Expressive Completeness and Computational Efficiency for Underspecified Representations*

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

Shalom Lappin
Dept. of Philosophy
King's College London
shalom.lappin@kcl.ac.uk

4th December 2006

Abstract

Cooper (1983) pioneered underspecified scope representation in formal and computational semantics through his introduction of quantifier storage into Montague semantics as an alternative to the syntactic operation of quantifying-in. In this paper we address an important issue in the development of an adequate formal theory of underspecified semantics. The tension between expressive power and computational tractability poses an acute problem for any such theory. Ebert (2005) shows that any reasonable current treatment of underspecified semantic representation either suffers from expressive incompleteness or produces a combinatorial explosion that is equivalent to generating the full set of possible scope readings in the course of disambiguation. In previous work we have presented an account of underspecified scope representations within Property Theory with Curry Typing (PTCT), an intensional first-order theory for natural language semantics. Here we show how filters applied to the underspecified-scope terms of PTCT permit both expressive completeness and the reduction of computational complexity in a significant class of non-worst case scenarios.

1 Introduction

Cooper (1983) pioneered underspecified scope representation in formal and computational semantics through his introduction of quantifier storage into Montague semantics as an alternative to the syntactic operation of quantifying-in. This work established the basis for a fruitful line of research in underspecified semantics over the past twenty-three years. In this paper we address an important issue in the development of an adequate formal theory of underspecified semantics. We are concerned with achieving expressive completeness in a system for underspecified scope representations in a way that maximizes computational efficiency.

*We are grateful to Christian Ebert for helpful comments on an earlier draft of this paper. His recent Ph.D. dissertation has stimulated and influenced much of the work we report here. We would also like to thank Dick Crouch and Ron Kaplan for useful discussion of the complexity issues we address.

In Fox & Lappin (2005a) we propose Property Theory with Curry Typing (PTCT) as a formal framework for the semantics of natural language. PTCT allows fine-grained distinctions of meaning without recourse to modal notions like (im)possible worlds. It also supports a unified dynamic treatment of pronominal anaphora and VP ellipsis, as well as related phenomena such as gapping and pseudo-gapping.

PTCT consists of three sublanguage components. The first component encodes a property theory within a language of terms (an untyped λ-calculus). The second adds dynamic Curry typing (Curry & Feys, 1958) to provide a system for expressing type judgements for terms. The third uses a first-order logic to specify the truth-conditions of the propositional subpart of the term language. Our semantic representation language is first-order in character, rather than higher-order. We achieve the sort of expressive power previously limited to higher-order theories within a formally more constrained system. This provides an effective procedure for modelling inference in natural language.

Fox & Lappin (2005a,b) use product types to generate underspecified semantic representations within PTCT, the representation language, rather than through meta-language devices, which are invoked in most current treatments of underspecification (Reyle, 1993; Bos, 1995; Blackburn & Bos, 2005; Copestake *et al.*, 2006). The expressive power of the language permits the formulation of filters on scope readings that cannot be captured in other theories of underspecification which rely on special purpose extra-linguistic operations and a weak system for constraint specification.

In Section 2 we summarise the main features of PTCT and our account of underspecified representations. Section 3 is devoted to showing how filters on underspecified scope terms can solve the problem of expressive incompleteness that Ebert (2005) raises for other theories of underspecification. In Section 4 we indicate how filters can be used to reduce the complexity involved in computing the set of possible scope readings that an underspecified term generates. Section 5 compares our account to other approaches to scope ambiguity current in the literature. Finally, in Section 6 we state the main conclusions of this work.

2 PTCT

2.1 Syntax

The core language of PTCT consists of the following sub-languages, where x ranges over a set of variables, c ranges over a set of constants, B is a basic type, and Prop characterises the type of propositions.

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

(2) Types $T ::= B \mid \mathsf{Prop} \mid T_1 \Longrightarrow T_2 \mid X \mid \{x \in T : \varphi'\} \mid \Pi X.T$
where X ranges over types excluding those of the form $\Pi X.T$.

(3) Wff $\varphi ::= \alpha \mid \sim\varphi \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid (\varphi_1 \rightarrow \varphi_2) \mid (\varphi_1 \leftrightarrow \varphi_2)$
$\mid (\forall x\varphi) \mid (\exists x\varphi) \mid (\forall X\varphi) \mid (\exists X\varphi)$
(atomic wff) $\alpha ::= t =_T s \mid t \in T \mid t \cong_T s \mid {}^{\mathsf{T}}t$

PTCT is a first-order theory in which types and propositions are terms over which we can quantify. This allows rich expressiveness whilst restricting the system to first-order resources (Fox & Lappin, 2005a, Chapter 9).

The language of terms is the untyped λ-calculus, enriched with logical constants. It is used to *represent* the interpretations of natural language expressions. It has no internal logic, but when we add a proof theory, the simple language of types together with the language of terms can be combined to produce a Curry-typed λ-calculus.

The syntactic rules of PTCT given here are flexible. They allow the generation of syntactic expressions that have no intuitively meaningful interpretation. This does not undermine the system. The rules give a minimal characterisation of the syntax while our proof theory and our model theory characterise the proper subset of well-formed PTCT terms that constitute meaningful expressions.

In a separation type $\{x \in T : \varphi'\}$, φ' is a term representable fragment of a wff, where term representability can be defined recursively. This restriction on separation types avoids semantic paradoxes of type membership which could otherwise emerge in the specification of these types. The values of bound type variables is limited to non-polymorphic types in order to avoid impredicative type membership statements.

In the first-order language of wffs we formulate type judgements for terms, and truth conditions for those terms judged to be in Prop.

It is important to distinguish between the notion of a proposition itself (in the language of wff), and that of a term that *represents* a proposition (in the language of terms). $\mathsf{T}(t)$ will be a true wff whenever the proposition represented by the term t is true, and a false wff whenever the proposition represented by t is false. The representation of a proposition t ($\in$ Prop) is distinct from its truth conditions ($\mathsf{T}(t)$). The identity criteria for propositions, taken as terms, are those of the λ-calculus with α, β, and η reduction.

