Expressive Completeness and Computational Efficiency for Underspecified Scope Representations

Chris Fox\textsuperscript{1} and Shalom Lappin\textsuperscript{2}

\textsuperscript{1}Department of Computer Science
University of Essex

\textsuperscript{2}Department of Philosophy
King’s College London

University of Sheffield, February 9, 2006
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
Outline

1. **Background**
   - Semantic Representation
     - The Standard Approach to Semantics
     - An Alternative Approach

2. **Underspecification in PTCT**
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. **Conclusions**
We want to translate between different modes of expression, e.g.:

- from Natural Language to some more formal, less ambiguous language;
- from Natural Language to a database query language;
- from Natural Language to a language that supports automatic/formal reasoning.

A systematic translation will tell us how to translate every grammatical construction into the target language.
Two Approaches
Ways of “building” representations in the target language

- We need mechanisms that allow us to build up the target representation.
- There are two general approaches.
  1. Unification
     - Widely used in grammar formalisms (e.g. HPSG).
     - A feature of Prolog.
     - Problematic for semantics. Lacks the expressive resources of abstraction and functional application.
  2. λ-calculus
     - A “calculus” for substitutions.
     - Widely used for generating robust semantic representations.
Essentially there are two kinds of expressions:

1. Applications $t(t')$
2. Abstractions $\lambda x (t)$

The latter means “if you give me an argument $t'$, substitute it for $x$ wherever $x$ occurs in the expression $t$ to find my value.”

So $(\lambda x (\ldots x \ldots))(t)$ is equal to $(\ldots t \ldots)$

...these operations provide a combinatorial engine for computing semantic representations in conjunction with syntactic structure.
Essentially there are two kinds of expressions

1. Applications $t(t')$
2. Abstractions $\lambda x(t)$

The latter means “if you give me an argument $t'$, substitute it for $x$ wherever $x$ occurs in the expression $t$ to find my value.

So $(\lambda x(\ldots x \ldots))(t)$ is equal to $(\ldots t \ldots)$

...these operations provide a combinatorial engine for computing semantic representations in conjunction with syntactic structure.
Basics of the $\lambda$-calculus

- Essentially there are two kinds of expressions:
  1. Applications $t(t')$
  2. Abstractions $\lambda x(t)$

- The latter means “if you give me an argument $t'$, substitute it for $x$ wherever $x$ occurs in the expression $t$ to find my value.

- So $(\lambda x(\ldots x \ldots))(t)$ is equal to $(\ldots t \ldots)$

  ... these operations provide a combinatorial engine for computing semantic representations in conjunction with syntactic structure.
A simple calculus of substitutions has some interesting properties. Terms in the calculus can be thought of as programs. Further, all computer programs can be encoded as terms in the calculus. Program execution is equivalent to performing substitutions in the calculus. This calculus is the basis for functional programming languages, like Lisp, Scheme, ML, Haskell, Miranda.
This “simple” calculus of substitutions has some interesting properties.

Terms in the calculus can be thought of as programs.

Further, all computer programs can be encoded as terms in the calculus.

Program execution is equivalent to performing substitutions in the calculus.

This calculus is the basis for functional programming languages, like Lisp, Scheme, ML, Haskell, Miranda.
An Interesting Feature of $\lambda$-calculus

- This “simple” calculus of substitutions has some interesting properties.
- Terms in the calculus can be thought of as programs.
- Further, all computer programs can be encoded as terms in the calculus.
- Program execution is equivalent to performing substitutions in the calculus.
- This calculus is the basis for functional programming languages, like Lisp, Scheme, ML, Haskell, Miranda.
Outline

1. **Background**
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. **Underspecification in PTCT**
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. **Conclusions**
A standard approach in formal semantics is to combine the target language with some version of the \( \lambda \)-calculus. In particular, this approach is used when formal logic is the target language. Unfortunately, combining substitution mechanisms and logic within the same language can produce paradoxes and inconsistency.
The usual way of avoiding inconsistency is to restrict the kinds of expressions that can be combined, and the kinds of substitutions that can be performed.

These restrictions are generally imposed by adding a notion of *type*, which constrains the set of well-formed expressions.

The result is a *Higher Order Logic* (HOL).
Some Problems

- The type system of HOL is overly restrictive for NL semantics: we need semantic types that correspond to the more flexible character of grammatical categories in natural language.

- HOL is not recursively enumerable: it is not possible to prove all true theorems.

- The usual semantic interpretations of HOL are too extensional for natural language. They do not permit fine-grained distinctions of meaning.

- The character of the \( \lambda \)-calculus as a device for generating and evaluating computable functions is lost

  \[ \text{but we want to sustain this feature so that our semantic theory is, effectively, a programming language.} \]
Some Problems

- The type system of HOL is overly restrictive for NL semantics: we need semantic types that correspond to the more flexible character of grammatical categories in natural language.
- HOL is not recursively enumerable: it is not possible to prove all true theorems.
- The usual semantic interpretations of HOL are too extensional for natural language. They do not permit fine-grained distinctions of meaning.
- The character of the λ-calculus as a device for generating and evaluating computable functions is lost . . . but we want to sustain this feature so that our semantic theory is, effectively, a programming language.
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
Distinct Sublanguages for Distinct Aspects of Meaning

- Separate out combinatorics, typing, and reasoning into three distinct components of the representation language.
  1. **λ-calculus of terms** for the combinatorial engine that generates representations of meaning.
  2. **Types** corresponding to natural language categories.
  3. **Well formed formulas** for reasoning.

- This division allows greater flexibility in coordinating the formal power and expressive resources of the theory with the semantic properties of natural language.

Some interesting issues arise in balancing power and expressiveness, while avoiding inconsistency.
Distinct Sublanguages for Distinct Aspects of Meaning

- Separate out combinatorics, typing, and reasoning into three distinct components of the representation language.
  1. \(\lambda\)-calculus of terms for the combinatorial engine that generates representations of meaning.
  2. Types corresponding to natural language categories.
  3. Well formed formulas for reasoning.

- This division allows greater flexibility in coordinating the formal power and expressive resources of the theory with the semantic properties of natural language.

- Some interesting issues arise in balancing power and expressiveness, while avoiding inconsistency.
The three sublanguages of Property Theory with Curry Typing (PTCT) are

- **Terms**: the untyped \( \lambda \)-calculus;
- **Types**: in the Curry-style, with polymorphism and sub-types;
- **Wff**: first-order logic.

Originally devised to develop a better treatment of intensionality and types within a recursively enumerable theory.

Many subtle issues in balancing and combining the three tiers so they have appropriate expressiveness without introducing paradoxes or explicit higher-order aspects.
The Untyped $\lambda$-calculus
Terms as Representations of Meanings

- PTCT uses the untyped $\lambda$-calculus.
- It is this form of the calculus that supports models of computation.
- Its terms represent intensions as expressions in the language of computable functions.
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
Semantic Ambiguity
A simple example of scope ambiguity

- Natural Language exhibits many kinds of ambiguity, including semantic ambiguity.
- Boring old example:
  
  *Every student wrote a program.*

  How many programs?

