An Expressive First-Order Logic with Flexible Typing for Natural Language Semantics

CHRIS FOX, *Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, Essex, CO4 3SQ.*
Email: foxcj@essex.ac.uk

SHALOM LAPPIN, *Department of Computer Science, King's College London, Strand, London WC2R 2LS.*
Email: lappin@dcs.kcl.ac.uk

Abstract

We present Property Theory with Curry Typing (PTCT), an intensional first-order logic for natural language semantics. PTCT permits fine-grained specifications of meaning. It also supports polymorphic types and separation types.[1] We develop an intensional number theory within PTCT in order to represent proportional generalized quantifiers like *most*. We use the type system and our treatment of generalized quantifiers in natural language to construct a type-theoretic approach to pronominal anaphora that avoids some of the difficulties that undermine previous type-theoretic analyses of this phenomenon.

1 Introduction

Much work in formal and computational semantics has been based on two assumptions. First, higher-order logic and type theory are necessary to achieve the expressive power required to represent the semantic properties of natural language. Second, intensions can best be characterized as functions from possible worlds (or situations) to extensions. We propose a logic that dispenses with both these assumptions and argue that it offers a more appropriate framework for formally representing the semantics of natural language. Property Theory with Curry Typing (PTCT) is a first-order system that incorporates flexible Curry typing. It is radically intensional in that its model theory takes intensions (represented as expressions in a term language) as primary rather than as defined in terms of extensions.

Let us briefly consider each of the two assumptions in turn. Montague's [43] IL is developed within an intensional version of Church's [13] Simple Theory of Types (SST) employing the typed λ-calculus with intension and extension forming operators, and modal operators. Gallin [21] simplifies this system by replacing the intensional and extensional operators with the basic type s. Barwise and Cooper [4] invoke a set theoretic counterpart of SST to develop an account of generalized quantifiers in natural language, and this framework is applied by, among others Keenan and Satavi [31], Keenan and Westerståhl [32], and Lappin [35]. Cooper [14] uses a version of the typed λ-calculus to construct a situation theoretic treatment of generalized quantifiers. Groenendijk and Stokhof [24] integrate their dynamic logic [25] into a variant of Montague's IL in order to represent certain kinds of discourse anaphora. The main reason for

[1] Separation types yield a form of *subtype*.

employing higher-order type systems is that they contain functional types, which are required to provide adequate semantic representations for several syntactic categories. A unified treatment of NPs is possible if they are assigned the type of generalized quantifier (functions from properties to propositions, or truth-values). Adjectival and adverbial modifiers correspond to functions from properties to properties (or sets). Sentential modifiers are interpreted as functions from propositions to propositions (truth-values).

Through its use of Curry typing PTCT contains the full range of functional types. Moreover, it allows for limited (non-iterated) polymorphism, so that it captures the fact that certain natural language expressions, like coordination, correspond to functional types that apply to a variety of distinct argument types. However, quantification in the language of well-formed formulas in PTCT is first-order. Quantification over functional entities and types is expressed through quantification over terms, which are elements of the domain. The model theory is an extension of a standard extensional model for the untyped λ-calculus, with the additional expressive power of Curry types added through a distinct component of the language of terms. The logic remains first-order in its formal power.

The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap [8]. Montague [43] gives it detailed formal expression. As has been frequently noted, this treatment of intensions yields a theory of meaning that is not sufficiently fine-grained. It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts, including the complements of propositional attitude predicates. However, such substitutions do not always hold. So, for example, the sentences in 1 are logically equivalent, but the non-equivalence of the sentences in 2 shows that those in 1 are not cointensional.

(1) (a) Every prime number is divisible only by itself and 1.
$\Leftrightarrow$
(b) If $A \subseteq B$ and $B \subseteq A$, then $A = B$.

(2) (a) John believes that every prime number is divisible only by itself and 1.
(b) John believes that if $A \subseteq B$ and $B \subseteq A$, then $A = B$.

Most attempts to construct fine-grained intensional theories have followed one of two approaches. The first involves positing impossible worlds or situations in which at least some some classically valid formulas do not hold. This enlargement of the set of worlds/situations permits distinctions to be made between expressions that are equivalent across the set of possible worlds/situations. Variants of this view are presented by Barwise and Etchemendy [5], Barwise [3], Muskens [44], and Gregory [23]. On the second approach, a hyperintensional model theory is constructed which appears to permit distinctions among logically equivalent expressions. However, a proof theory is not explicitly specified. As a result it is not clear how the system prevents intensional identity from collapsing into logical equivalence. This difficulty arises with the theories proposed by Thomason [52], Cresswell [15], Landman [33], and Larson and Segal [39].[2]

There are exceptions, such as Bealer [6], Turner [53, 55] and Zalta [58]. Notable among these, at least in the context of the present discussion, is Bealer [6], who proposes a first-order logic with an abstraction operator that forms names of intensional entities. Its model theory posits

[2] Fox and Lappin [19] and Lappin and Pollard [38] discuss these approaches in detail. They also show that Lappin and Pollard's [37] attempt to construct a hyperintensional semantics on the basis of a system that uses a free topos for its model theory leads to radical intensional collapse.

a domain of intensions (properties, propositions, etc.), and its proof theory (on one version) is designed to prevent the reduction of intensional identity to logical equivalence. Possible worlds or situations play no role in the theory. Bealer's logic does not contain types beyond those of classical first-order logic. Due to the absence of functional types it cannot express the full range of generalized quantifiers and modifiers that appear in natural language. It is also unable to handle polymorphism. Therefore, it lacks the expressive power required for an adequate framework for NL semantic representation.[3]

As in Bealer's logic, our model theory for PTCT takes intensions to be basic entities rather than functions from possible worlds (situations) to extensions. Our proof theory prevents the reduction of provable equivalence to identity, and intensions are represented independently of modality. However, PTCT supports functional types and (limited) polymorphism, and so it has the required expressive power. Fine-grained intensionality is achieved in a first-order system that does not require (im)possible worlds but is rich enough to sustain functional, separation, dependent, and comprehension types.[4] We prove that the basic PTCT logic (without number theory) is sound and complete.

In Section 11 we exhibit the expressive power of flexible Curry typing in PTCT by defining an intensional number theory, which we employ to specify the cardinality of properties. We then characterize generalized quantifiers as cardinality relations between properties. Finally, in Section 12, we combine our representation of generalized quantifiers with our treatment of separation and dependent types to present a unified treatment of pronominal anaphora that handles discourse anaphora relations dealt with by theories of dynamic semantics.

2 PTCT: A Curry-Typed Theory

There are various ways in which we can view PTCT. In terms of its presentation, it can be regarded as a development of Property Theory (PT) [55]. The differences being that PTCT has a fully-fledged language of types, whereas PT typically mimics types using properties. The addition of types requires changes to the syntax of the language, and the axioms, in order to give the appropriate behaviour to expressions in which types appear.

These additions mean that we can represent quantified propositions that explicitly restrict the domain of quantification. For example, if we wish to represent the sentence

John believe everything that Mary believes.

then the quantifier representing "everything" can be restricted to range only over propositions. In PT, such restrictions can only be expressed in the language of wffs.

There are other differences. PT has a universal type. It has been argued that this is appropriate for dealing with certain phenomena such as conjunctions, gerunds and infinitives [10, 11]. However, it could be argued that such an approach is unduly permissive, in that it imposes no constraints on the relevant types of the arguments and conjuncts. As we shall see, PTCT can

[3] There are additional formal problems with Bealer's logic. His model theory seems to permit the generation of semantic paradoxes. Moreover, the iterated use of his abstraction operator to generate singular terms constructed out of predicates that apply to property terms creates terms whose denotations seem to be computable only through the application of functions to intensions, where these functions are themselves elements of the domain of the model. This result runs counter to Bealer's claim to have constructed a model theory in which functions are excluded from any domain. We discuss these issues in Fox and Lappin [19].

[4] The Church-typed, higher-order, intensional logic FIL, presented in Fox, Lappin and Pollard [20] also allows for fine-grained intensionality independently of (im)possible worlds. However, it does not have separation, dependent, or comprehension types or polymorphism.

deal with most salient examples using polymorphic types. This allows us to impose linguistically motivated constraints on the types of the arguments and conjuncts.

A further difference is that unlike PT, PTCT does not permit self-application. This constraint is imposed to prevent paradoxes, that would otherwise follow from the nature of the type system (in particular, from having a type corresponding to propositions that can appear in terms within the theory).[5]

The presentation of PTCT differs from Higher-Order Logic, in that PTCT can be seen to adopt a *meta-theoretic* characterisation of a semantic theory. Basic derivations are presented as axioms in a meta-theory. To convey the basic idea, an inference of the form:

$$a, b \vdash c$$

is presented as something like the axiom

$$\mathsf{T}(a) \wedge \mathsf{T}(b) \rightarrow \mathsf{T}(c)$$

where $\wedge$ and $\rightarrow$ are meta-level connectives, and $\mathsf{T}(a)$ is a meta-theoretic proposition that asserts that the object level proposition a is true.

Formulating a logic by way of a meta-theory is not novel. See for example Turner [55], Smith [49]. Some of the differences with the Property Theory of Turner [55] have already been discussed. Smith [49] indicates how Martin-Löf's Type Theory (MLTT) can be interpreted in a Frege structure [1]. Although Smith internalises a "universe" of types, such types are precisely the small types of the Frege structure, which can be internalised in PT, thus the limitations of the typing are exactly those of PT, with the additional difference that Smith is capturing an intuitionistic theory rather than a classical one. Note that Turner [55] subsumes Smith [49].

The meta-theory for PTCT is first-order in character. By giving it an appropriate model, it makes clear that PTCT is a first-order equivalent language.[6]

Another departure from more conventional approaches lies in the nature of the types. Broadly speaking, types can be either Church-style or Curry-style. As already discussed, unlike most typed logics that have been applied in natural language semantics, PTCT adopts Curry typing for flexibility.

Conceptually, there is fairly close relationship between the underlying logic captured by the meta-theory of PTCT, and Turner's IHOL [56].

3 PTCT: Syntax of the basic theory

The core language of PTCT consists of the following sub-languages:

(3) Terms $t ::= x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t$
(logical constants) $l ::= \tilde{\wedge} \mid \hat{\wedge} \mid \hat{\vee} \mid \hat{\rightarrow} \mid \hat{\leftrightarrow} \mid \hat{\forall} \mid \hat{\exists} \mid \hat{=}_T \mid \hat{\cong}_T \mid \epsilon$

(4) Types $T ::= B \mid \mathsf{Prop} \mid T \Longrightarrow T'$

(5) Wff $\varphi ::= \alpha \mid \sim\varphi \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi') \mid (\varphi \rightarrow \varphi') \mid (\varphi \leftrightarrow \varphi') \mid (\forall x\varphi) \mid (\exists x\varphi)$
(atomic wff) $\alpha ::= (t =_T s) \mid t \in T \mid t \cong_T s \mid {}^{\mathsf{True}}t$

[5] If a system both has an impredicative notion of proposition, and allows terms involving self application to form propositions, then it will be inconsistent. PT allows self-application, but has a predicative notion of proposition. With PTCT, because the type that corresponds to propositions can appear within the terms of PTCT, its characterisation of propositions is impredicative, so self-application is not permitted.

[6] This is somewhat akin to giving a Henkin general model for a logic that is expressed in higher-order terms [27].

The language of terms is the untyped λ-calculus, enriched with constants c, and logical constants l. Types T are also terms.[7] The presence of constants c allows us to introduce additional λ-calculus terms in subsequent discussions without always being obliged to give an encoding of the new notion in pure (i.e. constant-free) λ-calculus.

The language of types contains the basic type of individuals B, propositions Prop, and general function space types $T \Longrightarrow T'$.

The language of wffs is a first-order language together with type judgements $t \in T$, typed identity $=_T$ (intensional equality) and equivalence $\cong_T$ (extensional equality), and truth judgments $^{\mathsf{True}}t$. In principle at least, the latter can apply to all terms t, but it only makes sense to apply it to terms that are intended to represent propositions (that is, $t \in \mathsf{Prop}$). To do otherwise would be to entertain a category mistake.

The language of terms is used to *represent* the interpretations of natural language expressions. It has no internal logic. By this, we mean that it is not possible to perform inferences *directly* with term representations of propositions. Inferences are concerned with establishing relationships between the truth conditions of propositions and terms representing propositions. In the case of PTCT, such relationships are expressed in the language of wffs, which acts as a *meta-language* for the language of terms.

With an appropriate proof theory, the simple language of types together with the language of terms can be combined to produce a Curry-typed λ-calculus. The first-order language of wffs is used to formulate type judgements for terms, and truth conditions for those terms judged to be in Prop.

