INTENSIONAL FIRST-ORDER LOGIC WITH TYPES

Chris Fox[a], Shalom Lappin[b*] and Carl Pollard[c]

[a]Dept. of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK
foxcj@essex.ac.uk
[b]Dept. of Computer Science, King's College London, Strand, London WC2R 2LS, UK
lappin@dcs.kcl.ac.uk
[c]Dept. of Linguistics, Ohio State University, 222 Oxley Hall, Columbus, OH 43210, USA
pollard@ling.ohio-state.edu

Abstract. The paper presents Property Theory with Curry Typing (PTCT) where the language of terms and well-formed formulæ are joined by a language of types. In addition to supporting fine-grained intensionality, the basic theory is essentially first-order, so that implementations using the theory can apply standard first-order theorem proving techniques. Some extensions to the type theory are discussed, type polymorphism, and enriching the system with sufficient number theory to account for quantifiers of proportion, such as "most."

1. Introduction

Higher-order theories that characterise intensions as functions from possible worlds to extensions have dominated formal semantics since Carnap (1947). There are problems with such theories:

1. the intensionality that they characterise is insufficiently fine grained—logically equivalent expressions are co-intensional and so intersubstitutable in all contexts, including the complements of propositional attitude predicates;

2. their higher-order nature does not aid implementation of automatic theorem proving.

3. the type system of such theories is very rigid, which does not accurately reflect the apparently flexible nature of the syntactic categories and semantic types of natural language.

All of these concerns have been addressed to a greater or lesser extent. There have been alternative proposals that have a more fine-grained analysis of intensionality which take propositions to be independent intensional entities, and truth to be a derived property. Such proposals include Thomason (1980), situation semantics (Barwise and Perry 1983; Barwise and Etchemendy 1990; Seligman and Moss 1997), Landman (1986), Lappin and Pollard (1999), Muskens (1995). There have also been proposals for explicitly first-order theories with fine-grained intensionality, such as property theory (Chierchia and Turner 1988; Turner 1987; Turner 1992). Work in categorial grammar, and related areas, has

*The second author's research is funded by grant number AN2687/APN 9387 from the Arts and Humanities Research Board of the United Kingdom.

addressed concerns with the requirements of type flexibility (Steedman 1996; Steedman 2000; Morrill 1994; Moortgat 1997; Carpenter 1998).

Here we sketch PTCT (Property Theory with Curry Typing), a formal semantic theory which seeks to address all three of the issues highlighted above simultaneously. Specifically PTCT supports fine-grained intensionality and flexible typing within a first-order framework.

Typed logics, based upon Church-style typing (Church 1940) typically have a basic set of (intensional) entities e, propositions P, and general functional types $T \Longrightarrow T'$. Such logics require that terms (or expressions) belong to exactly one type (monomorphism), and that there is no universal type. These requirements are imposed, in part, to avoid paradoxes in the underlying set-theoretic model: they lead to a ban on self-application. It is possible to avoid paradoxes whilst having a more flexible system of types.

In order to remain first-order, one option is to develop the logical theory as a first-order metatheory over representations of the appropriate propositions, properties, relations and other terms. This is the approach adopted by Property Theory (PT), which axiomatises Aczel's Frege Structures (Turner 1992; Aczel 1980). PT is essentially untyped, although it is possible to mimic types with a sortal system (Turner 1992). The language of terms is that of the untyped λ-calculus.

There are other logical systems that can be formulated as metatheories over the untyped λ-calculus, such as (constructive) programming logics. These logics include an explicit language of types. Rather than incorporating types into the language of the λ-calculus, as in Church (1940), such theories take the terms of untyped λ-calculus as given, and then adopt rules and axioms to assign types to terms (Curry and Feys 1958; Curry et al. 1958). This way of adding types to λ-calculus leads to a more expressive system (Hindley and Seldin 1986). As with PT, the metatheory is essentially first-order. The types will not lead to a higher-order system, provided that they do not include abstraction and universal quantification (over types).

