land \text{run } x) \rightarrow \neg \exists x (\text{man } x \land \text{laugh } x)))\)

c. \([[\text{every man}]\text{knows}[[\text{no unicorn}\text{runs}][\text{if}][\text{some man}\text{laughs}]]]]\]

d. \(\forall y (\text{man } y \rightarrow \text{know } y (\exists x (\text{man } x \land \text{laugh } x) \rightarrow \neg \exists x (\text{unicorn } x \land \text{run } x)))\)

Suppose that \(c\) is some constant of type \(e\). Then \(\text{know } c (1b) \Rightarrow \text{know } c (1d)\) is in fact a Hintikka sequent and is therefore refuted by an intensional model (addition of \((\alpha), (\beta)\) and \((\eta)\) does not change this). This intensional model can also be used to show that (2b) does not entail (2d). This is as desired, for even if (2a) holds there may well be a man who has not managed to draw the inference necessary to arrive at (1c). We have thus shown that the logic avoids the problem of logical omniscience in the sense that it does not exclude the possibility that a person knows one thing but fails to know another thing logically equivalent with it. Essential use was made of the failure of Extensionality in our logic ITL: terms of complex type can have the same extensions, even in all intensional models, without necessarily having the same intension.

This distinction between extension and intension does not extend to terms of basic type however and this raises the question how names are to be dealt with. If they are treated straightforwardly using constants of type \(e\) (e.g. \(b_c\), or in the present context preferably \(\lambda P.Pb\), for ‘Bill’) we run into the standard problems of the ‘Cicero–Tully’ or ‘Hesperus–Phosphorus’ kind. However, there are many reasonable translations that do not directly equate names with type \(e\) constants. The translations in Table 1, that send names to constants of the quantifier type \(\langle \langle e \rangle \rangle\), may serve as an example, provided some meaning postulates (additions to \(S\)) like the following are adopted.

(3) a. \(\forall P (\text{ann } P \leftrightarrow Pa)\)

b. \(\forall P (\text{bill } P \leftrightarrow Pb)\)

c. \(\forall P (\text{tully } P \leftrightarrow Pt)\)

d. \(\forall P (\text{cicero } P \leftrightarrow Pc)\)
The structure [Tully runs] translates as *tully run*, but given the meaning postulates just introduced, this is equivalent with *run t*. Similarly, [Cicero runs] translates as *cicero run*, equivalent with *run c*. And since [Tully[is Cicero]] is translated as *tully*(λx.cicero(λy.x = y)), which is equivalent with *t = c*, it is readily explained why the argument *Tully runs, Tully is Cicero, therefore Cicero runs* holds. But this reasoning essentially depends on extensional equivalence and therefore will not go through once propositional attitudes enter the picture. Consider the structure [Ann[believes[Tully runs]]]. It translates as *ann*(λx.belong x(*tully run*)) and this is equivalent with *belong a (*tully run*), while *belong a (*cicero run*) is equivalent with the translation of [Ann[believes [Cicero runs]]]. However, there is no co-entailment between these sentences, even in the presence of the postulates in (3) and the translation of [Tully[is Cicero]].

This shows that even for names the sense/reference distinction can be captured in this logic, provided one is willing to treat names with the help of predicates (Quine’s ‘primacy of predicates’ comes to mind). Treating them as being of type *(⟨e⟩)*, as we have done here, is one possible strategy. There may be others.

The present application of our intensional type theory to linguistic semantics has avoided the concept of possible worlds altogether, as it was not needed in order to illustrate our points. However, as possible worlds are obviously extremely useful for the analysis of a range of natural language constructions (though not for the true intensionality we have been concerned with in this paper), one might well want to combine them with the present approach. Muskens [20, chapter 4] gives a translation of what is essentially the fragment of Montague [16] into a two-sorted relational type theory, with possible worlds providing an additional basic type. Although the type theory in [20] validates Extensionality, its language essentially is the language employed here, so that the translation can also serve as a translation into ITL. A minor variation will treat names as they are treated above.

7 Conclusion

In this paper we have introduced an abstract and simple notion of intensional model that is a generalization of Henkin’s general models. Its definition does not involve concepts that have no immediate intuitive justification, such as the “abstraction designation functions” of Fitting [9] or the “application operators” of Benzmüller et al. [2]. These operators provide generalized, non-standard notions of abstraction in one case and of application in the other, but seem to have no justification other than a purely technical one. The present approach, in contrast, gives a kind of minimal logic of intension and extension, with ingredients that well-nigh *any* logic of intension and extension seems to need. Models are inhabited by intensions, a function *I* sends terms to their intensions and
functions $E_{\alpha}$ send intensions to the extensions they determine. If an additional requirement should be made that the $E_{\alpha}$ be injective, one essentially obtains Henkin’s general models, if, moreover, the $E_{\langle\alpha_1,\ldots,\alpha_n\rangle}$ should be required to be onto $\mathcal{P}(D_{\alpha_1} \times \cdots \times D_{\alpha_n})$, standard models are obtained.

The logic contrasts with other approaches to (hyper-)intensionality in two ways. Firstly, unlike other approaches, such as Thomason’s Intentional Logic (Thomason [28], see also Muskens [21]) or Property Theory (Turner [30], Chierchia and Turner [4], Fox and Lappin [10], Fox et al. [11]), the aim is not to set up a new logic, but to provide existing classical type theory with a wider class of models in order to invalidate an unwanted axiom. Secondly, the logic is agnostic about what intensions are. To this question various answers have been given and there are proposals for intensions as structured meanings (Carnap [3], Lewis [14], Cresswell [7]), intensions as sets of possible and impossible worlds (Montague [15], Cresswell [6], Hintikka [13], Muskens [19], Zalta [32]), intensions as constructions (Tichý [29]), and intensions as algorithms (Moschovakis [17], Muskens [21]), but here we have only provided an abstract characterization of the notion of intensionality. We have, in other words, focused on the logic rather than on the ontology of intensions.

References


Abstract

Higher order grammar (HOG) is a linguistic formalism that aims to combine the advantages of existing constraint-based formalisms (such as HPSG) and proof-theoretic ones (such as categorial grammar) by using higher-order logic (HOL) as the description language; the underlying intuitionistic type system plays a role analogous to that of a categorial type logic, while the classical higher-order term logic serves to impose constraints (analogous to the role played by RSRL in HPSG). Here we focus on semantics, showing how the use of a HOL with definable subtypes leads to a novel and surprisingly straightforward solution of the notorious granularity problem about natural-language (NL) meanings. We also call attention to a hitherto unnoticed problem in standard approaches to NL semantics having to do with nonprincipal ultrafilters and show why it does not arise under our proposal. The two main technical innovations that make the proposal work are (1) axiomatization of NL entailment as a preorder (as opposed to an order) on the set of (primitive) propositions, and (2) definition of the set of worlds as a certain subset of the powerset of the set of propositions.

1 Background on Higher Order Grammar

Higher Order Grammar (HOG, [19, 18, 17]) is a framework for linguistic theory developed by the author together with Jirka Hana since early 2001. It belongs to a loose assemblage of approaches to natural language that might be called multistratal type-theoretic approaches, which also include Abstract Categorial Grammar (ACG, [6]), Lambda Grammar [15], and Grammatical Framework [20, 13], all of which emerged independently around the turn of the millenium. Shared characteristics of these approaches include: distinguishing between tectogrammar (abstract syntax) and phenogrammar (concrete syntax); separate type theories each with its own Curry-Howard proof term calculus, for each
of (a) tectogrammar, (b) phenogrammar, and (c) meaning; and the interpretation of signs (tectostructures) into concrete linguistic forms (phenostructures) and meanings via structure-preserving translations of the term calculi.

What distinguishes HOG from the other approaches in this family is the use of full intuitionistic propositional logic for the type logics rather than linear logic, more specifically a bivalent boolean version of Lambek and Scott’s higher-order categorical logic \[12\]. This logic differs from the more familiar higher-order logics in the Church-Henkin-Montague tradition in providing for lambda-definable subtyping, which plays a central role in this paper. Set-theoretic models of theories in this kind of logic are just like the familiar Henkin models, but augmented with cartesian products and lambda-definable subsets. The simplicity and familiarity of such models makes this kind of logic accessible and practical for working linguistic semanticists. However, there are more general categorical models (bivalent boolean toposes), which make allowance for the possibility of uninhabited types (i.e. types other than the empty (counit) type for which there are no closed terms) should the need arise; and the boolean condition is easily dropped should one wish to experiment with intuitionistic theories of linguistic meaning.

2 Two Problems in Linguistic Semantics

The subtyping facility of the underlying logic has application to all levels of linguistic description, but in this paper, we limit attention to the HOG semantic theory, presenting first the underlying lambda calculus and its extension to a classical bivalent predicate logic (section 2), followed by an exposition of the semantic theory itself (section 3). The central notion of the theory is that of hyperintensions\(^1\), mathematical models of Fregean senses of a finer granularity than the familiar intensions (functions to extensions from worlds, where the worlds in turn are theoretical primitives) of mainstream Carnap/Montague-inspired semantics (hereafter called the standard theory, with which we assume general familiarity).

To illustrate the power of the theory, we show how it solves two perplexing problems of the standard theory, one long-standing and notorious, the other apparently heretofore unrecognized. The notorious one is the so-called granularity problem, which we briefly review in section 2.1. Although this problem has generated a vast literature, none of the solutions proposed so far has gained widespread acceptance; we believe the solution provided here to be unprecedently straightforward, accessible, and conservative (in the sense of preserving as much as possible of what is good about the standard theory). The new problem, which involves nonprincipal ultrafilters, will be introduced in section 2.2.

\(^{1}\)The theory improves on an earlier effort to formulate a simply-typed hyperintensional semantic theory \([3]\).
2.1 The Granularity Problem

The granularity problem is that in the standard theory there are not enough intensions to account for a wide range of robust intuitions about NL entailments. We illustrate with three examples.

(1) Heperus and Phosphorus

a. (The ancients realized that) Hesperus was Hesperus.

b. (The ancients realized that) Hesperus was Phosphorus.

(2) Frege’s analysis of Hesperus and Phosphorus:

1. The sense expressed by an expression depends on the senses expressed by its parts.

2. Although they have the same reference, the names *Hesperus* and *Phosphorus* express different senses.

3. Hence the sentences in (1)a-b express different propositions, so it is unsurprising that the ancients believed one but not the other.

(3) Hesperus and Phosphorus in the Standard Theory

1. The meanings of *Hesperus* and *Phosphorus* are functions from worlds to entities.

2. Assuming rigidity of names [11], they are constant functions.

3. At at least one world, both functions take the value Venus, and so they are the same constant function.

4. So *Hesperus* and *Phosphorus* mean the same thing, and consequently (*pace* Frege) the sentences in (1)a-b express the same proposition.

(4) Woodchucks and Groundhogs

a. Phil is a woodchuck.

b. Phil is a groundhog.
(5) **Woodchucks and Groundhogs in the Standard Theory**

1. Standard-Theory Meaning Postulate:

\[ \forall_{w \in \text{World}} \forall_{i \in \text{Ind}} (\text{woodchuck}(i)(w) \leftrightarrow \text{groundhog}(i)(w)) \]

2. By HOL, woodchuck = groundhog

3. Therefore (i) and (ii) express the same proposition:

   i. Jim believes Phil is a groundhog.

   ii. Jim believes Phil is a woodchuck.

(6) **Paris Hilton and the Riemann Hypothesis**


b. All nontrivial zeros of \( \zeta \) have real part 1/2.

(7) **Background for Paris Hilton and Riemann**

1. (6)b is the Riemann Hypothesis, the most famous unresolved conjecture in all mathematics.

2. For a declarative sentence R, to know whether R is to know that R (if R is true) or to know the denial of R (if R is false).

(8) **Paris Hilton and Riemann in the Standard Theory**

1. There is only one necessary truth, so whichever of (6)b and its denial is true expresses the same proposition as (6)a.


3. So if (6)b is true, then Paris Hilton knows that all nontrivial zeros of \( \zeta \) have real part 1/2.

4. And if (6)b is false, then Paris Hilton knows that not all nontrivial zeros of \( \zeta \) have real part 1/2.

5. Hence, Paris Hilton knows whether all nontrivial zeros of \( \zeta \) have real part 1/2.
As pointed out to the author and Shalom Lappin by Howard Gregory (p.c.),
the source of the granularity problem is the antisymmetry of entailment as
modelled in the standard theory, which arises directly from the modelling of
propositions as sets of worlds (which themselves are ontological primitives â
la Montague [14] and the later Kripke [10]) and of entailment as the subset
inclusion relation on the powerset of the set of worlds. There is simply no
getting around the fact that the inclusion relation on a powerset is an order,
and therefore antisymmetric.