- This is an example of a *scoping* ambiguity.
- There are other kinds of scoping ambiguity (quantified expressions, adjectives and adverbs, prepositional phrases), ...but we’ll stick with the simple example.
Natural Language exhibits many kinds of ambiguity, including semantic ambiguity.

Boring old example:

*Every student wrote a program.*

How many programs?

This is an example of a *scoping* ambiguity.

There are other kinds of scoping ambiguity (quantified expressions, adjectives and adverbs, prepositional phrases), ... but we’ll stick with the simple example.
We have many scope readings for individual sentences ($k!$ possibilities for a sentence with $k$ distinct scope-taking elements).

In general, there is no one reading that subsumes all the others.

This makes it difficult to produce and process semantic representations, and to reason with them.

We need an efficient system for compactly representing alternative scope readings.
Our approach to this problem seeks to

1. find some way of encoding the meanings of a sentence that does not enumerate all its scope readings;

2. allow fully specified (fully scoped) representations to be generated from this underspecified representation;

3. add filters to constrain the possible scoping patterns to those that are legitimate, and allowed by context;

4. use these filters for both expressive completeness and to restrict the search space of possible readings.
Our approach to this problem seeks to

1. find some way of encoding the meanings of a sentence that does not enumerate all its scope readings;
2. allow fully specified (fully scoped) representations to be generated from this underspecified representation;
3. add filters to constrain the possible scoping patterns to those that are legitimate, and allowed by context;
4. use these filters for both expressive completeness and to restrict the search space of possible readings.
Our approach to this problem seeks to

1. find some way of encoding the meanings of a sentence that does not enumerate all its scope readings;

2. allow fully specified (fully scoped) representations to be generated from this underspecified representation;

3. add filters to constrain the possible scoping patterns to those that are legitimate, and allowed by context;

4. use these filters for both expressive completeness and to restrict the search space of possible readings.
Some of the Major Issues

- How do we produce all these readings?
- Where do we produce all these readings?
- What is the status of the underspecified representations, and the mechanisms used to produce them?
- What happens to compositionality?
- How do we express constraints on the set of legitimate scope readings?
- Where are these constraints to be stated?

Most theories of underspecification use *meta-theoretic operations and conditions formulated outside of the semantic representation language.*
Some of the Major Issues

- How do we produce all these readings?
- Where do we produce all these readings?
- What is the status of the underspecified representations, and the mechanisms used to produce them?
- What happens to compositionality?
- How do we express constraints on the set of legitimate scope readings?
- Where are these constraints to be stated?

Most theories of underspecification use *meta-theoreteric operations and conditions formulated outside of the semantic representation language.*
The Basic Approach

Given that we have a rich language of terms in PTCT, we can use it to formulate compact underspecified representations as $\lambda$-terms in the representation language.

The constraints on possible scope readings can be formulated as filters (also $\lambda$-terms) on these representations.

The combinatorial operations required to specify underspecified scope representations, and the logical constants for the scope constraints are already contained in the representation language.
The Basic Approach

Given that we have a rich language of terms in PTCT, we can use it to formulate compact underspecified representations as $\lambda$-terms in the representation language.

The constraints on possible scope readings can be formulated as filters (also $\lambda$-terms) on these representations.

The combinatorial operations required to specify underspecified scope representations, and the the logical constants for the scope constraints are already contained in the representation language.
Some Additional Machinery

We need some additional machinery to implement this approach.

- **Polymorphic lists** to hold collections of scope taking elements.
- **Permutation** to produce the different scopings.
- **Relation reduction** to put the scope taking elements together to produce fully specified interpretations.
We need lists (or nested pairings) to encode the representations of scope taking elements, together with the core relation \( \langle Q_1, Q_2, \ldots Q_k, r_k \rangle \).

We implement this with product types and pairings to give a polymorphic analysis.

“Every student wrote a program” is initially represented by \( \langle Q_1, Q_2, r \rangle \), where

\[
\begin{align*}
Q_1 &= \lambda P. \forall x \in B(\text{student}^\prime(x) \rightarrow P(x)) \\
Q_2 &= \lambda Q. \exists y \in B(\text{program}^\prime(y) \land Q(y)) \\
r &= \lambda uv. \text{wrote}^\prime uv
\end{align*}
\]
Permutations

- We need a function \((perms\_scope)\) that allows us to permute the \(k\) scope-taking elements, and the positions that they bind in the core \(k\)-ary relation.

- The outcome is a \(k!\)-ary product of the different scope orderings/readings (each of which will be turned into a standard propositional form using the relation reduction machinery described later).

- \(perms\_scope\) can be characterized algorithmically, so we know it is term representable.
Permutations

- We need a function \((perms\_scope)\) that allows us to permute the \(k\) scope-taking elements, and the positions that they bind in the core \(k\)-ary relation.
- The outcome is a \(k!\)-ary product of the different scope orderings/readings (each of which will be turned into a standard propositional form using the relation reduction machinery described later).
- \(perms\_scope\) can be characterized algorithmically, so we know it is term representable.
We need a function \( \text{perms\_scope} \) that allows us to permute the \( k \) scope-taking elements, and the positions that they bind in the core \( k \)-ary relation.

The outcome is a \( k! \)-ary product of the different scope orderings/readings (\textit{each of which will be turned into a standard propositional form using the relation reduction machinery described later}).

\( \text{perms\_scope} \) can be characterized algorithmically, so we know it is term representable.
We need a function \(\text{perms}_\text{scope}\) that allows us to permute the \(k\) scope-taking elements, and the positions that they bind in the core \(k\)-ary relation.

The outcome is a \(k!\)-ary product of the different scope orderings/readings \(\text{(each of which will be turned into a standard propositional form using the relation reduction machinery described later)}\).

\(\text{perms}_\text{scope}\) can be characterized algorithmically, so we know it is term representable.
We adopt Ed Keenan’s (1992) “relation reduction” approach for composing a sequence of scope taking elements with the core relation in a uniform manner (adapted from van Eijck (2003))

The proposition corresponding to the resolved scope reading represented by $\langle Q_1, Q_2, \ldots Q_k, r_k \rangle$ is given by $RQ_1(RQ_2 \ldots (RQ_k r_k) \ldots )$

$R$ can be expressed algorithmically, so we know it is term representable.
We adopt Ed Keenan’s (1992) “relation reduction” approach for composing a sequence of scope taking elements with the core relation in a uniform manner (adapted from van Eijck (2003)).