We note that if $t \notin$ Prop, then $\mathsf{T}(t)$ will be false. We enforce a strictly bivalent Boolean evaluation in the proof theory and model theory. In principle we could modify this semantics. We might, for example, take the truth value of $\mathsf{T}(t)$ to be undefined when $t \notin$ Prop, whilst preserving Boolean negation (with the "law of excluded middle") for propositions. We will not pursue this issue here.

2.2 Proof Theory

The rules and axioms governing the logical behaviour of PTCT can be summarised as follows. The rules for the basic connectives of the wff have standard classical first-order behaviour. The axioms for identity of terms $=_T$ are those of α, β, and η reduction in the untyped λ-calculus. The rules for typing λ-terms are the rules/axioms of the Curry-typed calculus, augmented with rules governing those terms that represent propositions (Prop). Additional rules for the language of wffs govern the truth conditions of terms in Prop, which represent propositions. Finally, the rules for equivalence $\cong_T$ specify it as the relation of extensional equivalence.

We illustrate some of these rules as they apply to conjunction, as it appears in the language of terms ($\hat{\wedge}$), of type judgements, and of wff ($\wedge$).

(4) The basic connectives of the wff

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi} \wedge i \quad \frac{\varphi \wedge \psi}{\varphi} \wedge e \quad \frac{\varphi \wedge \psi}{\psi} \wedge e$$

(5) Typing rules for λ-terms

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

(6) Truth conditions for Propositions

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

We have encoded the proof theory of PTCT in a tableau system, which we present in Fox & Lappin (2005a, Chapter 5), together with proofs of soundness and completeness. A slightly earlier version of the proof theory appears in Fox & Lappin (2004).

2.3 Equivalence and Identity

There are two equivalence relations in this theory, intensional identity and extensional equivalence. $t \cong_T s$ states that the terms t, s are extensionally equivalent in type T. In the case where two terms t, s are propositions ($t, s \in \mathsf{Prop}$), then $t \cong_{\mathsf{Prop}} s$ corresponds to $t \leftrightarrow s$. In the case where two predicates of T ($t, s \in T \Longrightarrow \mathsf{Prop}$) are extensionally equivalent ($t \cong_{(T \Longrightarrow \mathsf{Prop})} s$), t, s each hold of all and only the same elements of T. Therefore $\forall x(x \in T \rightarrow ({}^{\mathsf{T}}t(x) \leftrightarrow {}^{\mathsf{T}}s(x)))$.

$t =_T s$ states that two terms are intensionally identical in type T. The proof system for PTCT permits us to derive $t =_T s \rightarrow t \cong_T s$ for all types inhabited by t, (s), but not $t \cong_T s \rightarrow t =_T s$. Therefore, two expressions (terms) can be provably equivalent but intensionally distinct. We have achieved this result without recourse to modal notions.

The fact that we can distinguish between equivalence and intensionality permits us to sustain differences in meaning in natural language that elude other intensional logics. The precise definition of equivalence and identity are given by our proof theory and model theory in Fox & Lappin (2005a, Chapter 5). See Fox & Lappin (2004) for a slightly earlier version of these theories.

2.4 Model Theory

We construct our set of models for PTCT on the basis of the model theory for the untyped λ-calculus given in Meyer (1982). $\mathcal{D} = \langle D, [D \rightarrow D], \Phi, \Psi \rangle$ where D is isomorphic to $[D \rightarrow D]$.

(7) (a) D is a non-empty set,
(b) $[D \rightarrow D]$ is some class of functions from D to D,
(c) $\Phi : D \rightarrow [D \rightarrow D]$,
(d) $\Psi : [D \rightarrow D] \rightarrow D$,

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

We can interpret the calculus using the following.

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

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

Note that we require functions of the form $\lambda d.[\![t]\!]_{g[d/x]}$ to be in the class $[D \to D]$ to ensure that the interpretation is well defined. Here we are just following Meyer (1982).

We interpret the types as terms in D that correspond to subsets of D. A model of PTCT is $\mathcal{M} = \langle \mathcal{D}, \mathsf{T}, \mathsf{P}, \mathsf{B}, \mathcal{B}, \mathcal{T}', \mathcal{T} \rangle$, where

(a) $\mathcal{D}$ is a model of the λ-calculus,

(b) $\mathsf{T} : D \to \{0, 1\}$ models the truth predicate $^{\mathsf{T}}$,

(c) $\mathsf{P} \subset D$ models the class of propositions,

(d) $\mathsf{B} \subset D$ models the class of basic individuals,

(e) $\mathcal{B}(\mathsf{B})$ is a set of sets whose elements partition B into equivalence classes of individuals,

(f) $\mathcal{T}' \subset \mathcal{T}$ models the class of non-polymorphic types

(g) $\mathcal{T} \subset D$ models the term representation of types,

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

In Fox & Lappin (2005a) we prove the soundness and completeness of PTCT with respect to the proof theory and model theory specified there.

2.5 Underspecified Representations in PTCT

We extend the type system of PTCT to include product types $S \otimes T$, which have elements of the form $\langle s, t \rangle$. We add the type $S \otimes T$, and a tableau rule corresponding to the following axiom.

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

Unlike monomorphic lists, the k-tuples that instantiate product types allow us to express polymorphic relations.

The appropriate notions of pairs and projections required for product types are λ-definable.

(10) $\langle x, y \rangle =_{\mathsf{def}} \lambda z(z(x)(y))$

(11) $\mathsf{fst} =_{\mathsf{def}} \lambda p(p\lambda xy(x))$

(12) $\mathsf{snd} =_{\mathsf{def}} \lambda p(p\lambda xy(y))$

We write $\langle t_1, t_2, \ldots, t_n\rangle$ for $\langle t_1, \langle t_2, \langle \ldots t_n\rangle\rangle \ldots\rangle$, and $T_1 \otimes T_2 \otimes \ldots \otimes T_n$ for $T_1 \otimes (T_2 \otimes (\ldots \otimes T_n)\ldots)$. We specify that for any k-tuple $\langle t_1, \ldots, t_k\rangle \in T_1 \otimes \ldots \otimes T_k$, the last element of the k-tuple, t_k, is a designated object, like 0 or $\perp$. This condition insures that it is possible to recognise the end of a k-tuple and so compute its arity. The designated element of a k-tuple plays the same role as the empty list does in the tail of every list. It renders the elements of product types equivalent to weak lists with elements of (possibly) distinct types. As in the case of lists, we generally suppress this final designated element when representing a k-tuple.