2.2 Nonprincipal Ultrafilters

The second problem with the standard theory seems to have been overlooked so
far. By way of background, there is another tradition, one which is both intu-
itively satisfying and of long standing [1, 9, 8, 21] of constructing worlds (or
equivalently, maximal consistent sets of propositions) as ultrafilters (i.e. meet-
closed and upper-closed proper subsets which for each proposition \( p \) contain
either \( p \) or its complement) over the boolean algebra of propositions. That is,
one takes the propositions as primitive and constructs worlds out of them rather
than the other way around as in the standard theory. Now if the boolean algebra
is finite, the two approaches coincide. This is because in a finite boolean algebra,
a subset is an ultrafilter iff it is a principal filter generated by an atom; in the
special case where the boolean algebra is the power set of the set of worlds, the
atoms, namely singleton sets of worlds, (and therefore the “constructed worlds”
themselves) are in one-to-one correspondence with the primitive worlds. How-
ever, if the ambient set theory has Choice, it is well known that any infinite
boolean algebra can be shown to have a nonprincipal ultrafilter. Since there are
uncontroversially infinitely many propositions, it follows (assuming Choice) that
there are some ways things might be (maximal consistent sets of propositions)
which are not being taken into account by limiting the domains of intensions
(as functions) to the set of primitive worlds.

2.3 Our Proposed Solution to Both Problems

The essence of our solution is (1) to follow the tradition of constructed worlds,
thereby eliminating the problem of excluding the nonprincipal ultrafilters, while
(2) not imposing antisymmetry on the algebra of propositions, thereby elimi-
nating the pernicious identification of mutually entailing propositions. In short:

(9) The Essence of our Proposal

1. Propositions are primitives;

2. they form a boolean preordered algebra (hereafter, boolean prealge-
   bra) preordered by entailment;
3. possible worlds are just the ultrafilters; and
4. ‘p is true in w’ just means \( p \in w \).

Here, by boolean prealgebra we mean a prelattice which satisfies all the conditions to be a boolean algebra but with equalities replaced by the prelattice equivalence relation (mutual entailment. where entailment is the preorder.) The axioms for such structures will be given below. The most familiar example of a boolean prealgebra is the set of formulas of classical propositional or predicate logic preordered by logical consequence.

Obviously the Stone Representation Theorem for boolean algebras cannot generalize to boolean prealgebras, since powerset algebras are antisymmetric. However, the principal lemma Stone used to prove the Representation Theorem does generalize, as does the following corollary, which will play a key role for us:

(10) Stone’s Lemma (There are Enough Ultrafilters)

If \( p \) and \( q \) are elements of a boolean prealgebra and \( p \not\equiv q \), then there is an ultrafilter \( w \) such that \( p \in w \) but \( q \not\in w \).

(11) Corollary (Boolean Equivalence and Ultrafilters)

If \( p \) and \( q \) are elements of a boolean prealgebra, then \( p \equiv q \) iff for every ultrafilter \( w \), \( p \in w \) iff \( q \in w \).

3 The Underlying Logic

We start with a standard simply typed lambda calculus with pairing\(^2\). But instead of Church’s [2] or Henkin’s [7] two basic types (which we will call Ent (entities) and Bool (truth values) or Gallin’s [4] three (Ent, Bool, and World), we have four: Ent, Bool, Ind (individual concepts, the things that have entities as extensions), and and Prop (propositions, the things that have truth values as extensions). Crucially, World is not a basic type!

Following a general strategy initiated by Church and adapted to a more general (categorical) setting by Lambek and Scott, we extend our lambda calculus to a classical higher-order predicate logic. The key points of this extension are as follows: First, we add equality constants \( =_A : (A \times A) \Rightarrow \text{Bool} \) and treat the usual lambda term equivalences as object-language axioms about equality. The usual (intuitionistic) logical constants (including the truth values true and false)

\(^2\)I.e. the type logic is positive intuitionistic propositional logic, so the type constructors are \( 1, \times, \Rightarrow \), and the term constructors include pairing and left and right projections in addition to the usual application and abstraction.
are then defined in terms of equality and lambda abstraction. These become classical with the addition of:

(12) **Axiom of Excluded Middle**
\[ \vdash \forall t \in \text{Bool} (t \lor \neg t) \]
We also need the following axiom, explicitly rejected by Church, which was added by Henkin (for completeness with respect to Henkin models) and by Lambek and Scott (for completeness with respect to the more general categorical (topos) models):

(13) **Axiom of Boolean Extensionality**
\[ \vdash \forall (x,y) \in \text{Bool} \times \text{Bool} [(x \leftrightarrow y) \rightarrow (x = y)] \]

Of crucial importance is the machinery for handling subtyping:

(14) **Subtypes and Characteristic Functions**
1. We have one more way of forming types: if \( a :: A \Rightarrow \text{Bool} \), then \( A_a \) is a type (intuitively: the subtype of \( A \) whose members satisfy the predicate \( a \));

2. We have one more way of forming terms: if \( a :: A \Rightarrow \text{Bool} \) is closed, then \( \text{ker}_a :: A_a \Rightarrow A \) (intuitively: the embedding of \( A_a \) into \( A \)); and

3. we have one further axiom schema
\[ \vdash \forall (x,a) \in A \times (A \Rightarrow \text{Bool}) \left( a(x) \leftrightarrow \exists y \in A_a x = \text{ker}_a(y) \right) \]
(Intuitively: \( a \) is the characteristic function of \( A_a \).)

The following axiom ensures that \text{true} and \text{false} are distinct:

(15) **Nondegeneracy**
\[ \vdash \neg (\text{true} = \text{false}) \]

And finally, we must ensure that \text{true} and \text{false} are the only truth values. In the presence of the axioms already imposed, this condition, known as **bivalence**, can be shown [12, 5] to be equivalent to the following:
(16) Disjunctivity

For all boolean terms $\phi$ and $\psi$, if $\vdash \phi \lor \psi$, then $\vdash \phi$ or $\vdash \psi$.

The categorical models of this kind of logic are the bivalent boolean toposes. Special cases of these, called well-pointed toposes, are (up to isomorphism) just the familiar Henkin models [7] with some extra structure (finite cartesian products and lambda-definable subsets).\(^3\)

4 A Hyperintensional Semantic Theory

4.1 Hyperintensional Meanings

Our theory starts by introducing types for hyperintensions, which replace intensions as our models of Fregean senses:

(17) Meanings are Hyperintensions (not Intensions)

The set of hyperintensional types is defined as follows:

1. 1 is a hyperintensional type;
2. Ind and Prop are hyperintensional types;
3. If $A$ and $B$ are hyperintensional types, so are $A \times B$ and $A \Rightarrow B$
4. If $a :: A \Rightarrow \text{Bool}$ is closed and $A$ is a hyperintensional type, so is $A_a$.
5. Nothing else is a hyperintensional type.

The corresponding extensional types are defined as follows:

(18) Extensional types corresponding to hyperintensional types\(^4\)

1. $\text{Ext}(1) = \text{def} 1$;
2. $\text{Ext}(\text{Ind}) = \text{def} \text{Ent}$;
3. $\text{Ext}(\text{Prop}) = \text{def} \text{Bool}$;

\(^3\)We remain agnostic for the time being as to whether the well-pointed toposes suffice for applications to NL semantics. But for familiarity, we will speak of the models as if they were well pointed, e.g. ‘set’ for ‘object’, ‘subset’ for ‘subobject’, ‘function’ for ‘arrow’, ‘preorder’ for ‘internal preorder object’, etc.

\(^4\)The definition of corresponding extensional type for a lambda-definable subtype of a given hyperintensional type is omitted here because of space limitations, but see [16].
4. \( \text{Ext}(A \times B) = \text{def} \, \text{Ext}(A) \times \text{Ext}(B); \) and

5. \( \text{Ext}(A \Rightarrow B) = \text{def} \, A \Rightarrow \text{Ext}(B) \)

In NL semantics, our central object of study is the entailment relation between propositions (declarative sentence meanings). This we axiomatize as follows:

(19) Axiomatizing the Entailment Relation

1. The entailment symbol is an object-language basic constant:

\[ \models:: (\text{Prop} \times \text{Prop}) \Rightarrow \text{Bool} \]

2. Entailment is an (internal) preorder (here \( p, q, r :: \text{Prop} \)):

   (a) \( \proves \forall_p (p \models p) \)

   (b) \( \proves \forall_{(p, q, r)} (p \models q) \rightarrow ((q \models r) \rightarrow (p \models r)) \)

3. Propositional equivalence is defined as mutual entailment:

\[ \equiv = \text{def} \lambda_{(p, q)} ((p \models q) \land (q \models p)) \]

4. Crucially, \( \equiv \) cannot be proven equal to \( =_{\text{Prop}} \).

5. Cf. (13), which says \( \leftrightarrow \) is equal to \( =_{\text{Bool}} \).

Next we introduce the constants used to translate English logic words, and suitable meaning postulates for them:

(20) Translations of English “Logic Words”

1. \text{truth} :: \text{Prop} abbreviates the translation of an arbitrarily chosen necessarily true English sentence.

2. \text{falsity} :: \text{Prop} abbreviates the translation of an arbitrarily chosen necessarily false English sentence.

3. \text{not’} :: \text{Prop} \Rightarrow \text{Prop} abbreviates the translation of \text{it is not the case that}.

4. \text{and’}, \text{or’} :: (\text{Prop} \times \text{Prop}) \Rightarrow \text{Prop} are the respective translations of (the sentential conjunctions) \text{and} and \text{or}.

5. \text{if’ . . . then’} translates the discontinuous sentential conjunction \text{if . . . then}. 

40
Our next set of axioms says that in a model, the interpretation of the type \( \text{Prop} \) forms a boolean prealgebra with the meanings of the logic words as the operations.\(^5\)

(21) The Entailment Preorder is a Boolean Prealgebra

1. \( \vdash \forall_p (p \models \text{truth}) \)
2. \( \vdash \forall_p (\text{falsity} \models p) \)
3. \( \vdash \forall_{(p,q)} ((p \land' q) \models p) \)
   \( \vdash \forall_{(p,q)} ((p \land' q) \models q) \)
4. \( \vdash \forall_{(p,q)} [(p \models q) \land (p \models r) \rightarrow (p \models (q \land' r))] \)
5. \( \vdash \forall_{(p,q)} (p \models (p \lor' q)) \)
   \( \vdash \forall_{(p,q)} (q \models (p \lor' q)) \)
6. \( \vdash \forall_{(p,q,r)} [(p \models q) \land (q \models r) \rightarrow ((p \lor' q) \models r)] \)
7. \( \vdash [(\text{if}' p \text{ then}' q) \land' p) \models q] \)
8. \( \vdash \forall_{(p,q,r)} [((r \land' p) \models q) \rightarrow (r \models (\text{if}' p \text{ then}' q))] \)
9. \( \vdash \forall_p ((\text{not}' p) \equiv (\text{if}' p \text{ then}' \text{falsity})) \)
10. \( \vdash \forall_p [(\text{not}' (\text{not}' p)) \models p] \)

4.2 Constructed Worlds

In order to conduct the usual semantic business with worlds (modality, counterfactuals, the taking of extensions at worlds, etc.), we need to have worlds in the theory. This might seem problematic, since we have no basic type for them. However, the existence of lambda-definable subtypes comes to our rescue, as follows:

(22) Constructing the Type of Worlds

1. In a model, worlds should be certain sets of propositions, so together they are a subset of the set that interprets \( \text{Prop} \Rightarrow \text{Bool} \).
2. More specifically: they should be the set of ultrafilters of the boolean prealgebra that interprets \( \text{Prop} \).

\(^5\)The first eight of these say that the propositions form an (internal) bicartesian-closed preorder (i.e. a heyting prelattice object), and the last two say that every proposition is equivalent (\textit{not} equal) to its own double negation.
3. So far this is just a set-theoretic construction on models, but we can internalize it by defining the type World to be the subtype

$$[\text{Prop} \Rightarrow \text{Bool}]_u$$

where $u :: (\text{Prop} \Rightarrow \text{Bool}) \Rightarrow \text{Bool}$ is the predicate on sets of propositions such that $u(s)$ says of $s$ that it is an ultrafilter.

This is possible because ultrafilterhood is a definable predicate of sets of propositions.

(23) Being an Ultrafilter is a Lambda-Definable Predicate:

$u$ is $\lambda_s[a(s) \land b(s) \land c(s)]$ where

1. $a(s)$ is $\forall_{(p,q)}[(s(p) \land p \models q) \rightarrow s(q)];$
2. $b(s)$ is $\forall_{(p,q)}[(s(p) \land s(q)) \rightarrow s(p \land q)];$ and
3. $c(s)$ is $\neg s(\text{falsity'}) \land \forall_p(s(p) \lor s(\text{not'} p)).$

Here $a(s)$ says $s$ is upper-closed (i.e. closed under entailment; $b(s)$ says $s$ is closed under binary least upper bounds (i.e. closed under propositional conjunction), and $c(s)$ says $s$ excludes necessarily false propositions and “settles every issue” (for each proposition, contains either that proposition or its negation).

We are now in a position to say what it means for a proposition to be true at a world.

(24) How to Say “$p$ is True at $w$”

1. In the standard approach: $p(w)$
2. Under our proposal: first guess would be $w(p)$, but this is ill-typed since $w :: \text{World}$, not $w :: \text{Prop} \Rightarrow \text{Bool}$.
3. But World = $[\text{Prop} \Rightarrow \text{Bool}]_u$ where $u$ is defined as in (23), so $\text{ker}_u :: \text{World} \Rightarrow (\text{Prop} \Rightarrow \text{Bool})$ denotes the embedding of the set of worlds into the set of sets of propositions.
4. So the right way to say “$p$ is true at $w$” is $\text{ker}_u(w)(p)$.
5. For this reason, we will usually abbreviate $\text{ker}_u(w)(p)$ to $p@w$.