The proposition corresponding to the resolved scope reading represented by \( \langle Q_1, Q_2, \ldots Q_k, r_k \rangle \) is given by

\[
R Q_1(R Q_2(\ldots(R Q_k r_k)\ldots))
\]

\( R \) can be expressed algorithmically, so we know it is term representable.
Underspecified representations in PTCT

First characterisation

Continuing with the previous example:

1. The representation of “every student wrote a program” is

\[ \text{perms}_{\text{scope}}(\langle Q_1, Q_2, \lambda uv.\text{wrote'}uv \rangle) \]

- Evaluating this, the intermediate permutations are
  \[ \langle \langle Q_1, Q_2, \lambda uv.\text{wrote'}uv \rangle, \langle Q_2, Q_1, \lambda vu.\text{wrote'}vu \rangle \rangle \]

- Applying relation reduction gives a list/product of propositions representing all the possible scopings:
  \[ \langle \exists x \in B(\text{student'}(x) \rightarrow \exists y \in B(\text{program'}(y) \& \text{wrote'}(x, y))), \exists y \in B(\text{program'}(y) \& \exists x \in B(\text{student'}(x) \& \text{wrote'}(x, y))) \rangle \]
Continuing with the previous example:

1. The representation of "every student wrote a program" is

   \[ \text{perms\_scope}(\langle Q_1, Q_2, \lambda uv.\text{wrote}'uv \rangle) \]

- Evaluating this, the intermediate permutations are
  \[ \langle \langle Q_1, Q_2, \lambda uv.\text{wrote}'uv \rangle, \langle Q_2, Q_1, \lambda vu.\text{wrote}'uv \rangle \rangle \]

- Applying relation reduction gives a list/product of propositions representing all the possible scopings:
  \[ \langle \forall x \in B(\text{student}'(x) \rightarrow \exists y \in B(\text{program}'(y) \& \text{wrote}'(x, y))), \exists y \in B(\text{program}'(y) \& \forall x \in B(\text{student}'(x) \& \text{wrote}'(x, y))) \rangle \]
Underspecified representations in PTCT
First characterisation

Continuing with the previous example:

1. The representation of "every student wrote a program" is

   \[ \text{perms\_scope}(\langle Q_1, Q_2, \lambda uv.\text{wrote}’uv \rangle) \]

- Evaluating this, the intermediate permutations are
  \[ \langle \langle Q_1, Q_2, \lambda uv.\text{wrote}’uv \rangle, \langle Q_2, Q_1, \lambda vu.\text{wrote}’uv \rangle \rangle \]

- Applying relation reduction gives a list/product of propositions representing all the possible scopings:
  \[ \langle \forall x \in B(\text{student}’(x) \rightarrow \exists y \in B(\text{program}’(y) \& \text{wrote}’(x, y))), \exists y \in B(\text{program}’(y) \& \forall x \in B(\text{student}’(x) \& \text{wrote}’(x, y))) \rangle \]
We can define an operation $\text{permute}^i_k$ that computes the $i$th permutation of a $k$-ary product (directly).

If we supply $\text{permute}^i_k$ with $k$ quantifiers and a $k$-ary relation, and “abstract” $i$, the result can be taken to be an underspecified representation.

2. The representation of “every student wrote a program” is then

$$\lambda i. \text{permute}^i_k (\langle Q_1, Q_2, \lambda uv. \text{wrote}'uv \rangle)$$

This representation denotes a (partial) function from integers to propositions.

If we make the function complete, then its type will will be $\text{Num} \Rightarrow \text{Prop}$. 
We can define an operation $\text{permute}_k^i$ that computes the $i$th permutation of a $k$-ary product (directly).

If we supply $\text{permute}_k^i$ with $k$ quantifiers and a $k$-ary relation, and “abstract” $i$, the result can be taken to be an underspecified representation.

2. The representation of “every student wrote a program” is then

$$\lambda i. \text{permute}_k^i(\langle Q_1, Q_2, \lambda uv. \text{wrote}' uv \rangle)$$

This representation denotes a (partial) function from integers to propositions.

If we make the function complete, then its type will be $\text{Num} \Rightarrow \text{Prop}$. 
Underspecified representations in PTCT

Second characterisation

- We can define an operation $\text{permute}_k^i$ that computes the $i$th permutation of a $k$-ary product (directly).
- If we supply $\text{permute}_k^i$ with $k$ quantifiers and a $k$-ary relation, and “abstract” $i$, the result can be taken to be an underspecified representation.

2. The representation of “every student wrote a program” is then

$$\lambda i. \text{permute}_k^i(\langle Q_1, Q_2, \lambda uv. \text{wrote}'uv \rangle)$$

- This representation denotes a (partial) function from integers to propositions.
- If we make the function complete, then its type will be $\text{Num} \Rightarrow \text{Prop}$. 

Chris Fox and Shalom Lappin
Underspecified Scope Representations
Underspecified representations in PTCT
Second characterisation

- We can define an operation $\text{permute}_k^i$ that computes the $i$th permutation of a $k$-ary product (directly).
- If we supply $\text{permute}_k^i$ with $k$ quantifiers and a $k$-ary relation, and “abstract” $i$, the result can be taken to be an underspecified representation.

2. The representation of “every student wrote a program” is then

$$\lambda i. \text{permute}_k^i(\langle Q_1, Q_2, \lambda uv. \text{wrote}'uv \rangle)$$

- This representation denotes a (partial) function from integers to propositions.
- If we make the function complete, then its type will be $\text{Num} \Rightarrow \text{Prop}$. 
We can define an operation $\text{permute}_k^i$ that computes the $i$th permutation of a $k$-ary product (directly).

If we supply $\text{permute}_k^i$ with $k$ quantifiers and a $k$-ary relation, and “abstract” $i$, the result can be taken to be an underspecified representation.

2. The representation of “every student wrote a program” is then

$$\lambda i. \text{permute}_k^i(\langle Q_1, Q_2, \lambda uv.\text{wrote}'uv \rangle)$$

This representation denotes a (partial) function from integers to propositions.

If we make the function complete, then its type will will be $\text{Num} \Rightarrow \text{Prop}$. 
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
We can express filters on possible scopings using the full power of PTCT.

This includes **lexical constraints** (e.g. “a certain” tends to take wide scope, as in “Every critic reviewed a certain book”),

... **syntactic constraints** (e.g. “every assignment” tends to take narrow scope in “A student who completed every assignment came first in the class”),

... and **anaphora resolution constraints**. Evaluation of an underspecified scope term can be delayed until a point in subsequent discourse where these filters apply.
Filtering readings

- We can express filters on possible scopings using the full power of PTCT.

- This includes **lexical constraints** (e.g. “a certain” tends to take wide scope, as in “Every critic reviewed a certain book”),

- **syntactic constraints** (e.g. “every assignment” tends to take narrow scope in “A student who completed every assignment came first in the class”),

- **anaphora resolution constraints**. Evaluation of an underspecified scope term can be delayed until a point in subsequent discourse where these filters apply.
Filtering readings

- We can express filters on possible scopings using the full power of PTCT.

- This includes *lexical constraints* (e.g. “a certain” tends to take wide scope, as in “Every critic reviewed a certain book”.),

- *syntactic constraints* (e.g. “every assignment” tends to take narrow scope in “A student who completed every assignment came first in the class.”),

- ... and *anaphora resolution constraints*. Evaluation of an underspecified scope term can be delayed until a point in subsequent discourse where these filters apply.
Filtering readings

- We can express filters on possible scopings using the full power of PTCT.
- This includes **lexical constraints** (e.g. “a certain” tends to take wide scope, as in “Every critic reviewed a certain book”),
- **syntactic constraints** (e.g. “every assignment” tends to take narrow scope in “A student who completed every assignment came first in the class”),
- and **anaphora resolution constraints**. Evaluation of an underspecified scope term can be delayed until a point in subsequent discourse where these filters apply.
Scope Disambiguation through Anaphora Resolution

A: Every student$_1$ wrote a program$_2$ for some professor$_3$.