It is important to distinguish between the notion of a proposition itself (in the language of wff), and that of a term that *represents* a proposition (in the language of terms). $^{\mathsf{True}}(t)$ will be a true wff whenever the proposition represented by the term t is true, and a false wff whenever the proposition represented by t is false. The representation of a proposition t ($\in \mathsf{Prop}$) is distinct from its truth conditions ($^{\mathsf{True}}(t)$).

Later, in Section 7, we will consider some extensions to the theory.

4 A Proof Theory for PTCT

We construct a tableau proof theory for PTCT.[8] Its rules can be broken down into the following kinds.

- The basic connectives of the wff: These have the standard classical first-order behaviour.
- Identity of terms ($=_T$): These are the usual rules of the untyped λ-calculus with α, β and η reduction, with the constraint that the related terms are known to be of the same type.
- Typing of λ-terms: These are essentially the rules of the Curry-typed calculus, augmented with rules governing those terms that represent propositions (Prop).
- Truth conditions for Propositions: Additional rules for the language of wffs that govern the truth conditions of terms in Prop (which represent propositions).

[7] It is possible to consider reformulating the theory using a typed *lambda*-calculus with abstraction of the form $\lambda x : T(t)$. Separation types, which are introduced later, could then be treated directly as sugar for a typed λ-abstract.

In the case of the untyped calculus used here, it is reasonable to consider adding untyped λ-identity as an atomic wff ($t =_\lambda t$), rather than adding the usual α, β, η rules as substitution rules in the tableau rules for the system.

[8] For an introduction to tableau proof procedures for first-order logic with identity see Jeffrey [28], based on work by Beth [7] and Smullyan [50]. Fitting [17] presents an implemented tableau theorem prover for first-order logic with identity, and he discusses its complexity properties.

- Equivalence ($\cong_T$): The theory has an internal notion of extensional equivalence which is given the expected behaviour.

Below we present the tableau rules for PTCT. The symbol * indicates that the corresponding proposition has been used, and does not need to be considered again.

4.1 General Rule Forms

There are four general kinds of rules, A, B, C, and D, as follows.

A-Rule

$$\begin{array}{c} A^* \\ | \\ A_1 \\ A_2 \end{array}$$

B-Rule

$$\begin{array}{ccc} & B^* & \\ \swarrow & & \searrow \\ B_1 & & B_2 \end{array}$$

$C(d)$-Rule

$$\begin{array}{c} C \\ | \\ C[d/x] \end{array}$$

where d occurs on the path (or is the only constant occurring in the path)

$D(d)$-Rule

$$\begin{array}{c} D \\ | \\ D[d/x] \end{array}$$

where d does not occur previously on the path

In addition, there are also closure rules which indicate when a branch is closed due to some contradiction. Partly due to the typing system, and partly to simplify the proof of completeness, additional closure rules are given in addition to the usual rule that states a path is closed if it contains the formulae A and $\sim A$ for any proposition A. Rules that require two premises are represented as single premise rules with a side-condition.

The specific rules for Wffs (the core classical logic, λ-equivalence, equivalence and identity), Types and Propositions are detailed next.

4.2 Rules for Wffs

4.2.1 Classical Rules

The following rules are standard, as we adopt the usual rules for the core logic of wff.

A-Rules

Conjunction

$$\begin{array}{c} (\varphi \wedge \psi)* \\ | \\ \varphi \\ \psi \end{array}$$

Bi-implication

$$\begin{array}{c} (\varphi \leftrightarrow \psi)* \\ | \\ (\varphi \rightarrow \psi) \\ (\psi \rightarrow \varphi) \end{array}$$

Disjunction′

$$\begin{array}{c} \sim(\varphi \vee \psi)* \\ | \\ \sim\varphi \\ \sim\psi \end{array}$$

Implication′

$$\begin{array}{c} \sim(\varphi \rightarrow \psi)* \\ | \\ \varphi \\ \sim\psi \end{array}$$

Double Negation

$$\begin{array}{c} \sim\sim\varphi* \\ | \\ \varphi \end{array}$$

B-**Rules**

Disjunction

$$\begin{array}{ccc} & (\varphi \vee \psi)* & \\ \swarrow & & \searrow \\ \varphi & & \psi \end{array}$$

Conjunction′

$$\begin{array}{ccc} & \sim(\varphi \wedge \psi)* & \\ \swarrow & & \searrow \\ \sim\varphi & & \sim\psi \end{array}$$

Implication

$$\begin{array}{ccc} & (\varphi \rightarrow \psi)* & \\ \swarrow & & \searrow \\ \sim\varphi & & \psi \end{array}$$

Bi-implication′

$$\begin{array}{ccc} & \sim(\varphi \leftrightarrow \psi)* & \\ \swarrow & & \searrow \\ \sim(\varphi \rightarrow \psi) & & \sim(\psi \rightarrow \varphi) \end{array}$$

C-**Rules**

$\forall$ **Quantification**

$$\begin{array}{c} \forall x.\varphi \\ | \\ \varphi[k/x] \end{array}$$

$\exists$ **Quantification′**

$$\begin{array}{c} \sim\exists x.\varphi \\ | \\ \sim\varphi[d/x] \end{array}$$

D-**Rules**

$\exists$ **Quantification**

$$\begin{array}{c} \exists x.\varphi \\ | \\ \varphi[d/x] \end{array}$$

$\forall$ **Quantification′**

$$\begin{array}{c} \sim\forall x.\varphi \\ | \\ \sim\varphi[d/x] \end{array}$$

$\perp$-**Rules**

Path closure can be derived under the following circumstances.

Contradiction (1)	**Contradiction (2)**
φ closes a branch if $\sim\varphi$	$\sim\varphi$ closes a branch if φ

Now that we have the standard rules in place, we can consider the rules for substituting λ-equivalent terms, and, subsequently, the rules that govern the PTCT notions of equivalence and identity.

4.2.2 λ Rules

These rules implement the usual α, β, and η rules of the λ-calculus as substitution rules.

***A*-Rules**

β-reduction

$$\begin{array}{l} \ldots(\lambda u.t)(a)\ldots \\ \quad\big| \\ \ldots t[a/u]\ldots \end{array}$$

η-reduction

$$\begin{array}{ll} \ldots\lambda u.(tu)\ldots & u \text{ not free in } t \\ \quad\big| & \\ \ldots t\ldots & \end{array}$$

***C*-Rules**

α-reduction

$$\begin{array}{ll} \ldots\lambda x.t\ldots & d \text{ not free in } t \\ \quad\big| & \\ \ldots\lambda d.t[d/x]\ldots & \end{array}$$

4.2.3 Equivalence and Identity Rules

These rules are intended to give the appropriate logical behaviour to statements of equivalence in the language of wffs.

***A*-Rules**

Substitution

$$\begin{array}{ll} \varphi & t =_T s \text{ or } s =_T t \\ \quad\big| & \\ \varphi[t/s] & \end{array}$$

Identity

$$\begin{array}{l} t \in T \\ \quad\big| \\ t =_T t \end{array}$$

Equivalence (1)—reflexivity

$$\begin{array}{l} t \in T \\ \quad\big| \\ t \cong_T t \end{array}$$

Equivalence (2)—symmetry

$$\begin{array}{ll} t \cong_T s & t, s \in T \\ \quad\big| & \\ s \cong_T t & \end{array}$$

Equivalence (3)—transitivity

$$\begin{array}{ll} t \cong_T s & t, s, u \in T;\ s \cong_T u \\ \quad\big| & \\ t \cong_T u & \end{array}$$

Equivalence (4)—in Prop

$$\begin{array}{l} t \cong_{\mathsf{Prop}} s \\ \quad\big| \\ \mathsf{True}_t \leftrightarrow \mathsf{True}_s \end{array}$$

Equivalence′ (5)—in Prop

$$\frac{\sim(t \cong_{\mathsf{Prop}} s)}{\sim({}^{\mathsf{True}}t \leftrightarrow {}^{\mathsf{True}}s)}$$

C-**Rules**

Co-extensionality (1)

$$\frac{s \cong_{(S \Longrightarrow T)} t \qquad d \in S;\ t, s \in (S \Longrightarrow T)}{s(d) \cong_T t(d)}$$

D-**Rules**

Co-extensionality′ (2)

$$\frac{\sim(s \cong_{(S \Longrightarrow T)} t) \qquad t, s \in (S \Longrightarrow T)}{\sim(s(d) \cong_T t(d)),\ d \in S}$$

⊥-**Rules**

Path closure can be derived under the following circumstances.

Equivalence within a Type	**Identity within a Type**
$t \cong_T t'$ closes a branch if $t \notin T$	$(t =_T t')$ closes a branch if $t \notin T$

4.3 Type Inference Rules

These rules govern the inference of type membership of terms in the language of wffs.

A-**Rules**

General Function Spaces (1)

$$\frac{t \in (S \Longrightarrow T)}{\forall x(x \in S \rightarrow tx \in T)}$$

General Function Spaces (2)

$$\frac{t \notin (S \Longrightarrow T)}{\exists x(x \in S \wedge tx \notin T)}$$

Negated Propositions

$$\frac{\sim t \notin \mathsf{Prop}}{t \notin \mathsf{Prop}}$$

Universal Propositions

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

Existential Propositions

$$\frac{(\hat{\exists} x \epsilon S.t) \notin \mathsf{Prop}}{(\lambda x.t) \notin (S \Longrightarrow \mathsf{Prop})}$$

Equivalence

$$\frac{t \in T \qquad t' \in T}{(t \mathrel{\hat{\cong}_T} t') \in \mathsf{Prop}}$$

Identity

$$\frac{t \in T \qquad t' \in T}{(t \mathrel{\hat{=}_T} t') \in \mathsf{Prop}}$$

B-**Rules**

Conjunctive Propositions

$$\frac{(t \mathrel{\hat{\wedge}} t') \notin \mathsf{Prop}}{t \notin \mathsf{Prop} \quad t' \notin \mathsf{Prop}}$$

Disjunctive Propositions

$$\frac{(t \mathrel{\hat{\vee}} t') \notin \mathsf{Prop}}{t \notin \mathsf{Prop} \quad t' \notin \mathsf{Prop}}$$

Implicative Propositions

$$\frac{(t \mathrel{\hat{\rightarrow}} t') \notin \mathsf{Prop}}{t \notin \mathsf{Prop} \quad t' \notin \mathsf{Prop}}$$

Bi-implicative Propositions

$$\frac{(t \mathrel{\hat{\leftrightarrow}} t') \notin \mathsf{Prop}}{t \notin \mathsf{Prop} \quad t' \notin \mathsf{Prop}}$$

$\perp$-**Rules**

True Propositions

$\mathsf{True}\, s$ closes a branch if $s \notin \mathsf{Prop}$

4.4 Truth Rules

A-**Rules**

Negation

$$\frac{\mathsf{True}(\hat{\sim} s)* \qquad s \in \mathsf{Prop}}{\sim \mathsf{True}\, s}$$

Conjunction

$$\frac{\mathsf{True}(s \mathrel{\hat{\wedge}} t)* \qquad s, t \in \mathsf{Prop}}{\mathsf{True}\, s \wedge \mathsf{True}\, t}$$

Disjunction

$$\frac{\mathsf{True}(s \mathrel{\hat{\vee}} t)* \qquad s, t \in \mathsf{Prop}}{\mathsf{True}\, s \vee \mathsf{True}\, t}$$

Implication

$$\frac{\mathsf{True}(s \mathrel{\hat{\rightarrow}} t)* \qquad s, t \in \mathsf{Prop}}{\mathsf{True}\, s \rightarrow \mathsf{True}\, t}$$

Bi-implication

$$\frac{\mathsf{True}(s \hat{\leftrightarrow} t)*}{\mathsf{True}\, s \leftrightarrow \mathsf{True}\, t} \quad s,t \in \mathsf{Prop}$$

∀ Quantification

$$\frac{\mathsf{True}(\hat{\forall} x \epsilon S.t)*}{\forall x(x \in S \rightarrow \mathsf{True}\, t)} \quad (\lambda x.t) \in (S \Longrightarrow \mathsf{Prop})$$

∃ Quantification

$$\frac{\mathsf{True}(\hat{\exists} x \epsilon S.t)*}{\exists x(x \in S \wedge \mathsf{True}\, t)} \quad (\lambda x.t) \in (S \Longrightarrow \mathsf{Prop})$$

Equivalence

$$\frac{\mathsf{True}(s \hat{\cong}_T t)*}{(s \cong_T t)} \quad s,t \in T$$

Identity

$$\frac{\mathsf{True}(s \hat{=}_T t)*}{(s =_T t)} \quad s,t \in T$$

Negation′

$$\frac{\sim \mathsf{True}(\hat{\sim} s)*}{\mathsf{True}\, s} \quad s \in \mathsf{Prop}$$

Conjunction′

$$\frac{\sim \mathsf{True}(s \hat{\wedge} t)*}{\sim(\mathsf{True}\, s \wedge \mathsf{True}\, t)} \quad s,t \in \mathsf{Prop}$$