Now that we have worlds, we can say what it means for something to be the extension of a given meaning (hyperintension) at a given world. The obvious move here is to treat the notion of extension as a family of functions (parametrized by the set of hyperintensional types) from hyperintension-world
pairs to other things. But we want to take into consideration the possibility that some meanings (e.g. meanings of names of fictional beings) may lack extensions at some worlds. To this end, we introduce instead a similarly parametrized family of realization functions from hyperintension-extension pairs to propositions, and then say that a meaning \( a \) has extension \( e \) at world \( w \) just in case the proposition that \( e \) realizes \( a \) is true at \( w \). (Remember that we can’t just “evaluate” \( a \) at \( w \), because our meanings are hyperintensions, not intensions!)

(25) Realization of a Hyperintension by an Extension

1. We introduce constants \( \text{realizes}_A :: (\text{Ext}(A) \times A) \Rightarrow \text{Prop} \) for each hyperintensional type \( A \). The intuition is that \( \text{realizes}(e, a)@w \) means \( e \) is the extension of \( a \) at \( w \) (if it has any there!).

2. Since presumably a meaning has at most a single extension at any world, we posit the axiom schema:

\[
\forall_{(w, a, e, f)} [(\text{realizes}(e, a)@w \land \text{realizes}(f, a)@w) \rightarrow (e = f)]
\]

3. The extension of a proposition \( p \) at a world \( w \) should be true iff \( p@w \), and so we posit the axiom:

\[
\forall_{(w, p, t)} \in \text{World} \times \text{Prop} \times \text{Bool} [\text{realizes}(t, p)@w \leftrightarrow (p@w \leftrightarrow t)]
\]

4. In general, a hyperintension need not have an extension at every world.

5. But propositions must, since worlds are ultrafilters.

One way to clarify the connection between our approach and the standard one is to introduce the simplifying assumption that every meaning has an extension at every world. (Of course this is the case on the standard approach, since then a meaning is a total function whose domain is the set of worlds.) Then analogs of some fairly standard assumptions about how extensions should work are captured by the following axiom schemata. Note that here \( \pi \) and \( \pi' \) denote the left and right projection functions on pairs.

(26) Simplifying Assumption: Meanings have an extension at every world

Then extensionalization can be treated as a (type-parametrized) function

\[
\text{ext}_A :: A \Rightarrow (\text{World} \Rightarrow \text{Ext}(A))
\]

that maps hyperintensions to the corresponding intensions (functions from worlds to extensions) subject to the following axioms:
1. $\vdash \forall w, a, e \left[ \left( \text{ext}(a)(w) = e \right) = \text{realizes}(a, e)@w \right]$
2. $\vdash \forall w, p \left( \text{ext}_{\text{Prop}}(p)(w) = p@w \right)$
3. $\vdash \text{ext}_1(w)(*) = *$
4. $\vdash \forall w, c \left( \text{ext}_{A \times B}(c)(w) = \left( \text{ext}_A(\pi(c))(w), \text{ext}_B(\pi'(c))(w) \right) \right)$
5. $\vdash \forall w, f \left( \text{ext}_{A \Rightarrow B}(f)(w) = \lambda x \in A \text{ext}_B(f(x))(w) \right)$

Unfortunately space and time considerations do not permit inclusion of illustrative linguistic examples; see [16] for brief discussion of extensional properties, generalized determiners, questions, and S5 modalities.

We conclude by considering what it means (still making the simplifying assumption that all meanings, not just propositions, have an extension at every world) to say that two meanings are equivalent.

(27) Equivalence of Hyperintensions

Two closed hyperintensional terms $a, b$ are equivalent iff Ext maps them to the same intension (or, equivalently, if they have the same extension at every world), i.e. $\text{ext}(a) = \text{ext}(b)$.

1. This generalizes the notion of equivalence for propositions.
2. Examples are (the meanings of):
   
   (a) *Hesperus* and *Phosphorus*
   
   (b) *woodchuck* and *groundhog*
   
   (c) *Paris Hilton is* *Paris Hilton* and whichever is true, the *Riemann Hypothesis* or its denial

3. Nothing in the theory enables us to prove equality for any of the pairs.
4. The essence of the Granularity Problem is that Ext is not injective.

To summarize: we propose an axiomatic theory of NL meaning within a simply-typed two-valued classical predicate logic. The set-theoretic models are the familiar Henkin models augmented with cartesian products and definable subtypes. The theory incorporates straightforward solutions to the notorious granularity problem as well as the heretofore unnoted yet perplexing problem of nonprincipal ultrafilters. We make no recourse to untyped lambda calculus, polymorphic typing, partial possible worlds, impossible worlds, giving up one or more of Gentzen’s structural rules, giving up possible worlds, or accepting (as the standard theory requires) that Paris Hilton knows whether the Riemann hypothesis is true.
References


Polymorphism in English Logical Grammar

Elias Ponvert
The University of Texas at Austin
efp@uts.cc.utexas.edu

Abstract

The quantifiers and connectives of English present difficulties to standard (Church-style) type theory in linguistics in that they can apply systematically to objects of different form. This paper sketches solutions to this problem with some extensions to the simply typed lambda calculus, including Damas-Milner types and System F or the polymorphic lambda calculus. The goal is to maintain the “proofs-as-readings” spirit of the logical grammar tradition while allowing flexible types for the logical terms.

1 Background and Outline

Type theory in linguistics grew out of the categorial investigations of Bar-Hillel [1], Lambek [18, 19, 20] and Curry [8]. Later, Montague’s PTQ semantics [21] utilized a simple Church-style type system to accompany the simple applicative categorial grammar used to define the fragment of English under investigation. This unification and the rediscovery of Lambek’s work led van Benthem [3, 4] and many others to develop the tradition of type-logical grammar. In this tradition (substructural) fragments of propositional intuitionistic logic are used to characterize both natural language (NL) syntax and as the type-theory accompanying lambda terms used to describe NL denotations. The easily defined (injective) functors from the associative syntactic calculus (often Lambek’s L) to the type theory (often Lambek’s associative syntactic calculus with permutations, LP, or the full positive propositional intuitionistic calculus J[→+]) became the generalized format of Montague’s rule-to-rule notion of compositionality. Also, implicitly or explicitly, type-logical grammar was inspired by the Curry-Howard-de Bruijn isomorphism between typed lambda terms and formulae of intuitionistic logic—the tradition characterized by the slogan “formulae as types, proofs as terms”. Applied in the setting of NL semantics, and the above theories of LP and J[→+], the isomorphism is relaxed to a correspondence [22, p.116].
Within the Montagovian tradition, Partee and Rooth [23] addressed the problem that the boolean connectives “and” and “or” in English induce the same basic intuition or reading irrespective of the types of arguments, e.g.

(1) Serge and Jane sing ≈ Serge sings and Jane sings
(2) Serge sings and smokes ≈ Serge sings and Serge smokes

(3) One or two girls are in the studio ≈
One girl is in the studio or two girls are in the studio

The monomorphic theory adopted by Montague and the type-logical tradition that followed did not have the expressive power to characterize polymorphic terms like connectives. Partee and Rooth sketched a solution in the following way: let $CT$ be the set of conjoinable types, that is, the set that satisfies

\[ CT ::= t | T \rightarrow CT \]

where $t$ is the type of propositions and $t \rightarrow u$ is the type of functional terms from $t$ to $u$. Thus,

\[ \phi \text{ and } \psi = \begin{cases} \phi \land \psi & \text{if } \phi \text{ and } \psi \text{ are of type } t \\ \lambda x^A (\phi \cdot x) \text{ and } (\psi \cdot x) & \text{if } \phi \text{ and } \psi \text{ are of type } A \rightarrow B \end{cases} \]

\[ \phi \text{ or } \psi = \begin{cases} \phi \lor \psi & \text{if } \phi \text{ and } \psi \text{ are of type } t \\ \lambda x^A (\phi \cdot x) \text{ or } (\psi \cdot x) & \text{if } \phi \text{ and } \psi \text{ are of type } A \rightarrow B \end{cases} \]

Along similar lines, more recently several authors [10, 11, 4] have noted that quantifiers like “every”, “one”, “two” etc. exhibit similarly polymorphic tendencies. Consider\(^1\):

(7) A present pleases every child. [4, 11]
(8) Every time Serge smokes, women swoon.
(9) Serge only sings for fun.
(10) Only Serge sings for fun.
(11) Serge sings only for fun.

The aim of this paper is to provide interpretations for these and other terms, that are both concise (ideally postulating a single interpretation for terms that exhibit a natural class of truth conditions) and expressive (flexible enough to apply to the variety of arguments that they seem to take).

\(^1\)Thanks are due to the anonymous reviewer who suggested investigating “only” in this context.
In what follows, I sketch a monomorphic theory based on Lambek’s categorial LP, using an operational semantics to characterize the relationship between terms and types. Then I extend it with sum types, Damas-Milner style polymorphic types and System F polymorphic types, proposing the use of the increasingly powerful systems for certain NL expressions, focusing on the quantifiers “every” and “only” and the basic connectives “and” and “or”.

2 A Type-theoretic Sketch

For a very long time linguists and philosophers have recognized that linguistic denotations can be characterized as mathematical functions or operations; for example Russell explicitly treated the logical constants as truth functions. As such, it is common to utilize the lambda calculus, a theory based around the λ operator for reifying functions anonymously, to express the form of linguistic denotations. As only certain combinations of linguistic terms are grammatical, so correspondingly only certain combinations of denotational terms are sensible; thus type theory, which characterizes such combinatorial constraints in terms of type-membership from Russell onwards, is commonly utilized in keeping with the Curry-Howard-de Bruijn correspondence described above.

2.1 Syntax

A theory well suited for this is the associative syntactic calculus with permutations LP of Lambek [20]. The correspondence between lambda terms and types in LP is given in the following schematization: Types T are

\[
T ::= e | t | T \rightarrow T | T \circ T
\]

where e is the type of entities of the model and t the type of propositions. Terms trm are

\[
trm ::= \text{var} | \text{const} | \lambda \text{var}^T \text{trm} | \text{trm} \cdot \text{trm}
\]

for a class var of term variables and const of constants. Types and terms are related by the type-tagging on variables in λ expressions, (thus this is a loosely-typed or Church-typed system), but more generally by type judgments. A typing context Γ is a string of variable-type pairs:

\[
\Gamma ::= (\text{var}_T)^* 
\]

You can treat contexts as a symbol-type-lookup table, searching from right to left, and as such a partial function \( \text{var} \rightarrow T \). \( \Gamma, \Delta \) means \( \Gamma \) and \( \Delta \) are concatenated. Concatenation is associative, so parentheses do not matter. \( \Gamma(\Delta) \) means \( \Delta \) is a substring of \( \Gamma \) and in particular,

\[
\frac{\Gamma(\Delta)}{\Gamma(\Delta')}
\]
means $\Delta'$ is substituted for $\Delta$ in $\Gamma$.

2.2 Judgments

LP typing judgments are given in the following natural deduction format:

$$\text{(Ax)} \quad x_A \vdash x : A$$

$$\text{(oE)} \quad \Gamma \vdash a : A \circ B \quad \Delta(x_A, y_B) \vdash c : C \quad \Delta(\Gamma) \vdash c[x \cdot y/a] : C$$

$$\Delta(x_A) \vdash f : B$$

$$\Gamma \vdash \lambda x^A f : A \rightarrow B$$

$$\text{(oI)} \quad \Gamma \vdash a : A \quad \Delta \vdash b : B \quad \Gamma(\Delta) \vdash a \cdot b : A \circ B$$

$$\text{(→ I)} \quad \Gamma \vdash \lambda x^A f : A \rightarrow B$$

$$\text{(→ E)} \quad \Gamma \vdash a : A \quad \Delta \vdash f : A \rightarrow B$$

Constants are assumed to have types already, that is, for $cn \in \text{const}$

$$\text{(Given)} \quad \vdash cn : A$$

for some $A$. Taken together, call this system LP_{ND}.

2.3 Gentzen Presentation and Decidability

There is also the Gentzen presentation LP_{g} [18, 20, 22], where, together with (Ax) and for what it’s worth (Given),

$$\text{(oL)} \quad \Gamma(x_A(y_B)) \vdash c : C \quad \Gamma(z_{A \circ B}) \vdash c[z/x \cdot y] :$$

$$\text{(oR)} \quad \Gamma \vdash a : A \quad \Delta \vdash b : B \quad \Gamma(\Delta) \vdash a \cdot b : A \circ B$$

$$\text{(→ L)} \quad \Gamma \vdash a : A \quad \Delta(x_B) \vdash c : C \quad \Delta(y_{A \rightarrow B}(\Gamma)) \vdash c[(y \cdot a)/z] : C$$

$$\text{(→ R)} \quad \Gamma(x_A) \vdash f : B \quad \Gamma \vdash \lambda x^A f : A \rightarrow B$$

Plus the familiar

$$\text{(Cut)} \quad \Gamma \vdash a : A \quad \Delta(x_A) \vdash c : C \quad \Gamma(\Delta) \vdash c[x/a] : C$$

Lambek [18] and Došen [9] show the equivalence of the two systems, the eliminability of (Cut), and consequently the decidability of both.