In effect, PTCT can be thought of as adopting such a logic, and extending its expressiveness to allow the representation of propositions within the language of terms, and so rendering it an appropriate vehicle for computational semantics.

The version of PTCT presented here is essentially the theory presented in Fox, Lappin and Pollard (2002a), but extended with modal operators and with a type system that is enriched to allow a logical analysis of proportional quantifiers, such as "most." This adapts the proposal of Fox and Lappin (2001).[1]

2. The Language of PTCT

Adopting the above proposal leads to the following language of Property Theory with Curry Typing (PTCT):

(logical constants)	l	::=	$\hat{\wedge} \mid \hat{\vee} \mid \hat{\rightarrow} \mid \hat{\leftrightarrow} \mid \hat{\bot} \mid \hat{\forall} \mid \hat{\exists} \mid \hat{=} \mid \hat{\cong}_T \mid \epsilon \mid \hat{\Box} \mid \hat{\Diamond}$
(terms)	t	::=	$x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t$
(Types)	T	::=	$B \mid \mathsf{Prop} \mid T \Longrightarrow S$
(atomic wff)	α	::=	$(t = s) \mid \bot \mid t \in T \mid t \cong_T s$
(wff)	φ	::=	$\alpha \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \rightarrow \psi) \mid (\varphi \leftrightarrow \psi)$
			$\mid (\forall x\varphi) \mid (\exists x\varphi) \mid \Box\varphi \mid \Diamond\varphi \mid {}^{\mathsf{true}}t$

[1] Fox, Lappin and Pollard (2002b) present a higher-order logic with fine grained intensionality (FIL). We are in the process of exploring the correspondences between PTCT and FIL.

The language of terms is embellished with logical constants that will be used to "represent" propositions, properties and relations.

We have made the assertion of truth a wff, rather than a type assignment in order to avoid complications when considering extensions to the theory. There are two notions of equality: $\hat{=}$ is intended to correspond to intensional identity, and $\hat{\cong}$ corresponds to extensional equivalence. This follows the proposal of Fox and Lappin (2001).[2] Note that equivalence in PTCT does not entail intensional identity.

Not every term will have a type, but every term that has a type in this basic theory will correspond with an expression in conventional higher-order logic.

The resultant theory may be more powerful than is required just for natural language semantics, but it makes it more convenient to show correspondences between conventional higher-order logic and a first-order language. The Curry typing is useful when we consider polymorphic variants. Once we have an explicit language of types independent of any set-theoretic interpretation, it is more convenient to explore more unusual types and type constructions.

2.1 Logic of Wff

We can give the language of wff the standard behaviour of classical first-order logic.[3]

2.1.1 Modal Operators

The modal operators are governed by the usual rule of necessitation, plus the appropriate axioms (for example, the standard axioms for S4 or S5).

2.2 Rules for Identity and Equivalence

Both identity $=$ and equivalence $\cong_T$ are reflexive, symmetric and transitive, although these properties only hold for $\cong_T$ with terms that are of type T.

2.3 Rules for λ-terms

The λ-terms care governed by the conventional α, β axioms of the untyped calculus:

α $\lambda x.t = \lambda y.t[y/x]$ Provided y is not free in t

β $(\lambda x.t)y = t[y/x]$

It is possible to add either of the following axioms to give the extensional version of the untyped calculus:

Ext $\forall x(tx = sx) \rightarrow (t = s)$

η $\lambda x.(tx) = t$

[2]Gilmore (2001) constructs an intensional simple theory of types (ITT) in which an intensional $(=)$ and an extensional $(=_e)$ identity predicate are defined. His proposal differs from that of Fox and Lappin (2001) in that (i) the extensional identity predicate is not type general, and is only defined for propositions and predicates, and (ii) for Fox and Lappin, identity and equivalence are primitive, whereas Gilmore defines them in terms of substitution and bi-implication.

[3]A full formalisation, with both natural-deduction and tableau-style rules is given in Fox, Pollard and Lappin (2002a).