2.6 Generalised Quantifiers

Generalised quantifiers (GQs) represent noun phrases. We follow Keenan (1992) and van Eijck (2003) in taking a GQ to be an arity reduction operator that applies to a relation r to yield either a proposition or a relation r' that is produced by effectively saturating one of r's argument with the GQ.[1] On this view, applying the GQ corresponding to "*every student*" ($\mathsf{every\_student}'$ or $\lambda Q\forall x(\mathsf{student}' \rightarrow Q(x))$) to the binary relation $\lambda yx(\mathsf{loves}'(x,y))$ gives the one-place relation $\lambda x(\mathsf{every\_student}'(\lambda y.\mathsf{loves}'(x,y)))$. Through β-reduction this gives $\lambda x(\forall y(\mathsf{student}'(y) \rightarrow \mathsf{love}'(x,y)))$, which is the property of loving every student.

GQs are of type $(X \Longrightarrow \mathsf{Prop}) \Longrightarrow \mathsf{Prop}$, which we write Quant^X for clarity (where X is typically B). Core propositional relations, such as verbs, are of type $X_1 \Longrightarrow \ldots \Longrightarrow X_n \Longrightarrow \mathsf{Prop}$. Slightly modifying van Eijck's Haskell-based treatment of GQs (van Eijck, 2003), we define an operator R to "lift" quantifiers to the appropriate level to combine with a relation.

(13) $R \in \mathsf{Quant}^X \Longrightarrow ((X \Longrightarrow T) \Longrightarrow T)$

(14) $Q \in \mathsf{Quant}^X \wedge r \in (X \Longrightarrow \mathsf{Prop}) \rightarrow RQr = Qr$

(15) $Q \in \mathsf{Quant}^X \wedge r \in (X \Longrightarrow T) \wedge (T \notin \mathsf{Prop}) \rightarrow RQr = \lambda xRQ(rx)$

We compose representations of n quantifiers with a relation r using

$$RQ_1(RQ_2 \ldots (RQ_n r)\ldots)$$

2.7 Indexed Permutations of GQ Scope Sequences

Natural language is ambiguous with respect to the scoping of quantifiers, modifiers, conjunction, and negation. Many of these scopings are purely semantic in nature. So, for example the following sentence (16) allows two alternative scope readings, given in (17) and (18).

(16) Every man loves a woman

[1] In Keenan's presentation, some generalised quantifiers can bind more that one of r's arguments, and so reduce its arity by more than 1. These GQ are formed from constituent quantifiers that exhibit relations of mutual dependence. Due to these relations, the GQ which they yield cannot be reduced to a simple functional composition of one quantifier with another. An example of such a GQ is $\langle$"*every student*", "*a different book*"$\rangle$ in "*Every student read a different book.*"

(17) $\forall x(\mathsf{man}'(x) \rightarrow \exists y(\mathsf{woman}'(y) \wedge \mathsf{loves}'(x,y)))$

(18) $\exists y(\mathsf{woman}'(y) \wedge \forall x(\mathsf{man}'(x) \rightarrow \mathsf{loves}'(x,y)))$

We want our theory to produce "*underspecified*" representations that subsume all the various readings, and from which the different readings can be generated. We can express computable functions in PTCT, and so we can incorporate the machinery of underspecified semantics directly into the representation language.

We specify a family of functions $perms\_scope_k$ (where $k > 1$) that generate all $k!$ indexed permutation products of a k-ary indexed product term $\langle t_1, \ldots, t_k \rangle$ as part of the procedure for generating the set of possible scope readings of a sentence. In Fox & Lappin (2005b) we specify a standard algorithm (following Campbell, 2004) for mapping a k-tuple $\langle 1, \ldots, k \rangle$ into the indexed $k!$-tuple of its permutations as part of the interpretation of $perms\_scope_k$. In Section 4 we formulate an alternative tree construction algorithm to generate the set of all possible permutations of scope taking elements to which $perms\_scope_k$ applies. We will use the factorial permutation trees that this algorithm generates to demonstrate the complexity reduction that filters on underspecified representations can achieve.

For our treatment of underspecification, $perms\_scope_k$ needs to take a k-ary product of scope taking elements (by default, in the order in which they appear in the surface syntax) and a k-ary relation representing the core proposition as its arguments. The scope taking elements and the core representation can be combined into a single product, e.g. as a pair consisting of the k-tuples of quantifiers as its first element and the core relation as its second. The permutation function $perms\_scope_k$ produces the $k!$-ary product of scoped readings. When a k-tuple of quantifiers is permuted, the λ-operators that bind the quantified argument positions in the core relation are effectively permuted in the same order as the quantifiers in the k-tuple. This correspondence is necessary to preserve the connection between each GQ and its argument position in the core relation across scope permutations.

A scope reading is generated by applying the elements of the k-tuple of quantifiers in sequence to the core proposition, reducing its arity with each such operation until a proposition results. The ith scope reading is identified by projecting the ith element of the indexed product of propositions that is the output by our $perms\_scope_k$ function. The PTCT term consisting of the application of $perms\_scope_k$ to an input pair of a k-tuple of GQs and a core relation therefore provides an underspecified representation of the sentence corresponding to this term. Below we describe a function that projects a fully specified scope reading. In principle we can follow van Eijck (2003) and give a uniform type to these representations by defining arbitrary arity product types to cover the type of the k-tuple of GQs that is the first element in the pair to which $perms\_scope_k$ applies and the $k!$-tuple which is its value.

Consider example (16) "*Every man loves a woman*". The GQs interpreting the subject NP, the object NP and the core relation are given in (19), (20) and (21), respectively, and the PTCT term expressing the underspecified representation of the sentence is given in (22).