Thm 1 (LP_{ND} \approx LP_{g}). If there is a proof of $\Gamma \vdash a : A$ in LP_{ND} then there is a proof of $\Gamma \vdash a : A$ in LP_{g}. (Lambek [18])

Thm 2 (Cut-elimination.). If there is a proof of $a \vdash A$ in LP_{g} with (Cut), there is also a proof without (Cut). (Lambek [18], Došen [9])
2.4 Reduction and Safety

Term reduction rules follow Pierce [24]'s presentation. Values in the system, a subset of terms, are given by

\[ \text{val ::= const} \mid \lambda \text{var}^T \text{trm} \]

and reduction rules are, for \( a, b, c \in \text{trm} \) and \( v \in \text{val} \),

\[
\begin{align*}
\text{(app1)} & \quad a \rightsquigarrow b \\
& \quad (a \cdot c) \rightsquigarrow (b \cdot c) \\
\text{(app2)} & \quad a \rightsquigarrow b \\
& \quad (v \cdot a) \rightsquigarrow (v \cdot b) \\
\text{(beta)} & \quad (\lambda x^A \ f) \cdot a \rightsquigarrow f[x/a]
\end{align*}
\]

The following two theorems relate term reductions to type judgments (c.f. Pierce [24]):

**Thm 3** (Progress). When \( \vdash a : A \) then either \( a \in \text{val} \) or \( \exists b, a \rightsquigarrow b \).

**Thm 4** (Preservation). If \( \Gamma \vdash a : A \) and \( a \rightsquigarrow b \) then \( \Gamma \vdash b : A \).

Taken together, the theorems show that LP admits the slogan “Safety = Progress + Preservation”.

2.5 Logical Constants

Following Turner [28] we introduce the following constant term to the system,

\[ \vdash \Rightarrow : t \to t \to t \]

and term schema

\[ \vdash \Pi_A : A \to t \to t \]

The language \( \text{val} \) is interpreted by a model \( \mathcal{M} = \langle D, P, T, I, \sqsubseteq, \Pi_A \rangle \). \( \cdot \)\(\mathcal{M}\) maps types \( T \) into \( D \) s.t.

\[
\begin{align*}
\lfloor e \rfloor_{\mathcal{M}} & \subseteq D \\
\lfloor t \rfloor_{\mathcal{M}} & = P \\
\lfloor A \to B \rfloor_{\mathcal{M}} & = ([B]_{\mathcal{M}}(([A]_{\mathcal{M}}))
\end{align*}
\]

For a model \( \mathcal{M} = \langle D, P, T, I, \sqsubseteq, \Pi_A \rangle \) the following holds,

1. \( D \) is a set, the universe,
2. \( P \subseteq D \) the set of propositions,
3. \( T \subseteq P \) the true propositions,
4. $I : \text{const} \rightarrow D,$

5. $\sqsubseteq$ a function on $P \times P \rightarrow P$ s.t. $x \sqsubseteq y \in T$ if $x \in T$ only if $y \in T$

6. $\Pi_A : P[A] \rightarrow P$ s.t. for $f : [A] \rightarrow P \Pi_A(f) \in T$ if for all $a \in [A], f(a) \in T.$

Equipped with a partial function $g : \text{var} \rightarrow D$, $[\cdot]^{\text{un}_g} : \text{val} \cup \text{var} \rightarrow D$ s.t.

\[
\begin{align*}
[\lambda x^A f]^{\text{un}_g} &= d \mapsto [f]^{\text{un}_g}_{y/x/d} \text{ for } d \in [A]^{\text{un}_g} \\
[x]^{\text{un}_g} &= g(x) \text{ for } x \in \text{var} \\
[a]^{\text{un}_g} &= I(x) \text{ for } a \in \text{const}, \text{ in particular} \\
[a \Rightarrow b]^{\text{un}_g} &= [a]^{\text{un}_g} \sqsubseteq [b]^{\text{un}_g} \\
[\Pi_A]^{\text{un}_g} &= \Pi_A
\end{align*}
\]

The rest of Turner’s Intuitionistic Higher Order Logic (ihol) is given by the following definitions

\[
(\forall x^T) \equiv \lambda \phi^{T \rightarrow} \Pi_T (\lambda x^T \phi) \\
\land \equiv \lambda \phi^T \lambda \psi^T ((\forall x^1) \cdot (\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow \chi) \\
\lor \equiv \lambda \phi^T \lambda \psi^T ((\forall x^1) \cdot (\phi \rightarrow \chi) \rightarrow ((\psi \rightarrow \chi) \rightarrow \chi)) \\
(\exists x^T) \equiv \lambda \phi^{T \rightarrow} \lambda x^T ((\forall y^T) \cdot (\phi \rightarrow \chi)) \\
\bot \equiv ((\forall \phi^T) \cdot \phi) \\
\sim \equiv \lambda \phi^T \phi \rightarrow \bot \\
=_{T} \equiv \lambda a^T \lambda b^T ((\forall f^{T \rightarrow}) \cdot (f \cdot a) \rightarrow (f \cdot b))
\]

It follows that the proof theory of IHOL (made into classical HOL with the inclusion of the law of excluded middle) applies here too.

3 Ad hoc Polymorphism: First Pass

Constructive conjunction and disjunction are added to the system as follows:

Extend $T$

\[
T ::= \cdots | T \times T | T + T
\]

with new term schemata, for $i \in \{0, 1\},$

\[
(\text{trm}) ::= \cdots | \langle \text{trm}, \text{trm} \rangle | \pi_i \text{trm} | \iota_i \text{trm} | (\text{trm} ? \text{trm} : \text{trm})
\]

Typing judgments are extended, for LP$_{\text{nd}},$

\[
\begin{align*}
(\times I) \quad & \Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash \langle a, b \rangle : A \times B \\
(\times E) \quad & \Gamma \vdash a : A_0 \times A_1 \\
(+) I \quad & \Gamma \vdash a : A_i \\
(+) E \quad & \Gamma \vdash c : A + B \quad \Gamma \vdash a : A \rightarrow C \quad \Gamma \vdash b : B \rightarrow C \\
& \Gamma \vdash (c ? a : b) : C
\end{align*}
\]
Alternatively, for \( \text{LP}_e \),

\[
\begin{align*}
\text{(×L)} & \quad \Gamma(x_{(A_i)}) \vdash b : B \\
\text{(×R)} & \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \\
\text{(+L)} & \quad \frac{\Gamma(x_A) \vdash a : C \quad \Gamma(y_B) \vdash b : C}{\Gamma(z_{A+B}) \vdash (z ? \lambda a : \lambda b : C) : (A + B)} \\
\text{(+R)} & \quad \frac{\Gamma \vdash a : A_i}{\Gamma \vdash \iota_i a : A_0 + A_1}
\end{align*}
\]

The continued equivalence of \( \text{LP}_{\text{sd}} \) and \( \text{LP}_e \) is relatively easy to show. For example, here is \((\times R)\)'s derivation in \( \text{LP}_{\text{sd}} \)

\[
\begin{align*}
\Gamma(x_{A_i}) & \vdash b : B \\
\frac{\Gamma \vdash \lambda x^{A_i} a : B}{\Gamma, y : A_0 \times A_1 \vdash (\lambda x^{A_i} a \cdot \pi_i y) : B}
\end{align*}
\]

Also, new evaluation rules:

\[
\begin{align*}
\text{(proj1)} & \quad \pi_i \langle a_0, a_1 \rangle \rightsquigarrow a_i \\
\text{(proj2)} & \quad \pi_i a \rightsquigarrow \pi_i a' \\
\text{(pair1)} & \quad \frac{a \rightsquigarrow a'}{\langle a, b \rangle \rightsquigarrow \langle a', b \rangle} \\
\text{(pair2)} & \quad \frac{a \rightsquigarrow a'}{\langle v, a \rangle \rightsquigarrow \langle v, a' \rangle} \\
\text{(inj-case)} & \quad (\iota_i v ? a_0 : A_1) \rightsquigarrow (a_i \cdot v) \\
\text{(case)} & \quad \frac{a \rightsquigarrow a'}{(a ? b : c) \rightsquigarrow (a' ? b : c)} \\
\text{(inj)} & \quad \frac{a \rightsquigarrow a'}{\iota_i a \rightsquigarrow \iota_i a'}
\end{align*}
\]

The properties sketched above, Decidability and Safety still hold for \( \text{LP} \) with \( \{\times, +\} \) \([25, 24]\). Also, it is standard to define arbitrary lists and variants directly rather than constructing them out of pairs and sums.

### 3.1 Conjunction

Ad hoc polymorphism can be applied to a first pass at the definition of generalized conjunction \textbf{and} and \textbf{or}. Let

\[
\begin{align*}
\text{and}_t & \overset{\text{def}}{=} \lambda \phi^{t \times t} . \pi_0 \phi \land \pi_1 \phi \\
\text{and}_e & \overset{\text{def}}{=} \lambda x^{e \times e} . (X \cdot \pi_0 x) \land (X \cdot \pi_1 x) \\
\text{or}_t & \overset{\text{def}}{=} \lambda \phi^{t \times t} . \pi_0 \phi \lor \pi_1 \phi \\
\text{or}_e & \overset{\text{def}}{=} \lambda x^{e \times e} . (X \cdot \pi_0 x) \lor (X \cdot \pi_1 x)
\end{align*}
\]
Polymorphic terms **and** and **or** are given by

\[
\begin{align*}
\text{and} & \overset{\text{def}}{=} \lambda x^e e + t x^t (x ? \text{and}_e : \text{and}_t) \\
\text{or} & \overset{\text{def}}{=} \lambda x^e e + t x^t (x ? \text{or}_e : \text{or}_t)
\end{align*}
\]

where $\times$ binds more closely than $\mp$. So, for example, “Serge and Brigitte” as in “Serge and Brigitte sing” is derived as follows. Given $\vdash \text{serge} : e$ and $\vdash \text{brigitte} : e$,

\[
\begin{align*}
x_e & \vdash \text{serge} : e \\
x_e & \vdash \text{brigitte} : e \\
x_e & \vdash (\text{serge}, \text{brigitte}) : e \times e \\
x_e & \vdash \iota_1 (\text{serge}, \text{brigitte}) : t \times t + e \times e \\
x_e & \vdash (x \cdot \text{serge}) \land (x \cdot \text{brigitte}) : t \\
x_e & \vdash \lambda x^{e+t} (x \cdot \text{serge}) \land (x \cdot \text{brigitte}) : t
\end{align*}
\]

**and** and **or** are legitimate combinators of LP: they would not be if $\cdot$ were substituted for $\times$ above.

However, **and** is not even a combinator, as it contains a free variable.

### 3.2 Quantification

Let's examine English quantifiers “every” and “only”. Let $i$ be a new primitive type (hence, $T ::= \cdots \mid i$) of time-indices and have it that verbs generally quantify over time-indices as sketched in [6], here oversimplified. That said, for the purposes of this presentation, assume that unless specified, verbal terms are applied to temporal constant `now`—i.e. in the present tense:

\[
\begin{align*}
((\text{loves} \cdot \text{serge}) \cdot \text{brigitte}) & = \overset{\text{def}}{( ((\text{loves} \cdot \text{now}) \cdot \text{serge}) \cdot \text{brigitte})} \\
((\text{every} \cdot \text{man}) \cdot \text{sleeps}) & = \\
((\text{every} \cdot (\lambda \text{id} \text{sleeps} \cdot \text{serge})) \cdot (\lambda \text{id} \text{dreams} \cdot \text{serge})) & = \\
((\text{only} \cdot \text{serge}) \cdot (\text{loves} \cdot \text{brigitte})) & = \\
((\text{only} \cdot \text{loves} \cdot \text{serge}) \cdot \text{brigitte}) & = \\
((\text{only} \cdot \text{forfun} \cdot ((\text{sings} \cdot \text{serge}))) & = \\
\end{align*}
\]

Ascribing a denotation for **every** is standard and throughout the literature [2]. It can be glossed as

\[
\text{every} = \overset{\text{def}}{\lambda x^{a+b} \lambda y^{a+b} \forall z^T (x \cdot z) \Rightarrow (y \cdot z)}
\]

\[\text{2Special thanks to the anonymous commenter on this paper who suggested looking into “only” here.}\]
only is trickier. “Only Serge loves Brigitte” suggests that Serge loves Brigitte, and no one else does, hence

\[
\text{only} =_{\text{def}} \lambda x^e \lambda Z^{e \to t} (Z \cdot x) \land (\forall Y^{e \to t}) \!
\]
\[(Y \cdot x) \Rightarrow Y =_{e \to t} Z
\]

but, “Serge only likes Brigitte” (in the sense that he doesn’t love her or hate her)

\[
\text{only} =_{\text{def}} \lambda x^{e \to e \to t} \lambda z^e \lambda y^e ((x \cdot z) \cdot y) \land (\forall w^{e \to e \to t})((w \cdot z) \cdot y) \Rightarrow w =_{e \to e \to e \to t} x
\]

The problem is not only in choosing a type for the full term, but for the sub-terms as well. Here, categorial fusion and its associativity and permutability, helps

\[
\text{only} =_{\text{def}} \lambda x^X \lambda y^B (x \cdot y) \land (\forall z^B)(x \cdot z) \Rightarrow y =_{B \to B} z
\]

However, rather than terms for “every” and “only” we have term schemata.