Disjunction′

$$\frac{\sim \mathsf{True}(s \hat{\vee} t)*}{\sim(\mathsf{True}\, s \vee \mathsf{True}\, t)} \quad s,t \in \mathsf{Prop}$$

Implication′

$$\frac{\sim \mathsf{True}(s \hat{\rightarrow} t)*}{\sim(\mathsf{True}\, s \rightarrow \mathsf{True}\, t)} \quad s,t \in \mathsf{Prop}$$

Bi-implication′

$$\frac{\sim \mathsf{True}(s \hat{\leftrightarrow} t)*}{\sim(\mathsf{True}\, s \leftrightarrow \mathsf{True}\, t)} \quad s,t \in \mathsf{Prop}$$

∀ Quantification′

$$\frac{\sim \mathsf{True}(\hat{\forall} x \epsilon S.t)*}{\sim \forall x(x \in S \rightarrow \mathsf{True}\, t)} \quad (\lambda x.t) \in (S \Longrightarrow \mathsf{Prop})$$

∃ Quantification′

$$\frac{\sim \mathsf{True}(\hat{\exists} x \epsilon S.t)*}{\sim \exists x(x \in S \wedge \mathsf{True}\, t)} \quad (\lambda x.t) \in (S \Longrightarrow \mathsf{Prop})$$

Equivalence′

$$\frac{\sim \mathsf{True}(s \hat{\cong}_T t)*}{\sim(s \cong_T t)} \quad s,t \in T$$

Identity′

$$\frac{\sim \mathsf{True}(s \hat{=}_T t)*}{\sim(s =_T t)} \quad s,t \in T$$

5 Example Proof

In Figure 1, we give an example of a proof using this tableau system. The example we use is intended to represent some agent's beliefs about another agent's beliefs. Specifically, we show

that

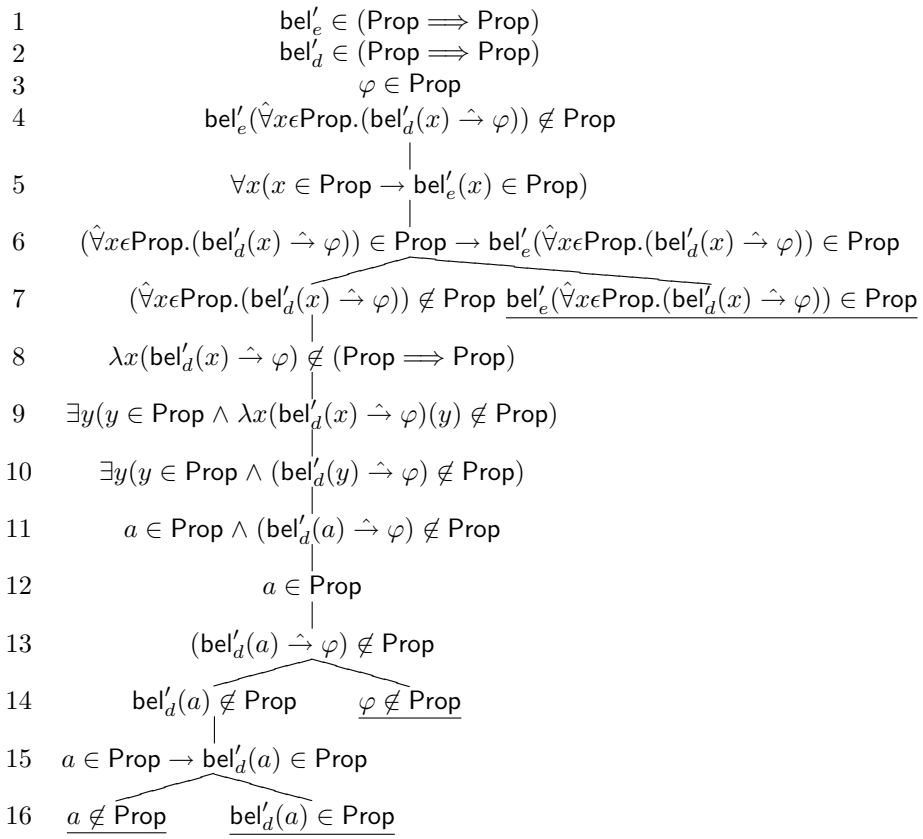
$$\mathsf{bel}'_e(\hat{\forall} x \epsilon \mathsf{Prop}.(\mathsf{bel}'_d(x) \hat{\rightarrow} \varphi)) \in \mathsf{Prop}$$

follows from

$$\begin{array}{l} \mathsf{bel}'_e \in (\mathsf{Prop} \Longrightarrow \mathsf{Prop}) \\ \mathsf{bel}'_d \in (\mathsf{Prop} \Longrightarrow \mathsf{Prop}) \\ \varphi \quad \in \mathsf{Prop} \end{array}$$

using the tableau rules given above.

Such proofs of sentences concerning a term's type are important, as we need to be able to show that a term represents a proposition before we can reason with its truth conditions.



Lines 1,2,3 are premises. 4 is the negated conclusion. 5 follows from 1 (General Function Spaces). The RHS of 6 contradicts 4. 9 uses Negated General Function Spaces. 10 uses β-reduction. The RHS of 14 contradicts 3. 15 is an instantiation of 5. RHS of 16 contradicts 14. LHS of 16 contradicts 12.

FIG. 1. Example Proof

6 Intensional Identity v. Extensional Equivalence

There are two equality notions in PTCT. $t \cong_T s$ states that the terms t, s are extensionally equivalent in type T. Extensional equivalence is represented in the language of terms by $t \mathrel{\hat{\cong}}_T s$. $t =_T s$ states that two terms are intensionally identical. The rules for intensional identity are essentially those of the $\lambda\alpha\beta\eta$-calculus. It is represented in the language of terms by $t \mathrel{\hat{=}}_T s$. It is necessary to type the intensional identity predicate in order to avoid paradoxes when we introduce comprehension types.

The rules governing equivalence and identity are such that we are able to derive $t =_T s \rightarrow t \cong_T s$ for all types inhabited by t (s), but not $t \cong_T s \rightarrow t =_T s$. As a result, PTCT can sustain fine-grained intensional distinctions among provably equivalent propositions. Therefore, we avoid the reduction of logically equivalent expressions to the same intension, a reduction which holds in classical intensional semantics, without invoking impossible worlds. Moreover, we do so within a first-order system that uses a flexible Curry typing system rather than a higher-order logic with Church typing.

7 Extending the Type System

What we have presented so far is a highly intensional formal logic with a simple type system, expressed in a Curry-style. Whilst this type system may be sufficient to deal with core issues in natural language semantics, there may be cases where a richer type system is more appropriate. Here we consider some possible extensions that are motivated by the concerns of natural language semantics.

7.1 A Universal Type

One possible extension that we could consider is to add a universal type Δ to the types, and rules corresponding to the following axiom.

(6) UT: $x \in \Delta \leftrightarrow x = x$

A reason for adopting such an extension is that it would make it possible to apply Chierchia's analysis of nominalisation [9] directly within PTCT. To be more specific, phrases such as "is fun" can take nouns, gerunds and infinitives as arguments, as in:

> Tennis is fun.
> Playing tennis is fun.
> To play tennis is fun.

The last two sentences are examples of nominalisation; the gerund and infinitive forms allow verbs to play the role of nouns. Chierchia accounts for this by arguing that such phrases should be represented by functions that take arguments of any type and yield a proposition, that is, the data can be accounted for if we have the type $\Delta \Longrightarrow \mathsf{Prop}$. This requires that the theory has a universal type Δ.[9]

As Chierchia observes, the universal type also allows for an analysis of natural language conjunction. Conjunctions such as "and" and "or" are not restricted to conjoining phrases of any particular category. We can capture this type generality of conjunction by giving it the type $\Delta \Longrightarrow (\Delta \Longrightarrow \Delta)$.

[9] According to Chierchia, this approach also allows us to account for apparent cases of self-predication, as in "fun is fun."

Unfortunately this extension is inconsistent in PTCT if Prop is a type, as it is in our proposed formulation. To see this, consider the term rr, where $r = \lambda x.\hat{\exists}y\epsilon(\Delta \Longrightarrow \mathsf{Prop})(x \mathrel{\hat{=}}_{\Delta \Longrightarrow \mathsf{Prop}} y \mathbin{\hat{\wedge}} \hat{\sim} xy)$.

Although Chierchia's solution for dealing with gerunds and infinitives is not available to us in PTCT, there are other consistent extensions that we can adopt. One of these, a restricted polymorphism (Section 7.4), provides an alternative means of addressing the nominalisation examples, and a more precise way of handling the typing of natural language conjunction.

First, we turn to several other types, one of which we will exploit extensively in an analysis of natural language anaphora (Section 12).

7.2 Separation Types

Separation types are a variety of subtype. They are expressed in the form $\{x \in T : \varphi\}$, for some T and φ. An element is of this type if it is a term of type T for which the proposition φ is true when the term is substituted for x in φ. The reason that we consider separation types here is that they will be exploited in Section 12, which adopts a proposal for the semantic representation of quantifiers suggested by Lappin [34], and Lappin and Francez [36].

To add separation types to PTCT, we add $\{x \in T : \varphi'\}$ to the types, and a tableau rule that implements the following axiom.

(7) SP: $z \in \{x \in T.\varphi'\} \leftrightarrow (z \in T \wedge \varphi'[z/x])$

That is:

Separation Types

$z \in \{x \in T.\varphi'\}^*$

$z \in T$

$\varphi'[z/x]$

Negated Separation Types

$z \notin \{x \in T.\varphi'\}^*$ *where $\notin$ has its usual intended meaning*

$z \notin T$ $\quad$ $\sim\varphi'[z/x]$

Note that there is an issue here concerning the nature of φ. To ensure the theory is first-order, this type needs to be term representable, so φ' must be term representable. To this end, we can define a term representable fragment of the language of wffs. First, we introduce syntactic sugar for typed quantification in the wffs.

(8) (a) $\forall_T x\varphi =_{\mathsf{def}} \forall x(x \in T \rightarrow \varphi)$
(b) $\exists_T x\varphi =_{\mathsf{def}} \exists x(x \in T \wedge \varphi)$

Wffs with these typed quantifiers, and no free-floating type judgements will then have direct intensional analogues—that is, term representations—which will always be propositions. We can define representable wffs by φ':

(9) $\varphi' ::= \alpha' \mid (\sim\varphi') \mid (\varphi' \wedge \psi') \mid (\varphi' \vee \psi') \mid (\varphi' \rightarrow \psi') \mid (\varphi' \leftrightarrow \psi') \mid (\forall_T x\varphi') \mid (\exists_T x\varphi) \mid {}^{\mathsf{True}}t$
(atomic representable wffs) $\alpha' ::= (t =_T s) \mid t \cong_T s$

The term representations of representable wffs $\lceil\alpha'\rceil$ are given by the following.

(10) (a) $\lceil\sim a\rceil = \hat{\sim}\lceil a\rceil$
(b) $\lceil a \wedge b\rceil = \lceil a\rceil \mathbin{\hat{\wedge}} \lceil b\rceil$

(c) $\lceil a \vee b \rceil = \lceil a \rceil \hat{\vee} \lceil b \rceil$
(d) $\lceil a \rightarrow b \rceil = \lceil a \rceil \hat{\rightarrow} \lceil b \rceil$
(e) $\lceil a \leftrightarrow b \rceil = \lceil a \rceil \hat{\leftrightarrow} \lceil b \rceil$
(f) $\lceil a \cong_T b \rceil = a \hat{\cong}_T b$
(g) $\lceil a =_T b \rceil = a \hat{=}_T b$
(h) $\lceil \mathsf{True} t \rceil = t$
(i) $\lceil \forall_T x.a \rceil = \hat{\forall} x \epsilon T \lceil a \rceil$
(j) $\lceil \exists_T x.a \rceil = \hat{\exists} x \epsilon T \lceil a \rceil$

Now we can express separation types as $\{x \in S.\varphi'\}$, which can be taken to be sugar for $\{x \epsilon S.\lceil \varphi' \rceil\}$.

The following theorem is an immediate consequence of the recursive definition of representable wffs and their term representations.

THEOREM 7.1 (Representability)
$\lceil \varphi' \rceil \in \mathsf{Prop}$ for all representable wffs φ', and furthermore $^{\mathsf{True}}\lceil \varphi' \rceil \leftrightarrow \varphi'$.

7.3 Comprehension Types

For completeness, we can consider comprehension types. These are types defined in terms of a proposition. They are usually written in the form $\{x : \varphi\}$. Elements are of this type if the proposition φ is true when the element is substituted for x in φ .

Usually comprehension can be derived from SP and UT. We are forgoing UT to avoid paradoxes (Section 7.1), so we have to define comprehension independently. The same arguments apply as for SP concerning representability

We add the type $\{x : \varphi'\}$ and a tableau rule corresponding to the following axiom.

