Natural Language Semantics in a Flexibly Typed Intensional Logic

Chris Fox
Dept. of Computer Science
University of Essex
foxcj@essex.ac.uk

Shalom Lappin
Dept. of Computer Science
King’s College London
lappin@dcs.kcl.ac.uk
Overview

- Two Traditional Assumptions of Formal Semantics
- Property Theory with Curry Typing (PTCT): An expressive first-order logic with fine-grained intensionality
- Syntax and proof theory
- Model theory
- Restricted polymorphic types
- An intensional number theory and generalized quantifiers
- A type-theoretical account of dynamic anaphora
- A type-theoretical of ellipsis
- Conclusions and future work
Background
Assumption I

First Assumption: Functional Types and Higher-Order Logic

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.
Assumption I

First Assumption: Functional Types and Higher-Order Logic

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.
- In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers.
Assumption I

First Assumption: Functional Types and Higher-Order Logic

• Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.

• In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers

• e.g. Montague (1974); Gallin (1975); Barwise and Cooper (1981); Keenan and Stavi (1986).
Assumption I

First Assumption: Functional Types and Higher-Order Logic

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.
- In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers.
- e.g. Montague (1974); Gallin (1975); Barwise and Cooper (1981); Keenan and Stavi (1986).
- If we are interested in a computationally viable theory, then we should be concerned about the formal power of the theory.
Assumption I

First Assumption: Functional Types and Higher-Order Logic

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.
- In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers.
- E.g. Montague (1974); Gallin (1975); Barwise and Cooper (1981); Keenan and Stavi (1986).
- If we are interested in a computationally viable theory, then we should be concerned about the formal power of the theory.
- Property Theory with Curry Typing (PTCT) is a first-order theory with function types.
Assumption I

*First Assumption: Functional Types and Higher-Order Logic*

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.
- In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers
- e.g. Montague (1974); Gallin (1975); Barwise and Cooper (1981); Keenan and Stavi (1986).
- If we are interested in a computationally viable theory, then we should be concerned about the formal power of the theory.
- Property Theory with Curry Typing (PTCT) is a first-order theory with function types
- ...and separation types (sub-types)
Assumption I

First Assumption: Functional Types and Higher-Order Logic

- Usually assumed that higher-order logic and type theory are necessary to achieve the expressive power required for NL semantic.

- In particular, the function types of higher-order systems are taken to be required for generalised quantifiers and modifiers.

- e.g. Montague (1974); Gallin (1975); Barwise and Cooper (1981); Keenan and Stavi (1986).

- If we are interested in a computationally viable theory, then we should be concerned about the formal power of the theory.

- Property Theory with Curry Typing (PTCT) is a first-order theory with function types.

- ...and separation types (sub-types)

- ...and polymorphic types.
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
Assumption II

*Second Assumption: Intensions and Possible Worlds*

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
- It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts.
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
- It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts.
- “Every prime number is divisible only by itself and 1”
  \[\Leftrightarrow\text{"If } A \subseteq B \text{ and } B \subseteq A, \text{ then } A = B\]
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
- It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts.
- “Every prime number is divisible only by itself and 1”
  ⇔ “If $A \subseteq B$ and $B \subseteq A$, then $A = B$”
- “John believes that every prime number is divisible only by itself and 1”
  ≠ “John believes that if $A \subseteq B$ and $B \subseteq A$, then $A = B$”
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
- It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts.
- “Every prime number is divisible only by itself and 1”
  \( \iff \) “If \( A \subseteq B \) and \( B \subseteq A \), then \( A = B \)”
- “John believes that every prime number is divisible only by itself and 1”
  \( \neq \) “John believes that if \( A \subseteq B \) and \( B \subseteq A \), then \( A = B \)”
- PTCT follows Bealer (1982), taking intensions to be basic (actually represented by first-order terms).
Assumption II

Second Assumption: Intensions and Possible Worlds

- The view that characterizes intensions as functions from possible worlds (situations) to extensions has been influential at least since Carnap (1947).
- It achieves detailed formal expression in Montague (1974).
- This treatment of intensions is not sufficiently fine-grained.
- It entails that logically equivalent expressions are cointensional and so intersubstitutable in all contexts.
- “Every prime number is divisible only by itself and 1”
  \(\iff\) “If \(A \subseteq B\) and \(B \subseteq A\), then \(A = B\)”
- “John believes that every prime number is divisible only by itself and 1”
  \(\neq\) “John believes that if \(A \subseteq B\) and \(B \subseteq A\), then \(A = B\)”
- PTCT follows Bealer (1982), taking intensions to be basic (actually represented by first-order terms).
- Intensions are independent of modality, and identity is not reduced to equivalence.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
- In which case we map NL to terms.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
- In which case we map NL to terms.
- The language of terms will have to be more complicated.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
- In which case we map NL to terms.
- The language of terms will have to be more complicated.
- Some terms will correspond to propositions.
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
- In which case we map NL to terms.
- The language of terms will have to be more complicated.
- Some terms will correspond to propositions.
- If the language of wffs and types proves a term represents a proposition...
The General Approach

- In formal semantics, NL is mapped to a formal logic.
- Such logics typically consist of wff, terms and types.
- NL is usually mapped directly to wffs.
- Intensions (e.g. in IL) are derived from wffs.
- Why not use terms for our notion of intensionality?
- In which case we map NL to terms.
- The language of terms will have to be more complicated.
- Some terms will correspond to propositions.
- If the language of wffs and types proves a term represents a proposition...
- ...then the truth conditions of such terms can be considered.
PTCT: Property Theory with Curry Typing
PTCT: Basic Syntax

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

**Terms**  \( t ::= x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t \)

(logical constants) \( l ::= \hat{\wedge} \mid \hat{\vee} \mid \hat{\rightarrow} \mid \hat{\leftrightarrow} \mid \hat{\neg} \mid \hat{\forall} \mid \hat{\exists} \mid \hat{T} \mid \hat{\neg}T \mid \epsilon T \)
PTCT: Basic Syntax

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

Terms  \( t ::= x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t \)

(\emph{logical constants}) \( l ::= \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow \mid \top \mid \bot \mid \neg \mid \exists \mid \equiv \mid \top \mid \bot \mid \epsilon T \)

- The language of terms is the untyped \( \lambda \)-calculus, enriched with logical constants.
  Used to \emph{represent} the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the \( \lambda \)-calculus.
PTCT: Basic Syntax

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

**Terms** \( t ::= x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t \)

( logical constants) \( l ::= \hat{\wedge} \mid \hat{\vee} \mid \hat{\to} \mid \hat{\iff} \mid \hat{\downarrow} \mid \hat{\forall} \mid \hat{\exists} \mid \hat{=}_T \mid \hat{\neq}_T \mid \epsilon T \)

- The language of terms is the untyped \( \lambda \)-calculus, enriched with logical constants. Used to *represent* the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the \( \lambda \)-calculus.

**Types** \( T ::= B \mid \text{Prop} \mid T \implies S \)
PTCT: Basic Syntax

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

**Terms** \( t ::= x \mid c \mid l \mid T \mid \lambda x(t) \mid (t)t \)

(logical constants) \( l ::= \hat{\wedge} \mid \hat{\vee} \mid \hat{\rightarrow} \mid \hat{\leftrightarrow} \mid \hat{\downarrow} \mid \hat{\vee} \mid \hat{\exists} \mid \hat{\exists}_T \mid \hat{\exists}_T \mid \epsilon T \)

- The language of terms is the untyped \( \lambda \)-calculus, enriched with logical constants. Used to *represent* the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the \( \lambda \)-calculus.

**Types** \( T ::= B \mid \text{Prop} \mid T \rightarrow S \)

- The languages of types and terms are combined with appropriate rules and axioms to produce a Curry-typed \( \lambda \)-calculus (e.g. Turner 1997).
PTCT: Basic Syntax

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

**Terms**  
\[ t ::= x | c | l | T | \lambda x(t) | (t)t \]

(logical constants)  
\[ l ::= \land | \lor | \rightarrow | \leftrightarrow | \perp | \top | \exists_T | \equiv_T | \epsilon_T \]

- The language of terms is the untyped \(\lambda\)-calculus, enriched with logical constants. Used to represent the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the \(\lambda\)-calculus.