B: Yes, she$_3$ taught the Haskell course this year.

C: The programs$_2$ were all list sorting procedures.

This resolution of the pronoun in B and the definite description in C yields the scope order $\langle$ some professor, every student, a program $\rangle$ for A.
A: Every student\textsubscript{1} wrote a program\textsubscript{2} for some professor\textsubscript{3}.
B: Yes, she\textsubscript{3} taught the Haskell course this year.
C: The programs\textsubscript{2} were all list sorting procedures.

This resolution of the pronoun in B and the definite description in C yields the scope order $\langle$ some professor, every student, a program $\rangle$ for A.
Scope Disambiguation through Anaphora Resolution

A: *Every student*$_1$ *wrote a program*$_2$ *for some professor*$_3$.

B: *Yes, she*$_3$ *taught the Haskell course this year.*

C: *The programs*$_2$ *were all list sorting procedures.*

This resolution of the pronoun in B and the definite description in C yields the scope order ⟨*some professor*, *every student*, *a program*⟩ for A.
Scope Disambiguation through Anaphora Resolution

A: Every student\(_1\) wrote a program\(_2\) for some professor\(_3\).
B: Yes, she\(_3\) taught the Haskell course this year.
C: The programs\(_2\) were all list sorting procedures.

This resolution of the pronoun in B and the definite description in C yields the scope order \(\langle\text{some professor, every student, a program}\rangle\) for A.
Let \( \langle Quants, Rel \rangle \) be a variable ranging over pairs in which \textit{Quants} is a \( k \)-tuple and \textit{Rel} is a \( k \)-ary relation.

As the \( k \)-tuples are indexed, there is a one-to-one correspondence between the elements of a \( k \)-tuple and their respective indices.

Let \textit{tuple\_element}(i, Quants) = \( Q_i \) if \( Q_i \) is the \( i \)th member of \textit{Quants}, and the distinguished term \( \omega \) otherwise.
Filters as $\lambda$-Terms in PTCT

- Let $\langle Quants, Rel \rangle$ be a variable ranging over pairs in which $Quants$ is a $k$-tuple and $Rel$ is a $k$-ary relation.
- As the $k$-tuples are indexed, there is a one-to-one correspondence between the elements of a $k$-tuple and their respective indices.
- Let $\text{tuple\_element}(i, Quants) = Q_i$ if $Q_i$ is the $i$th member of $Quants$, and the distinguished term $\omega$ otherwise.
Filters as $\lambda$-Terms in PTCT

- Let $\langle Quants, Rel \rangle$ be a variable ranging over pairs in which $Quants$ is a $k$-tuple and $Rel$ is a $k$-ary relation.
- As the $k$-tuples are indexed, there is a one-to-one correspondence between the elements of a $k$-tuple and their respective indices.
- Let $tuple\_element(i, Quants) = Q_i$ if $Q_i$ is the $i$th member of $Quants$, and the distinguished term $\omega$ otherwise.
We can formulate the filters contributed by the anaphora resolutions in B and C, respectively, as follows, where

\[ Q_1 = \text{every student}, \quad Q_2 = \text{a program}, \quad \text{and} \]

\[ Q_3 = \text{some professor}, \quad \text{and} \]

\[ GQ \text{ in } \hat{\sim}_{GQ} \text{ abbreviates the appropriate type of } Q_i, \]

\[ \lambda\langle\text{Quants}, \text{Rel}\rangle[\forall i \in \text{Num} \forall j \in \text{Num}(\text{tuple element}(i, \text{Quants}) \hat{\sim}_{GQ} Q_3 \land \text{tuple element}(j, \text{Quants}) \hat{\sim}_{GQ} Q_1) \rightarrow i \hat{<} j)]] \]

\[ \lambda\langle\text{Quants}, \text{Rel}\rangle[\forall i \in \text{Num} \forall j \in \text{Num}(\text{tuple element}(i, \text{Quants}) \hat{\sim}_{GQ} Q_1 \land \text{tuple element}(j, \text{Quants}) \hat{\sim}_{GQ} Q_2) \rightarrow i \hat{<} j)] \]
We can formulate the filters contributed by the anaphora resolutions in B and C, respectively, as follows, where

\( Q_1 = \text{every student} \), \( Q_2 = \text{a program} \), and

\( Q_3 = \text{some professor} \), and

\( GQ \) in \( \hat{=} \) abbreviates the appropriate type of \( Q_i \),

\[
\lambda \langle \text{Quants}, \text{Rel} \rangle [\forall i \in \text{Num} \forall j \in \text{Num}( (\text{tuple}_\text{element}(i, \text{Quants}) \hat{=} GQ Q_3 \land \text{tuple}_\text{element}(j, \text{Quants}) \hat{=} GQ Q_1 ) \rightarrow i < j )]
\]

\[
\lambda \langle \text{Quants}, \text{Rel} \rangle [\forall i \in \text{Num} \forall j \in \text{Num}( (\text{tuple}_\text{element}(i, \text{Quants}) \hat{=} GQ Q_1 \land \text{tuple}_\text{element}(j, \text{Quants}) \hat{=} GQ Q_2 ) \rightarrow i < j )]
\]
We can formulate the filters contributed by the anaphora resolutions in B and C, respectively, as follows, where $Q_1 = \text{every student}$, $Q_2 = \text{a program}$, and $Q_3 = \text{some professor}$, and $GQ$ in $\hat{\bowtie}_{GQ}$ abbreviates the appropriate type of $Q_i$.

$$\lambda\langle\text{Quants, Rel}\rangle[\hat{\forall}i \in \text{Num}\hat{\forall}j \in \text{Num}((\text{tuple}_\text{element}(i, \text{Quants}) \hat{\bowtie}_{GQ} Q_3 \& \text{tuple}_\text{element}(j, \text{Quants}) \hat{\bowtie}_{GQ} Q_1) \rightarrow i < j)]$$

$$\lambda\langle\text{Quants, Rel}\rangle[\hat{\forall}i \in \text{Num}\hat{\forall}j \in \text{Num}((\text{tuple}_\text{element}(i, \text{Quants}) \hat{\bowtie}_{GQ} Q_1 \& \text{tuple}_\text{element}(j, \text{Quants}) \hat{\bowtie}_{GQ} Q_2) \rightarrow i < j)]$$
The function \( \text{filter}_{\text{tuple}}(\langle F, T \rangle) \) maps a pair consisting of a \( j \)-tuple \( F \) of filters and a \( k \)-tuple \( T \) to a \( k' \)-tuple (possibly the empty tuple) of all the elements of \( T \) that satisfy each filter in \( F \).

\( \text{filter}_{\text{tuple}}(\langle F, \text{perms}_{\text{scope}}_k(\langle \text{Quants}_k, \text{Rel} \rangle) \rangle) \)

represents the \( k' \)-tuple obtained by applying the elements of \( F \) to the \( k! \)-tuple that is the value of \( \text{perms}_{\text{scope}}_k(\langle \text{Quants}_k, \text{Rel} \rangle) \).
A PTCT Term for Filtered $k!$-Tuples

- The function $\text{filter}_\text{tuple}(\langle F, T \rangle)$ maps a pair consisting of a $j$-tuple $F$ of filters and a $k$-tuple $T$ to a $k'$-tuple (possibly the empty tuple) of all the elements of $T$ that satisfy each filter in $F$.