Note that adding **Ext** to the untyped λ-calculus is equivalent to adding η. Such extensionality of the the underlying λ-calculus would not undermine the intensionality of the system as these axioms govern the behaviour of intensional identity, and are independent of the extensional notion of equivalence in the language of wff.[4]

2.4 Type Assignment

Function types are governed by the axiom of general function spaces

GFS $f \in (S \Longrightarrow T) \leftrightarrow \forall x(x \in S \rightarrow fx \in T)$

2.4.1 Identity and Types

We can also have the following, which links identity with types:

$$(t \in T \wedge t' = t) \rightarrow t' \in T$$

2.5 Prop *Typing Rules*

We need to be able to determine which terms represent propositions.

2.5.1 Absurdity

$$\hat{\bot} \in \mathsf{Prop}$$

2.5.2 Binary Connectives

We illustrate with disjunction:

$$(t \in \mathsf{Prop} \wedge t' \in \mathsf{Prop}) \rightarrow (t \hat{\vee} t') \in \mathsf{Prop}$$

2.5.3 Quantifiers

We illustrate with universal quantification:

$$(\lambda x.t) \in (S \Longrightarrow \mathsf{Prop}) \rightarrow (\hat{\forall} x \epsilon S.t) \in \mathsf{Prop}$$

2.5.4 Modal Operators

We illustrate with necessity:

$$t \in \mathsf{Prop} \rightarrow \hat{\Box} t \in \mathsf{Prop}$$

2.5.5 Identity and Equivalence

$$(t \in T \wedge t' \in T) \rightarrow (t \mathrel{\hat{=}} t') \in \mathsf{Prop}$$
$$(t \in T \wedge t' \in T) \rightarrow (t \mathrel{\hat{\cong}}_T t') \in \mathsf{Prop}$$

[4] We are grateful to Paul Gilmore for pointing this out to us.

2.6 Rules for Truth $^{\mathsf{true}}$

The following rules give the truth conditions of those terms that represent propositions. The truth conditions themselves are stated as expressions in the language of wff.

2.6.1 Propositions are in Prop

$$^{\mathsf{true}}t \rightarrow t \in \mathsf{Prop}$$

2.6.2 Absurdity

$$^{\mathsf{true}}\hat{\bot} \rightarrow \bot$$

2.6.3 Binary Connectives

We illustrate with disjunction:

$$(t \in \mathsf{Prop} \wedge t' \in \mathsf{Prop}) \rightarrow ({}^{\mathsf{true}}(t \mathbin{\hat{\vee}} t') \leftrightarrow ({}^{\mathsf{true}}t \vee {}^{\mathsf{true}}t'))$$

2.6.4 Quantifiers

We illustrate with universal quantification:

$$\lambda x.t \in (S \Longrightarrow \mathsf{Prop}) \rightarrow ({}^{\mathsf{true}}(\hat{\forall} x \epsilon S.t) \leftrightarrow \forall x(x \in S \rightarrow {}^{\mathsf{true}}(t)))$$

2.6.5 Modal Operators

We illustrate with necessity:

$$t \in \mathsf{Prop} \rightarrow ({}^{\mathsf{true}}\hat{\Box}t \leftrightarrow \Box {}^{\mathsf{true}}t)$$

2.6.6 Identity

$$(t \in T \wedge t' \in T) \rightarrow ({}^{\mathsf{true}}(t \mathbin{\hat{=}} t') \leftrightarrow t = t')$$

2.6.7 Equivalence

$$(t \in T \wedge t' \in T) \rightarrow ({}^{\mathsf{true}}(t \mathbin{\hat{\cong}}_T t') \leftrightarrow (t \cong_T t'))$$

$$(t \in (S \Longrightarrow T) \wedge t' \in (S \Longrightarrow T)) \rightarrow ((t \cong_{(S \Longrightarrow T)} t') \leftrightarrow \forall s(s \in S \rightarrow ts \cong_T t's))$$

$$(t \in \mathsf{Prop} \wedge t' \in \mathsf{Prop}) \rightarrow (t \cong_{\mathsf{Prop}} t' \leftrightarrow ({}^{\mathsf{true}}t \leftrightarrow {}^{\mathsf{true}}t'))$$