4 From Schemata to Terms

The goal of this section is an account of how to reify the schemata sketched above (32, 35) into terms. This will introduce parametric polymorphism into the lexicon.

4.1 Damas-Milner Polymorphism

Add to the system new terms

\[
\text{trm ::= } \cdots | \text{let } x \leftarrow \text{trm in trm}
\]

the new typing rule

\[
\frac{
\begin{array}{c}
\Gamma \vdash a[b/x] : A \\
\Gamma \vdash b : B
\end{array}}{
\Gamma \vdash \text{let } x \leftarrow b \text{ in } a : A}
\]

and evaluation rules

\[
\begin{align*}
\text{(let1) } & \quad v \in \text{val} \\
\text{(let2) } & \quad a \rightsquigarrow b
\end{align*}
\]
\[
\frac{
\begin{array}{c}
\text{let } x \leftarrow v \text{ in } a \rightsquigarrow a[x/v]
\end{array}}{
\text{let } x \leftarrow a \text{ in } c \rightsquigarrow \text{let } x \leftarrow b \text{ in } c}
\]

New type-place-holders ?X are added to the system and allowed to serve as type-tags on lambda terms. This is feasible in the system because an effective type-reconstruction algorithm exists for the system, choosing concrete types
(i.e. members of $T$) when a typing exists for a term. In fact, given $\Gamma$ a given term has a principal or most general solution to the question of how to type its constituent terms. In effect, this allows terms that do not have type-tags labeling them. Pierce [24] covers this topic.

So, now we have

$$\text{(37)} \quad \text{every} \equiv_{\text{def}} \lambda x \lambda y \ (\forall z)(x \cdot z) \Rightarrow (y \cdot z)$$

$$\text{(38)} \quad \text{only} \equiv_{\text{def}} \lambda x \lambda y \ (x \cdot y) \land (\forall z)(z \cdot x) \Rightarrow y = z$$

So, for example, “Every time Serge sleeps Serge dreams” is given by

$$\text{(39)} \quad \text{let } x \leftarrow \lambda t^i (((\text{sleeps} \cdot t) \cdot \text{serge}) \text{ in } y \leftarrow \lambda t^i (((\text{dreams} \cdot t) \cdot \text{serge}) \text{ in } (\forall z^i)(x \cdot z) \Rightarrow (y \cdot z)$$

Note that replacing $x$ and then $y$ with $\lambda t^i (((\text{sleeps} \cdot t) \cdot \text{serge})$ and then $\lambda t^i (((\text{dreams} \cdot t) \cdot \text{serge})$ reduces to

$$\text{(40)} \quad ((\text{every} \cdot \lambda t^i (((\text{sleeps} \cdot t) \cdot \text{serge}) \cdot \lambda t^i (((\text{dreams} \cdot t) \cdot \text{serge})$$

Since it is fairly straightforward, though tedious, to provide a typing for every s.t. $\vdash (40) : t$, it follows that

$$\text{(41)} \quad \vdash (40) : t$$

$$\text{(Let)} \quad \vdash (39) : t$$

### 4.2 Polymorphic Lambda Calculus

System F or the polymorphic lambda calculus [14, 15, 26, 27] adds to the type system a universal type schema and an abstraction operation on types. Now, in the object language there are two classes of variables, type variables $\text{TyVar}$ and term variables $\text{TmVar}$. Now, typing constants have type variables occur in them

$$\text{(42)} \quad \Gamma ::= (\text{var}_T \mid \text{TyVar})^*$$

and there are new terms that range over type symbols

$$\text{(43)} \quad T ::= e \mid t \mid i \mid \text{TyVar} \mid T \rightarrow T \mid \forall T. T$$

The other term schemata defined above are now unnecessary, as System F can define general terms for pairs, sums, even arithmetic via Church-encodings.

$$\text{(44)} \quad \text{TmVar} ::= \text{TmVar} \mid \text{const} \mid \Lambda \text{TmVar} \mid \text{trm} \mid (\text{trm} \cdot \text{trm}) \mid \Lambda \text{TyVar} \mid \text{trm} \mid (\text{trm} \cdot T)$$

New typing rules for F

$$\text{(\forall I)} \quad \Gamma, X \vdash f : A \quad \frac{}{\Gamma \vdash \Lambda X f : \forall X.A}$$

$$\text{(\forall E)} \quad \Gamma \vdash f : \forall X.A \quad \frac{}{\Gamma \vdash f_B : A[B/X]}$$
and reduction rules

\[
\begin{align*}
\text{(tyapp)} & \quad a \leadsto b \\
\ & \quad a_A \leadsto b_A \\
\text{(tybeta)} & \quad \Lambda X \ a_A \leadsto a[X/A]
\end{align*}
\]

System F is safe in the sense of “Preservation + Progress” [24], proofs are cut admissible and F has the property of normalization, that all well-typed terms terminate [14, 15].

The semantics of System F are strictly outside those we sketched for \textbf{LP}, as quantifiers range over propositional variables, then to comprehend a type \(\forall X.A\), it is necessary to comprehend all \(A[X/B]\), where \(B\) may be much more complicated than \(A\). In this sense F is \textit{impredicative}. There are two known classes of models of System F [27]. In one, types are mapped to partial equivalence relations on a model of the untyped lambda calculus. In the other, the meaning of type is a Scott domain.

Also, in general type reconstruction fails, that is, given a term \(a\) of the untyped lambda calculus, whether there is a term \(a'\) in System F s.t. \(a = a'\) with all the type-tags removed [24, 29].

Finally, in linguistics Martin Emms [11] discovered that in general the associative syntactic calculus \textbf{L} equipped with higher-order types is undecidable (in spite of showing that it is cut-admissible).

### 4.3 Examples of \(\forall \rightarrow \forall\) Polymorphism?

The general impredictativity of System F, and the results regarding its lack of type reconstruction and undecidability when constrained to \textbf{L} raise the question whether F is too powerful for linguistic tasks. Many linguists think so; van Bentham [4] prefers Damas-Milner style types, which he call “substitutional polymorphism” which are also studied in Emms [12] and elsewhere.

Several authors [7, 17, 13] have pointed out the impredictativity of certain English terms, such as “is fun”, and have generalized the type theory of their respective theories (or, in Chierchia’s case, removed it altogether).

In keeping with the emphasis on quantifiers and connectives in this paper, note the connectives can coordinate (polymorphic) quantifiers themselves:

(45) All and only men love Brigitte.

(46) One or two boys love Brigitte.
System F derivations for these examples are:

\[
\begin{align*}
&\frac{((\text{or} \cdot (\text{one}, \text{two})) \cdot \text{boy}) \cdot (\text{loves} \cdot \text{brigitte})}{(\Lambda X [(\text{or} \cdot (\text{one}_X, \text{two}_X)) \cdot \text{boy}) \cdot (\text{loves} \cdot \text{brigitte})]} \\
&\frac{((\text{or} \cdot (\text{one}_X, \text{two}_X)) \cdot \text{boy}) \cdot (\text{loves} \cdot \text{brigitte})}{((\text{or} \cdot (\text{one}_e, \text{two}_e)) \cdot (\text{loves} \cdot \text{brigitte})]}
\end{align*}
\]

Finally, considering “and/or” Is it the case that

\[
[\text{and/or}] = (\text{or} \cdot (\text{and}, \text{or}))?
\]

Of course,

\[
\begin{align*}
((\text{or} \cdot (\text{and}, \text{or})) \cdot (\phi, \psi)) &= ((\lambda x^t \cdot x) \cdot x) \cdot (\phi, \psi) \\
&= (\text{or} \cdot ((\phi \cdot (\phi, \psi)), (\text{or} \cdot (\phi, \psi)))) \\
&= (\phi \cdot (\phi \cdot (\phi \cdot (\phi, \psi))) \\
&= (\phi \cdot (\phi \cdot (\phi \cdot (\phi, \psi))) \\
&= \phi \lor \psi \text{ in classical reasoning}
\end{align*}
\]

In fact, even in constructive reasoning, or with an exclusive interpretation of “or”, \((\phi \land \psi) \lor (\phi \lor \psi)\) entails classical or, that is, (a witness to) \(\phi\), (a witness) to \(\psi\), or (a witness) to \(\phi \land \psi\).

References


[22] M. Moortgat. Categorial Type Logics. In van Bentham and ter Meulen [5].


Abstract

Infinite lambda calculi extend finite lambda calculus with infinite terms and transfinite reduction. In this paper we extend some classical results of finite lambda calculus to infinite terms. The first result we extend to infinite terms is Böhm Theorem which states the separability of two finite $\beta\eta$-normal forms. The second result we extend to infinite terms is the equivalence of the prefix relation up to infinite eta expansions and the contextual preorder that observes head normal forms. Finally we prove that the theory given by equality of $\infty\eta$-Böhm trees is the largest theory induced by the confluent and normalising infinitary lambda calculi extending the calculus of Böhm trees.

1 Introduction

Not all computations are finite. The calculation of the meaning of a linguistic expression can be infinite, whether the language is a natural language or not. Vicious circles can be a cause. In their book [3] Barwise and Moss give the following example:

The law school professor who had worked for him denounced the judge who had harassed her.

This sentence can be expressed using the formulæ:

$$D(P, J)$$

where $D(x, y)$ means $x$ denounces $y$, $P$ means the law school professor who had worked for him and $J$ means the judge who had harassed her. The calculations that lead to the exact references of a particular judge and professor can be performed with two rewrite rules:

$$P \to W(J)$$
Reduction Rules | Normal Forms | NF
---|---|---
Beta and Bottom for terms without tnf | Berarducci trees | $\text{BerT} = P_{\text{TN}}$
Beta and Bottom for terms without whnf | Lévy–Longo trees | $\text{LLT} = P_{\text{WN}}$
Beta and Bottom for terms without hnf | Böhm trees | $\text{BT} = P_{\text{HN}}$
Beta, Bottom parametric on $U$ | Parametric trees | $\text{NF} = P_{U}$
Beta, Bottom for terms w.o. hnf and Eta | $\eta$-Böhm trees | $\eta\text{BT}$
Beta, Bottom for terms w.o. hnf and EtaBang | $\infty\eta$-Böhm trees | $\infty\eta\text{BT}$

Figure 1: Infinitary Lambda Calculi

$\lambda x.x \rightarrow H(P)$

relating to the respective clauses who had worked for him and who had harassed her. We see that the calculation of the meaning of the whole sentence now does not terminate. It may go like this:

$$D(P, J) \rightarrow D(W(J), J) \rightarrow D(W(J), H(P)) \rightarrow D(W(H(P)), H(P)) \rightarrow$$

$$D(W(J), H(W(J)))) \rightarrow D(W(H(P)), H(W(J)))) \rightarrow \ldots$$

culminating in a limit $D(W(H(W(H(\ldots))))), H(W(H(W(\ldots)))))))$ which does no longer seem to refer to any judge or professor.

**Infinitary rewriting** is the branch of rewriting that deals explicitly with infinite terms and infinite reductions. Extending a finite rewriting system into a infinite system has to be done with care when one wants to preserve a useful property like confluence.

**In this paper** we focus on one particular theory of rewriting namely lambda calculus. Lambda calculus is confluent. Just extending finite lambda calculus with infinite lambda terms and infinite reduction destroys this confluence property.

Define $I = \lambda x.x$, $W = \lambda x.I(xx)$ and $\Delta = \lambda x.xx$ Then the term $\Delta W$ has a one step reduction to $\Delta\Delta$ and an infinite reduction to $I(I(\ldots))$, namely

$$\Delta W \rightarrow_\beta WW \rightarrow_\beta I(WW) \rightarrow_\beta I(I(WW)) \rightarrow_\beta I(I(I(\ldots)))$$

Both $\Delta\Delta$ and $I(I(I(\ldots)))$ reduce only to themselves, and have no common reduct.

To rescue the confluence property one has to extend the lambda calculus also with an extra $\perp$-rule that replaces a meaningless term by $\perp$. Interestingly there are many different choices for the set of meaningless terms. The set of terms without head normal form is the largest, the set of terms without top normal form is the smallest [6]. The infinite lambda calculi that we consider here have all the same set of finite and infinite terms $\Lambda_\infty$. Besides the variation that come with the choice of a set $U$ of meaningless terms, there is another source of variation in the infinitary setting that comes with the strength of extensionality.