**Types**  
\[ T ::= B | Prop | T \rightarrow S \]

- The languages of types and terms are combined with appropriate rules and axioms to produce a Curry-typed \(\lambda\)-calculus (e.g. Turner 1997).

**Wff**  
\[ \varphi ::= \alpha | (\varphi \land \psi) | (\varphi \lor \psi) | (\varphi \rightarrow \psi) | (\varphi \leftrightarrow \psi) | (\forall x \varphi) | (\exists x \varphi) | \text{true} \]

(atomic wff)  
\[ \alpha ::= t \equiv_T s | \perp | t \in T | t \equiv_T s \]
PTCT: Basic Syntax

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

**Terms** \( t ::= x | c | l | T | \lambda x(t) | (t)t \)

(logical constants) \( l ::= \hat{\text{\&}} | \hat{\text{\lor}} | \hat{\text{\rightarrow}} | \hat{\text{\leftrightarrow}} | \hat{\text{\textend{itemize}}}

- The language of terms is the untyped \( \lambda \)-calculus, enriched with logical constants. Used to *represent* the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the \( \lambda \)-calculus.

**Types** \( T ::= B | \text{Prop} | T \implies S \)

- The languages of types and terms are combined with appropriate rules and axioms to produce a Curry-typed \( \lambda \)-calculus (e.g. Turner 1997).

**Wff** \( \phi ::= \alpha | (\phi \land \psi) | (\phi \lor \psi) | (\phi \rightarrow \psi) | (\phi \leftrightarrow \psi) | (\forall \phi) | (\exists \phi) | \text{true}t \)

(atomic wff) \( \alpha ::= t =_T s | \bot | t \in T | t \equiv_T s \)

- The first-order language of wffs will be used to formulate type judgements for terms, and truth conditions for those terms judged to be in \( \text{Prop} \).
PTCT: Basic Syntax

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

**Terms** $ t ::= x | c | l | T | \lambda x(t) | (t)t 

(logical constants) $ l ::= \hat{\top} | \hat{\bot} | \hat{\rightarrow} | \hat{\leftrightarrow} | \hat{\forall} | \hat{\exists} | \hat{=} | \hat{\equiv} | \epsilon T $

- The language of terms is the untyped $\lambda$-calculus, enriched with logical constants. Used to *represent* the interpretations of natural language expressions. It has no internal logic! The identity criteria are those of the $\lambda$-calculus.

**Types** $ T ::= B | \text{Prop} | T \implies S $

- The languages of types and terms are combined with appropriate rules and axioms to produce a Curry-typed $\lambda$-calculus (e.g. Turner 1997).

**Wff** $ \varphi ::= \alpha | (\varphi \land \psi) | (\varphi \lor \psi) | (\varphi \rightarrow \psi) | (\varphi \leftrightarrow \psi) | (\forall x \varphi) | (\exists x \varphi) | \text{true} t $

(atomic wff) $ \alpha ::= t =_T s | \bot | t \in T | t \equiv_T s $

- The first-order language of wffs will be used to formulate type judgements for terms, and truth conditions for those terms judged to be in $\text{Prop}$.

- If a term $t$ represents a proposition, $\text{true}(t)$ is a wff that denotes its truth conditions. The identity criteria of wffs are those of their truth conditions.
PTCT: Rules and Axioms

Here we exemplify some of these kinds of rules as they apply to conjunction, both as it appears in the language of wff ($\land$), and in the language of terms ($\hat{\land}$)

- The basic connectives of the wff

\[
\frac{\varphi \quad \psi}{\varphi \land \psi} \quad \land i \quad \frac{\varphi \land \psi}{\varphi} \quad \land e \quad \frac{\varphi \land \psi}{\psi} \quad \land e
\]
PTCT: Rules and Axioms

Here we exemplify some of these kinds of rules as they apply to conjunction, both as it appears in the language of wff (\(\land\)), and in the language of terms (\(\hat{\land}\)).

- The basic connectives of the wff

\[
\frac{\varphi \quad \psi}{\varphi \land \psi} \quad \land i \quad \frac{\varphi \land \psi}{\varphi} \quad \land e \quad \frac{\varphi \land \psi}{\psi} \quad \land e
\]

- Typing rules for \(\lambda\)-terms

\[t \in \text{Prop} \land t' \in \text{Prop} \rightarrow (t \hat{\land} t') \in \text{Prop}\]
PTCT: Rules and Axioms

Here we exemplify some of these kinds of rules as they apply to conjunction, both as it appears in the language of wff (\(\land\)), and in the language of terms (\(\hat{\land}\))

- The basic connectives of the wff
  \[
  \frac{\varphi \quad \psi}{\varphi \land \psi} \land i \quad \frac{\varphi \land \psi}{\varphi} \land e \quad \frac{\varphi \land \psi}{\psi} \land e
  \]

- Typing rules for \(\lambda\)-terms
  \[
  t \in \text{Prop} \land t' \in \text{Prop} \rightarrow (t \hat{\land} t') \in \text{Prop}
  \]

- Truth conditions for Propositions
  \[
  t \in \text{Prop} \land t' \in \text{Prop} \rightarrow (\text{true}(t \hat{\land} t') \leftrightarrow \text{true} t \land \text{true} t')
  \]
Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- $t \equiv_T s$ states that the terms $t, s$ are extensionally equivalent in type $T$. 
Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- \( t \cong_T s \) states that the terms \( t, s \) are extensionally equivalent in type \( T \)

- **Propositions:** in the case where two terms \( t, s \) are propositions (\( t, s \in \text{Prop} \)), then \( t \cong_{\text{Prop}} s \) corresponds to \( t \leftrightarrow s \)
Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- $t \equiv_T s$ states that the terms $t, s$ are extensionally equivalent in type $T$
  - **Propositions:** in the case where two terms $t, s$ are propositions ($t, s \in \text{Prop}$), then $t \equiv_{\text{Prop}} s$ corresponds to $t \leftrightarrow s$
  - **Predicates:** in the case where two predicates of $T$ are extensionally equivalent $t \equiv_{(T \rightarrow \text{Prop})} s$ then $t, s$ each hold of the same elements of $T$.

Therefore $\forall x (x \in T \rightarrow (\text{true}_t(x) \leftrightarrow \text{true}_s(x)))$
Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- $t \equiv_T s$ states that the terms $t, s$ are extensionally equivalent in type $T$
  - **Propositions:** in the case where two terms $t, s$ are propositions ($t, s \in \text{Prop}$), then $t \equiv_{\text{Prop}} s$ corresponds to $t \leftrightarrow s$
  - **Predicates:** in the case where two predicates of $T$ are extensionally equivalent $t \equiv_{(T \rightarrow \text{Prop})} s$ then $t, s$ each hold of the same elements of $T$. Therefore $\forall x (x \in T \rightarrow (\text{true}_t(x) \leftrightarrow \text{true}_s(x)))$

- $t =_T s$ states that two terms are intensionally identical in type $T$. 

Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- $t \equiv_T s$ states that the terms $t, s$ are extensionally equivalent in type $T$
  - Propositions: in the case where two terms $t, s$ are propositions ($t, s \in \text{Prop}$), then $t \equiv_{\text{Prop}} s$ corresponds to $t \leftrightarrow s$
  - Predicates: in the case where two predicates of $T$ are extensionally equivalent $t \equiv_{(T \rightarrow \text{Prop})} s$ then $t, s$ each hold of the same elements of $T$.
    Therefore $\forall x (x \in T \rightarrow (\text{true}_t(x) \leftrightarrow \text{true}_s(x)))$

- $t =_T s$ states that two terms are intensionally identical in type $T$.
  - The rules for intensional identity are essentially those of the $\lambda\alpha\beta\eta$-calculus.
Equivalence and Identity

There are two equivalence notions in this theory: intensional identity and extensional equivalence.