We can derive:

$$(t \in (S \Longrightarrow \mathsf{Prop}) \wedge t' \in (S \Longrightarrow \mathsf{Prop})) \rightarrow ((t \cong t') \leftrightarrow \forall s(s \in S \rightarrow ({}^{\mathsf{true}}ts \leftrightarrow {}^{\mathsf{true}}t's)))$$

3. Extensions to the Language of Types

There are many ways in which this system can be extended and strengthened, for example we can define types for general functions, pairs ($\langle t, t' \rangle$) (and n-tuples), disjoint union ($t \oplus t'$), Cartesian product $t \ominus t'$, dependent product ($\pi(x)tt'$) and dependent sum ($\sigma(x)tt'$). We can define a Curry-Howard isomorphism, mapping from logical connectors to type operators. Here, we will look at a small selection of types and type operations that are of special interest. Universal type, and polymorphic types are useful in the analysis of type general expressions in natural language. Comprehension types, or separation types together with a universal type, will be required for the subsequent analysis of proportional quantifiers, such as "most."

3.1 Universal Type

We can add Δ to the language of types, which is governed by the following:

UT $x \in \Delta \leftrightarrow x = x$

At first, this might appear inconsistent as in standard ZF set theory, but we can see that it is not by noting that $x \in x$ is not in the language.

There have been proposals to use a universal type in the analysis of type-general expressions in natural language, ex amplified by "is fun," which accepts individuals, infinitives and gerunds as in the sentences "John is fun," "Tennis is fun," "Playing tennis is fun," and "To play tennis is fun," as well as conjunctions of arbitrary categories (Chierchia and Turner 1988).

3.2 Separation Types

It can be useful to have 'subtypes'. These can be defined using Separation Types. These embody wff within a type expression. Essentially, we add $\{x \in T : \varphi\}$ to the language of types. Such types are governed by the following:

SP $z \in \{x \in T : \varphi\} \leftrightarrow (z \in T \wedge \varphi[z/x])$

Adding **UT** (§3.1) by itself does not really seem to increase the power of the logic (it just appears to say something fairly vacuous), but when combined with **SP**, we then have the means to define pairs, disjoint union, Cartesian product, dependent sum, dependent product and more, including 'classical' intersection ($\cap$), union ($\cup$) and difference ($-$). For example, we can define intersection, union and difference by way of the following:

$$\begin{array}{lll} T \cap S & =_{\mathsf{def}} & \{T \in \Delta : x \in S \wedge x \in T\} \\ T \cup S & =_{\mathsf{def}} & \{T \in \Delta : x \in S \vee x \in T\} \\ T - S & =_{\mathsf{def}} & \{T \in \Delta : x \in S \wedge x \notin T\} \end{array}$$

3.3 Comprehension Types

Comprehension allows us to capture a type just using a wff (property). We add $\{\varphi\}$ to the language of types, and the following axiom:

COMP $z \in \{x : \varphi\} \leftrightarrow \varphi[z/x]$

It turns out that **COMP** is equivalent to **SP+UT** (see Turner (1991) for example). The proofs are trivial.

Theorem 1 ***COMP*** *supports* ***SP*** *and* ***UT***

This can be shown by adopting the definitions $\{x \in S : \varphi\} =_{\mathsf{def}} \{x : x \in S \wedge \varphi\}$ and $\Delta =_{\mathsf{def}} \{x : x = x\}$.

Theorem 2 ***SP+UT*** *supports* ***COMP***

This is demonstrated by adopting the definition $\{x : \varphi\} =_{\mathsf{def}} \{x \in \Delta : \varphi\}$.

3.4 Polymorphic Types

To add polymorphic types, first we have to enrich the language of types to include type variables X, and our language of wff to include quantification over types (using these type variables) $\forall X\varphi, \exists X\varphi$.