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

The effect of this axiom can be achieved by the following tableau rules.

Comprehension Types

$z \in \{x : \varphi'\}^*$
|
$\varphi'[z/x]$

Negated Comprehension Types

$z \notin \{x : \varphi'\}^*$ *where $\notin$ has its usual intended meaning*
|
$\sim \varphi'[z/x]$

Given that COMP = SP + UT, where UT is the Universal Type $\Delta = \{x : x \hat{=} x\}$, we would derive a paradox if = was not typed. This is because in PTCT Prop is a type. So rr, where $r = \lambda x.\hat{\exists} y \in (\Delta \Longrightarrow \mathsf{Prop})[x \hat{=} y \hat{\wedge} \hat{\sim} xy]$ produces a paradoxical propositional. Our use of a typed intensional identity predicate filters out the paradox because it must be possible to prove that the two expressions for which $=_T$ is asserted are of type T independently of the identity assertion. $s =_T t$ iff $s, t \in T$ and $s = t$.

7.4 Polymorphic Types

As we have observed, natural language exhibits flexibility in the categories to which different syntactic elements can belong. Chierchia's examples of nominalisation, and natural language conjunction, discussed in Section 7.1, exemplify this phenomena.

In formal systems, this kind of behavour is often characterised as *polymorphism*. Here we introduce polymorphism, formalise a particular form of polymorphism in PTCT, and indicate how it can address at least some of the data for which a universal type has been proposed.

There are many varieties of polymorphism, including schematic polymorphism, implicit polymorphism, explicit polymorphism, and impredicative polymorphism.

Schematic polymorphism is a syntatic device whereby a meta-theoretic symbol is used to abbreviate a range of types. Such symbols are, effectively, syntactic sugar for a disjunction of expressions in which the schematic types are consistently replaced by ground types.

The other forms of polymorphism are genuine extensions to the type system, and require the addition of type variables. In the case of implicit polymorphism, the type variables are universally quantified. It is this kind of polymorphism that we shall formulate here.

With explicit polymorphism, relations and functions in effect contain type variables that can be instantied by arguments. It might be that this is actually a more appropriate form of polymorphism for theories of natural language semantics.

Impredicative polymorphism is a particularly powerful device, which is discussed below. It appears that this power is neither required, nor appropriate, for natural language semantics.

To add implicit polymorphic types to PTCT, we enrich the language of types to include type variables X, and the wffs to include quantification over types $\forall X\varphi, \exists X\varphi$. We add following tableau rules.

Universal Type-Quantification

$\forall X.\varphi$ *where K is a type that occurs on the path (or is the only type occuring in the path)*

$\varphi[K/X]$

Existential Type-Quantification

$\exists X.\varphi^*$ *where type K does not occur on the path (suggests we need Type equivalence)*

$\varphi[K/X]$

Negated Type-Quantification

$\sim QX.\varphi^*$ *where Q' is the dual of Q*

$Q'X.\sim\varphi$

We add $\Pi X.T$ to the language of types, governed by the tableau rule corresponding to the following axiom:

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

The effect of this axiom can be achieved by the following tableau rules.

Polymorphic Types

$f \in \Pi X.T^*$

$\forall X(f \in T)$

Negated Polymorphic Types

$f \notin \Pi X.T^*$ *where $\notin$ has its usual intended meaning*

$\exists X(f \notin T)$

Polymorphic types permit us to accommodate the fact that natural language expressions like coordination and certain verbs can can apply as functions to arguments of different types. In particular, we can account for the data given in Section 7.1. In the case of the nominalisation data of Chierchia [9]

Tennis is fun.
Playing tennis is fun.
To play tennis is fun.

we can argue that "is fun" is of type $\Pi X.X \Longrightarrow \mathsf{Prop}$. Conjunctions such as "and," which can combine categories of any type to give an expression of the same type, can be given the type $\Pi X.X \Longrightarrow (X \Longrightarrow X)$.[10]

Note that PM is impredicative, ie. the type quantification ranges over the types that are being defined. Impredicativity greatly increases the power of the language, which is problematic if we wish to sustain a recursively enumerable theory. It can also lead to paradoxes under certain conditions. To avoid these difficulties, we adopt a restricted form of polymorphism.

(13) PM′: $f \in \Pi X.T \leftrightarrow \forall X(f \in T)$ where X ranges only over non-polymorphic types.

This constraint limits quantification over types to type variables that take non-polymorphic types as values. Therefore, we rule out iterated type polymorphism in which functional polymorphic types apply to polymorphic arguments. This weak version of polymporphism seems entirely adequate to express the instances of multiple type assignment that occur in natural languages [57]. The tableau rules do not need to be changed to incorporate PM′, provided that only non-polymorphic types are substituted for type variables in the rules for type quantification.

Even predicative, implicit polymorphism may not be a perfect match for natural language polymorphism. *Explicit* polymorphism may be more appropriate if we wish to deal elegantly with the fact that generalised quantifiers appear to be able to range over different types. It seems natural to say that the noun-phrase "every belief" ranges over propositions, whereas "every book" ranges over individuals. The type of the determiner "every" is then determined by the type of its argument ("book," and "belief" in these examples). This kind of constraint is expressible with explicit polymorphic types. The determiner "every" would then have an argument position that can be filled in by the type of the noun with which it combines. We leave amending PTCT to include implicit polymorphic types as an exercise for the interested reader.

7.5 Product Types

Product types of the form $S \otimes T$ are useful if we wish to allow terms to accepts more then one argument at a time. Elements of such a type are pairs of terms, where the first element of the pair is in S, and the second is in T. This type is different in kind from the other types presented so far in that it imposes a structural constraint on its elements: they must be pairs. λ-calculus allows us to define the required notions of pairs and terms.

$$\begin{array}{ll} \langle x, y\rangle & =_{\mathsf{def}}\ \lambda z(z(x)(y)) \\ \mathsf{fst} & =_{\mathsf{def}}\ \lambda p(p\lambda xy(x)) \\ \mathsf{snd} & =_{\mathsf{def}}\ \lambda p(p\lambda xy(y)) \end{array}$$

Pairs can be nested, and product type formation can be iterated. This allows arbitrary finite product types to be defined.

We use product types in our treatment of ellipsis and our account of semantic underspecification [19].

[10] This does not immediately account for Chierchia's example of apparent self-predication "Fun is fun." Nor does it deal with alledged cases of cross-categorial conjunction [22, 48].

To incorporate product types in PTCT, we add the type $S \otimes T$, and a tableau rule corresponding to the following axiom.

(14) PROD: $\langle x, y\rangle \in (S \otimes T) \leftrightarrow x \in S \wedge y \in T$

The effect of this axiom can be achieved by the following tableau rules.

Product Types

$\langle x, y\rangle \in (S \otimes T)^*$

$x \in S$

$y \in T$

Negated Product Types

$\langle x, y\rangle \notin (S \otimes T)^*$ *where $\notin$ has its usual intended meaning*

$x \notin S$ | $y \notin T$

For completeness, it may be appropriate to consider adding curry and uncurry operators to the language. We do not exploit such operators.

7.6 Final Syntax

Adopting the extensions discussed above, which are constrained to prevent the generation of paradoxes, we arrive at the following syntax for PTCT..

(15)	(logical constants)	l	::=	$\hat{\sim} \mid \hat{\wedge} \mid \hat{\vee} \mid \hat{\rightarrow} \mid \hat{\leftrightarrow} \mid \hat{\forall} \mid \hat{\exists} \mid \hat{=}_T \mid \hat{\cong}_T \mid \epsilon$
	(terms)	t	::=	$x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t$
	(Types)	T	::=	$B \mid \mathsf{Prop} \mid T \Longrightarrow S \mid X \mid \{x \in T.\varphi'\} \mid \{x.\varphi'\}$
				$\mid \Pi X.T \mid S \otimes T$
	(atomic wff)	α	::=	$(t =_T s) \mid t \in K \mid t \cong_T s \mid {}^{\mathsf{True}}t$
	(wff)	φ	::=	$\alpha \mid \sim\varphi \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \rightarrow \psi) \mid (\varphi \leftrightarrow \psi)$
				$\mid (\forall x\varphi) \mid (\exists x\varphi) \mid (\forall X\varphi) \mid (\exists X\varphi)$

where φ' is as defined in Section 7.2, and type variables X are intended to range only over non-polymorphic types.

8 A Model Theory for PTCT

In order to give a model for PTCT, we first need a model of the untyped λ-calculus, which will provide the model for PTCT's language of terms. For convenience and simplicity we adopt Meyer's model [42] (readers are free to substitute their favourite models of the untyped λ-calculus).

8.1 Models of the (Extensional) λ-Calculus

Definition 8.1 (General Functional Models)
A functional model is a structure of the form $\mathcal{D} = \langle D, [D \rightarrow D], \Phi, \Psi\rangle$ where

(i) D is a non-empty set,
(ii) $[D \rightarrow D]$ is some class of functions from D to D,
(iii) $\Phi : D \rightarrow [D \rightarrow D]$,
(iv) $\Psi : [D \rightarrow D] \rightarrow D$,

(v) $\Psi(\Phi(d)) = d$ for all $d \in D$

We can interpret the calculus using the following.

(16) $[\![x]\!]_g = g(x)$
$[\![\lambda x.t]\!]_g = \Psi(\lambda d.[\![t]\!]_{g[d/x]})$
$[\![ts]\!]_g = \Phi([\![t]\!]_g)[\![s]\!]_g$

where g is an assignment function from variables to elements of D. This interpretation exploits the fact that Φ maps every element of D into a corresponding function from D to D, and Ψ maps functions from D to D into elements of D.

Note that we require functions of the form $\lambda d.[\![t]\!]_{g[d/x]}$ to be in the class $[D \to D]$ to ensure that the interpretation is well defined. Here we are just following Meyer [42].

In the case where we permit constant terms, then we add the clause

(17) $[\![c]\!]_g = i(c)$

where i assigns elements of D to constants.

THEOREM 8.2
If $t = s$ in the extensional untyped λ-calculus (with ξ and η), then $[\![t]\!]_g = [\![s]\!]_g$ for each assignment g.

PROOF. By induction on the derivations. ∎

A model $\mathcal{M}$ of PTCT is constructed on the basis of a simple extensional model of the untyped λ-calculus [42, 2, 56], with additional structure added to capture the type rules and the relation between the sublanguages of PTCT.

DEFINITION 8.3 (Model of PTCT)
A model of PTCT is $\mathcal{M} = \langle \mathcal{D}, \mathsf{T}, \mathsf{P}, \mathsf{B}, \mathcal{B}, \mathcal{T}', \mathcal{T} \rangle$, where

(i) $\mathcal{D}$ is a model of the λ-calculus
(ii) $\mathsf{T} : D \to \{0, 1\}$ models the truth predicate $^{\mathsf{True}}$
(iii) $\mathsf{P} \subset D$ models the class of propositions
(iv) $\mathsf{B} \subset D$ models the class of basic individuals
(v) $\mathcal{B}(\mathsf{B})$ is a set of sets whose elements partition B into equivalence classes of individuals
(vi) $\mathcal{T}' \subset \mathcal{T}$ models the class of non-polymorphic types
(vii) $\mathcal{T} \subset D$ models the class of types

with sufficient structural constraints on T, P and $\mathcal{T}'$ to validate the tableau rules of PTCT.

In the following, we give the structural constraints required to sustain the proof theory given above.

8.2 Types

Types can be interpreted as subsets of D. To ensure that polymorphic types are predicative (i.e. to avoid an implicit circularity in their definition), quantification is only over non-polymorphic types.

To give a first-order account of type quantification, we consider the interpretation of term *representations* of types. We take types in PTCT to denote terms in $\mathcal{T} \subset D$. If type T in PTCT denotes an individual $S \in \mathcal{T}$, the underlying type (or set of individuals) will be denoted by $\mathcal{E}S$.

The types in the model, and the interpretation of PTCT's types are specified by the following rules.