62
Figure 1 summarises the infinitary lambda calculi studied so far [4, 6, 7, 5, 10, 9]. An interesting aspect of infinitary lambda calculus is the possibility of capturing the notion of tree (such as Böhm and Lévy–Longo trees) as a normal form. These trees were originally defined for finite lambda terms only, but in the infinitary lambda calculus we can also consider normal forms of infinite terms. The three infinitary lambda calculi mentioned in the first three rows of Figure 1 capture the well-known cases of Böhm, Lévy–Longo and Berarducci trees [4, 6, 7]. In the fourth row, there is an uncountable class of infinitary lambda calculi with a $\perp$-rule parametrised by a set $\mathcal{U}$ of meaningless terms [8, 5]. By changing the parameter set $\mathcal{U}$ of the $\perp$-rule, we obtain different infinitary lambda calculi. If $\mathcal{U}$ is the set $\mathcal{H}_N$ of terms without head normal form, we capture the notion of Böhm tree. If $\mathcal{U}$ is the set $\mathcal{W}_N$ of terms without weak head normal form we obtain the Lévy–Longo trees. And if $\mathcal{U}$ is the set $\mathcal{T}_N$ of terms without top head normal form to $\perp$, we recover the Berarducci trees.

The infinitary lambda calculus sketched in the one but last row incorporates the $\eta$-rule [10]. This calculus captures the notion of $\eta$-Böhm tree. The last row in Figure 1 mentions the infinitary lambda calculus incorporating the $\eta$!-rule, a strengthened form of the $\eta$-rule [9], whose normal forms correspond to the $\infty\eta$-Böhm trees.

In this paper we lift three classical results of finite lambda calculus to infinite lambda calculus. First we extend Böhm’s Theorem concerning separability of finite $\beta\eta$-normal forms, also known as finite $\eta$Böhm trees, to possibly infinite $\infty\eta$Böhm trees. Two terms $M$ and $N$ are separable if for any pair of finite terms $P, Q$ there exists a context $C$ such that $C[M] \rightarrow_\beta P$ and $C[N] \rightarrow_\beta Q$. This statement is extended to infinite terms by considering the variation of $\eta$-reduction called $\eta!$. We prove then that two (possibly infinite) $\beta\eta!$-normal forms (also known as $\infty\eta$Böhm) $M$ and $N$ can be separated, i.e. for any (possible infinite) pair of terms $P$ and $Q$ there exists a finite context $C$ such that $C[M] \rightarrow_\beta P$ and $C[N] \rightarrow_\beta Q$. The terms $M, N$ subject to separability may be infinite. However the discriminating context remains finite and only finite $\beta$-reduction is necessary to ”separate them”. The method for finding such contexts is called the Böhm out technique [1].

The second result that we extend to infinite terms is the equivalence of the prefix relation up to infinite eta expansions and the contextual preorder that observes head normal forms. It is natural to compare terms, in particular normal forms, with help of the prefix relation $\preceq$. When terms are represented as trees, prefixes of a tree are obtained by pruning some of its subtrees and replacing them by $\perp$. In [11] we prove that the function $\mathcal{B}_D$ is monotone in $(\Lambda_\infty, \preceq)$ and that the function $\infty\eta\mathcal{B}_D$ is not so. For $\infty\eta\mathcal{B}_D$ it is the prefix relation up to infinite eta expansions denoted by $\preceq_{\eta!}$ that is monotone. Another basic preorder between terms is the contextual preorder with respect to head normal forms. denoted by $M \subseteq_h N$ which means that for all contexts $C$ if $C[M]$ has a head normal form then $C[N]$ has a head normal form. In [13] Wadsworth, generalising Böhm’s theorem, shows that the equivalence between $\subseteq_h$ and $\preceq_{\eta!}$ on the set $\Lambda$ of finite $\lambda$-terms (which is part of the Characterisation Theorem for $D_\infty$). In this paper we will show the equivalence between $\subseteq_h$ and $\preceq_{\eta!}$ on the
set $\Lambda^\infty$ of finite and infinite lambda terms. One direction uses only properties of the reduction. The other direction extends the Böhm out technique to infinite terms.

Finally we prove that the theory given by equality of $\infty\eta$-Böhm trees is the largest theory induced by a confluent and normalising infinitary lambda calculus extending the calculus of the Böhm trees. The analogous result for finite lambda calculus is that the theory $H^\ast = \{(M, N) \in \Lambda \mid M \subseteq_h N\}$ is the unique Hilbert-Poincaré complete lambda theory extending the theory $H$ which equates the unsolvables [1].

2 Infinitary Lambda Calculus

In this section we will briefly recall some notions and facts of infinite lambda calculus from our earlier work [6, 7, 5, 10, 9]. We assume familiarity with basic notions and notations from [1].

Let $\Lambda$ be the set of $\lambda$-terms and $\Lambda_\bot$ be the set of finite $\lambda$-terms with $\bot$ given by the inductive grammar:

$$M ::= \bot \mid x \mid (\lambda x M) \mid (MM)$$

where $x$ is a variable from some fixed set of variables $\mathcal{V}$. We follow the usual conventions on syntax. Terms and variables will respectively be written with (super- and subscripted) letters $M, N$ and $x, y, z$. Terms of the form $(M_1M_2)$ and $(\lambda x M)$ will respectively be called applications and abstractions. A context $C\[\]$ is a term with a hole in it, and $C[M]$ denotes the result of filling the hole by the term $M$, possibly by capturing some free variables of $M$. If $\sigma : \mathcal{V} \to \Lambda^\infty$ then $M^\sigma$ is the simultaneous substitution of the variables in $M$ by $\sigma$.

The set $\Lambda_\bot^\infty$ of finite and infinite lambda terms is defined by coinduction using the same grammar as for $\Lambda_\bot$. This set contains the three sets of Böhm, Lévy–Longo and Berarducci trees. In [7, 8, 5], an alternative definition of the set $\Lambda^\infty_\bot$ is given using a metric. The coinductive and metric definitions are equivalent [2]. In this paper we consider only one set of $\lambda$-terms, namely $\Lambda^\infty_\bot$, in contrast to the formulations in [7, 8] where several sets (which are all subsets of $\Lambda^\infty_\bot$) are considered. The paper [5] shows that the infinitary lambda calculi can be formulated using a common set $\Lambda^\infty$, confluence and normalisation still hold since the extra terms added by the superset $\Lambda^\infty_\bot$ are meaningless and equated to $\bot$.

We define several rules we use to define various infinite lambda calculi. The $\beta, \eta$ and $\eta^{-1}$-rules apply to finite and infinite terms as well. The extra power of the $\eta$-rule becomes visible on infinite terms. The $\bot$-rule is parametric on a set $\mathcal{U} \subset \Lambda^\infty$ of meaningless terms [8, 5] where $\Lambda^\infty$ is the set of terms in $\Lambda^\infty_\bot$ that do not contain $\bot$.

Definition 1. We define the following rewrite rules on $\Lambda^\infty_\bot$:

$$(\lambda x M)N \to M[x := N] \quad (\beta) \quad \frac{M[\bot := \Omega] \in \mathcal{U}}{M \neq \bot \quad (M \to \bot) \bot}$$
In this paper we need various rewrite relations constructed from these rules on the set $\Lambda^\infty_\bot$. These are defined in the standard way, eg. $\rightarrow^\beta_\bot$ is the smallest binary relation containing the $\beta$, $\bot$ and $\eta!$-rules which is closed under contexts. Reduction sequences can be of any transfinite ordinal length $\alpha$: $M_0 \rightarrow M_1 \rightarrow \ldots M_\omega \rightarrow M_{\omega+1} \rightarrow \ldots M_{\omega+\omega} \rightarrow M_{\omega+\omega+1} \rightarrow \ldots M_\alpha$. This makes sense if the limit terms $M_\omega, M_{\omega+\omega}, \ldots$ in such sequence are all equal to the corresponding Cauchy limits, $\lim_{\beta \rightarrow \lambda} M_\beta$, in the underlying metric space for any limit ordinal $\lambda \leq \alpha$. If this is the case, the reduction is called Cauchy converging. We need the stronger concept of a strongly converging reduction that in addition satisfies that the depth of the contracted redexes goes to infinity at each limit term: $\lim_{\beta \rightarrow \lambda} d_\beta = \infty$ for each limit ordinal $\lambda \leq \alpha$, where $d_\beta$ is the depth in $M_\beta \rightarrow M_\beta+1$ of the contracted redex in $M_\beta \rightarrow M_\beta+1$. Any finite reduction is, then, strongly converging. We use the following notation:

1. $M \rightarrow N$ denotes a one step reduction from $M$ to $N$;
2. $M \rightarrow N$ denotes a finite reduction from $M$ to $N$;
3. $M \rightarrow N$ denotes a strongly converging reduction from $M$ to $N$.

Variations on the reduction rules give rise to different calculi (see Figure 1). The resulting infinite lambda calculus $(\Lambda^\infty_\bot, \rightarrow^\rho_\bot)$ we will denote by $\lambda^\infty_\rho$ for any $\rho \in \{\beta_\bot, \beta_\bot \eta, \beta_\bot \eta!\}$. Since the $\bot$-rule is parametric, each set $U$ of meaningless terms gives a different infinitary lambda calculus $\lambda^\infty_{\beta_\bot}$.

The notions of head normal form, weak head normal form and top normal form are defined as follows:

1. A head normal form (hnf) is a term of the form $\lambda x_1 \ldots x_n. y M_1 \ldots M_k$.
2. A weak head normal form (whnf) is either a hnf or an abstraction $\lambda x. M$.
3. A top normal form (tnf) is either a whnf or an application $(MN)$ if there is no $P$ such that $M \rightarrow^\beta_\bot \lambda x. P$.

We define the following sets:

$\mathcal{HN} = \{ M \in \Lambda^\infty \mid M \rightarrow^\beta N \text{ and } N \text{ in head normal form} \}$

$\mathcal{WN} = \{ M \in \Lambda^\infty \mid M \rightarrow^\beta N \text{ and } N \text{ in weak head normal form} \}$

$\mathcal{T N} = \{ M \in \Lambda^\infty \mid M \rightarrow^\beta N \text{ and } N \text{ in top normal form} \}$

Instances of $U \subseteq \Lambda^\infty$ are $\overline{\mathcal{HN}}$, $\overline{\mathcal{WN}}$ and $\overline{\mathcal{T N}}$ the respective complements of $\mathcal{HN}$, $\mathcal{WN}$ and $\mathcal{T N}$.

**Definition 2.** 1. We say that a term $M$ in $\lambda^\infty_\rho$ is in $\rho$-normal form if there is no $N$ in $\lambda^\infty_\rho$ such that $M \rightarrow^\rho N$.

2. We say that $\lambda^\infty_\rho$ is confluent (Church-Rosser) if $(\Lambda^\infty_\bot, \rightarrow^\rho_\bot)$ satisfies the diamond property, i.e. $\rho \leftrightarrow \circ \rightarrow^\rho \subseteq \rightarrow^\rho \circ \rho \leftrightarrow$. 

65
3. We say that $\lambda^\infty_\rho$ is normalising if for all $M \in \Lambda^\infty_\perp$ there exists an $N$ in $\rho$-normal form such that $M \rightarrow^* \rho N$.

**Theorem 1.** [7, 8, 5] Let $U$ be a set of meaningless terms. The calculi $\lambda^\infty_{\beta\perp}$ with a parametric $\perp$-rule on the set $U$ are confluent, normalising and satisfy postponement of $\perp$ over $\beta$.

In [5], confluence of the parametric calculi is proved for any Cauchy converging reduction, not only strongly converging ones.

**Theorem 2.** [10, 9] The infinite lambda calculi of $\infty\eta$-Böhm and $\eta$-Böhm trees are confluent and normalising.

We will use the following properties of $\eta!$-reduction proved in [9].

**Theorem 3.** [9]

1. Inverse reductions: $M \rightarrow^* \eta! N$ if and only if $N \rightarrow^* \eta^{-1} M$.

2. The infinitary lambda calculus $\lambda^\infty_{\eta!}$ is confluent.

3. The relations $\rightarrow^* \eta!$ and $\rightarrow^* \beta$ commute.

4. The transfinite $\eta!$ and $\eta^{-1}$-reductions preserves the property of having head normal form, i.e. if $M \rightarrow^* \eta! N$ then the following are equivalent:

   (a) there exists a head normal form $M'$ such that $M \rightarrow^* \beta M'$,

   (b) there exists a head normal form $N'$ such that $N \rightarrow^* \beta N'$.

3 The standard prefix relation

Each of the confluent and normalising extensions of finite lambda calculus gives rise to a normal form function that assigns to a lambda term in $\Lambda^\infty_\perp$ a corresponding normal form. We will denote these various functions with notation used in Figure 1.

**Definition 3.** Let $M, N \in \Lambda^\infty_\perp$.

1. We say that $M$ is a prefix of $N$ (we write $M \preceq N$) if $M$ is obtained from $N$ by replacing some subterms of $N$ by $\perp$.

2. $M \preceq_{\mathrm{NF}} N$ if $\mathrm{NF}(M) \preceq \mathrm{NF}(N)$.

**Definition 4.** Let $M \in \Lambda^\infty_\perp$. We define the truncation of $M$ at depth $n$, denoted as $M^n$, as the result of replacing in $M$ all subterms at depth $n$ by $\perp$.

The following lemmas are particular cases of general lemmas proved in [11, 12] to deduce continuity of $\mathcal{B}T$ and $\mathcal{L}T$ on the cpo $(\Lambda^\infty_\perp, \preceq)$.