(19) $Q_1 = \lambda P \hat{\forall} x \epsilon B(\mathsf{man}'(x) \hat{\rightarrow} P(x))$

(20) $Q_2 = \lambda Q \hat{\exists} y \epsilon B(\mathsf{woman}'(y) \hat{\wedge} Q(y))$

(21) $\lambda uv.\mathsf{loves}'uv$

(22) $perms\_scope_2(\langle\langle Q_1, Q_2\rangle, \lambda uv.\mathsf{loves}'uv)\rangle$

The permutations of the quantifiers and the core representation that we produce are

(23) $\langle\langle\langle Q_1, Q_2\rangle, \lambda uv.\mathsf{loves}'uv\rangle\langle\langle Q_2, Q_1\rangle, \lambda vu.\mathsf{loves}'uv\rangle\rangle$

Applying relation reduction to computing the final propositions gives us a product containing the two readings.

(24) $perms\_scope_2(\langle\langle Q_1, Q_2\rangle, \lambda uv.\mathsf{loves}'uv)\rangle) =$
$\langle\hat{\forall}x\epsilon B(\mathsf{man}'(x) \hat{\rightarrow} \hat{\exists}y\epsilon B(\mathsf{woman}'(y) \hat{\wedge} \mathsf{loves}'(x,y))),$
$\hat{\exists}y\epsilon B(\mathsf{woman}'(y) \hat{\wedge} \hat{\forall}x\epsilon B(\mathsf{man}'(x) \hat{\wedge} \mathsf{loves}'(x,y)))\rangle$

To obtain resolved scope readings from an underspecified representation, we define a family of functions $project\_scope_k(i)$ that compute the ith permutation of a k-ary product of propositions. Specifically, a function of this kind returns the ith proposition in the product of scope readings that $perms\_scope_k$ gives as its value. We extend the type system to include the type Num of natural numbers. We can then define $project\_scope_k$'s type as

$$\langle \mathsf{Prop}_1, \ldots, \mathsf{Prop}_k\rangle \Rightarrow \mathsf{Num} \Rightarrow \mathsf{Prop}$$

where $\mathsf{Num} \leq k$. To ensure that the function is total, we can define $project\_scope_k(i)$ so that it projects the (i mod k)th term, for example. A detailed proposal for the inclusion of natural numbers into PTCT is provided in Fox & Lappin (2005a, Chapter 6).

3 Filters and Expressive Completeness

There are various kinds of constraints that limit the set of possible scope readings for a particular sentence to a proper subset of the set of $k!$ orderings of the k scope taking elements which appear in it. A common condition on relative scope is the strong preference for wide scope assignment to certain quantifiers by virtue of their lexical semantic properties, like "*a certain N′*".

A second kind of condition depends upon the syntactic domain in which a GQ appears. So, for example, a quantified NP within a relative clause cannot take scope over a quantified NP in which the relative clause is embedded.

The following two examples illustrate these constraints.

(25) Every critic reviewed a certain book.

(26) A student who completed every assignment came first in the class.

The strongly preferred reading of (25) is the one on which "*a certain book*" takes wide scope relative to "*every critic*". In (26) "*every assignment*" can only take narrow scope relative to "*a student who completed every assignment*".

Scope constraints of these kinds can be formulated as filters on the $k!$-tuple of permutations $\langle\langle Qtuple_1, Rel_1\rangle, \ldots, \langle Qtuple_{k!}, Rel_{k!}\rangle\rangle$ that $perms\_scope_k$ generates for an argument pair $\langle Qtuple_1, Rel_1\rangle$. Each such filter is a Boolean property function that imposes a condition on the elements of the $k!$-tuple.[2]

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. We take $a\_certain$ to be a PTCT property that is true of all and only GQs that represent "*a certain N′*", and is false of anything else. 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 ith member of $Quants$, and the distinguished term ω otherwise.

We can specify the lexical scope constraint illustrated in (25) as the filter in (27), where i and j are variables ranging over integers (type Num).

(27) $\lambda\langle Quants, Rel\rangle[\hat{\sim}(\hat{\exists} i \epsilon \mathsf{Num} \hat{\exists} j \epsilon \mathsf{Num}(a\_certain(tuple\_element(i, Quants)) \hat{\wedge}$
$\hat{\sim} a\_certain(tuple\_element(j, Quants)) \hat{\wedge}$
$j \hat{<} i))]$

This condition requires that no element of a $k!$-tuple of scope readings contains a k-tuple of GQs in which the index of a $a\_certain$ GQ is higher than that of a non-$a\_certain$ GQ (and so outscoped by it). Notice that we have only quantified over integers (elements of the type Num) in this filter. We have taken advantage of the isomorphism between k-tuples of integers and k-tuples of indexed GQs to avoid quantifying over GQ expressions. Therefore, we have remained within the first-order expressive resources of PTCT.[3]

In order to formulate the condition illustrated in (26) we must introduce syntactic relations. Let $relcl\_embed(Q_1, Q_2)$ hold iff the NP corresponding to Q_2 appears in a relative clause contained in the NP corresponding to Q_1. We can formulate the constraint as in (28).

(28) $\lambda\langle Quants, Rel\rangle[\hat{\sim}(\hat{\exists} i \epsilon \mathsf{Num} \hat{\exists} j \epsilon \mathsf{Num}(relcl\_embed(tuple\_element(i, Quants)$
$tuple\_element(j, Quants))$
$\hat{\wedge}\ j \hat{<} i))]$

This filter prevents a GQ that interprets an NP in a relative clause from having scope over a GQ that interprets an NP in which the relative clause is embedded.

(27) and (28) achieve partial disambiguation of an underspecified representation to which they apply (non-vacuously) by ruling out a subset of the set of possible scope readings that this representation generates independently of the filters.

Underspecified representations can also be disambiguated by information acquired through subsequent discourse. So, for example, resolving anaphoric expressions like pronouns and definite descriptions in sentences following a statement that exhibits scope ambiguity may eliminate certain readings of the antecedent.