(18) (a) For all $t \in \mathcal{T}$, $\mathcal{E}t \subset D$
(b) $[\![B]\!]_{g,\tau} \in \mathcal{T}'$
(c) $[\![\mathsf{Prop}]\!]_{g,\tau} \in \mathcal{T}'$
(d) $[\![X]\!]_{g,\tau} = \tau(X) \in \mathcal{T}'$
(e) If $[\![S]\!]_{g,\tau}, [\![U]\!]_{g,\tau} \in \mathcal{T}'$, then $[\![S \Longrightarrow U]\!]_{g,\tau} \in \mathcal{T}'$
(f) If $[\![S]\!]_{g,\tau} \in \mathcal{T}'$ and $[\![\lceil\varphi'\rceil]\!]_{g,\tau} \in \mathsf{P}$ then $[\![\{x \in S.\varphi'\}]\!]_{g,\tau} \in \mathcal{T}'$
(g) If $[\![\lceil\varphi'\rceil]\!]_{g,\tau} \in \mathsf{Prop}$ then $[\![\{x.\varphi'\}]\!]_{g,\tau} \in \mathcal{T}'$
(h) If $[\![S]\!]_{g,\tau} \in \mathcal{T}'$ then $[\![\Pi X.S]\!]_{g,\tau} \in \mathcal{T}$
(i) If $[\![S]\!]_{g,\tau}, [\![T]\!]_{g,\tau} \in \mathcal{T}'$ then $[\![S \otimes T]\!]_{g,\tau} \in \mathcal{T}'$
(j) $\mathcal{E}[\![B]\!]_{g,\tau} = \mathsf{B} \subseteq D$
(k) $\mathcal{E}[\![\mathsf{Prop}]\!]_{g,\tau} = \mathsf{P} \subseteq D$
(l) $\mathcal{E}[\![S \Longrightarrow U]\!]_{g,\tau} = \{d \in D : \forall e \in \mathcal{E}[\![S]\!]_{g,\tau}.(\Phi(d))e \in \mathcal{E}[\![U]\!]_{g,\tau}\}$
(m) $\mathcal{E}[\![\{x \in S.\varphi'\}]\!]_{g,\tau} = \{d \in \mathcal{E}[\![S]\!]_{g,\tau}.\mathcal{M}_{g[d/x],\tau} \models \varphi'\}$
(n) $\mathcal{E}[\![\{x.\varphi'\}]\!]_{g,\tau} = \{d \in D.\mathcal{M}_{g[d/x],\tau} \models \varphi'\}$
(o) $\mathcal{E}[\![\Pi X.S]\!]_{g,\tau} = \{d \in D : \forall U \in \mathcal{T}'.d \in \mathcal{E}[\![S]\!]_{g,\tau[U/X]}\}$
(p) $\mathcal{E}[\![S \otimes T]\!]_{g,\tau} = \{d \in D : \{d' \in D : d \in \mathcal{E}[\![S]\!]_{g,\tau}$ and $d' \in \mathcal{E}[\![T]\!]_{g,\tau}\}\}$

Here, τ is an assignment from type variables to elements of $\mathcal{T}$, and $\lceil\cdot\rceil$ is as defined in Section 7.2, and Φ is as defined in the Meyer model of the untyped λ-calculus (Section 8.1).

8.3 Propositions

The typing rules for Prop are supported by the following structural constraints on models.

(19) (a) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$, then $[\![\tilde{\sim} t]\!]_{g,\tau} \in \mathsf{P}$.
(b) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $[\![t \mathbin{\hat{\wedge}} s]\!]_{g,\tau} \in \mathsf{P}$.
(c) if $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $[\![t \mathbin{\hat{\vee}} s]\!]_{g,\tau} \in \mathsf{P}$.
(d) if $[\![t]\!]_{g,\tau} \in \mathsf{P}$, and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $[\![t \mathbin{\hat{\rightarrow}} s]\!]_{g,\tau} \in \mathsf{P}$.
(e) if $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$ then $[\![t \mathbin{\hat{\leftrightarrow}} s]\!]_{g,\tau} \in \mathsf{P}$.
(f) If $[\![S]\!]_{g,\tau} \in \mathcal{T}$, and $[\![t]\!]_{g[d/x],\tau} \in \mathsf{P}$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$, then $[\![\hat{\forall} x \epsilon S.t]\!]_{g,\tau} \in \mathsf{P}$.
(g) If $[\![S]\!]_{g,\tau} \in \mathcal{T}$, and $[\![t]\!]_{g[d/x],\tau} \in \mathsf{P}$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$, then $[\![\hat{\exists} x \epsilon S.t]\!]_{g,\tau} \in \mathsf{P}$.
(h) If $[\![S]\!]_{g,\tau} \in \mathcal{T}$, then $[\![t \mathbin{\hat{\cong}}_S s]\!]_{g,\tau} \in \mathsf{P}$ iff $[\![s]\!]_{g,\tau}, [\![t]\!]_{g,\tau} \in \mathcal{E}[\![S]\!]_{g,\tau}$.
(i) If $[\![S]\!]_{g,\tau} \in \mathcal{T}$, then $[\![t \mathbin{\hat{=}}_S s]\!]_{g,\tau} \in \mathsf{P}$ iff $[\![s]\!]_{g,\tau}, [\![t]\!]_{g,\tau} \in \mathcal{E}[\![S]\!]_{g,\tau}$.

8.4 Truth

The rules for $^{\mathsf{True}}$ are supported by the following conditions.

(20) (a) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$, then $\mathsf{T}([\![\tilde{\sim} t]\!]_{g,\tau}) = 1$ iff $\mathsf{T}([\![t]\!]_{g,\tau}) = 0$.
(b) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $\mathsf{T}([\![t \mathbin{\hat{\wedge}} s]\!]_{g,\tau}) = 1$
iff $\mathsf{T}([\![t]\!]_{g,\tau}) = 1$ and $\mathsf{T}([\![s]\!]_{g,\tau}) = 1$.

(c) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $\mathsf{T}([\![t \hat{\vee} s]\!]_{g,\tau}) = 1$ iff either $\mathsf{T}([\![t]\!]_{g,\tau}) = 1$ or $\mathsf{T}([\![s]\!]_{g,\tau}) = 1$.
(d) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $\mathsf{T}([\![t \hat{\rightarrow} s]\!]_{g,\tau}) = 1$ iff either $\mathsf{T}([\![t]\!]_{g,\tau}) = 0$ or $\mathsf{T}([\![s]\!]_{g,\tau}) = 1$.
(e) If $[\![t]\!]_{g,\tau} \in \mathsf{P}$ and $[\![s]\!]_{g,\tau} \in \mathsf{P}$, then $\mathsf{T}([\![t \hat{\leftrightarrow} s]\!]_{g,\tau}) = 1$ iff $[\![t]\!]_{g,\tau} = [\![s]\!]_{g,\tau}$.
(f) If $[\![S]\!]_{g,\tau} \in \mathcal{T}$ and $[\![t]\!]_{g[d/x],\tau} \in \mathsf{P}$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$, then $\mathsf{T}([\![\hat{\forall} x \epsilon S.t]\!]_{g,\tau}) = 1$ iff $\mathsf{T}([\![t]\!]_{g[d/x],\tau}) = 1$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$.
(g) if $[\![S]\!]_{g,\tau} \in \mathcal{T}$ and $[\![t]\!]_{g[d/x],\tau} \in \mathsf{P}$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$, then $\mathsf{T}([\![\hat{\exists} x \epsilon S.t]\!]_{g,\tau}) = 1$ iff $\mathsf{T}([\![t]\!]_{g[d/x],\tau}) = 1$ for some $d \in [\![S]\!]_{g,\tau}$.
(h) i. If $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in \mathsf{B}$ then $\mathsf{T}([\![t \hat{\cong}_{\mathsf{B}} s]\!]_{g,\tau}) = 1$ iff there is a set S such that $S \in \mathcal{B}(\mathsf{B})$ and $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in S$.
ii. If $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in \mathsf{P}$ then $\mathsf{T}([\![t \hat{\cong}_{\mathsf{Prop}} s]\!]_{g,\tau}) = 1$ iff $\mathsf{T}([\![t]\!]_{g,\tau}) = \mathsf{T}([\![s]\!]_{g,\tau})$.
iii. If $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in [\![S \Longrightarrow U]\!]_{g,\tau}$, where $[\![S]\!]_{g,\tau}, [\![U]\!]_{g,\tau} \in \mathcal{T}$ then $\mathsf{T}([\![t \hat{\cong}_{(S \Longrightarrow U)} s]\!]_{g,\tau}) = 1$ iff $\mathsf{T}([\![tx \hat{\cong}_{U} sx]\!]_{g[d/x],\tau}) = 1$ for all $d \in \mathcal{E}[\![S]\!]_{g,\tau}$.
(i) If $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in \mathcal{E}[\![S]\!]_{g,\tau}$, then $\mathsf{T}([\![t \hat{=}_{S} s]\!]_{g,\tau}) = 1$ iff $[\![t]\!]_{g,\tau} = [\![s]\!]_{g,\tau}$.
(j) If $\mathsf{T}([\![t]\!]_{g,\tau}) = 1$ then $[\![t]\!]_{g,\tau} \in \mathsf{P}$.

Note that we give no rules for the extensional equivalence of elements of comprehension types, separation types or polymorphic types.

8.5 Well-Formed Formulae

The language of wffs can now be given truth conditions. Some examples follow.

(21)

$\mathcal{M}_{g,\tau} \models s =_T t$	iff $[\![t]\!]_{g,\tau}, [\![s]\!]_{g,\tau} \in \mathcal{E}[\![T]\!]_{g,\tau}$ and $[\![t]\!]_{g,\tau} = [\![s]\!]_{g,\tau}$
$\mathcal{M}_{g,\tau} \models s \cong_T t$	iff $\mathsf{T}([\![s]\!]_{g,\tau} \hat{\cong}_T [\![t]\!]_{g,\tau}) = 1$
$\mathcal{M}_{g,\tau} \models s \cong_{\mathsf{Prop}} t$	iff $[\![s]\!]_{g,\tau}, [\![t]\!]_{g,\tau} \in \mathsf{P}$ and $\mathcal{M}_{g,\tau} \models s \leftrightarrow t$
$\mathcal{M}_{g,\tau} \models s \cong_{(T \Longrightarrow \mathsf{Prop})} t$	iff $[\![s]\!]_{g,\tau}, [\![t]\!]_{g,\tau} \in [\![T \Longrightarrow \mathsf{Prop}]\!]_{g,\tau}$ and for all $x \in T, \mathcal{M}_{g,\tau} \models sx \leftrightarrow tx$ (x not free in s, t)
$\mathcal{M}_{g,\tau} \models {}^{\mathsf{True}}(t)$	iff $\mathsf{T}([\![t]\!]_{g,\tau}) = 1$
$\mathcal{M}_{g,\tau} \models t \in T$	iff $[\![t]\!]_{g,\tau} \in \mathcal{E}[\![T]\!]_{g,\tau}$[11]
$\mathcal{M}_{g,\tau} \models \sim \varphi$	iff $\mathcal{M}_{g,\tau} \not\models \varphi$
$\mathcal{M}_{g,\tau} \models \varphi \wedge \psi$	iff $\mathcal{M}_{g,\tau} \models \varphi$ and $\mathcal{M}_{g,\tau} \models \psi$
$\mathcal{M}_{g,\tau} \models \varphi \vee \psi$	iff $\mathcal{M}_{g,\tau} \models \varphi$ or $\mathcal{M}_{g,\tau} \models \psi$
$\mathcal{M}_{g,\tau} \models \varphi \rightarrow \psi$	iff $\mathcal{M}_{g,\tau} \not\models \varphi$ or $\mathcal{M}_{g,\tau} \models \psi$
$\mathcal{M}_{g,\tau} \models \varphi \leftrightarrow \psi$	iff $\mathcal{M}_{g,\tau} \models \varphi$ exactly when $\mathcal{M}_{g,\tau} \models \psi$
$\mathcal{M}_{g,\tau} \models \forall x \varphi$	iff for all $d \in D$, $\mathcal{M}_{g[d/x],\tau} \models \varphi$
$\mathcal{M}_{g,\tau} \models \exists x \varphi$	iff for some $d \in D$, $\mathcal{M}_{g[d/x],\tau} \models \varphi$
$\mathcal{M}_{g,\tau} \models \forall X \varphi$	iff for all $S \in \mathcal{T}'$, $\mathcal{M}_{g,\tau[S/X]} \models \varphi$
$\mathcal{M}_{g,\tau} \models \exists X \varphi$	iff for some $S \in \mathcal{T}'$, $\mathcal{M}_{g,\tau[S/X]} \models \varphi$

Here, type quantification is restricted to non-polymorphic types. Type variables are already constrained to denote types in $\mathcal{T}'$.

Definition 8.4 (Validity)
A wff φ of PTCT is *valid* iff $\mathcal{M}_{g,\tau} \models \varphi$ for all models $\mathcal{M}$ and all assignment functions g, τ.

[11] Syntactic constraints on T guarantee that $[\![T]\!]_{g,\tau} \in \mathcal{T}$.

THEOREM 8.5 (Soundness of PTCT)
If $\mathsf{PTCT} \vdash \phi$, then ϕ is valid.

PROOF. To prove that the tableau proof procedure for PTCT is sound we have to prove the following lemma.

LEMMA 8.6
If the initial sentence S of a tableau T is satisfied in a model $\mathcal{M}$ for PTCT, then there is an open (possibly infinite) path P in T in which every full sentence in P is satisfied.