- $t \approx_T s$ states that the terms $t, s$ are extensionally equivalent in type $T$
  - **Propositions:** in the case where two terms $t, s$ are propositions ($t, s \in \text{Prop}$), then $t \cong_{\text{Prop}} s$ corresponds to $t \leftrightarrow s$
  - **Predicates:** in the case where two predicates of $T$ are extensionally equivalent $t \cong_{(T \rightarrow \text{Prop})} s$ then $t, s$ each hold of the same elements of $T$.
    Therefore $\forall x (x \in T \rightarrow (\text{true}_t(x) \leftrightarrow \text{true}_s(x)))$

- $t =_T s$ states that two terms are intensionally identical in type $T$.
  - The rules for intensional identity are essentially those of the $\lambda\beta\eta$-calculus.

- We are able to derive $t =_T s \rightarrow t \approx_T s$ for all types inhabited by $t, (s)$, but not $t \cong_T s \rightarrow t =_T s$. 
Extension of the Type System
Separation Types

• Add \( \{ x \in T : \varphi' \} \) to the types
Separation Types

- Add \( \{ x \in T : \phi' \} \) to the types
- \( \text{SP: } z \in \{ x \in T : \phi \} \iff (z \in T \land \phi'[z/x]) \)

Note that there is an issue here concerning the nature of \( \phi' \). To ensure the theory is first-order, this type needs to be term representable, so \( \phi' \) must be term representable. To this end, we define a term representable fragment of the language of wffs. [We won’t go into the details here. They are in the paper.]

We use separation types for dynamic analyses of anaphora and ellipsis.
Separation Types

- Add \( \{x \in T : \varphi'\} \) to the types
- \( \text{SP}: z \in \{x \in T : \varphi\} \leftrightarrow (z \in T \land \varphi'[z/x]) \)
- Note that there is an issue here concerning the nature of \( \varphi \). To ensure the theory is first-order, this type needs to be term representable, so \( \varphi' \) must be term representable.
Separation Types

- Add \( \{ x \in T : \varphi' \} \) to the types
- SP: \( z \in \{ x \in T : \varphi \} \leftrightarrow (z \in T \land \varphi'[z/x]) \)
- Note that there is an issue here concerning the nature of \( \varphi \). To ensure the theory is first-order, this type needs to be term representable, so \( \varphi' \) must be term representable.
- To this end, we define a term representable fragment of the language of wffs. [We won’t go into the details here. They are in the paper.]
Separation Types

- Add \( \{x \in T : \varphi'\} \) to the types
- SP: \( z \in \{x \in T : \varphi\} \iff (z \in T \land \varphi'[z/x]) \)

- Note that there is an issue here concerning the nature of \( \varphi \). To ensure the theory is first-order, this type needs to be term representable, so \( \varphi' \) must be term representable.

- To this end, we define a term representable fragment of the language of wffs. [We won’t go into the details here. They are in the paper.]

- We use separation types for dynamic analyses of anaphora and ellipsis.
Polymorphic Types

- Enrich the language of types to include type variables $X$, and the wffs to include quantification over types $\forall X \phi, \exists X \phi$. 

Note that PM is impredicative (the type quantification ranges over the types that are being defined). This can be avoided by using Polymorphic Kinds.
Polymorphic Types

• Enrich the language of types to include type variables \( X \), and the wffs to include quantification over types \( \forall X \varphi, \exists X \varphi \).

• Add \( \Pi X.T \) to the language of types, governed by the following axiom:
Polymorphic Types

- Enrich the language of types to include type variables $X$, and the wffs to include quantification over types $\forall X \varphi, \exists X \varphi$.
- Add $\Pi X. T$ to the language of types, governed by the following axiom:
  \[ \text{PM} \quad f \in \Pi X. T \iff \forall X (f \in T) \]
Polymorphic Types

- Enrich the language of types to include type variables $X$, and the wffs to include quantification over types $\forall X \phi, \exists X \phi$.
- Add $\Pi X.T$ to the language of types, governed by the following axiom:
  \[ \text{PM} \quad f \in \Pi X.T \iff \forall X(f \in T) \]
- Note that PM is impredicative (the type quantification ranges over the types that are being defined).
Polymorphic Types

- Enrich the language of types to include type variables $X$, and the wffs to include quantification over types $\forall X \varphi, \exists X \varphi$.

- Add $\Pi X.T$ to the language of types, governed by the following axiom:
  
  $PM \quad f \in \Pi X.T \iff \forall X (f \in T)$

- Note that PM is impredicative (the type quantification ranges over the types that are being defined).

- This can be avoided by using Polymorphic Kinds. Quantification can range over types, but not the Polymorphic Kinds themselves.
Polymorphism in NL

Polymorphism in Natural Language

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).
Polymorphism in NL

Polymorphism in Natural Language

• NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).

• Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.
Polymorphism in Natural Language

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).

- Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.

- More “natural” treatments of such phenomena are possible if we allow a flexible system of types (such as Curry-style typing), where
Polymorphism in NL

Polymorphism in Natural Language

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).
- Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.
- More “natural” treatments of such phenomena are possible if we allow a flexible system of types (such as Curry-style typing), where
  1. expressions can belong to more than one type;
Polymorphism in NL

Polymorphism in Natural Language

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).
- Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.
- More “natural” treatments of such phenomena are possible if we allow a flexible system of types (such as Curry-style typing), where
  1. expressions can belong to more than one type;
  2. functional types can apply to arguments of several types.
**Polymorphism in NL**

*Polymorphism in Natural Language*

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).

- Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.

- More “natural” treatments of such phenomena are possible if we allow a flexible system of types (such as Curry-style typing), where
  1. expressions can belong to more than one type;
  2. functional types can apply to arguments of several types.

- “Is fun” can have the type $\Pi X. X \Rightarrow Prop$. 


Polymorphism in NL

Polymorphism in Natural Language

- NL expressions act as if they belong to more than one semantic type: *playing tennis is fun, to play tennis is fun, tennis is fun* (Chierchia 1982; Turner 1997).
- Some NL expressions (such as conjunctions) can combine expressions of different types. There are constraints on the types of the combined expressions and the resultant expression.
- More “natural” treatments of such phenomena are possible if we allow a flexible system of types (such as Curry-style typing), where
  1. expressions can belong to more than one type;
  2. functional types can apply to arguments of several types.
- “Is fun” can have the type $\Pi X. X \Rightarrow Prop$.
- “and” can have the type $\Pi X. X \Rightarrow X \Rightarrow X$. 


Model Theory
Model Theory for PTCT

Sketch of a model

- A model of the untyped $\lambda$-calculus (e.g. General Functional Models), $\mathcal{D} = \langle D, [D \to D], \Phi, \Psi \rangle$ where $D$ is isomorphic to $[D \to D]$

1. $D$ is a non-empty set,
2. $[D \to D]$ is some class of functions from $D$ to $D$,
3. $\Phi : D \to [D \to D]$,
4. $\Psi : [D \to D] \to D$,
5. $\Psi(\Phi(d)) = d$ for all $d \in D$

(Meyer 1982).

- Interpret the types as terms in $D$ that correspond to subsets of $D$. 