**Lemma 4.** Let $C[M] \in \Lambda^\infty_\perp$. Then, $(C[M])^n = C^n[M^k]$ where $k = \max(0, n-d)$ and $d$ is the depth of the hole in $C$. 
Lemma 5. If \( M \preceq N \) and \( M \rightarrow_{\beta} M' \) then \( N \rightarrow_{\beta} N' \) and \( M' \preceq N' \) for some \( N' \).

Lemma 6. Let \( M, N \in \mathcal{K}_\perp^\infty \). If \( M \preceq N \) then \( M \preceq_{\text{BT}} N \).

Lemma 7. Let \( P \in \mathcal{K}_\perp^\infty \). For all \( n \) there exists \( i \) such that \( (\text{BT}(P))^n \preceq_{\text{BT}} (P)^{n+i} \).

4 Prefix up to infinite eta expansions

In this section we define the relation \( \preceq_{\eta!} \) on \( \beta\perp \)-normal forms which is equivalent to the relation \( \eta \subseteq_{\eta} \) on Böhm-like trees defined in [1] (up to change of representation from terms to trees).

Definition 5. Let \( M, N \in \mathcal{K}_\perp^\infty \). Then, \( M \preceq_{\eta!} N \) if \( \text{BT}(M) \rightarrow_{\eta-1} P \preceq Q \eta-1 \leftarrow \leftarrow \text{BT}(N) \) for some \( P, Q \in \text{BT}(\mathcal{K}_\perp^\infty) \). \(^1\)

If \( M \preceq_{\eta!} N \) then there exists “a canonical pair of terms” \( P, Q \) such that \( M \rightarrow_{\eta-1} P \preceq Q \eta-1 \leftarrow \leftarrow N \). To find this pair of terms we use a bisimulation that imposes the number of abstractions and arguments to be the same. This bisimulation is used to simplify the Böhm-out technique.

Definition 6 (Honest bisimulation). Let \( R \) be a binary relation on the set of \( \beta\perp \)-normal forms. Then \( R \) is called a honest bisimulation if whenever \( M \mathrel{R} N \),

- if \( M = \lambda x_1 \ldots x_n y M_1 \ldots M_m \), \( N = \lambda x_1 \ldots x_{n'} y N_1 \ldots N_{m'} \) and \( n - n' = m - m' \) then \( n = n' \), \( m = m' \) and \( M_i \mathrel{R} N_i \) for all \( 1 \leq i \leq m \).

The maximal honest bisimulation \( R \) is denoted by \( \sim \).

We give some examples of bisimilar terms:

- The constant \( \bot \) is bisimilar to any \( \beta\perp \)-normal form.
- We say that two \( \beta\perp \)-normal forms \( M \) and \( N \) are distinguishable if \( M = \lambda x_1 \ldots x_n y M_1 \ldots M_m \), \( N = \lambda x_1 \ldots x_{n'} y' N_1 \ldots N_{m'} \) and either the head variables \( y \) and \( y' \) are different or \( n - n' \neq m - m' \). Then, distinguishable terms are bisimilar.
- The terms \( y \) and \( \lambda x . y z \) are not bisimilar but by \( \eta \)-expanding the variable \( y \) we get the term \( \lambda x . y x \) which is bisimilar to \( \lambda x . y z \).

Theorem 8 (Existence of bisimilar terms). Let \( M, N \) be in \( \beta\perp \)-normal form. Then there are \( P, Q \) such that \( M \rightarrow_{\eta-1} P \sim Q \eta-1 \leftarrow \leftarrow N \).

Proof. We define a function \( \text{bisim} : \mathcal{K}_\perp^\infty \times \mathcal{K}_\perp^\infty \rightarrow \mathcal{K}_\perp^\infty \times \mathcal{K}_\perp^\infty \) by coinduction such that \( \text{bisim}(M, N) = (P, Q) \) and \( M \rightarrow_{\eta-1} P \sim Q \eta-1 \leftarrow \leftarrow N \).

We have two cases:

\(^1\)By Theorem 3, we could replace \( \rightarrow_{\eta!} \) by \( \eta-1 \leftarrow \leftarrow \) and get an equivalent definition.
1. If \( M = \lambda x_1 \ldots x_n. y M_1 \ldots M_m, N = \lambda x_1 \ldots x_{n'}. y N_1 \ldots N_{m'} \) and \( n - n' = m - m' = k \) then we define

\[
\text{bisim}(M, N) = (\lambda x_1 \ldots x_n. y P_1 \ldots P_m, \lambda x_1 \ldots x_n. y Q_1 \ldots Q_m)
\]

where \( P_i \) and \( Q_i \) are obtained as follows:

(a) If \( n \geq n' \) and \( m \geq m' \) then we have to eta expand \( N \) until the number of abstractions and applications coincide with the ones in \( M \). Hence,

\[
\text{bisim}(M_i, N_i) = (P_i, Q_i) \quad \text{for} \quad 1 \leq i \leq m'
\]

and

\[
\text{bisim}(M_{m'+j}, x_{m'+j}) = (P_{m'+j}, Q_{m'+j}) \quad \text{for} \quad 1 \leq j \leq k
\]

(b) If \( n < n' \) and \( m < m' \) then we have to eta expand \( M \) until the number of abstractions and applications coincide with the ones in \( N \). Hence,

\[
\text{bisim}(M_i, N_i) = (P_i, Q_i) \quad \text{for} \quad 1 \leq i \leq m
\]

and

\[
\text{bisim}(x_{n+j}, N_{m+j}) = (P_{m+j}, Q_{m+j}) \quad \text{for} \quad 1 \leq j \leq -k
\]

2. Otherwise, \( \text{bisim}(M, N) = (M, N) \).

It is easy to see that the function \( \text{bisim} \) as a relation is an honest bisimulation and hence, \( P \sim Q \).

\[ \square \]

**Corollary 9.** Let \( M, N \) be in \( \beta\perp \)-normal form and \( (P, Q) = \text{bisim}(M, N) \). Then \( M \preceq_{\eta} N \) if and only if \( P \preceq Q \).

**Proof.** Note that if \( M \preceq_{\eta} N \) and \( M \sim N \) then \( M \preceq N \). \[ \square \]

We give some examples of how to find the pair \( (P, Q) = \text{bisim}(M, N) \). Let \( J \) be a term satisfying the recursive equation \( Jx = \lambda y. x(Jy) \) and \( E_x \) be the \( \beta\perp \)-normal form of \( Jx \), i.e. \( E_x = \lambda y_1. x(\lambda y_2.y_2(\ldots)) \). Note that \( x \rightarrow_{\eta-1} E_x \).

<table>
<thead>
<tr>
<th>( M )</th>
<th>( N )</th>
<th>( P )</th>
<th>( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>( \lambda x.y.z )</td>
<td>( \lambda x.y.x )</td>
<td>( \lambda x.y.z )</td>
</tr>
<tr>
<td>( x\perp(\lambda y.zy) )</td>
<td>( \lambda u.xzu )</td>
<td>( \lambda u.x\perp(\lambda y.zy)u )</td>
<td>( \lambda u.x(\lambda y.zy)u )</td>
</tr>
<tr>
<td>( x\perp(\lambda y.zE_y) )</td>
<td>( \lambda u.xzE_a )</td>
<td>( \lambda u.x\perp(\lambda y.zE_y)E_u )</td>
<td>( \lambda u.x(\lambda y.zE_y)E_u )</td>
</tr>
</tbody>
</table>
5 Contextual Preorders

In this section we give two definitions for contextual preorder in the infinitary lambda calculus and prove that they are equivalent using the previous truncation lemmas. The definitions will differ in the sets of contexts over which will be quantified. In the definition of $\subseteq_h$ the quantification is restricted to finite contexts, in the definition of $\subseteq^\infty_h$ the quantification runs over finite and infinite contexts.

Definition 7. 1. We say that $M \subseteq_h N$ if for all finite contexts $C$, if $C[M]$ $\beta$-reduces to a head normal form then so does $C[N]$.

2. We say that $M \subseteq^\infty_h N$ if for all (finite or infinite) contexts $C$, if $C[M]$ $\beta$-reduces to a head normal form then so does $C[N]$.

Both notions of contextual preorder coincide:

Theorem 10. The following statements are equivalent for any terms $M, N \in \Lambda^\infty$:

1. $M \subseteq_h N$.

2. $M \subseteq^\infty_h N$.

Proof. (2) implies (1) trivially. We show (1) implies 2. Let $C$ be an infinite context. We will construct a finite context (a truncation of $C$) that behaves like $C$. Observe that:

$$\begin{align*}
\text{BT}^i(C[M]) &\preceq_B (C[M])^{1+i} \quad \text{for some } i \text{ by Lemma 7} \\
&= C^{1+i}[(M)^k] \quad \text{by Lemma 4} \\
&\preceq_B C^{1+i}[M] \quad \text{by Lemma 6}
\end{align*}$$

where $d$ is the depth of the hole in $C$ and $k = \max(0, n + 1 - d)$.

If $C[M]$ has a head normal form, so does $C^{1+i}[M]$. Since $C^{1+i}$ is finite, by (1) we have that $C^{1+i}[N]$ has a head normal form. By Lemma 6, we have that $C^{1+i}[N] \preceq_B C[N]$ and hence $C[N]$ has a head normal form too.

6 Separability

In this section we extend Böhm Theorem to infinite terms. We first extend the notion of separability so that it applies to infinite terms as well. Despite this extension the discriminating contexts that we will construct will be finite so that only finite $\beta$-reduction will be needed to "fully separate them". Böhm-ing out and the separability of distinguishable terms follow along the lines of [1]. However infinite terms complicate matters because it is not possible to give a bound for the number $q$ of the permutator $P_q$ used to Böhm-out the subterms of a term.
Definition 8. Let $M, N \in \mathcal{L}_1^\infty$. We say that the terms $M$ and $N$ are separable if for any $P, Q \in \mathcal{L}^\infty$ there exists a finite context $C$ such that $C[M] \rightarrow_{\beta} P$ and $C[N] \rightarrow_{\beta} Q$.

We will use the following notation and terminology:

1. **permutators** are terms of the form $P_q = \lambda x_1 \ldots x_{q+1}.x_{q+1}x_1 \ldots x_q$;
2. **selectors** are terms of the form $U_i^q = \lambda x_1 \ldots x_q.x_i$; and
3. **constants** are terms of the form $K^m = \lambda x_1 \ldots x_{m+1}.x_{m+1}$.

The way we select ("Bohm-out") subterms in the infinitary lambda calculus is similar to the finite case:

Theorem 11 (Bohm-ing-out finite and infinite terms). Let $M$ be a $\beta_\perp$-normal form such that $M = \lambda x_1 \ldots x_n.y M_1 \ldots M_m$. Take $q \geq m$. Then there are contexts $C_i$ for $1 \leq i \leq q$ and a term $P$ such that:

1. $C_i[M] \rightarrow_{\beta} M_i^\sigma$ when $1 \leq i \leq m$,
2. $C_i[M] \rightarrow_{\beta} z_i$ when $m + 1 \leq i \leq q$;

where $\sigma = [y := P]$ and $z_{m+1} \ldots z_q z_{q+1}$ are fresh variables.

Proof. Put $C_i[ ] = (\lambda y.[ ]x_1 \ldots x_n)P z_{m+1} \ldots z_q z_{q+1}$ and $C_i[ ] = (\lambda z_{q+i}. C_i[ ] ) U_i^q$. Then $C_i[M] \rightarrow_{\beta} M_i[y := P]$ for $1 \leq i \leq m$ and $C_i[M] \rightarrow_{\beta} z_i$ for $m + 1 \leq i \leq q$.

Also the following lemma generalises immediately from finite to infinite lambda calculus.

Lemma 12. Distinguishable terms are separable.

Proof. Let $M = \lambda x_1 \ldots x_n.y M_1 \ldots M_m$ and $N = \lambda x_1 \ldots x_{n'}.y' N_1 \ldots N_{n'}$. Then, $y \neq y'$ or $n - m \neq n' - m'$. We construct a discriminating context $C$ such that $C[M] \rightarrow_{\beta} P$ and $C[N] \rightarrow_{\beta} Q$ for any $P$ and $Q$ as follows:

1. Case $y \neq y'$. Suppose $n \geq n'$. Let $k = n - n'$. Then for $C[ ]$ we chose the context $(\lambda y'[ ]x_1 \ldots x_n)(\lambda x_1 \ldots x_m.P)(\lambda x_1 \ldots x_{m'}x_{m'+1} \ldots x_{m'+k}.Q)$.
2. Case $y = y'$ and $n - n' \neq m - m'$. Suppose $n \geq n'$. Since $n - n' \neq m - m'$, we can suppose $n > m' + n - n'$. Let $k = m - (m' + n - n')$. Now we chose for $C[ ]$ the context $(\lambda y'[ ]x_1 \ldots x_n(\lambda y_1 \ldots y_k.P)b_1 \ldots b_{k-1}Q)K^m$ where $y_1 \ldots y_k, b_1 \ldots b_{k-1}$ are fresh variables.