- $\text{filter}_\text{tuple}(\langle F, \text{perms}_\text{scope}_k(\langle \text{Quants}_k, \text{Rel} \rangle) \rangle)$ represents the $k'$-tuple obtained by applying the elements of $F$ to the $k!$-tuple that is the value of $\text{perms}_\text{scope}_k(\langle \text{Quants}_k, \text{Rel} \rangle)$.
Ebert (2005) shows that most current theories of underspecification are expressively incomplete. They cannot identify the proper subset of possible scope readings specified by Boolean operations other than conjunction, and in particular by negation.

Every market manager showed five sales representatives a sample.

In Ebert’s example, real world knowledge allows all scope permutations except the one corresponding to $\langle \exists, 5, \forall \rangle$. 
Filters with Boolean Conditions other than Conjunction

- Ebert (2005) shows that most current theories of underspecification are expressively incomplete.
- They cannot identify the proper subset of possible scope readings specified by Boolean operations other than conjunction, and in particular by negation.

Every market manager showed five sales representatives a sample.

In Ebert’s example, real world knowledge allows all scope permutations except the one corresponding to $\langle \exists, 5, \forall \rangle$. 
Ebert (2005) shows that most current theories of underspecification are expressively incomplete.

They cannot identify the proper subset of possible scope readings specified by Boolean operations other than conjunction, and in particular by negation.

*Every market manager showed five sales representatives a sample.*

In Ebert’s example, real world knowledge allows all scope permutations except the one corresponding to $\langle \exists, 5, \forall \rangle$. 
Filters with Boolean Conditions other than Conjunction

- Ebert (2005) shows that most current theories of underspecification are expressively incomplete.
- They cannot identify the proper subset of possible scope readings specified by Boolean operations other than conjunction, and in particular by negation.

Every market manager showed five sales representatives a sample.

In Ebert’s example, real world knowledge allows all scope permutations except the one corresponding to $\langle \exists, 5, \forall \rangle$. 
Filters with Boolean Conditions other than Conjunction

Ebert demonstrates that

- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)

cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Ebert demonstrates that

- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)

cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Filters with Boolean Conditions other than Conjunction

Ebert demonstrates that

- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)

cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Ebert demonstrates that

- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)

cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Ebert demonstrates that

- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)

cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Ebert demonstrates that
- storage (Cooper 1983, Pereira 1990),
- hole semantics (Bos 1995, Blackburn and Bos 2005),
- Minimal Recursion Semantics (Copestake et al. 1997), and
- Normal Dominance Conditions (Koller et al. 2003)
cannot formulate underspecified representations that express the set containing only the five remaining scope readings.
Filters with Boolean Conditions other than Conjunctions

By contrast, it is straightforward to formulate a filter in PTCT that rules out the problematic scope sequence.

\[ \lambda \langle Quants, Rel \rangle [ \forall i \in \text{Num} \forall j \in \text{Num} \forall k \in \text{Num} ( (\text{tuple}_\text{element}(i, Quants) \cong_{Q3} Q_3 \land \text{tuple}_\text{element}(j, Quants) \cong_{Q5} Q_5 \land \text{tuple}_\text{element}(k, Quants) \cong_{Qv} Q_v) \rightarrow \sim(i > j \land j > k) ) ] \]
Filters with Boolean Conditions other than Conjunctions

By contrast, it is straightforward to formulate a filter in PTCT that rules out the problematic scope sequence.

\[
\lambda\langle Quants, Rel\rangle[ \\
\forall i \in \text{Num} \forall j \in \text{Num} \forall k \in \text{Num}( \\
\text{tuple}\_\text{element}(i, Quants) \cong_{GQ} Q_{\exists} \land \\
\text{tuple}\_\text{element}(j, Quants) \cong_{GQ} Q_{5} \land \\
\text{tuple}\_\text{element}(k, Quants) \cong_{GQ} Q_{\forall}) \rightarrow \\
\sim(i \gtrsim j \land j \gtrsim k) ]
\]
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
A Tree Generation Algorithm for Producing the Indexed $k!$-Tuple of Possible Scope Permutations

- It is possible to formulate a procedure for generating all possible permutations of scope taking elements as a tree construction algorithm.
- Given a $k$-tuple $\langle Q_1, \ldots, Q_k \rangle$, a tree is generated breadth first, starting with the root of the tree at $Q_1$.
- At each new level $i$ of the tree, $Q_i$ is successively added in each possible position, from right to left, to the $(i - 1)$ – *tuples* at the daughter nodes of the previous level, $i - 1$.
- The tree is finished when the $k$ – *th* level of the tree is generated.
It is possible to formulate a procedure for generating all possible permutations of scope taking elements as a tree construction algorithm.

Given a $k$-tuple $\langle Q_1, \ldots, Q_k \rangle$, a tree is generated breadth first, starting with the root of the tree at $Q_1$.

At each new level $i$ of the tree, $Q_i$ is successively added in each possible position, from right to left, to the $(i - 1)$ - tuples at the daughter nodes of the previous level, $i - 1$.

The tree is finished when the $k$ - th level of the tree is generated.
A Tree Generation Algorithm for Producing the Indexed $k!$-Tuple of Possible Scope Permutations

- It is possible to formulate a procedure for generating all possible permutations of scope taking elements as a tree construction algorithm.
- Given a $k$-tuple $\langle Q_1, \ldots, Q_k \rangle$, a tree is generated breadth first, starting with the root of the tree at $Q_1$.
- At each new level $i$ of the tree, $Q_i$ is successively added in each possible position, from right to left, to the $(i - 1)$-tuples at the daughter nodes of the previous level, $i - 1$.
- The tree is finished when the $k$-th level of the tree is generated.
A Tree Generation Algorithm for Producing the Indexed $k!$-Tuple of Possible Scope Permutations

- It is possible to formulate a procedure for generating all possible permutations of scope taking elements as a tree construction algorithm.
- Given a $k$-tuple $\langle Q_1, \ldots Q_k \rangle$, a tree is generated breadth first, starting with the root of the tree at $Q_1$.
- At each new level $i$ of the tree, $Q_i$ is successively added in each possible position, from right to left, to the $(i - 1)$ – *tuples* at the daughter nodes of the previous level, $i - 1$.
- The tree is finished when the $k – th$ level of the tree is generated.
The permutation tree for $\langle Q_1, Q_2, Q_3 \rangle$ is as follows.
The permutation tree for \( \langle Q_1, Q_2, Q_3 \rangle \) is as follows.

\[
\begin{align*}
\langle Q_1 \rangle \\
\langle Q_1, Q_2 \rangle \\
\langle Q_1, Q_2, Q_3 \rangle & \langle Q_1, Q_3, Q_2 \rangle \\
\langle Q_3, Q_1, Q_2 \rangle & \langle Q_2, Q_1, Q_3 \rangle \\
\langle Q_2, Q_3, Q_1 \rangle & \langle Q_3, Q_2, Q_1 \rangle
\end{align*}
\]
Filters can apply in this algorithm as constraints on nodes. If a node violates a scope ordering filter, then the subtree that it dominates is not generated. Consider the filter $Q_1 < Q_2$. 

\[
\langle Q_1 \rangle \\
\langle Q_1, Q_2 \rangle \\
\langle Q_1, Q_2, Q_3 \rangle, \langle Q_1, Q_3, Q_2 \rangle, \langle Q_3, Q_1, Q_2 \rangle, \langle Q_2, Q_1 \rangle
\]
Filters can apply in this algorithm as constraints on nodes.