A Model for PTCT

A model of PTCT is $\mathcal{M} = \langle \mathcal{D}, \mathcal{T}, \mathcal{P}, \mathcal{B}, \mathcal{B}, \mathcal{T}, \mathcal{K} \rangle$, where

1. $\mathcal{D}$ is a model of the $\lambda$-calculus.
2. $\mathcal{T} : \mathcal{D} \to \{0, 1\}$ models the truth predicate $\text{true}$.
3. $\mathcal{P} \subseteq \mathcal{D}$ models the class of propositions.
4. $\mathcal{B} \subseteq \mathcal{D}$ models the class of basic individuals.
5. $\mathcal{B}(\mathcal{B})$ is a set of sets whose elements partition $\mathcal{B}$ into equivalence classes of individuals.
6. $\mathcal{T} \subseteq \mathcal{K}$ models the term representation of types.
7. $\mathcal{K} \subseteq \mathcal{D}$ models the kinds.

with sufficient structural constraints on $\mathcal{T}$, $\mathcal{P}$ and $\mathcal{T}$ to validate the rules of PTCT.
Representing Cardinality and Generalized Quantifiers
Intensional Number Theory

We can add an intensional number theory to PTCT

**Terms** 0 | succ | pred | add | mult | most | \cdot | _B

**Types** Num

**Wffs** zero(t) | t \cong_{\text{Num}} t' | t <_{\text{Num}} t' | most(p)(q)

**Axioms for Num** The usual Peano axioms, adapted to PTCT

**Axioms for <_{\text{Num}}**

\[
\begin{align*}
y \in \text{Num} & \rightarrow 0 <_{\text{Num}} \text{succ}(y) \\
x \in \text{Num} & \rightarrow x \not<_{\text{Num}} 0 \\
x \in \text{Num} \land y \in \text{Num} & \rightarrow (\text{succ}(x) <_{\text{Num}} \text{succ}(y) \leftrightarrow x <_{\text{Num}} y)
\end{align*}
\]
Proportional Quantifiers

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

**Cardinality of properties** $|p|_B$

1. $p \in (B \Rightarrow \text{Prop}) \land \sim \exists x(x \in B \land \text{true } px) \rightarrow$
   
   $|p|_B \cong_{\text{Num}} 0$

2. $p \in (B \Rightarrow \text{Prop}) \land b \in B \land \text{true } pb \rightarrow$
   
   $|p|_B \cong_{\text{Num}} \text{add}(|\lambda x(px \land \sim x \approx_b b)|_B)(\text{succ}(0))$

The cardinality of types can be defined in a similar way.
Proportional Quantifiers

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

**Cardinality of properties** \(|p|_B\)

1. \(p \in (B \rightarrow \text{Prop}) \land \exists x (x \in B \land \text{true}\ p x) \rightarrow \)
   \(|p|_B \equiv_{\text{Num}} 0\)

2. \(p \in (B \rightarrow \text{Prop}) \land b \in B \land \text{true}\ pb \rightarrow \)
   \(|p|_B \equiv_{\text{Num}} \text{add}(\langle \lambda x (px \sim x \hat{=} B b) \rangle_B)(\text{succ}(0))\)

The cardinality of types can be defined in a similar way.

**Analysis of most(p)(q)**

\(p \in (B \rightarrow \text{Prop}) \land q \in (B \rightarrow \text{Prop}) \rightarrow \)

\(\text{most}(p)(q) \leftrightarrow\)

\(|\{x \in B. \text{true} px \sim \text{true} qx\}|_B \leq_{\text{Num}} |\{x \in B. \text{true} px \land \text{true} qx\}|_B\)
Anaphora
Anaphora and Type Theory

A Type-Theoretical Approach to Anaphora

- Ranta (1994) uses Martin-Löf Type Theory (MLTT) to represent cases of pronominal anaphora which motivated DRT (Kamp, 1981; Kamp and Reyle, 1993) and dynamic logic (Groenendik and Stokhof, 1990 and 1991).
- As Ranta acknowledges, by adopting the DRT treatment of donkey anaphora as (intuitionistic) universal quantification over pairs his analysis inherits the proportion problem (Heim, 1990; Kadmon, 1990).
- It provides the wrong results for cases like “Most men who own a donkey, beat it.”
- It also does not capture the existential reading of sentences like “Every person who had a quarter in his/her pocket put it in the parking meter.” (Pelletier and Schubert, 1989).
- It is not obvious how this approach can be extended to plural anaphora as in “Every man arrived. They sang.”
An Alternative Account

An Alternative Type-Theoretic Account of Pronominal Anaphora

- Represent the truth conditions of quantified NPs as cardinality relations (cf. treatment of “most” in PTCT).
- “Every student sang.”
- $\{x \in B. \text{true student'}(x) \wedge \text{true sang'}(x)\}_B$
  $\equiv_{\text{Num}} \{x \in B. \text{true student'}(x)\}_B$
- Pronouns are represented as appropriately typed free-variables.
Bound Readings of Pronouns

An Alternative Type-Theoretic Account of Pronominal Anaphora

- If the free pronoun 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 set operator, then the variable can be interpreted as bound by the operator through substitution under α identity.

- This interpretation yields the bound reading of the pronoun.

"Every man loves his mother."

\[ \{x \in B.\text{true} \text{man}'(x) \land \text{true} \text{love}'(x, \text{mother-of}'(y))\}_{B} \]

\[ \approx_{\text{Num}} \{x \in B.\text{true} \text{man}'(x)\}_{B} \]

- Representations of this kind are generated by compositional semantic operations as described in (for example) Lappin (1989), and Lappin and Francez (1994).
Bound Readings of Pronouns

An Alternative Type-Theoretic Account of Pronominal Anaphora

- If the free pronoun 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 set operator, then the variable can be interpreted as bound by the operator through substitution under $\alpha$ identity.

- This interpretation yields the bound reading of the pronoun.

"Every man loves his mother."

$$\models_{\text{Num}} \{ x \in B. \text{true man'}(x) \wedge \text{true love'}(x, \text{mother-of'}(y)) \} |_B$$

- Representations of this kind are generated by compositional semantic operations as described in (for example) Lappin (1989), and Lappin and Francez (1994).
Bound Readings of Pronouns

An Alternative Type-Theoretic Account of Pronominal Anaphora

- If the free pronoun 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 set operator, then the variable can be interpreted as bound by the operator through substitution under α identity.

- This interpretation yields the bound reading of the pronoun.

  “Every man loves his mother.”

  \[
  \{x \in B. \text{true} \text{man}'(x) \land \text{true} \text{love}'(x, \text{mother-of}'(x))\}|_B \\
  \cong_{\text{Num}} \{x \in B. \text{true} \text{man}'(x)\}|_B
  \]

- Representations of this kind are generated by compositional semantic operations as described in (for example) Lappin (1989), and Lappin and Francez (1994).
Quantified NP Antecedents

Unbound Pronouns with Quantified NP Antecedents

- When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.
- By the default case, the pronoun variable is bound by a universal quantifier.
- “Every student arrived.”
  \[
  \{ x \in B. \text{true student'}(x) \land \text{true arrived'}(x) \} \mid_B \\
  \equiv_{\text{Num}} \{ x \in B. \text{true student'}(x) \} \mid_B
  \]
- “They sang.”
  \[
  \forall y \in A. (\text{true sang'}(y))
  \]
When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.

By the default case, the pronoun variable is bound by a universal quantifier.

"Every student arrived."
\[
\{x \in B. \text{true} \text{ student'}(x) \land \text{true} \text{ arrived'}(x)\} \mid_B \\
\cong \text{Num} \mid \{x \in B. \text{true} \text{ student'}(x)\} \mid_B
\]

"They sang."
\[
\forall y \in A. (\text{true} \text{ sang'}(y))
\]
Quantified NP Antecedents

Unbound Pronouns with Quantified NP Antecedents

- When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.
- By the default case, the pronoun variable is bound by a universal quantifier.
- "Every student arrived."
  \[ \{ x \in B.\text{true student'}(x) \land \text{true arrived'}(x) \} \models_B \]
  \[ \cong_{\text{Num}} \{ x \in B.\text{true student'}(x) \} \models_B \]
- "They sang."
  \[ \forall y \in A.\text{true sang'}(y) \]
Quantified NP Antecedents

Unbound Pronouns with Quantified NP Antecedents

- When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.
- By the default case, the pronoun variable is bound by a universal quantifier.
- “Every student arrived.”
  \[ \{ x \in B. \text{true student'}(x) \land \text{true arrived'}(x) \} \]
  \[ \cong_{\text{Num}} \{ x \in B. \text{true student'}(x) \} \]
- “They sang.”
  \[ \forall y \in A. (\text{true sang'}(y)) \]
Quantified NP Antecedents

Unbound Pronouns with Quantified NP Antecedents

- When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.
- By the default case, the pronoun variable is bound by a universal quantifier.
- “Every student arrived.”
  \[ \{ x \in B. \text{true student'}(x) \land \text{true arrived'}(x) \} \mid_B \]
  \[ \equiv_{\text{Num}} \{ x \in B. \text{true student'}(x) \} \mid_B \]
- “They sang.”
  \[ \forall y \in \{ x \in B. \text{true student'}(x) \land \text{true arrived'}(x) \}.(\text{true sang'}(y)) \]
Quantified NP Antecedents