(29) (a) *Speaker 1:* Every student wrote a program for some professor.

(b) *Speaker 2:* Yes, I know the professor. She taught the Haskell course.

[2] See van Eijck (2003) for examples of filters on lists specified as Boolean functions on the elements of a list.

[3] In Fox & Lappin (2005a, Chapter 6) we formulate a version of Num using Presburger arithmetic (Presburger, 1929), so avoiding a commitment to the full power of Peano arithmetic.

(c) *Speaker 3:* I saw the programs, and they were all list-sorting procedures.

Identifying "*some professor*" in (29a) as the antecedent for "*the professor*" and "*she*" in (29b) gives "*some professor*" scope over "*every student*" in (29a). Interpreting "*a program*" in (29a) as the antecedent for "*the programs*" and "*they*" in (29c) causes "*a program*" to have narrow scope relative to "*every student*" in (29a). Therefore, taken conjointly (29b) and (29c) forces on (29a) the fully resolved scope order

$$\langle \text{“}\textit{some professor}\text{”}, \text{“}\textit{every student}\text{”}, \text{“}\textit{a program}\text{”} \rangle$$

Let "*every student*" $= Q_1$, "*a program*" $= Q_2$, and "*some professor*" $= Q_3$. We can formulate the filters contributed by (29b) and (29c) as (30) and (31), respectively (where GQ in $\hat{=}_{GQ}$ abbreviates the appropriate type of Q_i).

(30) $\lambda\langle Quants, Rel\rangle[\hat{\forall} i \epsilon \mathsf{Num} \hat{\forall} j \epsilon \mathsf{Num}((tuple\_element(i, Quants) \hat{=}_{GQ} Q_3 \hat{\wedge}$
$tuple\_element(j, Quants) \hat{=}_{GQ} Q_1) \hat{\rightarrow}$
$i \hat{<} j)]$

(31) $\lambda\langle Quants, Rel\rangle[\hat{\forall} i \epsilon \mathsf{Num} \hat{\forall} j \epsilon \mathsf{Num}((tuple\_element(i, Quants) \hat{=}_{GQ} Q_1 \hat{\wedge}$
$tuple\_element(j, Quants) \hat{=}_{GQ} Q_2) \hat{\rightarrow}$
$i \hat{<} j)]$

We specify the function *filter_tuple*$(\langle F, T\rangle)$ which 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.[4] We construct a PTCT term of the form (32) to represent the k'-tuple obtained by applying the elements of F to the $k!$-tuple that is the value of $perms\_scope_k(\langle Quants_k, Rel\rangle)$.

(32) $filter\_tuple(\langle F, perms\_scope_k(\langle Quants_k, Rel\rangle)\rangle)$

Ebert (2005) shows that most current theories of underspecification are expressively incomplete to the extent that they cannot identify the proper subset of possible scope readings specified by Boolean operations other than conjunction, and in particular by negation. He cites the following example to illustrate the problem.

(33) Every market manager showed five sales representatives a sample.

Ebert stipulates that, in his example, real world knowledge allows all scope permutations except the one corresponding to $\langle \exists, 5, \forall \rangle$, where *a sample* takes wide scope, *five sales representatives* intermediary position, and *every market manager* narrow scope. He demonstrates that storage (Cooper, 1983; Pereira, 1990), hole semantics (Bos, 1995; Blackburn & Bos, 2005), Minimal Recursion Semantics (Copestake *et al.*, 2006), and Normal Dominance Conditions (Koller *et al.*, 2003) cannot formulate underspecified representations that express the set containing only the five remaining scope readings.

By contrast it is straightforward to formulate a filter in PTCT that rules out the problematic scope sequence in Ebert's case while permitting the five other readings.

[4] In fact, this will be a family of functions $filter\_tuple_{j,k}(\langle F_j, T_k\rangle)$. In the interests of simplicity we will suppress the j and k indices on *filter_tuple* in the text.

(34) $\lambda\langle Quants, Rel\rangle[\hat{\forall} i \epsilon \mathsf{Num} \hat{\forall} j \epsilon \mathsf{Num} \hat{\forall} k \epsilon \mathsf{Num}((tuple\_element(i, Quants) \hat{=}_{GQ} Q_{\exists} \hat{\wedge}$
$tuple\_element(j, Quants) \hat{=}_{GQ} Q_5 \hat{\wedge}$
$tuple\_element(k, Quants) \hat{=}_{GQ} Q_{\forall}) \hat{\rightarrow}$
$\hat{\sim}(i \hat{<} j \hat{\wedge} j \hat{<} k))]$

PTCT is, in principle, able to achieve expressive completeness in Ebert's (2005) sense.[5]

4 Efficient Computation of Possible Scope Readings

At first glance it might seem that it is, in general, necessary to generate the full k!-tuple that is the value of $perms\_scope_k(\langle Quants_k, Rel\rangle)$ before applying the filters of F to the elements of this k!-tuple in order to compute the value of (32). If this were true, filters would never reduce the search space of possible scope readings that must be accessed in the course of their application. Fortunately, this is not the case.

In Fox & Lappin (2005b) we specify an algorithm based on Campbell (2004) for generating the indexed list of all possible permutations of an input list. This procedure was used to partially characterise the computable function $perms\_scope_k$. It is possible to use an alternative algorithm to implement this function, where the indexed k!-tuple of possible permutations of an initial k-tuple is obtained through the construction of a tree.

(35) (a) Given a k-tuple $\langle Q_1, \ldots Q_k\rangle$, a tree is generated breadth first, starting by creating the root of the tree, then producing successive levels, continuing until level k is generated, as follows.

Base Case Take the tuple $\langle Q_1\rangle$ consisting of the initial element of the k-tuple $\langle Q_1, \ldots, Q_k\rangle$ to be the root of the tree. Let this be level 1 of the tree.

Recursive Case Level $(i + 1)$ of the tree is created from level i by considering each node n_m of level i in turn (starting with the left-most node n_1, and continuing to the right-most node), and constructing all the daughters of each node n_m as follows.