If a node violates a scope ordering filter, then the subtree that it dominates is not generated. Consider the filter $Q_1 < Q_2$.
Filters can apply in this algorithm as constraints on nodes.

If a node violates a scope ordering filter, then the subtree that it dominates is not generated. Consider the filter $Q_1 < Q_2$. 

![Diagram](https://via.placeholder.com/150)

- $\langle Q_1 \rangle$
- $\langle Q_1, Q_2 \rangle$
- $\langle Q_1, Q_2, Q_3 \rangle$
- $\langle Q_1, Q_3, Q_2 \rangle$
- $\langle Q_3, Q_1, Q_2 \rangle$
- $\langle Q_2, Q_1 \rangle$
Identifying the size of a tree with the number of its nodes, we can compute the size of a tree $T$, $|T|$, through the formula

$$|T| = \sum i!,$$

where $i$ is the index of the $i$th element of the initial $k$-tuple which the algorithm takes as its input.

- The size of the tree for $\langle Q_1, Q_2, Q_3 \rangle$ is $1! + 2! + 3! = 9$.
- The size of the tree produced by pruning this tree with the filter $Q_1 < Q_2$ is 6, which is a reduction of 30%.
- The size of a subtree $ST$ dominated by a node $n$ at level $i$, but not including $n$, is given by the formula

$$|ST| = \prod j \ (i < j \leq k) + \sum j' \ (i < j' < k)$$

Chris Fox and Shalom Lappin

Underspecified Scope Representations
Computing the Size of a Tree

- Identifying the size of a tree with the number of its nodes, we can compute the size of a tree $T$, $|T|$, through the formula $|T| = \sum i!$, where $i$ is the index of the $i$th element of the initial $k$-tuple which the algorithm takes as its input.

- The size of the tree for $\langle Q_1, Q_2, Q_3 \rangle$ is $1! + 2! + 3! = 9$.

- The size of the tree produced by pruning this tree with the filter $Q_1 < Q_2$ is $6$, which is a reduction of $30\%$.

- The size of a subtree $ST$ dominated by a node $n$ at level $i$, but not including $n$, is given by the formula $|ST| = \prod j \ (i < j \leq k) + \sum j' \ (i < j' < k)$.
Computing the Size of a Tree

- Identifying the size of a tree with the number of its nodes, we can compute the size of a tree $T$, $|T|$, through the formula
  
  $$|T| = \sum i!,$$

  where $i$ is the index of the $i$th element of the initial $k$-tuple which the algorithm takes as its input.

- The size of the tree for $\langle Q_1, Q_2, Q_3 \rangle$ is $1! + 2! + 3! = 9$.

- The size of the tree produced by pruning this tree with the filter $Q_1 < Q_2$ is 6, which is a reduction of 30%.

- The size of a subtree $ST$ dominated by a node $n$ at level $i$, but not including $n$, is given by the formula
  
  $$|ST| = \prod j \ (i < j \leq k) + \sum j' \ (i < j' < k)$$
Computing the Size of a Tree

- Identifying the size of a tree with the number of its nodes, we can compute the size of a tree \( T \), \( |T| \), through the formula
  \[ |T| = \sum i! \]
  where \( i \) is the index of the \( i \)th element of the initial \( k \)-tuple which the algorithm takes as its input.

- The size of the tree for \( \langle Q_1, Q_2, Q_3 \rangle \) is \( 1! + 2! + 3! = 9 \).

- The size of the tree produced by pruning this tree with the filter \( Q_1 < Q_2 \) is 6, which is a reduction of 30%.

- The size of a subtree \( ST \) dominated by a node \( n \) at level \( i \), but not including \( n \), is given by the formula
  \[ |ST| = \prod j \ (i < j \leq k) + \sum j' \ (i < j' < k) \]
The tree for \( \langle Q_1, Q_2, Q_3, Q_4 \rangle \) has an indexed \( k! \)-tuple of 24 \( k \)-tuples as the leaves of a tree \( T_4 \) with 4 levels and 33 nodes.

If a filter like \( Q_1 < Q_2 \) applies at level 2, the first branching node of \( T_4 \), it prunes the right-half of \( T_4 \) under \( \langle Q_2, Q_1 \rangle \).

This pruning eliminates a subtree of 15 nodes and reducing \( T_4 \) by \( 15/33 = 45.4\% \).
Pruning with Several Filters

- The tree for \( \langle Q_1, Q_2, Q_3, Q_4 \rangle \) has an indexed \( k! \)-tuple of 24 \( k \)-tuples as the leaves of a tree \( T_4 \) with 4 levels and 33 nodes.

- If a filter like \( Q_1 < Q_2 \) applies at level 2, the first branching node of \( T_4 \), it prunes the right-half of \( T_4 \) under \( \langle Q_2, Q_1 \rangle \).

- This pruning eliminates a subtree of 15 nodes and reducing \( T_4 \) by \( 15/33 = 45.4\% \).
Pruning with Several Filters

- The tree for \(\langle Q_1, Q_2, Q_3, Q_4 \rangle\) has an indexed \(k!\)-tuple of 24 \(k\)-tuples as the leaves of a tree \(T_4\) with 4 levels and 33 nodes.
- If a filter like \(Q_1 < Q_2\) applies at level 2, the first branching node of \(T_4\), it prunes the right-half of \(T_4\) under \(\langle Q_2, Q_1 \rangle\).
- This pruning eliminates a subtree of 15 nodes and reducing \(T_4\) by \(15/33 = 45.4\%\).
Pruning with Several Filters

- The remaining left side of $T_4$ has the three nodes $\langle Q_1, Q_2, Q_3 \rangle$, $\langle Q_1, Q_3, Q_2 \rangle$, $\langle Q_3, Q_1, Q_2 \rangle$ at level 3.
- Applying the filter $Q_2 < Q_3$ at this level removes the 8 leaf nodes under $\langle Q_1, Q_3, Q_2 \rangle$ and $\langle Q_3, Q_1, Q_2 \rangle$.
- Therefore the conjunction of the filters $Q_1 < Q_2$ and $Q_2 < Q_3$ reduces $T_4$ by $15 + 8 = 23$ nodes, which is (approximately) 70% of the full tree.
The remaining left side of $T_4$ has the three nodes $\langle Q_1, Q_2, Q_3 \rangle$, $\langle Q_1, Q_3, Q_2 \rangle$, $\langle Q_3, Q_1, Q_2 \rangle$ at level 3.

Applying the filter $Q_2 < Q_3$ at this level removes the 8 leaf nodes under $\langle Q_1, Q_3, Q_2 \rangle$ and $\langle Q_3, Q_1, Q_2 \rangle$.