Unbound Pronouns with Quantified NP Antecedents

- When the pronoun is interpreted as dependent upon an NP which does not bind it, we represent the pronoun variable as constrained by a type taken from the predicative part of the antecedent clause representation.
- By the default case, the pronoun variable is bound by a universal quantifier.
- “Every student arrived.”
  \[ \{ x \in B. \text{true student}'(x) \land \text{true arrived}'(x) \} |_B \]
  \[ \equiv_{\text{Num}} \{| x \in B. \text{true student}'(x)\} |_B \]
- “They sang.”
  \[ \forall y \in \{ x \in B. \text{true student}'(x) \land \text{true arrived}'(x) \}. (\text{true sang}'(y)) \]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- "John arrived."
  \[\text{true} \quad \text{arrived}(\text{john})\]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”

\[ \text{true} \ \text{arrived}'( \text{john} ) \]

- “He sang.”

\[ \forall y \in A. (\text{true} \ \text{sang}'(y)) \]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[\text{true} \text{ arrived}'(\text{john})\]
- “He sang.”
  \[\forall y \in A. (\text{true} \text{ sang}'(y))\]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[ \text{true} \text{arrived}'( \text{john} ) \]
- “He sang.”
  \[ \forall y \in A. (\text{true} \text{sang}'(y)) \]
- \[ A = \{ x \in B. x =_B \text{john} \} \]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[ \text{true} \text{arrived'}( \text{john} ) \]
- “He sang.”
  \[ \forall y \in A. (\text{true} \text{sang'}(y)) \]
- \( A = \{x \in B. x =_B \text{john}\} \)
- “Some man arrived.”
  \[ | \{x \in B. \text{true man'}(x) \land \text{true} \text{arrived'}(x) \land \text{true} \phi(x)\} |_B >_{\text{Num}} 0 \]
- \( \phi \) is a predicate that is specified in context and uniquely identifies a man who arrived in that context.
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[
  \text{true } \text{arrived}'(\text{john})
  \]

- “He sang.”
  \[
  \forall y \in A. (\text{true } \text{sang}'(y))
  \]

- \( A = \{x \in B. x =_B \text{john}\} \)

- “Some man arrived.”
  \[
  | \{x \in B. \text{true } \text{man}'(x) \land \text{true } \text{arrived}'(x) \land \text{true } \phi(x)\} |_B >_{\text{Num}} 0
  \]

- \( \phi \) is a predicate that is specified in context and uniquely identifies a man who arrived in that context.

- “He sang.”
  \[
  \forall y \in A. (\text{true } \text{sang}'(y))
  \]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[ \text{true} \text{arrived}'( \text{john} ) \]
- “He sang.”
  \[ \forall y \in A. (\text{true} \text{sang}'(y)) \]
- \[ A = \{ x \in B. x =_B \text{john} \} \]
- “Some man arrived.”
  \[ \{ x \in B. \text{true} \text{man}'(x) \land \text{true} \text{arrived}'(x) \land \text{true} \phi(x) \} \]  \[ B > _\text{Num} 0 \]
- \( \phi \) is a predicate that is specified in context and uniquely identifies a man who arrived in that context.
- “He sang.”
  \[ \forall y \in A. (\text{true} \text{sang}'(y)) \]
Names and Existential NPs

Proper Name and Existentially Quantified NP Antecedents

- “John arrived.”
  \[\text{true} \, \text{arrived}'(\text{john})\]

- “He sang.”
  \[\forall y \in A. (\text{true} \, \text{sang}'(y))\]
  \[A = \{x \in B. x =_B \text{john}\}\]

- “Some man arrived.”
  \[|\{x \in B. (\text{true} \, \text{man}'(x) \land \text{true} \, \text{arrived}'(x) \land \text{true} \, \phi(x))\}|_B > \text{Num} 0\]
  \[\phi\] is a predicate that is specified in context and uniquely identifies a man who arrived in that context.

- “He sang.”
  \[\forall y \in A. (\text{true} \, \text{sang}'(y))\]
  \[A = \{x \in B. (\text{true} \, \text{man}'(x) \land \text{true} \, \text{arrived}'(x) \land \text{true} \, \phi(x))\}\]
Donkey Anaphora in PTCT

- “Every man who owns a donkey beats it.”
Donkey Anaphora in PTCT

- "Every man who owns a donkey beats it."

\[ \{ x \in B. \text{true \textit{man'}}(x) \]
\[ \land (\{ y \in B. \text{true \textit{own'}}(x, y) \land \text{true \textit{donkey'}}(y) \} \mid_B >_{\text{Num}} 0) \]
\[ \land \forall z \in A(\text{true \textit{beat'}}(x, z)) \} \mid_B \]
\[ \equiv_{\text{Num}} \]
\[ \{ x \in B. \text{true \textit{man'}}(x) \]
\[ \land (\{ y \in B. \text{true \textit{own'}}(x, y) \land \text{true \textit{donkey'}}(y) \} \mid_B >_{\text{Num}} 0) \} \mid_B \]
Donkey Anaphora in PTCT

- “Every man who owns a donkey beats it.”

\[
\{x \in B. \text{true} \text{man'}(x) \wedge (\{y \in B. \text{true} \text{own'}(x, y) \wedge \text{true} \text{donkey'}(y)\}_B > \text{Num} 0) \wedge \forall z \in A(\text{true} \text{beat'}(x, z))\}_B
\]

\[\equiv_{\text{Num}}\]

\[
\{x \in B. \text{true} \text{man'}(x) \wedge (\{y \in B. \text{true} \text{own'}(x, y) \wedge \text{true} \text{donkey'}(y)\}_B > \text{Num} 0)\}_B
\]
Donkey Anaphora in PTCT

- “Every man who owns a donkey beats it.”

$\{x \in B. \text{true man}'(x) \land (\{y \in B. \text{true own}'(x, y) \land \text{true donkey}'(y)\} |_{B > \text{Num } 0}) \land \forall z \in A(\text{true beat}'(x, z))|_B \approx \text{Num}$

$\{x \in B. \text{true man}'(x) \land (\{y \in B. \text{true own}'(x, y) \land \text{true donkey}'(y)\}|_{B > \text{Num } 0})\}|_B$

- $A = \{y \in B. \text{true own}'(x, y) \land \text{true donkey}'(y)\}$
Donkey Anaphora in PTCT

- “Every man who owns a donkey beats it.”

\[
\{ x \in B. \text{true}\text{man}'(x) \\
\quad \land (\{ y \in B. \text{true}\text{own}'(x, y) \land \text{true}\text{donkey}'(y) \} \mid_B >_{\text{Num}} 0) \\
\quad \land \forall z \in A(\text{true}\text{beat}'(x, z))\} \mid_B \\
\equiv_{\text{Num}} \\
\{ x \in B. \text{true}\text{man}'(x) \\
\quad \land (\{ y \in B. \text{true}\text{own}'(x, y) \land \text{true}\text{donkey}'(y) \} \mid_B >_{\text{Num}} 0)\} \mid_B
\]

- \( A = \{ y \in B. \text{true}\text{own}'(x, y) \land \text{true}\text{donkey}'(y) \} \)

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

Existential Readings of Donkey Sentences

- The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

"Every person who had a quarter put it in a parking meter."
Existential Readings

Existential Readings of Donkey Sentences

- The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

- “Every person who had a quarter put it in a parking meter.”
Existential Readings

Existential Readings of Donkey Sentences

- The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

- “Every person who had a quarter put it in a parking meter.”