Base Case′ Construct the left-most daughter d_1 of the current node n_m by adding the $(i+1)$th tuple in which the $(i+1)$th element of $\langle Q_1, \ldots, Q_k\rangle$ is concatenated with the i-tuple at n_m in the right-most position of the $(i + 1)$-tuple at d_1.

Recursive Case′ Obtain the daughter d_{j+1} of the current node n_m immediately to the right of d_j by moving the $(i + 1)$th element of $\langle Q_1, \ldots, Q_k\rangle$, added at level $(i + 1)$, one place to left.
In this way, construct all daughters of n_m until the right-most daughter is generated as the $(i + 1)$-tuple in which the $(i + 1)$th element of $\langle Q_1, \ldots, Q_k\rangle$ appears in the left-most position.

The tree is finished when the kth level has been generated.

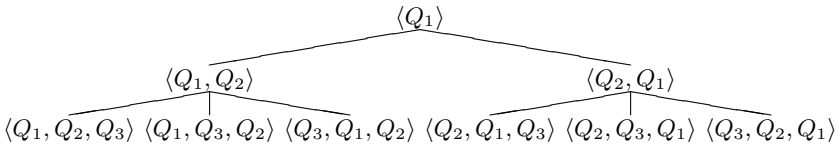
[5] Ebert (2005) observes that to actually arrive at expressive completeness it is necessary to extend PTCT to deal with nested quantificational structures, like the subject NP in "*Two representatives from three companies saw most samples*". He sketches a proposal for doing this.

(b) To obtain the $k!$-tuple of all possible permutations of $\langle Q_1, \ldots, Q_k \rangle$ concatenate the k-tuples at the leaves of the tree from left to right into a $k!$-tuple.

(c) Indexing: assign each k-tuple element of the $k!$-tuple an index i, starting with 1, in the the left-to-right order in which they appear as leaves of the finished tree.

If this algorithm takes as its input the triple $\langle Q_1, Q_2, Q_3 \rangle$, then it generates the following tree.

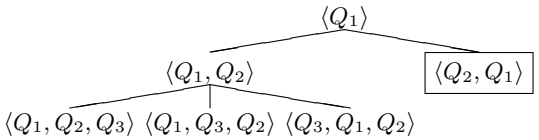
(36)



Filters can apply as constraints to nodes in the tree as the algorithm produces them. If a node violates a filter, then it is deleted, and the subtree that it dominates is not generated. In this way filters can reduce the size of the tree, and so limit the search space of possible scope readings explored for a $perms\_scope_k(\langle Quants_k, Rel \rangle)$ term to a proper subset of the elements of the $k!$-tuple that is its value.

So, for example, the filter $Q_1 < Q_2$ prunes the tree in (36) to give the one in (37).

(37)



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

(38) $|T| = \Sigma_{i=1}^{k} i!$, where i is the index of the ith element of the initial k-tuple which the algorithm takes as its input.

Therefore, the size of the tree in (36) is $1! + 2! + 3! = 9$. The size of the tree in (37) 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

(39) $|ST| = \Pi_{j=i+1}^{k} j + \Sigma_{j'=i+1}^{k-1} j'$

Consider the quadruple $\langle Q_1, Q_2, Q_3, Q_4 \rangle$. The algorithm in (35) produces 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 $\langle Q_2, Q_1 \rangle$, and so it eliminates a subtree of 15 nodes, reducing T_4 by $15/33 = 45.4\%$. 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. If the filter $Q_2 < Q_3$ applies at this level, the 8 leaf nodes under $\langle Q_1, Q_3, Q_2 \rangle$ and $\langle Q_3, Q_1, Q_2 \rangle$ are pruned. 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.

(40) (a) *Speaker 1:* It's amazing. A critic recently reviewed two plays for every newspaper in a major city.

(b) *Speaker 2:* Yes, I wonder how he got away with that. He published the same reviews of the current productions of "*A Midsummer Night's Dream*" and "*New-Found-Land*" in every major paper in New York last week.

Clearly, the earlier in the tree construction process (the higher up in the tree) that a filter applies, the greater the reduction in search space of possible scope readings that it achieves. It is also 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 point in the generation of the tree. For example, if the algorithm takes as its input the triple $\langle Q_1, Q_2, Q_3 \rangle$ and one of the filters that apply to this triple is $Q_2 < Q_3$, then the reordering operation will map the triple into $\langle Q_2, Q_3, Q_1 \rangle$. We will leave the formulation of this operation for future work.

Ebert (2005) proves a theorem that 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 for PTCT in the limit case, where no filters have been applied to a $perms\_scope_k(\langle Quants_k, Rel \rangle)$ term, or they do not operate early enough in the tree construction algorithm to restrict the scope permutation tree. However, as we have seen, there is a large class of cases in which filters significantly reduce the search space through tree pruning, and so they offer a mechanism for rendering scope disambiguation computationally efficient.

5 Other Treatments of Scope Ambiguity

5.1 Quantifier Storage

Quantifier storage as defined in Cooper (1983) and Pereira (1990) is perhaps the first system for generating non-compositional underspecified scope representations. A generalised quantifier (GQ) is stored as the first element of a pair whose second element identifies the variable that is used to mark the GQ's argument position in the syntactic structure of the sentence. The representation produced for the clause consists of the core propositional relation and a set of stored GQ pairs. When a GQ is discharged from storage, it is applied to the core relation, binding the variable in its original position. As the elements of the storage set are unordered, they can be discharged in any sequence, where each sequence yields a possible scope reading.

Storage provides an elegant and straightforward way of generating underspecified scope representations for a sentence. However, there are (at least) three difficulties with this approach. First, storage is an additional mechanism defined outside of the semantic representation language as such. The expressions that it produces are not themselves part of this language (a typed λ-calculus) but stages in the derivation of well-formed terms of the representation language. While storage is easily implemented

in a declarative fashion, as in Pereira (1990) and Blackburn & Bos (2005), it remains an essentially procedural device that is added to a compositional semantic theory as a means of obtaining scope ambiguity without attaching alternative scope readings to distinct syntactic structures, as in Montague (1974).