Therefore the conjunction of the filters $Q_1 < Q_2$ and $Q_2 < Q_3$ reduces $T_4$ by $15 + 8 = 23$ nodes, which is (approximately) 70% of the full tree.
The remaining left side of $T_4$ has the three nodes $\langle Q_1, Q_2, Q_3 \rangle$, $\langle Q_1, Q_3, Q_2 \rangle$, $\langle Q_3, Q_1, Q_2 \rangle$ at level 3.

Applying the filter $Q_2 < Q_3$ at this level removes the 8 leaf nodes under $\langle Q_1, Q_3, Q_2 \rangle$ and $\langle Q_3, Q_1, Q_2 \rangle$.

Therefore the conjunction of the filters $Q_1 < Q_2$ and $Q_2 < Q_3$ reduces $T_4$ by $15 + 8 = 23$ nodes, which is (approximately) 70% of the full tree.
It is not difficult to construct a plausible case in which the interpretation of a sentence containing four quantified NPs is disambiguated by a conjunction of two filters of this kind through anaphora resolution in subsequent discourse.

A: A critic recently reviewed two plays for every newspaper in a major city.

B: Yes, he published the same reviews of the current productions of ‘A Midsummer Night’s Dream’ and ‘New-Found-Land’ in every paper in New York last week.
It is not difficult to construct a plausible case in which the interpretation of a sentence containing four quantified NPs is disambiguated by a conjunction of two filters of this kind through anaphora resolution in subsequent discourse.

**A:** A critic recently reviewed two plays for every newspaper in a major city.

**B:** Yes, he published the same reviews of the current productions of ‘A Midsummer Night’s Dream’ and ‘New-Found-Land’ in every paper in New York last week.
It is not difficult to construct a plausible case in which the interpretation of a sentence containing four quantified NPs is disambiguated by a conjunction of two filters of this kind through anaphora resolution in subsequent discourse.

**A:** A critic recently reviewed two plays for every newspaper in a major city.

**B:** Yes, he published the same reviews of the current productions of ‘A Midsummer Night’s Dream’ and ‘New-Found-Land’ in every paper in New York last week.
The earlier in the tree construction process (the higher up in the tree) that a filter applies, the greater the reduction in search space that it achieves.

It is possible to optimise the interaction of filters and the tree construction algorithm by specifying a procedure that reorders the elements of the input \( k \)-tuple to permit the filters to apply at the earliest possible point in the generation of the tree.

Assume, for example, that the algorithm takes as its input the triple \( \langle Q_1, Q_2, Q_3 \rangle \) and one of the filters that applies to this triple is \( Q_2 < Q_3 \).

The reordering operation will map the triple into \( \langle Q_2, Q_3, Q_1 \rangle \).
The earlier in the tree construction process (the higher up in the tree) that a filter applies, the greater the reduction in search space that it achieves.

It is possible to optimise the interaction of filters and the tree construction algorithm by specifying a procedure that reorders the elements of the input $k$-tuple to permit the filters to apply at the earliest possible point in the generation of the tree.

Assume, for example, that the algorithm takes as its input the triple $\langle Q_1, Q_2, Q_3 \rangle$ and one of the filters that applies to this triple is $Q_2 < Q_3$.

The reordering operation will map the triple into $\langle Q_2, Q_3, Q_1 \rangle$. 

Chris Fox and Shalom Lappin
Optimizing the Application of Filters

The earlier in the tree construction process (the higher up in the tree) that a filter applies, the greater the reduction in search space that it achieves.

It is possible to optimise the interaction of filters and the tree construction algorithm by specifying a procedure that reorders the elements of the input \(k\)-tuple to permit the filters to apply at the earliest possible point in the generation of the tree.

Assume, for example, that the algorithm takes as its input the triple \(\langle Q_1, Q_2, Q_3 \rangle\) and one of the filters that applies to this triple is \(Q_2 < Q_3\).

The reordering operation will map the triple into \(\langle Q_2, Q_3, Q_1 \rangle\).
The earlier in the tree construction process (the higher up in the tree) that a filter applies, the greater the reduction in search space that it achieves.

It is possible to optimise the interaction of filters and the tree construction algorithm by specifying a procedure that reorders the elements of the input $k$-tuple to permit the filters to apply at the earliest possible point in the generation of the tree.

Assume, for example, that the algorithm takes as its input the triple $\langle Q_1, Q_2, Q_3 \rangle$ and one of the filters that applies to this triple is $Q_2 < Q_3$.

The reordering operation will map the triple into $\langle Q_2, Q_3, Q_1 \rangle$. 

Chris Fox and Shalom Lappin

Underspecified Scope Representations
Ebert (2005) proves a theorem which entails that if a theory is expressively complete, then it will, in the worst case, produce a combinatorial explosion equivalent to generating all \( k! \) scope readings for a sentence.

This result holds in the limit cases where (i) no filters apply, or (ii) they do not operate early enough in the tree construction algorithm to prevent the generation of any nodes.

However, there is a significant class of cases in which filters substantially reduce the search space through tree pruning.

For these non-worst cases filters offer a mechanism for rendering scope disambiguation computationally efficient.
Ebert (2005) proves a theorem which entails that if a theory is expressively complete, then it will, in the worst case, produce a combinatorial explosion equivalent to generating all $k!$ scope readings for a sentence.

This result holds in the limit cases where

(i) no filters apply, or
(ii) they do not operate early enough in the tree construction algorithm to prevent the generation of any nodes.

However, there is a significant class of cases in which filters substantially reduce the search space through tree pruning.

For these non-worst cases filters offer a mechanism for rendering scope disambiguation computationally efficient.
Efficiency in Non-Worst Case Scenarios

- Ebert (2005) proves a theorem which entails that if a theory is expressively complete, then it will, in the worst case, produce a combinatorial explosion equivalent to generating all $k!$ scope readings for a sentence.

- This result holds in the limit cases where
  (i) no filters apply, or
  (ii) they do not operate early enough in the tree construction algorithm to prevent the generation of any nodes.

- However, there is a significant class of cases in which filters substantially reduce the search space through tree pruning.

- For these non-worst cases filters offer a mechanism for rendering scope disambiguation computationally efficient.
Ebert (2005) proves a theorem which entails that if a theory is expressively complete, then it will, in the worst case, produce a combinatorial explosion equivalent to generating all $k!$ scope readings for a sentence.

This result holds in the limit cases where (i) no filters apply, or (ii) they do not operate early enough in the tree construction algorithm to prevent the generation of any nodes.

However, there is a significant class of cases in which filters substantially reduce the search space through tree pruning.