$\exists x \in B.\text{true} \text{man}'(x) \\
\land (\exists y \in B.\text{true} \text{had}'(x, y) \land \text{true} \text{quarter}'(y)) |_B > \text{Num} \ 0) \\
\land \exists z \in A(\text{true} \text{put-in-a-parking-meter}'(x, z))|_B \\
\approx \text{Num}$

$\exists x \in B.\text{true} \text{person}'(x) \\
\land (\exists y \in B.\text{true} \text{had}'(x, y) \land \text{true} \text{quarter}'(y))|_B > \text{Num} \ 0))|_B$
Existential Readings

Existential Readings of Donkey Sentences

• The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

• “Every person who had a quarter put it in a parking meter.”

\[
\begin{align*}
\{x & \in B.\text{true } \text{man}'(x) \\
\land & (\{y \in B.\text{true } \text{had}'(x, y) \land \text{true } \text{quarter}'(y)\} _B > \text{Num } 0) \\
\land & \exists z \in A(\text{true } \text{put-in-a-parking-meter}'(x, z)))\}_B \\
\end{align*}
\]

\[
\begin{align*}
\{x & \in B.\text{true } \text{person}'(x) \\
\land & (\{y \in B.\text{true } \text{had}'(x, y) \land \text{true } \text{quarter}'(y)\} _B > \text{Num } 0)\}_B \\
\end{align*}
\]
Existential Readings

Existential Readings of Donkey Sentences

- The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

- “Every person who had a quarter put it in a parking meter.”

$$\{x \in B. \text{true } man'(x)$$

$$\land ( \{y \in B. \text{true } had'(x, y) \land \text{true } quarter'(y)\} |_B > \text{Num } 0)$$

$$\land \exists z \in A(\text{true } put-in-a-parking-meter'(x, z)) |_B \equiv \text{Num} \equiv \text{Num}$$

$$\{x \in B. \text{true } person'(x)$$

$$\land (\{y \in B. \text{true } had'(x, y) \land \text{true } quarter'(y)\} |_B > \text{Num } 0)\} |_B$$

- $$A = \{y \in B. \text{true } had'(x, y) \land \text{true } quarter'(y)\}$$
Existential Readings

Existential Readings of Donkey Sentences

• The existential reading of a donkey sentence can be obtained by binding the variable representing the pronoun by an existential quantifier.

• “Every person who had a quarter put it in a parking meter.”

\[
\{\{x \in B.\text{true man'}(x) \\
\land (\{y \in B.\text{true had'}(x, y) \land \text{true quarter'}(y)\} \uparrow_B >_{\text{Num}} 0) \\
\land \exists z \in A(\text{true put-in-a-parking-meter'}(x, z))\} \uparrow_B =_{\text{Num}}
\{\{x \in B.\text{true person'}(x) \\
\land (\{y \in B.\text{true had'}(x, y) \land \text{true quarter'}(y)\} \uparrow_B >_{\text{Num}} 0)\} \uparrow_B
\]

• \( A = \{y \in B.\text{true had'}(x, y) \land \text{true quarter'}(y)\} \)

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

\( \triangle \)
Proportional Donkey Sentences

- The proportionality problem does not arise on our account.
Proportional Donkey Sentences

- 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.
Proportional Donkey Sentences

- 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.
- “Most men who own a donkey beat it.”
Proportional Donkey Sentences

- 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.
- “Most men who own a donkey beat it.”

\[
\{(x \in B. \text{true man'}(x))
\land (\{y \in B. \text{true own'}(x, y) \land \text{true donkey'}(y)\} |_{B > \text{Num}} 0)
\land \forall z \in A \sim (\text{true beat'}(x, z))\}|_B > \text{Num}
\]

\[
\{(x \in B. \text{true man'}(x))
\land (\{y \in B. \text{true own'}(x, y) \land \text{true donkey'}(y)\} |_{B > \text{Num}} 0)\}|_B
\]
Proportional Donkey Sentences

- 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.
- "Most men who own a donkey beat it."

\[
\begin{align*}
\{x \in B. \text{true} & \text{man'}(x) \\
\wedge (\{y \in B. \text{true} & \text{own'}(x, y) \wedge \text{true donkey'}(y)\}_{B > \text{Num} 0}) \\
\wedge \forall z \in A & \sim (\text{true beat'}(x, z))\}_{B > \text{Num} 0}
\end{align*}
\]

This representation states that most men who own a donkey beat all of the donkeys they own, and so it is false in the model which causes problems for the universal quantification over pairs analysis.
Proportional Donkey Sentences

- 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.
- “Most men who own a donkey beat it.”
  \[\|\{x \in B. \text{true man}'(x)\} \land (\{y \in B. \text{true own}'(x, y) \land \text{true donkey}'(y)\} |_B >_{\text{Num}} 0) \land \forall z \in A \sim (\text{true beat}'(x, z))|_B >_{\text{Num}} 0)\]  
  \[A = \{y \in B. \text{true own}'(x, y) \land \text{true donkey}'(y)\}\]

This representation states that most men who own a donkey beat all of the donkeys they own, and so it is false in the model which causes problems for the universal quantification over pairs analysis.
Proportional Donkey Sentences

- 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.
- "Most men who own a donkey beat it."

\[
\begin{align*}
|x \in B.\text{true} man'(x) \\
&\land (\{y \in B.\text{true} own'(x, y) \land \text{true donkey}'(y)\} \mid_B >_{\text{Num}} 0) \\
&\land \forall z \in A \sim(\text{true beat}'(x, z)) \mid_B >_{\text{Num}} \\
|\{x \in B.\text{true} man'(x) \\
&\land (\{y \in B.\text{true} own'(x, y) \land \text{true donkey}'(y)) \mid_B >_{\text{Num}} 0)\} |_B
\end{align*}
\]

- \( A = \{y \in B.\text{true} own'(x, y) \land \text{true donkey}'(y)\} \)
- This representation states that most men who own a donkey beat all of the donkeys they own, and so it is false in the model which causes problems for the universal quantification over pairs analysis.
Free-floating Type Judgements

- In order to compositionally compute the given representations, it may be necessary to allow for intermediate term representations containing free-floating type judgements for subtypes.
Free-floating Type Judgements

- In order to compositionally compute the given representations, it may be necessary to allow for intermediate term representations containing free-floating type judgements for subtypes.
- Such an extension of PTCT should not be problematic if, for example, we limit such assertions of type membership to terms that have a normal form.
Free-floating Type Judgements

- In order to compositionally compute the given representations, it may be necessary to allow for intermediate term representations containing free-floating type judgements for subtypes.
- Such an extension of PTCT should not be problematic if, for example, we limit such assertions of type membership to terms that have a normal form.
- The free floating type judgements can also be given an alternative property-theoretic presentation, where membership is represented by functional application.
Ellipsis
Let $S$ be a parameter that is instantiated by separation types.
Ellipsis and Separation Types

- Let $S$ be a parameter that is instantiated by separation types.
- We can represent a clause containing an elided VP like

as
Ellipsis and Separation Types

- Let $S$ be a parameter that is instantiated by separation types.
- We can represent a clause containing an elided VP like
  
  *John sings, and Mary does too.*

  as
Ellipsis and Separation Types

- Let $S$ be a parameter that is instantiated by separation types.
- We can represent a clause containing an elided VP like
  
  *John sings, and Mary does too.*

  as

  $$\text{true} \ sings'(john) \wedge \text{mary} \in S$$
Ellipsis and Separation Types

- Let $S$ be a parameter that is instantiated by separation types.
- We can represent a clause containing an elided VP like
  
  *John sings, and Mary does too.*

  as

- $\text{true} \; \text{sings'}(john) \land \text{mary} \in S$
- We can abstract on $john$ to obtain the separation type $\{x \in B. \text{true} \; \text{sings'}(x)\}$ from the antecedent in order to resolve $S$. 
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
- Product types can be added to PTCT, or the examples can be represented using an equivalent curried function form.
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
- Product types can be added to PTCT, or the examples can be represented using an equivalent curried function form.
- *Mary reviewed Principia and Max Ulysses.*
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
- Product types can be added to PTCT, or the examples can be represented using an equivalent curried function form.
- *Mary reviewed Principia and Max Ulysses.*
- $\text{true } \text{reviewed'}(\text{john, principia}) \land (\text{max, ulysses}) \in S$
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
- Product types can be added to PTCT, or the examples can be represented using an equivalent curried function form.
- *Mary reviewed Principia and Max Ulysses.*
- \( \text{true} \text{ reviewed'}(\text{john, principia}) \land (\text{max, ulysses}) \in S \)
- \( S = \{(x, y) \in B \otimes B. \text{true} \text{ reviewed'}(x, y)\} \)
Gapping

- Our treatment of VP ellipsis extends directly to gapping.
- Here we use product types.
- Product types can be added to PTCT, or the examples can be represented using an equivalent curried function form.
- *Mary reviewed Principia and Max Ulysses.*
- \[ \text{true} \text{ reviewed'}(john, principia) \land (max, ulysses) \in S \]
- \[ S = \{(x, y) \in B \otimes B. \text{true} \text{ reviewed'}(x, y)\} \]
- \[ \text{true} \text{ reviewed'}(john, principia) \land \text{true} \text{ reviewed'}(max, ulysses) \]
Pseudogapping

- Our account also applies to pseudogapping.
Our account also applies to pseudogapping.