Lemma 8.6 follows by induction on the downward correctness of the tableau rules relative to the model theory of PTCT. For each full sentence F in P, if F is true in $\mathcal{M}$, then the rules of the model theory insure that the sentences in the extension of the open path derived from F by application of the tableau rules are also satisfied in $\mathcal{M}$. We illustrate downward correctness of the tableau rules with three examples, one for each type of rule.

Rules for Wffs Universal Quantification
The truth condition given in (21) for universal quantification over individuals entails that if $\forall x\varphi$ is true in $\mathcal{M}$, then $\varphi^{x/c}$ is true in $\mathcal{M}$ for every individual constant c such that $i(c) \in \mathcal{D}$.

Type Inference Rules General Function Spaces
(18l) and the truth condition for type membership formulas given in (21) entail that if $t \in (S \Longrightarrow T)$ and $t' \in S$ are true in $\mathcal{M}$, then $tt' \in T$ is true in $\mathcal{M}$.

Truth Rules Conjunction
The truth conditions for $^{\mathsf{True}}(t)$ and $\varphi \wedge \psi$ in (21), and (20b) entail that if $^{\mathsf{True}}(s \mathbin{\hat{\wedge}} t)$ and $s, t \in \mathsf{Prop}$ are true in $\mathcal{M}$, then $^{\mathsf{True}}s \wedge {}^{\mathsf{True}}t$ is true in $\mathcal{M}$.

From Lemma 8.6 it follows that if the tableau for a formula S is closed, then S is not satisfiable in a model of PTCT. Therefore, if there is a proof of S (i.e. the tableau for $\sim S$ is closed), then S is satisfied in all models of PTCT. ∎

THEOREM 8.7 (Completeness of PTCT)
If ϕ is valid, then $\mathsf{PTCT} \vdash \phi$.

PROOF. To prove that the tableau proof procedure for PTCT is complete we must prove the following lemma.

LEMMA 8.8
If there is an open (possibly infinite) path in a tableau for a sentence S, then S is satisfiable in a model of PTCT.

Lemma 8.8 follows by induction on the upward correctness of the tableau rules relative to the model theory for PTCT. For each set Γ of formulas in an extension of an open path derived by the application of the tableau rules to a formula F, if the elements of Γ are true in a model $\mathcal{M}$, then the definition of the model theory insures that F is satisfiable in $\mathcal{M}$. We illustrate the upward correctness of the tableau rules with the same examples that we used for downward correctness. In these illustrations correctness runs in the opposite direction, from the hypothesis that the conclusions of the rule are true in $\mathcal{M}$ to the result that its premise(s) are satisfiable in $\mathcal{M}$.

Rules for Wffs Universal Quantification
The truth condition given in (21) for universal quantification over individuals entails that

if $\varphi^{x/c}$ is true in $\mathcal{M}$ for every individual constant c that appears in the open path, where $i(c) \in \mathcal{D}$, then $\forall x\varphi$ is satisfiable in $\mathcal{M}$.

Type Inference Rules General Function Spaces

18l and the truth condition for type membership formulas given in (21) entail that if $tt' \in T$ is true in $\mathcal{M}$, then $t \in (S \Longrightarrow T)$ and $t' \in S$ are true in $\mathcal{M}$.

Truth Rules Conjunction

The truth conditions for $^{\mathsf{True}}(t)$ and $\varphi \wedge \psi$ in (21), together with (20b) and (20j) entail that if $^{\mathsf{True}}s \wedge {}^{\mathsf{True}}t$ is true in $\mathcal{M}$, then $^{\mathsf{True}}(s \mathbin{\hat{\wedge}} t)$ and $s, t \in \mathsf{Prop}$ are true in $\mathcal{M}$.

Lemma 8.8 entails that if a branch B in a tableau for S is open, then all the sentences in B, including S, are satisfiable in a model of PTCT. It follows that if S is valid ($\sim S$ is not satisfiable in a model of PTCT), then there is a tableau proof of S. ∎

9 Types and Properties

On initial examination, it appears that there are clear parallels between properties and types in PTCT, along the following lines.

$$\begin{array}{lcl} x\epsilon T & \cong & T(x) \\ (T \Longrightarrow S) & \cong & \hat{\forall} x\epsilon T.S(x) \\ (T \Longrightarrow S) & \cong & \hat{\forall} x(T(x) \mathbin{\hat{\wedge}} S(x)) \\ \{x\epsilon T.\varphi\} & \cong & \lambda x(x\epsilon T \mathbin{\hat{\wedge}} \varphi) \\ \{x.\varphi\} & \cong & \lambda x(\varphi) \end{array}$$

There is a sense in which types could be thought of as properties of type $\Delta \Longrightarrow \mathsf{Prop}$, although as we have seen above (Section 7), adding the universal type directly to PTCT is problematic.

This apparent similarity between types and properties leads to the notion of property-theoretic definability of a type.

Definition 9.1 (Property-Theoretic Definability of a Type)
A type T is definable as an unrestricted property p iff $\forall x(x \in T \leftrightarrow {}^{\mathsf{True}}p(x))$, where an unrestricted property is one that forms a proposition with any argument.

If we allow type membership to be represented by predication in this way, then we have effectively allowed free-floating type judgements in the language of terms, of the form $x\epsilon T$. We might consider axiomatising the behaviour of free-floating type judgements directly as follows.

$$\begin{array}{l} \mathsf{Type}(T) \rightarrow (x\epsilon T) \in \mathsf{Prop} \\ \mathsf{Type}(T) \rightarrow {}^{\mathsf{True}}(x\epsilon T) \leftrightarrow x \in T \end{array}$$

We could add further constraints to these axions, such as requiring that the term on the left of the membership symbol is not a type.

Unfortunately, allowing both free floating type judgements, and full property-theoretic definability of types leads to a paradox in the case of the type Prop.[12]

Theorem 9.2
Prop cannot appear in free-floating type judgements of the form $x\epsilon$Prop.

[12] Such paradoxes would also arise if we were to treat truth as a type.

PROOF. Consider the term

$$R = \lambda x(\hat{\sim}(xx\epsilon\mathsf{Prop}))$$

Assuming that $\mathsf{Type}(\mathsf{Prop})$, then we can show that $\forall x(R(x) \in \mathsf{Prop})$. Therefore $RR \in \mathsf{Prop}$. The axioms governing truth then allow us to show that $^{\mathsf{True}}(RR) \leftrightarrow (RR \notin \mathsf{Prop})$, which yields a contradiction. ∎

COROLLARY 9.3
Prop has no property-theoretic definition.

These problems with free-floating type judgements also indicate why we require intensional identity to be typed. If it was not, then we could define free-floating type judgements as follows.

$$x\epsilon T =_{\mathsf{def}} \hat{\exists} y\epsilon T.x \mathrel{\hat{=}} y$$

As it stands, we require the type of y to be imposed independently of the identity statement.

10 Separation Types and Internal Type Judgements

The preceding discussion suggests that it is not possible to represent free-floating type judgements in PTCT's language of terms. However, as we shall now see, there are some kinds of type judgements that *can* be incorporated into the language of terms.

The types in question are separation types. Recall that these types have the following form.

$$\{x\epsilon T.\varphi'\}$$

where φ' is an internally representable wff. More pedantically, we write this as

$$\{x\epsilon T.\ulcorner\varphi'\urcorner\}$$

where $\ulcorner\varphi'\urcorner$ is the term representation of the representable wff φ', as defined in Section 7.2.

As previously described, a term z is a member of this type under the following circumstances.

$$z \in \{x\epsilon T.\ulcorner\varphi'\urcorner\} \leftrightarrow (z \in T \wedge \varphi'[z/x])$$

φ' is term representable (it contains no free-floating type judgements), and so the only part of the right-side of this equivalence that is not term representable is $z \in T$. If we knew independently that $z \in T$, then we would have the following.

$$z \in \{x\epsilon T.\ulcorner\varphi'\urcorner\} \leftrightarrow \varphi'[z/x]$$

This statement is equivalent to

$$z \in T \rightarrow (z \in \{x\epsilon T.\ulcorner\varphi'\urcorner\} \leftrightarrow \varphi'[z/x])$$

Observe that $\varphi'[z/x]$ is equivalent to $^{\mathsf{True}}((\lambda x\ulcorner\varphi'\urcorner)z)$. Thus, it turns out that $z \in \{x\epsilon T.\ulcorner\varphi'\urcorner\}$ is equivalent to $^{\mathsf{True}}((\lambda x\ulcorner\varphi'\urcorner)z)$, in the event that $z \in T$.

We conclude that it is safe to have a *restricted* form of free-floating type judgement in the language of terms. We use the symbol ϵ' to represent a restricted type membership relation of this kind. We can axiomatise it in the usual way, describing when such type judgements are propositions, and when they are true propositions.

$$t \in S \rightarrow (t\epsilon'\{x \in S.\varphi\}) \in \mathsf{Prop}$$

and

$$t \in S \rightarrow ({}^{\mathsf{True}}(t\epsilon'\{x \in S.\varphi\}) \leftrightarrow \varphi[t/x])$$

Conceptually, this internalisable type judgement can be thought of as exploiting a combination of the notion of a *universe* or *small type* as used in MLTT [40, 41] and Frege Structures [1] (that is, we restrict which types can appear in internal type judgements) with Turner's S5-like treatment of the internalisation of troublesome predicates, such as those for propositions and truth in PT [55] (the internal type judgement only "makes sense" if we already know something about the type of the term t). There is a clear connection between this approach to accommodating restricted free floating type judgements and our use of typed identity and equivalence predicates to avoid paradox. They are both instances of the same strategy.

So internal type judgements $t\epsilon' S$ are felicitous, provided S is a separation type $\{x\epsilon T.\varphi\}$, and that we are in a context where it can be shown independently that $t \in T$.

There is a choice about how to incorporate these observations into PTCT. We could change the language, add new rules and revise the model, or we can take expressions involving $t\epsilon'\{x\epsilon T.\varphi\}$—where we know $t \in T$—to be "syntactic sugar" for an equivalent representation that does not involve these new judgements, namely $(\lambda x(\lceil\varphi'\rceil))t$.

On the former approach, one way of proceeding is to add the logical constant ϵ' to the language of terms[13], and to adopt the the following tableau rule to determine that a judgement is felicitous.

Internal Separation Types

$$\frac{p \in \mathsf{Prop}}{z\epsilon'\{x\varepsilon T.p\} \in \mathsf{Prop}} \quad \textit{where } z \in T$$

The following two rules determine when the judgement is true.

Internal Separation Types

$$\frac{{}^{\mathsf{True}}(z\epsilon'\{x\varepsilon T.p\})^* \quad \textit{where } z \in T, p \in \mathsf{Prop}}{{}^{\mathsf{True}}p[z/x]}$$

Negated Internal Separation Types

$$\frac{\sim {}^{\mathsf{True}}(z\epsilon'\{x\varepsilon T.p\})^* \quad \textit{where } z \in T, p \in \mathsf{Prop}}{\sim {}^{\mathsf{True}}p[z/x]}$$

On the alternative approach, we can use the following definition.

Definition 10.1 (Restricted Free Floating Type Judgements)
In the context where $t \in T$, terms of the form $t\epsilon'\{x\epsilon T.\varphi\}$ are taken to be "syntactic sugar" for $(\lambda x(\lceil\varphi'\rceil))t$.

For presentational reasons, we here adopt the latter approach. In places we also use the notation $\in$ for ϵ' where the context makes it clear what is meant.

Whichever approach is taken, it is possible to revise our recursive definition of translation for term representable wffs to terms so that it includes a translation rule for unproblematic type judgements, for example[14]

$$\lceil a \in \{x \in T.\varphi\}\rceil = a\epsilon'\{x\epsilon T.\lceil\varphi\rceil\}$$

[13] An alternative would be to "overload" the existing ϵ constant.

[14] The original recursive translation is defined only for term-representable wffs. Adding this new rule for type judgements gives a translation whose result is specified for some non-representable wffs, namely type judgements of the form $a \in \{x \in T.\varphi\}$ where it cannot be shown that $a \in T$, but the result will not be a proposition. If the translation procedure is to be constrained to avoid such cases, then it needs to be revised so that it keeps track of the relevant typing context.

11 An Intensional Number Theory

We add an intensional number theory to PTCT, incorporating rules corresponding to the axioms in (26).

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

(23) Types: Num

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

(25) Axioms for Num: The usual Peano axioms, adapted to PTCT

(26) Axioms for $<_{\mathsf{Num}}$:

(a) $y \in \mathsf{Num} \rightarrow 0 <_{\mathsf{Num}} \mathit{succ}(y)$

(b) $x \in \mathsf{Num} \rightarrow x \not<_{\mathsf{Num}} 0$

(c) $x \in \mathsf{Num} \wedge y \in \mathsf{Num} \rightarrow (\mathit{succ}(x) <_{\mathsf{Num}} \mathit{succ}(y) \leftrightarrow x <_{\mathsf{Num}} y)$

The model theory can be extended in a straightforward way to support these new rules of the proof theory.