Lemma 13. Let $C$ be a finite context in $\beta_\perp$-normal form and let $M, N$ be distinguishable terms. Then $C[M]$ and $C[N]$ are separable.

Proof. We proceed by induction on the depth of the context $C$.
1. The base case, when $C[\ ] = [\ ]$, follows from Lemma 12.

2. The inductive case is when $C[\ ] = \lambda x_1 \ldots x_n . y M_1 \ldots C'[\ ] \ldots M_n$, Using Theorem 11, we can find a context to Böhm-out $C'[M]^\sigma$ and $C'[N]^\sigma$ where $\sigma = [y := P_q]$. Since $C$ is finite, we can find $q$ big enough so the depth of $C'^\sigma$ is equal to depth of $C'$ and the two terms $M^\sigma$ and $N^\sigma$ are still distinguishable.

\[ \square \]

**Theorem 14** (Böhm Theorem extended to infinite terms). If $M, N$ are two different $\beta\eta!$-normal forms without $\bot$ then $M$ and $N$ are separable.

**Proof.** By Theorem 8, there exist $P$ and $Q$ such that $M \xrightarrow{\eta-1} P \sim Q \xleftarrow{\eta-1} N$. Since $M \neq N$ and by confluence of $\beta\eta!$, we have that $P$ and $Q$ are different $\beta\perp$-normal forms. Let $d$ be the minimal depth where $P$ and $Q$ differ. We truncate the common part of $P$ and $Q$ at depth $d$ and obtain a finite context $C$ (possibly containing $\perp$) such that $C[P_0] \preceq P$ and $C[Q_0] \preceq Q$. Since $P$ and $Q$ are bisimilar we have that $P_0$ and $Q_0$ are distinguishable. By Lemma 13, we have that $C[P_0]$ and $C[Q_0]$ are separable. By Lemma 5, we see that $P$ and $Q$ are also separable. Since $\xrightarrow{\eta!}$ commutes with $\xrightarrow{\beta}$, we also have that $M$ and $N$ are separable.

\[ \square \]

We give some examples of separable terms and how to find the separating context:

1. The discriminating context for the terms $x$ and $z$ is $C[\ ] = (\lambda x z . [\ ]) PQ$.

2. Let $M = y$ and $N = \lambda x . y z$. Then, $M$ $\eta$-expands to $\lambda x . y x$ which is bisimilar to $\lambda x . y z$. The context $C[\ ] = (\lambda y . [\ ] x) U_1^1$ can be used to Böhm-out the variables $x$ and $z$. Then, we proceed as in the first part.

3. Let $M = yy(yx)$ and $N = yy(yz)$. In this case we would need to substitute the first occurrence of the variable $y$ by $U_2^2$ and the second occurrence by $U_1^1$. This is of course not possible. For this, Böhm invented the trick of the permutators. Since the greatest number of arguments of the variable $y$ is 2, we can make use of the permutator $P_2$. The context $C[\ ] = (\lambda y . [\ ] x) P_2 U_2^2$ does not Böhm-out exactly $yx$ and $yz$. What it gives is the result of substituting these terms by $P_2$, i.e. we get $P_2 x$ and $P_2 z$. Then, $C'[\ ] = [\ ] a U_1^2$ can be used to select $x$ and $z$ from $P_2 x$ and $P_2 z$.

4. Consider the infinite term $R_1$ defined using the following recurrence relation: $R_1 = y R_2 R_2$, $R_2 = y R_3 R_3$ and so on, where in general, $R_{k+1}$ adds $k$ arguments $R_k$ to $y$. Clearly, the number of arguments that $y$ can have in $R_1$ has no bound. To find the discriminating context for the terms $M = y R_1(yx)$ and $N = y R_1(yz)$, we first consider the truncations $M_0 = y \perp (yx)$ and $N_0 = y \perp (yz)$. Now a bound on the number of arguments of the variable $y$ is 2 and in this case we can, then, use
the permutator $P_2$ to Böhm-out the second argument of the first head variable. The discriminating context is, then, exactly the same as for the third part.

7 Equivalence between $\preceq_\eta$ and the contextual preorder $\subseteq_h$

We show that the relations $\preceq_\eta$ and $\subseteq_h$ are the same in the infinitary lambda calculus.

Lemma 15 (Propagation of $\subseteq_h$ to substitutions of subterms). Let $M, N \in \text{BT}(\Lambda^\infty)$ such that $\lambda x_1 \ldots x_n . y M_1 \ldots M_m$ and $N = \lambda x_1 \ldots x_n . y N_1 \ldots N_m$. If $M \subseteq_h N$ then $M_i^\sigma \subseteq_h N_i^\sigma$ for $1 \leq i \leq m$ where $\sigma = [y := P_q]$.

Proof. If follows from Theorem 11.

$M^\sigma \subseteq_h N^\sigma$ does not imply that $M \subseteq_h N$. Take $M = x$, $N = y x I$ and $\sigma = [y := P_1]$. As in the previous section, we have to find the appropriate permutator $P_q$ to Böhm-out the subterms of a term without changing their meaning.

Lemma 16. Let $M, N \in \text{BT}(\Lambda^\infty)$ such that $M$ is finite and $M \equiv_d N$ where $d$ is the depth of $M$. If $M \subseteq_h N$ then $M \preceq_\eta N$.

Proof. We proceed by induction on the depth of $M$. If the depth of $M$ is 0 then $M = \bot \preceq \subseteq N$. If the depth of $M$ is not 0 then $M \neq \bot$. Hence, $M = \lambda x_1 \ldots x_n . y M_1 \ldots M_m$ and $N = \lambda x_1 \ldots x_n . y' N_1 \ldots N_m'$. By Lemma 12, $y = y'$ and $n - n' = m - m'$. Since $M$ and $N$ have even eta expansions, we have that $n = n'$ and $m = m'$. By Lemma 15, $M_i^\sigma \subseteq_h N_i^\sigma$ for $1 \leq i \leq m$ and $\sigma = [y := P_q]$. Take $q$ greater than the number of symbols of $M$. Hence the depth of $M_i$ and $M_i^\sigma$ are the same and by induction hypothesis, $M_i^\sigma \preceq N_i^\sigma$ for $1 \leq i \leq m$. Again $q$ is big enough so $M_i \preceq N_i$. Hence $M \preceq N$.

Theorem 17. The following statements are equivalent for terms $M, N$ in $\Lambda^\infty$:

1. $M \preceq_\eta N$.

2. $M \subseteq_h N$.

Proof. (1) $\Rightarrow$ (2): Let $M' \preceq N'$ be such that $\text{BT}(M) \rightarrow_{\eta^{-1}} M'$ and $\text{BT}(N) \rightarrow_{\eta^{-1}} N'$. By confluence of $\beta \bot$ and the fact that $\rightarrow_{\eta^{-1}}$ preserves the property of having head normal form, if $C[M]$ has $\beta$-head normal form, so does $C[M']$. By Lemma 6, $C[N']$ has $\beta$-head normal form. Hence, again by confluence of $\beta \bot$ and $\rightarrow_{\eta^1}$ preserves the property of having head normal form, $C[M]$ has $\beta$-head normal form.

(2) $\Rightarrow$ (1): Suppose $M \subseteq_h N$. There exists $M'$ and $N'$ such that $\text{BT}(M) \rightarrow_{\eta^{-1}} M' \sim N'_{\eta^{-1}} \iff \text{BT}(N)$. Since we have that $M'^m \preceq_\eta M$ and
\(N =_\eta N',\) by the previous part, we also have that \(M'^n \subseteq_h M \subseteq_h N =_h N'.\) By Lemma 16, we have that \(M'^n \preceq N'.\) Hence,

\[
\text{BT}(M) \rightarrow \eta^{-1} M' = \bigcup \{M'^n \mid n \in \omega\} \preceq N' \eta^{-1} \iff \text{BT}(N)
\]

The previous result is part of the infinitary Characterisation Theorem for \(D_\infty\) in [11]. In particular, it says that two terms that have different \(\infty\eta\)-Böhm trees can be discriminated. The complication with the application of the Böhm-out technique to \(\infty\eta\)-Böhm trees is clear. The \(\infty\eta\)-Böhm trees of finite terms can be infinite. The proof in [1] deals with the problem using a relation \(\equiv_\alpha\) which coincides with the equality between \(\infty\eta\)-Böhm trees\(^2\). We have solved the problem in a slightly different way using a bisimulation and properties of the truncations.

8 Theories induced by infinitary lambda calculi

We have seen that confluent and normalising infinite extensions (where normal forms can now be infinite too!) induce a normal form function \(\text{NF} : \Lambda_\infty^\perp \rightarrow \Lambda_\perp^\infty\) that maps a term to its unique normal form. Each normal form function gives rise to an lambda theory:

\[
\text{Eq}(\text{NF}) = \{(M, N) \in \Lambda \times \Lambda \mid \text{NF}(M) = \text{NF}(N)\}
\]

Because \(\overline{T\Lambda} \subset \overline{W\Lambda} \subset \overline{H\Lambda}\) we get the following strict inclusions:

\[
\text{Eq(BerT)} \subset \text{Eq(LLT)} \subset \text{Eq(BT)} \subset \text{Eq(\eta BT)} \subset \text{Eq(\infty\eta BT)}
\]

We say that \(\text{Eq(NF)}\) is consistent if it is not the set of all equations \(\Lambda \times \Lambda\).

**Theorem 18.** If \(\text{Eq(NF)}\) is consistent and \(\text{Eq(BT)} \subseteq \text{Eq(NF)}\) then \(\text{Eq(NF)} \subseteq \text{Eq(\infty\eta BT)}\).

**Proof.** Suppose there are two finite terms \(M, N \in \Lambda\) such that \(\text{NF}(M) = \text{NF}(N)\) and \(\infty\eta\text{BT}(M) \neq \infty\eta\text{BT}(N)\). Then by Theorem 17 there exists a context \(C\) such that \(C[M] \beta\)-reduces to a head normal form and \(C[N]\) does not. Then it is easy to see that for any \(P \in \text{BT}(\Lambda_\perp^\infty)\) we have that \(P = \text{NF}(C[M]) = \text{NF}(C[N]) = \perp\).

It is tempting to conjecture that the smallest lambda theory that is induced by a confluent and normalising extension of the finite lambda calculus is the one related to the Berarducci trees.

**Conjecture 1.** \(\text{Eq(BerT)} \subseteq \text{Eq(NF)} \subseteq \text{Eq(\infty\eta BT)}\).

\(^2\text{The relation }\equiv_\alpha\text{ gives a syntactic characterisation of two }\perp\beta\text{-normal forms with the same }\infty\eta\text{-Böhm tree.}
The previous generalises to infinitary theories. These are defined as follows:

\[ \text{Eq}^\infty(\text{NF}) = \{(M, N) \in \Lambda_\infty^\perp \times \Lambda_\infty^\perp | \text{NF}(M) = \text{NF}(N)\} \]

It is clear that the previous theorem generalises with help of Theorem 17:

**Theorem 19.** If \( \text{Eq}^\infty(\text{NF}) \) is consistent and \( \text{Eq}^\infty(\text{BT}) \subseteq \text{Eq}^\infty(\text{NF}) \) then \( \text{Eq}^\infty(\text{NF}) \subseteq \text{Eq}^\infty(\infty\eta\text{BT}) \).

The exact relationship between finitary and infinitary theories is not clear yet.

**References**


Computationalism

Raymond Turner
The University of Essex

Computationalism is the research program in mathematical modelling that restricts itself to mathematics that can be implemented. It poses the question: how much can we do with such mathematics — and what can’t we do? It is not doctrinal: it does not say that mathematical modelling that goes beyond such mathematics is wrong or meaningless. This is unlike the restrictive programs behind finitism and constructivism. Indeed, our motivation is positive in that it is minimalist i.e., we seek to investigate, in the spirit of reverse mathematics, what we can do with notions that have a computational interpretation. Such minimalism has the potential to generate more accurate mathematical models: using set theory to do addition misses out on something.

There is another way, via the underlying theories of programming languages (PLs), to grasp what is involved. Every PL has an associated theory of data. Slightly more exactly, every PL has a collection of types: i.e., basic types (e.g. numbers, strings, characters) together with a battery of type constructors such as products, operations, sets and recursive types. In addition, each PL comes equipped with some basic relations and functions that operate over its types e.g., addition and less than for numbers, and concatenate, head and tail, for lists etc. Any axiomatisation of these types and their associated functions and relations, determines the underlying theory of data for the PL. However, while such theories may distinguish between different languages, they do not necessarily separate out the different styles of PL. The process of theory abstraction, abstracts the underlying notion of type, but does not mark whether a language is imperative, functional or logic based. Fortunately, for mathematical modelling, this is precisely what we require: the underlying axiomatic theory provides us with the ontological content of the language; it informs us of the supported building blocks of our mathematical modelling. In these terms, computationalism is restricted to using such theories of data. Indeed, given such a theory of data, we have a guarantee that we can implement the theory in the source PL.

Our objective in this paper is to examine this idea more carefully. We shall develop an abstract framework for such theories of data, provide some simple and, not so simple examples. In particular, for the latter, we shall study examples from program specification and natural language semantics. Finally, we shall say something about the relationship between computationalism, finitism and constructivism.