*Max introduced Rosa to Sam before Bill did Mary to John.*
Pseudogapping

- Our account also applies to pseudogapping.

  \[ \text{Max introduced Rosa to Sam before Bill did Mary to John.} \]

- true \text{introduced}'(max, rosa, sam) before (bill', mary, john) \in S \]
Pseudogapping

- Our account also applies to pseudogapping.

Max introduced Rosa to Sam before Bill did Mary to John.

- \true\ introduced'(max, rosa, sam) before (bill', mary, john) \in S

- \( S = \{ (x, y, z) \in B \otimes B \otimes B. \true\ introduced'(x, y, z) \} \)
Strict and Sloppy

Strict and Sloppy Pronominal Anaphora

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.
Strict and Sloppy

**Strict and Sloppy Pronominal Anaphora**

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.
- *John loves his mother, and Bill does too.*
Strict and Sloppy

Strict and Sloppy Pronominal Anaphora

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.
- \( \text{John loves his mother, and Bill does too.} \)
- \( \forall x \in A(\text{true loves'}(\text{john, mother-of'}(x)) \land \text{bill} \in S) \)
Strict and Sloppy

Strict and Sloppy Pronominal Anaphora

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.
- *John loves his mother, and Bill does too.*
- $\forall x \in A(\text{true\_loves'}(\text{john, mother-of'}(x)) \land \text{bill} \in S$
- $S$ is defined by $S = \{y \in B.\forall x \in A(\text{true\_loves'}(y, \text{mother-of'}(x))\}$
Strict and Sloppy Pronominal Anaphora

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.
- **John loves his mother, and Bill does too.**
- $$\forall x \in A(\text{true} \, \text{loves'}(\text{john, mother-of'}(x)) \land \text{bill} \in S)$$
- **S** is defined by $$S = \{y \in B. \forall x \in A(\text{true} \, \text{loves'}(y, \text{mother-of'}(x)))\}$$
- When the type parameter $$A$$ on the variable $$x$$ is specified as
  $$\{w \in B. \text{true} \, w \overset{=}B \text{john}\}$$
  in the antecedent clause prior to the resolution of $$S$$, then a strict reading of the pronoun results.
Strict and Sloppy

Strict and Sloppy Pronominal Anaphora

- If we combine our treatment of ellipsis with our account of pronominal anaphora, the representation of the distinction between strict and sloppy readings of pronouns under ellipsis is straightforward.

- *John loves his mother, and Bill does too.*

- $\forall x \in A^{\text{true}\text{loves'}(\text{john}, \text{mother-of'}(x))} \land \text{bill } \in S$

- $S$ is defined by $S = \{ y \in B. \forall x \in A^{\text{true}\text{loves'}(y, \text{mother-of'}(x))}\}$

- When the type parameter $A$ on the variable $x$ is specified as

  $$\{ w \in B. \text{true} w \models_B \text{john}\}$$

  in the antecedent clause prior to the resolution of $S$, then a strict reading of the pronoun results.

- If $A$ is determined after the value of $S$ is identified, then it can be taken as

  $$\{ w \in B. \text{true} w \models_B \text{bill}\}$$

which provides the sloppy reading.
Antecedent Contained Ellipsis

*Antecedent Contained Ellipsis (ACE)*

- *Mary read every book that John did.*
Antecedent Contained Ellipsis

Antecedent Contained Ellipsis (ACE)

- *Mary read every book that John did.*
- We impose the condition that the conjunct corresponding to a restrictive relative clause in the propositional part of the sub-type expression of a GQ contain an occurrence of the variable bound by the set operator of the sub-type.
Antecedent Contained Ellipsis

Antecedent Contained Ellipsis (ACE)

- *Mary read every book that John did.*
- We impose the condition that the conjunct corresponding to a restrictive relative clause in the propositional part of the sub-type expression of a GQ contain an occurrence of the variable bound by the set operator of the sub-type.
- This is, in effect, a non-vacuousness constraint on relative clause modification.
Antecedent Contained Ellipsis

Antecedent Contained Ellipsis (ACE)

- *Mary read every book that John did.*
- We impose the condition that the conjunct corresponding to a restrictive relative clause in the propositional part of the sub-type expression of a GQ contain an occurrence of the variable bound by the set operator of the sub-type.
- This is, in effect, a non-vacuousness constraint on relative clause modification.
- It requires that a relative clause be interpreted as a modifier that contributes a restriction to the head noun.
ACE

- Given this constraint, the representation of
ACE

• Given this constraint, the representation of

Mary read every book that John did.

is
Given this constraint, the representation of

Mary read every book that John did.

is

\[ \{\{ x \in B. \text{truebook}'(x) \land (\text{john}, x) \in S \} \land \text{trueread}'(\text{mary}, x)\} \}_{B} \]

\[ \cong_{\text{Num}} \{\{ x \in B. \text{truebook}'(x) \land (\text{john}, x) \in S \} \}_{B} \]
Given this constraint, the representation of

Mary read every book that John did.

is

\[ |\{ x \in B. \text{true book}'(x) \land (john, x) \in S \} \land \text{true read}'(mary, x) | \]_B

\[ \equiv_{\text{Num}} |\{ x \in B. \text{true book}'(x) \land (john, x) \in S \} | \]_B

Taking the conjunct that corresponds to the matrix clause as the antecedent and abstracting over both its arguments we obtain

\[ S = \{ (y, w). \text{true read}'(y, w) \} \]
Given this constraint, the representation of

\[ \textit{Mary read every book that John did.} \]

is

\[ \{(x \in B.\text{true book'} (x) \land (john, x) \in S) \land ^\text{true read'} (mary, x)\}_B \]

\[ \equiv_{\text{Num}} \{(x \in B.\text{true book'} (x) \land (john, x) \in S)\}_B \]

Taking the conjunct that corresponds to the matrix clause as the antecedent and abstracting over both its arguments we obtain

\[ S = \{(y, w).\text{true read'} (y, w)\} \]

The interpretation that we derive for the ACE structure is

\[ \{(x \in B.\text{true book'} (x) \land ^\text{true read'} (john, x) \land ^\text{true read'} (mary, x))\}_B \]

\[ \equiv_{\text{Num}} \{(x \in B.\text{true book'} (x) \land ^\text{true read'} (john, x))\}_B \]
Given this constraint, the representation of

\[\text{Mary read every book that John did.}\]

is

\[\{(x \in B. \text{true book'}(x) \land (john, x) \in S) \land \text{true read'}(mary, x)\}\]_B

\[\equiv_{\text{Num}} \{(x \in B. \text{true book'}(x) \land (john, x) \in S)\}_B\]

Taking the conjunct that corresponds to the matrix clause as the antecedent and abstracting over both its arguments we obtain

\[S = \{(y, w). \text{true read'}(y, w)\}\]

The interpretation that we derive for the ACE structure is

\[\{(x \in B. \text{true book'}(x) \land \text{true read'}(john, x) \land \text{true read'}(mary, x))\}_B\]

\[\equiv_{\text{Num}} \{(x \in B. \text{true book'}(x) \land \text{true read'}(john, x))\}_B\]