We also need to have comprehension types of the form $\{x : \varphi'\}$, where φ' are first order, and do not contain bound type variables. This excludes expressions of the form $\{x : \exists X(x \in X)\}$ (the set of terms that have a type).

By adopting these restricted comprehension types, rather than full second-order comprehension, the system remains first-order: type variables range over terms corresponding to types, and the types themselves are not permitted to embody quantification over types.[5]

The proof rules for quantification over types are as expected.

Now we can express universal (implicit) polymorphic types, by adding $\Pi X.T$ to the language of types, which is governed by the following axiom:

PM $f \in \Pi X.T \leftrightarrow \forall X(f \in T)$

Such types represent infinite intersections of types. As an example, $\Pi X(X \Longrightarrow X)$ is the type of function in every general function space (such as $\lambda x.x$). The expression "is fun" can now be given the type $\Pi X.(X \Longrightarrow P)$.

To allow conjunctions of terms and nominalised expressions (such as gerunds and infinitives) to appear as arguments to type general verbs, natural language conjunction "and" can be given the type $\Pi X(X \Longrightarrow X \Longrightarrow X)$.

The two approaches to dealing with nominalisation (polymorphic types, and a universal type) are not mutually exclusive: if we adopt additional extensions, we can obtain a theory in which there is both a universal type and polymorphic types.

4. Proportional Quantifiers

We can adapt the approach of Fox and Lappin (2001) for number quantifiers to PTCT. Things are slightly complicated by the fact that terms can be of more than one type. We assume that we are judging whether most ps are qs when p and q are felicitously interpreted as properties of objects of type B.

Note that numeric equality of terms is judged independently of syntactic identity; this is an intensional number theory.

[5]Note that if we were to adopt existential types (Abstract Data Types) of the form $\Sigma X.T$—where $f \in \Sigma X.T \leftrightarrow \exists X(f \in T)$—in addition to first-order comprehension and polymorphic types, then we would produce a system that was equivalent to PTCT with second-order comprehension. See Turner (1991) for the proof.

4.1 Syntax for Numbers

The following additions can be made to PTCT:

Terms $0 \mid \mathit{succ} \mid \mathit{pred} \mid \mathit{add} \mid \mathit{mult} \mid \hat{\mathit{most}} \mid \mid \cdot \mid_B$

Types Num

Wffs $\mathit{zero}(t) \mid t \cong_{\mathsf{Num}} t' \mid t <_{\mathsf{Num}} t' \mid \mathit{most}(p)(q)$

In the language of types, Num is the type of terms that represent numbers.

The terms *succ*, *pred*, *add* and *mult* represent, respectively, the operations of successor, predecessor, addition, and multiplication, and 0 is the term that represents zero. The term $\mid \cdot \mid_B$ represents a cardinality operator. Given a term p of type $(B \Longrightarrow \mathsf{Prop})$, $|p|_B$ is intended to represent the number of terms t of type B for which ${}^{\mathsf{true}}pt$ holds.

In the language of wffs, $\mathit{zero}(t)$ tests whether the term t is numerically equal to 0, $t \cong_{\mathsf{Num}} t'$ determines whether the terms t, t' are numerically equal (corresponding to extensional equivalence for numbers), and $t <_{\mathsf{Num}} t'$ determines whether the term t is numerically less than the term t'. The wff $\mathit{most}(t)(t')$ represents the proposition "most ts are t's," and $\hat{\mathit{most}}$ is the term representation of the same quantifier. The axioms governing the behaviour of numbers must be formulated in such a way that t, t', p and q are restricted to terms of appropriate types.

4.2 Axioms for Numbers

We can add the following axioms to obtain the expected behaviour, adapting the usual Peano axioms of first-order arithmetic to PTCT.