By contrast, on our account underspecified representations are themselves terms of PTCT, the representation language. Therefore, this issue does not arise; the underspecified representation is expressed directly in the representation language.

Second, because storage is a mechanism constructed outside of the representation language, it is necessary to specify an additional constraint language for stating the Boolean conditions required to restrict the set of possible scope readings derived from the storage set.[6]

Again, this problem does not arise on our treatment of underspecification. The filters that express constraints on scope readings are λ-terms of PTCT, and so the resources required for the formulation of these constraints are available within the representation language.

Finally, even if a constraint language is added to storage, it will be necessary to specify an additional mechanism to prevent the generation of the full set of possible scope orderings prior to the application of the filters.

As we have seen in Section 4, when $perms\_scope_k$ is interpreted as a computable function whose application involves the tree generation algorithm in (35), filters can be acquired incrementally in discourse and applied to underspecified representations in PTCT in a straightforward way that can significantly reduce the search space of possible scope readings.

5.2 Hole Semantics, Minimal Recursion Semantics, and Normal Dominance Constraints

Bos (1995), and Blackburn & Bos (2005) develop a constraint-based system for underspecified representation for first-order logic that they refer to as *Predicate Logic Unplugged* (PLU). This system is a generalisation of the *hole semantics* approach to underspecification which Reyle (1993) first developed within the framework of Underspecified Discourse Representation Theory. Copestake *et al.*'s (2006) Minimal Recursion Semantics is an application of hole semantics within a typed feature structure grammar (HPSG). Koller *et al.*'s (2003) Normal Dominance Conditions can be seen as a refinement and development of the central ideas of hole semantics. The problems that we identify with the hole semantics model apply to all three theories, and so, in the interests of simplicity, we will summarise (a version of) PLU as the representative of this approach.[7]

An underspecified representation of a quantified first-order formula in PLU is an ordered tripe $\langle LH, F, R\rangle$. LH is a set of labels for formulas and of holes, which are

[6]Keller (1988) defines a type of storage that encodes relations of syntactic nesting within the stored GQ corresponding to an NP that contains another quantified NP. Although these nested stores avoid certain problems of variable binding encountered with Cooper storage, they do not, in themselves, impose constraints on possible scope readings of the sort that we have discussed in the previous section. See Blackburn & Bos (2005) for a discussion and an implementation of Keller stores.

[7]See Ebert (2005) for detailed discussion and results concerning the formal relations among these theories with respect to their expressive power.

(essentially) metavariables that take formulas as values. F is a set of labelled formulas, which may contain holes for subformulas. R is a set of scope constraints expressed as partial order relations on labels and holes. The PLU representation of (41) is (42).

(41) Every student wrote a program.

(42) $\langle \{l_1, l_2, l_3, h_0, h_1, h_2\},$
$\{l_1 : \forall x(\mathsf{student}'(x) \rightarrow h_1), l_2 : \exists y(\mathsf{program}'(y) \wedge h_2), l_3 : \mathsf{wrote}'(x, y)\},$
$\{l_1 \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2\}\rangle$

The partial ordering constraints in (42) define a bounded lattice with h_0 as $\top$, the propositional core of the formula, l_3 as $\bot$, and l_1 and l_2 as midpoints of the lattice between $\top$ and $\bot$. As l_1 and l_2 are not ordered with respect to each other, either formula can be substituted for the hole in the other formula. l_3 must be substituted last in the remaining hole. If l_1 is taken as the value of h_0, l_2 is substituted for h_1, and then l_3 is substituted for h_2, the result is a wide scope reading of the universal quantifier, as in (43). Alternatively, if l_2 is taken as the value of h_0, l_1 is assigned to h_2, and l_3 to h_1, we obtain (44).

(43) $\forall x(\mathsf{student}'(x) \rightarrow \exists y(\mathsf{program}'(y) \wedge \mathsf{wrote}'(x, y)))$

(44) $\exists y(\mathsf{program}'(y) \wedge \forall x(\mathsf{student}'(x) \rightarrow \mathsf{wrote}'(x, y)))$

These are the only two scope resolutions that satisfy the partial order conditions in (42).

Hole semantics provides a more expressive and flexible system for constructing underspecified representations than storage. It generalises naturally to scope elements other than GQs, like negation and modifiers. It is possible to identify a subset of scope readings that satisfy the constraints of an underspecified hole semantic representation by imposing a particular order of substitution of labels for holes in a schematic formula set. However, it does suffer from the first and second difficulties which we raised against storage. Underspecified representations are constructed out of metavariables, schematic formulas, and partial ordering statements in a metalanguage that is distinct from the semantic representation language. The substitutions of labelled formulas for holes that generate the well-formed formulas of the representation language which correspond to scope readings are also metalinguistic operations added to the representation language.

More seriously, as we have observed, Ebert (2005) shows that PLU and other hole semantics theories are expressively incomplete because their constraint languages do not permit the formulation of Boolean conditions on scope like those given in (27), (28), and (34). As in the case of storage, it is possible to add a constraint language with sufficient expressive power required to state conditions of this kind.[8] But this requires further enrichment and complication of the theory. As we have seen, these problems do not arise on our account.

[8]We are grateful to Ian Pratt-Hartman for helpful discussion of this point.

5.3 Glue Language Semantics and Packed Scope Representations

Dalrymple *et al.* (1999) and Crouch & van Genabith (1999) suggest a theory on which representations of GQs and core relations are expressed as premises in an underspecified semantic glue language. These premises are combined by the natural deduction rules of linear logic in order to yield a formula that represents the scope reading of a sentence. The rules can apply to premises in different orders of derivation to generate alternative scope readings. Unlike PLU, the glue language can be higher-order. Although their formal properties differ, glue language semantics is closely related to hole semantics in the general view of underspecification that it adopts. It would seem that in order achieve expressive completeness in the sense of Ebert (2005), glue language semantics must add a system for stating constraints on the linear logic proof theory which it employs to derive fully specified interpretations.