This asserts that every book that John read Mary read, which is the intended reading.
ACE

- We generate interpretations of ACE structures without using a syntactic operation of quantifier raising (Fiengo and May, 1994) or a semantic procedure of storage (Dalrymple et al, 1991; Shieber et al, 1996).
ACE

- We generate interpretations of ACE structures without using a syntactic operation of quantifier raising (Fiengo and May, 1994) or a semantic procedure of storage (Dalrymple et al, 1991; Shieber et al, 1996).
- We also do not require a syntactic trace (Lappin, 1996) or a SLASH feature in the ellipsis site (Lappin, 1999).
ACE

- We generate interpretations of ACE structures without using a syntactic operation of quantifier raising (Fiengo and May, 1994) or a semantic procedure of storage (Dalrymple et al, 1991; Shieber et al, 1996).
- We also do not require a syntactic trace (Lappin, 1996) or a SLASH feature in the ellipsis site (Lappin, 1999).
- The presence of the variable bound by the set operator of the sub-type as the second argument of the function which assigns a value to the elided PTCT expression is motivated by a general condition on the representation of restrictive relative clauses as non-vacuous conjuncts in a GQ.
Comparison with Higher Order Unification
Similarities

Comparison with Higher-Order Unification (HOU)

- Our PTCT-based analysis of ellipsis is similar in approach to the Higher-Order Unification analysis presented in Dalrymple et al. (1991) and Shieber et al. (1996).
Similarities

Comparison with Higher-Order Unification (HOU)

- Our PTCT-based analysis of ellipsis is similar in approach to the Higher-Order Unification analysis presented in Dalrymple et al. (1991) and Shieber et al. (1996).
- In both cases correspondences are set up between a sequence of phrases in an ellipsis site and an antecedent clause, and a predicate term is abstracted from the antecedent for application to elements in the elided clause.
Differences

How our approach differs from Higher-Order Unification (HOU)

- While HOU solves an equation with a higher-order variable to obtain a lambda expression, our PTCT-based account uses a parameter that is resolved to a separation type expression.
Differences

How our approach differs from Higher-Order Unification (HOU)

- While HOU solves an equation with a higher-order variable to obtain a lambda expression, our PTCT-based account uses a parameter that is resolved to a separation type expression.
- In our model theory type variables take terms as values.
Differences

How our approach differs from Higher-Order Unification (HOU)

- While HOU solves an equation with a higher-order variable to obtain a lambda expression, our PTCT-based account uses a parameter that is resolved to a separation type expression.
- In our model theory type variables take terms as values.
- Even if a separation type parameter is construed as a variable of PTCT, we remain within the first-order resources of PTCT.
Differences

*How our approach differs from Higher-Order Unification (HOU)*

- While HOU solves an equation with a higher-order variable to obtain a lambda expression, our PTCT-based account uses a parameter that is resolved to a separation type expression.
- In our model theory type variables take terms as values.
- Even if a separation type parameter is construed as a variable of PTCT, we remain within the first-order resources of PTCT.
- HOU requires storage to extract a quantified NP from its antecedent-contained position in the semantic representation of an ACE structure.
Differences

*How our approach differs from Higher-Order Unification (HOU)*

- While HOU solves an equation with a higher-order variable to obtain a lambda expression, our PTCT-based account uses a parameter that is resolved to a separation type expression.
- In our model theory type variables take terms as values.
- Even if a separation type parameter is construed as a variable of PTCT, we remain within the first-order resources of PTCT.
- HOU requires storage to extract a quantified NP from its antecedent-contained position in the semantic representation of an ACE structure.
- We are able to interpret these NPs *in situ* by virtue of the presence of a bound variable in the part of a sub-type in a GQ representation that corresponds to the relative clause of the ACE.
Summary
Conclusions

- We have developed type-theoretic treatments of pronominal anaphora and ellipsis within the framework of PTCT, a first-order fine-grained intensional logic with flexible Curry typing.
Conclusions

- We have developed type-theoretic treatments of pronominal anaphora and ellipsis within the framework of PTCT, a first-order fine-grained intensional logic with flexible Curry typing.
- Our account of anaphora has wider empirical coverage than Ranta’s (1994) MLTT analysis.
Conclusions

- We have developed type-theoretic treatments of pronominal anaphora and ellipsis within the framework of PTCT, a first-order fine-grained intensional logic with flexible Curry typing.
- Our account of anaphora has wider empirical coverage than Ranta’s (1994) MLTT analysis.
- Our account of ellipsis avoids the higher-order variables of HOU, and we do not require an operation of storage to handle ACE structures.
Conclusions

- We have developed type-theoretic treatments of pronominal anaphora and ellipsis within the framework of PTCT, a first-order fine-grained intensional logic with flexible Curry typing.
- Our account of anaphora has wider empirical coverage than Ranta’s (1994) MLTT analysis.
- Our account of ellipsis avoids the higher-order variables of HOU, and we do not require an operation of storage to handle ACE structures.
- The primary advantage of PTCT is that it provides the expressiveness of a higher-order system with rich typing while remaining a first-order logic with limited formal power.
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
- then examine extensions that establish correspondences between types and properties — one aim of this work would be to show an equivalence between the type-theoretic and property-theoretic approaches to anaphora and ellipsis;
- explore PTCT as a semantic representation language in implemented systems of natural language interpretation — as part of this research we will be constructing a theorem prover that uses PTCT’s tableau proof theory;
- investigate the implementation of our proposed approaches to anaphora and ellipsis resolution.
Future Work

We will:

• investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
Future Work

*We will:*

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
Future Work

We will:

• investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
• determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
• consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
• then examine extensions that establish correspondences between types and properties.
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
- then examine extensions that establish correspondences between types and properties — one aim of this work would be to show an equivalence between the type-theoretic and property-theoretic approaches to anaphora and ellipsis;
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
- then examine extensions that establish correspondences between types and properties — one aim of this work would be to show an equivalence between the type-theoretic and property-theoretic approaches to anaphora and ellipsis;
- explore PTCT as a semantic representation language in implemented systems of natural language interpretation.
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
- then examine extensions that establish correspondences between types and properties — one aim of this work would be to show an equivalence between the type-theoretic and property-theoretic approaches to anaphora and ellipsis;
- explore PTCT as a semantic representation language in implemented systems of natural language interpretation — as part of this research we will be constructing a theorem prover that uses PTCT’s tableau proof theory;
Future Work

We will:

- investigate the possibility of incorporating product types into PTCT without taking it out of the class of first-order systems;
- determine the most appropriate way of incorporating the representation of free-floating type judgements into the language of terms;
- consider a property-theoretic variant of the treatment of anaphora and ellipsis which exploits properties and application rather than the types and type-membership used in the type-theoretic treatment presented here;
- then examine extensions that establish correspondences between types and properties — one aim of this work would be to show an equivalence between the type-theoretic and property-theoretic approaches to anaphora and ellipsis;
- explore PTCT as a semantic representation language in implemented systems of natural language interpretation — as part of this research we will be constructing a theorem prover that uses PTCT’s tableau proof theory;
- investigate the implementation of our proposed approaches to anaphora and ellipsis resolution.
Papers and Books

Papers on our websites and . . .


Underspecification in PTCT: Underspecified Interpretations in a Curry-Typed Representation Language. Chris Fox and Shalom Lappin. Accepted for publication in the Journal of Logic and Computation, subject to revisions.

Underspecification in PTCT

We also have a treatment of underspecification in PTCT.

- Builds on work by Ed Keenan and Jan van Eijck.
- Incorporates a Cooper-storage like approach within the theory.
- The theory then becomes available to express constraints/filters on acceptable scopings. (We can express constraints that cannot be stated in some other theories.)
- Underspecified representations can be viewed as a function from integers to propositions (\[\text{Num} \rightarrow \text{Prop}\]).
- Exploits the fact that the language of terms can encode computable functions.
- Uses polymorphic lists to allow underspecification with quantifiers of different types (individuals, propositions etc.) “Someone believes everything that Mary believes.”
- Some details of the polymorphism remain to be presented.