When we incorporate the intensional number theory into PTCT we lose completeness of the proof theory. However, it is important to recognize that incompleteness sets in only when tableau rules that encode number-theoretic inferences are applied. The basic logic and type system of PTCT without these rules and their corresponding definitions in the model theory remains complete.

By defining the cardinality of properties, we can express the truth conditions of proportional quantifiers in PTCT.

(27) Cardinality of properties $|p|_B$:

(a) $p \in (B \Longrightarrow \mathsf{Prop}) \wedge \sim \exists x(x \in B \wedge {}^{\mathsf{True}}px) \rightarrow |p|_B \cong_{\mathsf{Num}} 0$

(b) $p \in (B \Longrightarrow \mathsf{Prop}) \wedge b \in B \wedge {}^{\mathsf{True}}pb \rightarrow$
$|p|_B \cong_{\mathsf{Num}} \mathit{add}(|\lambda x(px \mathbin{\hat{\wedge}} \hat{\sim}(x \mathbin{\hat{=}}_B b)|_B)(\mathit{succ}(0))$

Note that we can give an equivalent type-theoretic formulation of cardinality. This is apparent if we observe the close relationship between properties and separation types.

(28) Cardinality of types $|\{x \epsilon B.px|_B$:

(a) $p \in (B \Longrightarrow \mathsf{Prop}) \wedge \sim \exists x(x \in B \wedge {}^{\mathsf{True}}px) \rightarrow$
$|\{x \epsilon B.px\}|_B \cong_{\mathsf{Num}} 0$

(b) $p \in (B \Longrightarrow \mathsf{Prop}) \wedge b \in B \wedge {}^{\mathsf{True}}pb \rightarrow$
$|\{x \epsilon B.px\}|_B \cong_{\mathsf{Num}} \mathit{add}(|\{x \epsilon B.px \mathbin{\hat{\wedge}} \hat{\sim}(x \mathbin{\hat{=}}_B b)\}|_B)(\mathit{succ}(0))$

We represent $\mathit{most}(p)(q)$ as follows:

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

Given that PTCT is a first-order theory in which all quantification is limited to first-order variables, this characterization of *most* effectively encodes a higher-order generalized quantifier within a first-order system.

11.1 Presburger Arithmetic

If we wish to avoid the incompleteness that results from using Peano arithmetic, we can adopt a weaker number theory, such as Presburger arithmetic [46].

The conventional presentations of Presburger arithmetic adopt the constants:

$$+, 1, 0, =$$

together with the following axioms

(30) $\forall x \sim(0 = x + 1)$
(31) $\forall xy(\sim(x = y) \rightarrow \sim(x + 1 = y + 1))$
(32) $\forall x(x + 0 = x)$
(33) $\forall xy((x + y) + 1 = x + (y + 1))$

and the axiom schema for induction.

(34) $(\phi[0] \wedge \forall x(\phi[x] \rightarrow \phi[x + 1])) \rightarrow \forall x(\phi[x])$

Presburger arithmetic does not include multiplication, and so it is not as strong as Peano arithmetic. It is not allow the formulation of theorems about prime numbers, for example. However, it is adequate for the purpose of determining the truth conditions of proportional cardinality quantifiers.

Adapting these axioms to give a number theory in PTCT will result in a system that is complete, and remains semi-decidable, but which is sufficiently expressive to cope with proportional cardinality quantifiers. To incorporate Presburger arithmetic PTCT, the language has to be extended to include $+, 0, 1$ as terms. As before, the wff $t \cong_{\mathsf{Num}} s$ can serve as the notion of numerical identity. All that remains is to add closure axioms for Num and the new terms, and adapt the Presburger axioms to PTCT.

(35) Closure axioms:
  (a) $0 \in \mathsf{Num}$
  (b) $1 \in \mathsf{Num}$
  (c) $(t \in \mathsf{Num} \wedge s \in \mathsf{Num}) \rightarrow (t + s) \in \mathsf{Num}$

(36) Presburger axioms:
  (a) $x \in \mathsf{Num} \rightarrow \sim(0 \cong_{\mathsf{Num}} x + 1)$
  (b) $(x \in \mathsf{Num} \wedge y \in \mathsf{Num}) \rightarrow (\sim(x \cong_{\mathsf{Num}} y) \rightarrow \sim(x + 1 \cong_{\mathsf{Num}} y + 1))$
  (c) $x \in \mathsf{Num} \rightarrow (x + 0 \cong_{\mathsf{Num}} x)$
  (d) $(x \in \mathsf{Num} \wedge y \in \mathsf{Num}) \rightarrow ((x + y) + 1 \cong_{\mathsf{Num}} x + (y + 1))$
  (e) $(\phi[0] \wedge \forall x((x \in \mathsf{Num} \wedge \phi[x]) \rightarrow \phi[x + 1])) \rightarrow$
  $\forall x(x \in \mathsf{Num} \rightarrow \phi[x])$

Cardinality and $<_{\mathsf{Num}}$ can be treated as before.

Although Presburger arithmetic is decidable, it has some unfortunate computational properties. Fischer and Rabin [16] show that the time required to decide the truth of a statement of length n is at least $2^{2^{cn}}$, for some constant c. It may be possible to optimise the implementation of the theory in a computational system for the evaluation of the truth conditions of proportional quantifiers.

12 A Type-Theoretical Approach to Anaphora

To illustrate the expressiveness of PTCT we combine our treatment of generalised quantifiers with our characterisation of separation types to provide a unified type-theoretic account of anaphora.[15]

We assume that all quantified NPs are represented as cardinality relations on the model of our treatment of *most*. Pronouns are taken to be appropriately typed free variables. If the free variable is within the scope of a set forming operator that specifies a subtype and it meets the same typing constraints as the variable bound by the operator, the variable can be interpreted as bound by the operator through substitution under α identity. This interpretation yields the bound reading of the pronoun.

(37) Every man loves his mother.

(38) $|\{x \in B.^{\mathsf{True}} man'(x) \wedge {}^{\mathsf{True}} love'(x, mother\text{-}of'(y))\}|_B \cong_{Num} |\{x \in B.^{\mathsf{True}} man'(x)\}|_B \Rightarrow$
$|\{x \in B.^{\mathsf{True}} man'(x) \wedge {}^{\mathsf{True}} love'(x, mother\text{-}of'(x))\}|_B \cong_{Num} |\{x \in B.^{\mathsf{True}} man'(x)\}|_B$

Representations of this kind are generated by compositional semantic operations as described by (for example) Lappin [34], and Lappin and Francez [36].

When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a dependent type parameter whose value is supplied by context. Generally, the value of this type parameter is obtained from the predicative part of the antecedent clause representation. In the default case, we treat the pronoun variable as bound by a universal quantifier in the language of wffs.

(39) Every student arrived.

(40) $|\{x \in B.^{\mathsf{True}} student'(x) \wedge {}^{\mathsf{True}} arrived'(x)\}|_B \cong_{Num} |\{x \in B.^{\mathsf{True}} student'(x)\}|_B$

(41) They sang.

(42) $\forall y \in A.({}^{\mathsf{True}} sang'(y))$
where $A = \{x \in B.^{\mathsf{True}} student'(x) \wedge {}^{\mathsf{True}} arrived'(x)\}$

In the case of proper names and existentially quantified NP antecedents we obtain the following.

(43) John arrived.

(44) ${}^{\mathsf{True}} arrived'(john)$

(45) He sang.

(46) $\forall y \in A.({}^{\mathsf{True}} sang'(y))$
where $A = \{x \in B.x \hat{=}_B john \wedge {}^{\mathsf{True}} arrived'(x)\}$

(47) Some man arrived.

(48) $|\{x \in B.^{\mathsf{True}} man'(x) \wedge {}^{\mathsf{True}} arrived'(x) \wedge {}^{\mathsf{True}} \phi(x)\}|_B >_{Num} 0$

(49) He sang.

(50) $\forall y \in A.({}^{\mathsf{True}} sang'(y))$
where $A = \{x \in B.^{\mathsf{True}} man'(x) \wedge {}^{\mathsf{True}} arrived'(x) \wedge {}^{\mathsf{True}} \phi(x)\}$

ϕ is a predicate that is specified in context and uniquely identifies a man who arrived in that context.

We handle donkey anaphora in PTCT through a type constraint on the variable corresponding to the pronoun.

[15] This approach to anaphora is extended to ellipsis in Fox and Lappin [18].

(51) Every man who owns a donkey beats it.

(52) $|\{x \in B.^{\mathsf{True}} man'(x) \wedge (|\{y \in B.^{\mathsf{True}} own'(x,y) \wedge {}^{\mathsf{True}} donkey'(y)\}|_B >_{Num} 0) \wedge$
$\forall z \in A({}^{\mathsf{True}} beat'(x,z))\}|_B$
$\cong_{Num}$
$|\{x \in B.^{\mathsf{True}} man'(x) \wedge (|\{y \in B.^{\mathsf{True}} own'(x,y) \wedge {}^{\mathsf{True}} donkey'(y)\}|_B >_{Num} 0)\}|_B$
where $A = \{y \in B.^{\mathsf{True}} own'(x,y) \wedge {}^{\mathsf{True}} donkey'(y)\}$

The representation asserts that every man who owns at least one donkey beats all of the donkeys that he owns.

Our type-theoretic account of donkey anaphora is similar in spirit to the E-type analysis proposed by Lappin and Francez [36]. There is, however, an important difference. Lappin and Francez [36] interpret an E-type pronoun as a choice function from the elements of an intersective set specified by the clause containing the antecedent NP to a range of values determined by this NP. Both the domain and range of the function are described informally in terms of the semantic representation of the antecedent clause. On the type-theoretic approach proposed here the interpretation of the E-type pronoun is specified explicitly through type constraints on variables in the semantic representation language. Therefore our account provides a more precise and properly formalized treatment of pronominal anaphora.

Ranta develops an analysis of anaphora within Martin-Löf Type Theory (MLTT) [47]. He represents donkey sentences as universal quantification over product types.[16]

(53) $\Pi z : (\Sigma x : man)(\Sigma y : donkey)(x\ owns\ y))(p(z)\ beats\ p(q(z))$

In this example, z is a variable over product pairs, and p and q are left and right projections, respectively, on the product pair, where

(54) (a) $p(z) : man$
(b) $q(z) : (\Sigma y : donkey)(p(z)\ owns\ y)$
(c) $p(q(z)) : donkey$
(d) $q(q(z)) : (p(z)\ owns\ p(q(z)))$.

Ranta's account does not generate the existential reading of donkey sentences [45]. On the preferred interpretation of (55), for example, every person who had a quarter put at least one quarter in a parking meter.

(55) Every person who had a quarter put it in a parking meter.

We can generate these readings by treating the principle that the free variable representing a pronoun is bound by a universal quantifier as defeasible. We can then substitute an existential for a universal quantifier.

(56) $|\{x \in B.^{\mathsf{True}} man'(x) \wedge (|\{y \in B.^{\mathsf{True}} had'(x,y) \wedge {}^{\mathsf{True}} quarter'(y)\}|_B >_{Num} 0)$
$\wedge \exists z \in A({}^{\mathsf{True}} put\text{-}in\text{-}a\text{-}parking\text{-}meter'(x,z))\}|_B$
$\cong_{Num}$
$|\{x \in B.^{\mathsf{True}} person'(x) \wedge (|\{y \in B.^{\mathsf{True}} had'(x,y) \wedge {}^{\mathsf{True}} quarter'(y)\}|_B >_{Num} 0)\}|_B$
where $A = \{y \in B.^{\mathsf{True}} had'(x,y) \wedge {}^{\mathsf{True}} quarter'(y)\}$

[16] Note that here expressions of the form $\Pi x : (T)(S)$ denote a dependent product type. This is not to be confused with PTCT's polymorphic types, whose form $(\Pi X.T)$ is superficially similar.

This representation asserts that every person who had a quarter put at least one quarter that he/she had in a parking meter.

By taking the binding of the free variable corresponding to a pronoun by a universal quantifier as defeasible through lexical semantic and real world knowledge factors we are adopting a version of the pragmatic maximality condition on the interpretation of unbound pronouns proposed in Lappin and Francez [36]. Therefore, the choice between universal and existential readings of donkey pronouns is a pragmatic effect, and not the result of semantic ambiguity.[17]

As Ranta acknowledges, his universal quantification-over-pairs analysis follows DRT [30] in inheriting the proportionality problem in a sentence like the following (57) [26, 29].

(57) Most men who own a donkey beat it.

On the universal quantification-over-pairs account of donkey anaphora, contrary to the desired interpretation, the sentence is true in a model in which ten men own donkeys, nine men own a single donkey each and do not beat it, while the tenth man owns ten donkeys and beats them all. The preferred reading of the sentence requires that it is false in this model.