4.3 Axioms for $<_{\mathsf{Num}}$

In order to formalise the truth conditions of $\mathit{most}(p)(q)$, we need a characterisation of what it means for one term to be numerically less than another.

$$\begin{gathered} y \in \mathsf{Num} \rightarrow 0 <_{\mathsf{Num}} \mathit{succ}(y) \\ x \in \mathsf{Num} \rightarrow x \not<_{\mathsf{Num}} 0 \\ x \in \mathsf{Num} \wedge y \in \mathsf{Num} \rightarrow (\mathit{succ}(x) <_{\mathsf{Num}} \mathit{succ}(y) \leftrightarrow x <_{\mathsf{Num}} y) \end{gathered}$$

4.4 "Most" in PTCT

To obtain an analysis of $\mathit{most}(p)(q)$, we must first define the cardinality of properties. This recursive definition exploits comprehension types (§3.3), together with separation types to define differences (§3.2).

$$\begin{gathered} p \in (B \Longrightarrow \mathsf{Prop}) \wedge {\sim}\exists x(x \in B \wedge {}^{\mathsf{true}}px) \rightarrow |p|_B \cong_{\mathsf{Num}} 0 \\ p \in (B \Longrightarrow \mathsf{Prop}) \wedge b \in B \wedge {}^{\mathsf{true}}pb \rightarrow |p|_B \cong_{\mathsf{Num}} \mathit{add}(|p - \{x.x \mathrel{\hat{\cong}}_B b\}|_B)(\mathit{succ}(0)) \end{gathered}$$

Now we can give the appropriate behaviour to $\mathit{most}(p)(q)$ by way of the following:

$$\begin{gathered} p \in (B \Longrightarrow \mathsf{Prop}) \wedge q \in (B \Longrightarrow \mathsf{Prop}) \rightarrow \\ \mathit{most}(p)(q) \leftrightarrow |\{x \in B.{}^{\mathsf{true}}px \wedge {\sim}{}^{\mathsf{true}}qx\}|_B <_{\mathsf{Num}} |\{x \in B.{}^{\mathsf{true}}px \wedge {}^{\mathsf{true}}qx\}|_B \end{gathered}$$

In this analysis, it is presumed that all terms representing the singular natural language denotable objects are of the basic type B (although they may also belong to other types). It would be possible to devise an explicitly polymorphic (§3.4) version of this analysis.

5. Future Work

There are extensions to PTCT that are of relevance to natural language semantics. We could incorporate separation types and dependent types into the language of types in order to model discourse phenomena (Ranta 1991; Ranta 1994; Dávila-Pérez 1995; Fox 1994; Fox 2000). Of more immediate concern is a proof of consistency. Fox, Lappin and Pollard (2002a) sketch an approach to demonstrating the consistency of PTCT by finding sound and complete mappings between PTCT and a higher-order logic.[6] An alternative would be to build directly upon a model for the Curry-typed λ-calculus.

6. Conclusions

The paper presents a highly intensional theory that can emulate some key aspects of typed logics within a first-order framework. In this system, extensional equivalence does not entail intensional identity. The theory is sufficiently expressive to lend itself to extensions that incorporate universal and polymorphic types (which can be used to model some natural language nominalisation phenomena). The paper shows how the theory can be extended with modal operators, and how an intensional number theory can be formalised to allow an analysis of the full range of generalised quantifiers.

References

Aczel, P.: 1980, Frege structures and the notions of proposition, truth and set, in Barwise, Keisler, and Keenan (eds.), *The Kleene Symposium*, North Holland Studies in Logic, pp 31–39, North Holland

Barwise, J. and Etchemendy, J.: 1990, Information, infons, and inference, in R. Cooper, K. Mukai, and J. Perry (eds.), *Situation Theory and Its Applications*, Vol. 1, pp 33–78, CSLI, Stanford, CA

Barwise, J. and Perry, J.: 1983, *Situations and Attitudes*, MIT Press (Bradford Books), Cambridge, MA

Carnap, R.: 1947, *Meaning and Necessity*, University of Chicago Press, Chicago

Carpenter, B.: 1998, *Type-Logical Semantics*, MIT Press

Chierchia, G. and Turner, R.: 1988, Semantics and property theory, *Linguistics and Philosophy* 11, 261–302

Church, A.: 1940, A formulation of the simple theory of types, *Journal of Symbolic Logic* 5, 56–68