Crouch (2005) describes a procedure for using the linear logic derivations of glue language semantics to generate all scoped interpretations for a sentence. These interpretations are encoded 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 a graph for non-redundant representation of the full set of possible syntactic structures for a parsed phrase. In this system the choice space of Boolean combinations of clauses in a packed representation that are to be tested for satisfiability is optimised using Maxwell & Kaplan's (1995) method for rendering disjunctive constraint satisfaction efficient.

Packing offers an 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 of a sentence. 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. In PTCT filters can be extracted from new discourse information, where some of these filters significantly reduce the search space of interpretations.

5.4 Relation Reduction

van Eijck (2003) develops an approach to underspecified representations, in the functional programming language Haskell, which uses relation reduction and arbitrary arity relations. This inspired our account, which we have developed within a more restrictive formal theory.

We give a fully general treatment of scope and generalise van Eijck's approach in certain respects. In particular, we introduce a function for selecting specific scope readings, and we make explicit the mechanisms for constraining scope readings using filters. Our approach to underspecification is also polymorphic, which leaves open the possibility of dealing with core relations whose arguments are of different types.

We developed PTCT to have a rich system of types, broadly comparable to that of Haskell, but within a language that we have shown to be of more restricted formal power.

6 Conclusion

We have presented a treatment of underspecified scope representation within PTCT which uses product types to represent sequences of scope taking terms. These types permit us to accommodate polymorphism in the core relation arguments.

We have characterised an underspecified representation as a PTCT term in which a function $perms\_scope_k$ applies to a pair containing an initial sequence of scope taking elements and a core relation. It returns as its value an indexed $k!$-product of possible scope readings. $project\_scope(perms\_scope_k(\langle Qs, R\rangle), i)$ projects the ith scope reading in the $k!$-tuple of scope readings $perms\_scope_k(\langle Qs, R\rangle)$.

We have formulated constraints on scope readings as filters on the $k!$-tuples that $perms\_scope_k$ produces. These filters are PTCT property terms which encode Boolean conditions and quantification over the integers of indexed k-tuples. In principle, they permit PTCT to achieve expressive completeness in the sense of Ebert (2005).

We have also specified a tree generation algorithm to characterise (the permutation part of) the computable function that $perms\_scope_k$ denotes. When filters are applied as constraints on nodes in the tree that the algorithm generates, they can significantly reduce the search space of possible scope readings given by an underspecified representation.

Underspecified representations, the projection of a particular scope interpretation, and constraints on possible scope readings are all specified by appropriately typed λ-terms within the semantic representation language, PTCT, rather than through operations on schematic metalinguistic objects. Our proposed treatment of underspecified representations within PTCT achieves both significant expressive power and and efficient computation of possible scope interpretations.

References

Blackburn, P. & J. Bos (2005), *Representation and Inference for Natural Language*, CSLI, Stanford.

Bos, J. (1995), Predicate logic unplugged, in *Proceedings of the Tenth Amsterdam Colloquium*, Amsterdam, Holland.

Campbell, W. H. (2004), Indexing permutations, *Journal of Computing in Small Colleges* 19:296–300.

Cooper, Robin (1983), *Quantification and Syntactic Theory*, Synthese Language Library, D. Reidel, Dordrecht.

Copestake, A., D. Flickinger, & I. A. Sag (2006), Minimal recursion semantics, *Research on Language and Computation* 3:281–332.

Crouch, D. (2005), Packed rewriting for mapping semantics to KR, in *Proceedings of the Sixth International Workshop on Computational Semantics*, Tilburg, (103–114).

Crouch, D. & J. van Genabith (1999), Context change, underspecification, and structure of glue language derivations, in M. Dalrymple (ed.), *Semantics and Syntax in Lexical Functional Grammar*, MIT, Cambridge, MA, (117–189).

Curry, H. B. & R. Feys (1958), *Combinatory Logic*, volume 1 of *Studies in Logic*, North Holland.

Dalrymple, M., J. Lamping, F. Pereira, & V. Saraswat (1999), Quantification, anaphora, and intensionality, in M. Dalrymple (ed.), *Semantics and Syntax in Lexical Functional Grammar*, MIT, Cambridge, MA, (39–89).

Ebert, C. (2005), *Formal Investigation of Underspecified Representations*, Ph.D. thesis, Department of Computer Science, King's College London, unpublished.

van Eijck, J. (2003), *Computational Semantics and Type Theory*, unpublished ms., CWI, Amsterdam.

Fox, C. & S. Lappin (2004), An expressive first-order logic with flexible typing for natural language semantics, *Logic Journal of the Interest Group in Pure and Applied Logics* 12(2):135–168.

Fox, C. & S. Lappin (2005a), *Formal Foundations of Intensional Semantics*, Blackwell, Oxford.

Fox, C. & S. Lappin (2005b), Underspecified interpretations in a Curry-typed representation language, *The Journal of Logic and Computation* 15:131–143.

Keenan, E. (1992), Beyond the Fregean boundary, *Linguistics and Philosophy* 15:199–221.

Keller, W. (1988), Nested cooper storage: The proper treatment of quantification in ordinary noun phrases, in U. Reyle & C. Rohrer (eds.), *Natural Language Parsing and Linguistic Theories*, Reidel, Dordrecht.

Koller, A., J. Niehren, & S. Thater (2003), Bridging the gap between underspecified formalisms: Hole semantics as dominance constraints, in *Proceedings of 11th EACL*, Budapest.

Maxwell, J. & R. Kaplan (1995), A method for disjunctive constraint satisfaction, in M. Dalrymple, R. Kaplan, J. Maxwell, & A. Zaenen (eds.), *Formal Issues in Lexical Functional Grammar*, CSLI, Stanford, CA.

Meyer, A. (1982), What is a model of the lambda calculus?, *Information and Control* 52:87–122.

Montague, R. (1974), *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, CT/London, UK, edited with an introduction by R. H. Thomason.

Pereira, F. (1990), Categorial semantics and scoping, *Computational Linguistics* 16:1–10.

Presburger, Mojzesz (1929), Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt, in *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves, Warszawa*, (92–101).

Reyle, U. (1993), Dealing with ambiguities by underspecification: Construction, representation and deduction, *Journal of Semantics* 10:123–179.