Ranta cites Sundholm's [51] solution to the proportionality problem. This suggestion involves positing a second quantifier *most* on product pairs $(\Sigma x : A)(B(x))$ that is interpreted as applying an A injection to the pairs, where this injection identifies only the first element of each pair as within the domain of quantification. Defining an additional mode of quantification as a distinct reading of *most* in order to generate the correct interpretation of (57) would seem to be an ad hoc approach to the difficulty. There is no evidence for taking *most* as ambiguous between two quantificational readings beyond the need to avoid the inability of the quantification-over-pairs analysis to yield the correct results for this case. Assuming two modes of quantification adds considerable complication to the type-theoretic approach to anaphora without independent motivation.

The proportionality problem does not arise on our account. *Most* is represented as a cardinality relation (generalized quantifier) in which quantification is over the elements of the set corresponding to the subject restriction rather than over pairs. Therefore the sentence is false in this model.

13 Relevance of PTCT to Computational Semantics

We believe that the features of PTCT that we have discussed here make it particularly appropriate for implementing theorem proving systems for natural language semantics. It adds appropriate "expressiveness" to a first-order theory, without an undesirable increase in formal power. Historically, higher-order systems have dominated the field in natural language semantics. In general, the set of theorems of such systems is not r.e. For this reason, a first-order system is preferable, if it is sufficiently expressive for the relevant domain. Basic first-order logic by itself does not provide the features that are required for natural language semantics. It is insufficiently expressive for this purpose. PTCT is a first-order system whose expressiveness is designed for natural language semantics. It directly supports fine-grained intensionality and a flexible system of types. We have formulated tableau rules for PTCT, which provide an effective theorem proving procedure for the underlying logic and type system of the theory (i.e. PTCT without the intensional number theory).

[17] This approach contrasts with that of Chierchia [12] who uses two distinct formal mechanisms, dynamic binding and E-type pronoun choice-functions, to represent each of these readings, thus treating donkey sentences as systematically ambiguous.

We have also characterized intensionality and achieved find grained intensional distinctions without invoking (im)possible worlds. Therefore, PTCT provides a treatment of intensions that is entirely independent of modality.

14 Conclusion

We have constructed a first-order fine-grained intensional logic with flexible Curry typing, PTCT, for the semantic representation of natural languages. PTCT contains typed predicates for intensional identity and extensional equality. Its proof theory permits us to prove that identity of intension entails identity of extension, but that the converse does not hold.

The theory can be distinguished from Aczel's Frege Structures [1] and related, weakly typed theories of properties (PT) [54] in two ways. First, there is an explicit notion of polymorphic type within the theory, which is more appropriate for natural language semantics than the universal type of PT. Second, the type Prop can appear in intensional representations of propositions. This allows us to express the fact that, for example, the universal quantification in statements of the form *John believes everything that Mary believes* ranges only over propositions. In PT, this requirement can only be expressed as an external constraint [56].

We have provided a model theory for PTCT using extensional models for the untyped λ-calculus enriched with interpretations of Curry types. The restrictions that we impose on separation types, comprehension types, quantification over types, and the relation between the three sublanguages of PTCT insure that it remains a first-order system in which its enriched expressive power comes largely through quantification over terms and the representation of types as terms within the language.

Unlike alternative hyperintensionalist frameworks that have been proposed, this logic distinguishes among provably equivalent propositions without resorting to impossible worlds to sustain the distinction. The incorporation of Curry typing into the logic allows us to sustain weak polymorphism. Subtypes also permit a us to develop a uniform type-theoretical account of pronominal anaphora with wide empirical coverage. This application illustrates the expressive power of the system for expressing complex semantic phenomena of natural language in a straightforward and formally integrated way.

Acknowledgements

Earlier versions of the ideas discussed in this paper were presented at LACL2001 (Le Crosic), Seventh International Workshop on Natural Language Understanding and Logic Programming 2002 (Copenhagen), Seventh Symposium for Logic and Language 2002 (Pécs), Sens du Représentation 2003 (Montreal), Fields Workshop on Mathematical Linguistics 2003 (Ottawa), Logical Foundations of Computational Linguistics Workshop at LICS 2003 (Ottawa), Recent Advances in Natural Language Processing 2003 (Borovets), the Cognitive Science Colloquium, University of Osnabrück (2003), the Artificial Intelligence Colloquium of the Computer Science Department, Harvard University (2004), and the Human Communication Research Centre Colloquium of the University of Edinburgh (2004). We are grateful to the participants of these forums for helpful discussion. We would also like to thank Peter Aczel, Paul Gilmore, Graham Katz, Kai-Uwe Kühnberger, Michael Kolhasen, Jim Lambek, Tom Maibaum, Gerald Penn, Ian Pratt, Michael Rabin, Dana Scott, Phil Scott, Ray Turner, and Yoad Winter for helpful advice on a number of significant formal issues. We are particularly indebted to Robert Schubert for

careful, detailed and constructive comments on our proposed proof and model theories. This paper developed out of joint work with Carl Pollard on the formal foundations of intensional semantics. He has played an important role in shaping our ideas on general issues of intensionality and type theory, and we would like to thank him for his contribution to our work. Needless to say, we bear sole responsibility for any errors that may remain in the paper. The research of the second author has been supported by grant number AN/2687/APN from the Arts and Humanities Research Board of the UK, and grant number RES-000-23-0065 from the Economic and Social Research Council of the UK.

References

[1] P. Aczel. Frege structures and the notions of proposition, truth and set. In Barwise, Keisler, and Keenan, editors, *The Kleene Symposium*, North Holland Studies in Logic, pages 31–39. North Holland, 1980.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North Holland, Amsterdam, second edition, 1984.

[3] J. Barwise. Information and impossibilities. *The Notre Dame Journal of Formal Logic*, 38:488–515, 1997.

[4] J. Barwise and R. Cooper. Generalised quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

[5] J Barwise and J. Etchemendy. Information, infons, and inference. In R. Cooper, K. Mukai, and J. Perry, editors, *Situation Theory and Its Applications*, volume 1, pages 33–78. CSLI, Stanford, CA, 1990.

[6] G. Bealer. *Quality and Concept*. Clarendon Press, Oxford, 1982.

[7] E. W. Beth. L'existence en mathématiques. In *Collection de logique mathématiques, série A*. Gauthier-Villars/Naewelaerts, Paris, Louvain, 1956. (Beth's lecturs at Sorbonne University (Paris), March 29 – April 2, 1954).

[8] R. Carnap. *Meaning and Necessity*. University of Chicago Press, Chicago, 1947.

[9] G. Chierchia. Nominalisation and Montague grammar: a semantics without types for natural languages. *Linguistics and Philosophy*, 5:303–354, 1982.

[10] G. Chierchia. *Topics in the Semantics of Infinitives and Gerunds*. PhD thesis, University of Massachusetts, Amherst, 1984. Published in 1989 by Garland, New York.

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

[12] Gennaro Chierchia. *Dynamics of Meaning*. University of Chicago Press, Chicago, 1995.

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

[14] R. Cooper. The role of situations in generalized quantifiers. In S. Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 65–86. Blackwell, Oxford, 1996.

[15] M. J. Cresswell. *Stuctured Meanings*. MIT Press, Cambridge, Massachusetts, 1985.

[16] M. J. Fischer and M. O. Rabin. Super-exponential complexity of presburger arithmetic. In *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*, volume 7, pages 27–41, 1974.

[17] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, Berlin, 1996.

[18] C. Fox and S. Lappin. A type-theoretic approach to anaphora and ellipsis. In G. Angelova, K. Bontcheva, R. Mitkov, N. Nicolov, and N. Nikolov, editors, *Proceedings of Recent Advances in Natural Language Processing*, pages 1–10. Bulgarian Academy of Science, Borovets, Bulgaria, 2003.

[19] C. Fox and S. Lappin. *Formal Foundations of Intensional Semantics*. Blackwell, Oxford, to appear.

[20] C. Fox, S. Lappin, and C. Pollard. A higher-order, fine-grained logic for intensional semantics. In G. Alberti, K. Balough, and P. Dekker, editors, *Proceedings of the Seventh Symposium for Logic and Language*, pages 37–46, Pecs, Hungary, 2002.

[21] D. Gallin. *Intensional and Higher-Order Modal Logic*. North-Holland, Amsterdam, 1975.

[22] G. Gazdar. A cross-categorial semantics for coordination. *Linguistics and Philosophy*, 3:407–409, 1980.

[23] H. Gregory. Relevance logic and natural language semantics. In *Proceedings of Formal Grammar 2002*, Trento, Italy, 2002.

[24] J. Groenendijk and M. Stokhof. Dynamic Montague grammar. In L. Kálmán and L. Polós, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48, Budapest, Hungary, 1990. Akadémiai Kiadó. Also available as ITLI Prepublication Series LP–90–02.

[25] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.
[26] I. Heim. E-type pronouns and donkey anaphora,. *Linguistics and Philosophy*, 13:137–177, 1990.
[27] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:323–344, 1950.
[28] R. Jeffrey. *Formal Logic: Its Scope and Limits*. McGraw-Hill, New York, 1982.
[29] N. Kadmon. Uniqueness. *Linguistics and Philosophy*, 13:237–324, 1990.
[30] H. Kamp and U. Reyle. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht, 1993.
[31] E. Keenan and K. Stavi. A semantic characterization of natural language determiners. *Linguistics and Philosophy*, 9:253–326, 1986.
[32] E. Keenan and D. Westerståhl. Generalized quantifiers in linguistics and logic. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 838–893. Elsevier, Amsterdam, 1997.
[33] F. Landman. *Towards a Theory of Information*. Foris, Dordrecth, 1986.
[34] S. Lappin. Donkey pronouns unbound. *Theoretical Linguistics*, 15:263–286, 1989.
[35] S. Lappin. An intensional parametric semantics for vague quantifiers. *Linguistics and Philosophy*, 23:599–620, 2000.
[36] S. Lappin and N. Francez. E-type pronouns, I-sums, and donkey anaphora. *Linguistics and Philosophy*, 17:391–428, 1994.
[37] S. Lappin and C. Pollard. A hyperintensional theory of natural language interpretation without indices or situations. ms., King's College, London and Ohio State University, 1999.
[38] S. Lappin and C. Pollard. Strategies for hyperintensional semantics. ms., King's College, London and Ohio State University, 2000.
[39] R. Larson and G. Segal. *Knowledge of Meaning*. MIT Press, Cambridge, MA, 1995.
[40] P. Martin-Löf. Constructive mathematics and computer programming. In Cohen, Los, Pfeiffer, and Podewski, editors, *Logic, Methodology and Philosophy of Science VI*, pages 153–179. North Holland, 1982.
[41] P. Martin-Löf. *Studies in Proof Theory (Lecture Notes)*. Bibliopolis, Napoli, 1984.
[42] A. Meyer. What is a model of the lambda calculus? *Information and Control*, 52:87–122, 1982.
[43] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, CT/London, UK, 1974. Edited with an introduction by R. H. Thomason.
[44] R. Muskens. *Meaning and Partiality*. CSLI and FOLLI, Stanford, CA, 1995.
[45] J. Pelletier and L. Schubert. Generically speaking. In G. Chiercihia, B. H. Partee, and R. Turner, editors, *Propteries, Types, anmd Meaning*, volume 2. Kluwer, Dordrecht, 1989.
[46] Mojzesz Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves, Warszawa*, pages 92–101, 1929.
[47] A. Ranta. *Type Theoretic Grammar*. Oxford University Press, Oxford, 1994.
[48] I. A. Sag. Coordination and underspecification. In Jong-Bok Kim and Stephen Wechsler, editors, *Proceedings of the Ninth International Conference on HPSG*, pages 267–291, Stanford, CA, 2003.
[49] J. M. Smith. An interpretation of Martin-Löf's Type Theory in a type-free theory of propositions. *Journal of Symbolic Logic*, 49, 1984.
[50] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlin, 1968.
[51] G. Sundholm. Constructive generalised quantifiers. *Synthese*, 79:1–12, 1989.
[52] R. Thomason. A modeltheory for propositional attitudes. *Linguistics and Philosophy*, 4:47–70, 1980.
[53] R. Turner. A theory of properties. *Journal of Symbolic Logic*, 52(2):455–472, June 1987.
[54] R. Turner. Properties, propositions, and semantic theory. In *Proceedings of Formal Semantics and Computational Linguistics*, Switzerland, 1988.
[55] R. Turner. Properties, propositions and semantic theory. In M. Rosner and R. Johnson, editors, *Computational Linguistics and Formal Semantics*, Studies in Natural Language Processing, pages 159–180. Cambridge University Press, Cambridge, 1992.
[56] R. Turner. Types. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 535–586. Elsevier, 1997.
[57] J. van Benthem. *Language in Action*. Studies in Logic. North-Holland, Amsterdam, 1991.
[58] Edward N. Zalta. *Intensional Logic and the Metaphysics of Intensionality*. MIT Press/Bradford Books, Cambridge, MA, 1988.