Curry, H. and Feys, R.: 1958, *Combinatory Logic*, Vol. 1 of *Studies in Logic*, North Holland

Curry, H., Hindley, R., and Seldin, J.: 1958, *Combinatory Logic*, Vol. 2 of *Studies in Logic*, North Holland

Dávila-Pérez, R.: 1995, *Semantics and parsing in intuitionistic categorial grammar*, Ph.D. thesis, University of Essex

Fox, C.: 1994, Discourse representation, type theory and property theory, in H. Bunt, R. Muskens, and G. Rentier (eds.), *Proceedings of the International Workshop on Computational Semantics*, pp 71–80, ITK, Tilburg

[6] This follows the proof of soundness of HOLCU (a higher-order Curry-typed logic) described by Turner (1997).

Fox, C.: 2000, *The Ontology of Language*, CSLI Lecture Notes, CSLI, Stanford

Fox, C. and Lappin, S.: 2001, A framework for the hyperintensional semantics of natural language with two implementations, in P. de Groote, G. Morrill, and C. Retore (eds.), *Logical Aspects of Computational Linguistics*, Springer Lecture Notes in Artificial Intelligence, pp 175–192, Springer-Verlag, Berlin and New York

Fox, C., Lappin, S., and Pollard, C.: 2002a, First-order Curry-typed logic for natural language semantics, in *Proceedings of the Workshop on Natural Language Understanding and Logic Programming*, Copenhagen, Denmark, (to appear, at time of writing)

Fox, C., Lappin, S., and Pollard, C.: 2002b, A higher-order fine-grained logic for intensional semantics, in *Proceedings of the Seventh Symposium on Logic and Language*, Pécs, Hungary, (to appear, at time of writing)

Gilmore, P.: 2001, An intensional type theory: Motivation and cut-elimination, *Journal of Symbolic Logic* 66, 383–400

Hindley, R. and Seldin, J.: 1986, *Introduction to Combinators and the Lambda Calculus*, London Mathematical Society Student Texts 1, Cambridge University Press

Landman, F.: 1986, Pegs and alecs, in *Towards a Theory of Information. The Status of Partial Objects in Semantics*, Groningen-Amsterdam Studies in Semantics, pp 97–136, Foris, Dordrecht

Lappin, S. and Pollard, C.: 1999, *A Hyperintensional Theory of Natural Language Interpretation without Indices or Situations*, ms., King's College, London and Ohio State University

Moortgat, M.: 1997, Categorial type logics, in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, pp 93–177, Elsevier, North Holland, Amsterdam

Morrill, G.: 1994, *Type Lexical Grammar*, Kluwer

Muskens, R.: 1995, *Meaning and Partiality*, CSLI and FOLLI, Stanford, CA

Ranta, A.: 1991, Intuitionistic categorial grammar, *Linguistics and Philosophy* 14, 203–239

Ranta, A.: 1994, *Type Theoretic Grammar*, Oxford University Press

Seligman, J. and Moss, L.: 1997, Situation theory, in J. van Bentham and A. ter Meulen (eds.), *Handbook of Logic and Language*, Elsevier, North Holland, Amsterdam

Steedman, M.: 1996, *Surface Structure and Interpretation*, MIT Press

Steedman, M.: 2000, *Syntactic Process*, MIT Press

Thomason, R.: 1980, A modeltheory for propositional attitudes, *Linguistics and Philosophy* 4, 47–70

Turner, R.: 1987, A theory of properties, *Journal of Symbolic Logic* 52(2), 455–472

Turner, R.: 1991, *Constructive Foundations for Functional Languages*, McGraw-Hill

Turner, R.: 1992, Properties, propositions and semantic theory, in M. Rosner and R. Johnson (eds.), *Computational Linguistics and Formal Semantics*, Studies in Natural Language Processing, pp 159–180, Cambridge University Press, Cambridge

Turner, R.: 1997, Types, in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, pp 535–586, Elsevier, North Holland, Amsterdam