For these non-worst cases filters offer a mechanism for rendering scope disambiguation computationally efficient.
Outline

1. Background
   - Semantic Representation
   - The Standard Approach to Semantics
   - An Alternative Approach

2. Underspecification in PTCT
   - Representing Scope Ambiguity in PTCT
   - Expressive Completeness
   - Computational Efficiency
   - Comparison with Other Proposals

3. Conclusions
Other Theories of Underspecified Semantic Representation

There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
- **Glue Languages**: Dalrymple et al. (1999), Crouch and van Genabith (1999).
Other Theories of Underspecified Semantic Representation

There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
- **Glue Languages**: Dalrymple *et al.* (1999), Crouch and van Genabith (1999).
Other Theories of Underspecified Semantic Representation

There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
Other Theories of Underspecified Semantic Representation

There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
- **Glue Languages**: Dalrymple *et al.* (1999), Crouch and van Genabith (1999).
Other Theories of Underspecified Semantic Representation

There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
- **Glue Languages**: Dalrymple *et al.* (1999), Crouch and van Genabith (1999).
There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
- **Glue Languages**: Dalrymple *et al.* (1999), Crouch and van Genabith (1999).
There are many other proposals, e.g.:

- **Underspecified DRT**: Reyle (1993).
- **Minimal Recursion Semantics**: Copestake et al. (1997).
- **Normal Dominance Conditions**: Koller et al. (2003).
Other Theories of Underspecified Semantic Representation

These accounts have significant limitations:

- They require meta-theoretic operations and additional constraint languages.
- They posit objects whose formal status is unclear (e.g. indexed place-holders).
- They lack the resources for stating the full range of Boolean conditions on possible readings required for achieving expressive completeness in the sense of Ebert (2005).
Other Theories of Underspecified Semantic Representation

These accounts have significant limitations:

- They require meta-theoretic operations and additional constraint languages.
- They posit objects whose formal status is unclear (e.g. indexed place-holders).
- They lack the resources for stating the full range of Boolean conditions on possible readings required for achieving expressive completeness in the sense of Ebert (2005).
These accounts have significant limitations:

- They require meta-theoretic operations and additional constraint languages.
- They posit objects whose formal status is unclear (e.g. indexed place-holders).
- They lack the resources for stating the full range of Boolean conditions on possible readings required for achieving expressive completeness in the sense of Ebert (2005).
Other Theories of Underspecified Semantic Representation

These accounts have significant limitations:

- They require meta-theoretic operations and additional constraint languages.
- They posit objects whose formal status is unclear (e.g. indexed place-holders).
- They lack the resources for stating the full range of Boolean conditions on possible readings required for achieving expressive completeness in the sense of Ebert (2005).
Crouch (2005) describes a procedure for generating all scoped interpretations for a sentence as a set of packed clauses in which components of meaning shared by several readings are expressed as a single common clause.

Scope readings are distinguished by clauses in the set that encode their distinctive elements.

Packing uses the approach that is applied in chart parsing to construct non-redundant representations of the full set of possible syntactic structures for a parsed phrase.
Packed Scope Representations

- Crouch (2005) describes a procedure for generating all scoped interpretations for a sentence as a set of packed clauses in which components of meaning shared by several readings are expressed as a single common clause.

- Scope readings are distinguished by clauses in the set that encode their distinctive elements.

- Packing uses the approach that is applied in chart parsing to construct non-redundant representations of the full set of possible syntactic structures for a parsed phrase.
Crouch (2005) describes a procedure for generating all scoped interpretations for a sentence as a set of packed clauses in which components of meaning shared by several readings are expressed as a single common clause.

Scope readings are distinguished by clauses in the set that encode their distinctive elements.

Packing uses the approach that is applied in chart parsing to construct non-redundant representations of the full set of possible syntactic structures for a parsed phrase.
In the system that Crouch cites the choice space of Boolean combinations of clauses to be tested for satisfiability is optimized using Maxwell and Kaplan’s (1995) method for rendering disjunctive constraint satisfaction efficient.
Packing vs. Underspecification

- Packing offers an elegant and efficient way of representing and reasoning with the full set of possible scope readings for a sentence.
- However, it requires that this set be computed as part of the parsing and compositional interpretation.
- With underspecified representations it is possible to avoid computing the set of scoped readings until a subsequent point in the processing of a discourse or text.
- Filters acquired in subsequent discourse can reduce the set of possible scope interpretations to be computed.
Packing offers an elegant and efficient way of representing and reasoning with the full set of possible scope readings for a sentence.

However, it requires that this set be computed as part of the parsing and compositional interpretation.

With underspecified representations it is possible to avoid computing the set of scoped readings until a subsequent point in the processing of a discourse or text.

Filters acquired in subsequent discourse can reduce the set of possible scope interpretations to be computed.
Packing vs. Underspecification

- Packing offers an elegant and efficient way of representing and reasoning with the full set of possible scope readings for a sentence.
- However, it requires that this set be computed as part of the parsing and compositional interpretation.
- With underspecified representations it is possible to avoid computing the set of scoped readings until a subsequent point in the processing of a discourse or text.
- Filters acquired in subsequent discourse can reduce the set of possible scope interpretations to be computed.
Packing offers an elegant and efficient way of representing and reasoning with the full set of possible scope readings for a sentence.

However, it requires that this set be computed as part of the parsing and compositional interpretation.

With underspecified representations it is possible to avoid computing the set of scoped readings until a subsequent point in the processing of a discourse or text.

Filters acquired in subsequent discourse can reduce the set of possible scope interpretations to be computed.
The design of PTCT allows us to incorporate a treatment of underspecification within the semantic representation language.

This avoids the need for additional meta-theoretic machinery whose status may be unclear.

Filters, formulated as $\lambda$-terms in PTCT, permit the theory to achieve expressive completeness by using Boolean constraints to identify any subset of possible scope readings associated with a sentence.

Filters also render scope disambiguation efficient, in non-worst case scenarios, by restricting the search space of possible scope permutations.
Conclusions

The design of PTCT allows us to incorporate a treatment of underspecification within the semantic representation language.

This avoids the need for additional meta-theoretic machinery whose status may be unclear.

Filters, formulated as λ-terms in PTCT, permit the theory to achieve expressive completeness by using Boolean constraints to identify any subset of possible scope readings associated with a sentence.

Filters also render scope disambiguation efficient, in non-worst case scenarios, by restricting the search space of possible scope permutations.
Conclusions

- The design of PTCT allows us to incorporate a treatment of underspecification within the semantic representation language.
- This avoids the need for additional meta-theoretic machinery whose status may be unclear.
- Filters, formulated as $\lambda$-terms in PTCT, permit the theory to achieve expressive completeness by using Boolean constraints to identify any subset of possible scope readings associated with a sentence.
- Filters also render scope disambiguation efficient, in non-worst case scenarios, by restricting the search space of possible scope permutations.
Conclusions

- The design of PTCT allows us to incorporate a treatment of underspecification within the semantic representation language.
- This avoids the need for additional meta-theoretic machinery whose status may be unclear.
- Filters, formulated as $\lambda$-terms in PTCT, permit the theory to achieve expressive completeness by using Boolean constraints to identify any subset of possible scope readings associated with a sentence.
- Filters also render scope disambiguation efficient, in non-worst case scenarios, by restricting the search space of possible scope permutations.
Further Reading

[Anaphora and Ellipsis in PTCT:] Fox and Lappin
“Doing Natural Language Semantics in An Expressive First-Order Logic with Flexible Typing”

[PTCT theory:] Fox and Lappin
“An Expressive First-Order Logic with Flexible Typing for Natural Language Semantics”
Further Reading

[Underspecification in PTCT:] Fox and Lappin
“Underspecified Interpretations in a Curry-Typed Representation Language”

[Monograph:] Fox and Lappin
*Foundations of Intensional Semantics*
THE END