Expressiveness and Complexity in Underspecified Semantics*

Chris Fox and Shalom Lappin
University of Essex and King's College London

Abstract

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. Generating the full set of resolved scope readings from an underspecified representation produces a combinatorial explosion that undermines the efficiency of these representations. Moreover, Ebert (2005) shows that most current theories of underspecified semantic representation suffer from expressive incompleteness. In previous work we present an account of underspecified scope representations within Property Theory with Curry Typing (PTCT), an intensional first-order theory for natural language semantics. We review this account, and we show that filters applied to the underspecified-scope terms of PTCT permit expressive completeness. While they do not solve the general complexity problem, they do significantly reduce the search space for computing the full set of resolved scope readings in non-worst cases. We explore the role of filters in achieving expressive completeness, and their relationship to the complexity involved in producing full interpretations from underspecified representations.

*We dedicate this paper to Jim Lambek, whose pioneering contributions to mathematical linguistics have inspired generations of researchers in the theory of formal grammar.

Earlier versions of this paper were presented at The Symposium on Logic and Language, Debrecen, 2004; The Amsterdam Colloquium, 2005; the Computer Science Colloquium, University of Sheffield, 2006, the Computer Science Colloquium, University of Haifa, 2006; the Linguistics Colloquium, Yale University, 2007; the Dialogue Matters Workshop, King's College, London, 2008; the Computational Linguistics Colloquium, IRIT, Toulouse, 2008; and the Computational Linguistics Colloquium of the School of Informatics, University of Edinburgh, 2008. We are grateful to the participants of these forums for helpful comments and advice. We would also like to thank Christian Ebert, Richard Crouch, Josef van Genabith, Ron Kaplan, Alexander Koller, Nati Linial, Massimo Poesio, and Ian Pratt-Hartmann for invaluable discussion of the complexity issues discussed here. We bear sole responsibility for the ideas presented in this paper and any mistakes that it may contain.

Linguistic Analysis, 36 1-4

P.O. Box 2237, Vashon, WA 98070

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-five 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 maximises computational efficiency.

We will not discuss the syntax-semantics interface but focus on properties of the semantic representation language. The semantic representations that we use can be mapped onto, and built up compositionally in tandem with, the syntactic structures provided by most contemporary theories of formal grammar. Our semantic representations are particularly commensurate with the syntactic structures of a categorial grammar, as both are terms of a λ-calculus. However, our general framework could easily be adapted to other syntactic systems.

In Fox and 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 and 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 and Lappin (2005a,b) use product types to generate under-

specified 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 and 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. Section 4 discusses the general complexity problem posed by underspecified scope representations and explores three initially attractive strategies for dealing with it. None of these turns out to be viable. In Section 5 we indicate how filters can be used to reduce the search space involved in computing the set of possible scope readings that an underspecified term generates. Section 6 compares our account to other approaches to scope ambiguity current in the literature. Finally, in Section 7 we state the main conclusions of this work.

2. PTCT

In this section we provide an overview of PTCT and our account of underspecified scope representations within this framework. Scope-taking elements occur in a variety of syntactic categories, which include NPs, quantified adverbial phrases, negation, tense, and prepositional phrases. When the interaction of expressions from all of these categories is taken into account, the degree of scope ambiguity exhibited by even relatively simple sentences can be quite high. However, for ease of exposition we limit ourselves here to the generalised quantifiers used to interpret NPs.

2.1. Syntax

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 and Lappin, 2005a, Chapter 9).

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 p characterises the type of propositions.

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

(2) Types
$T ::= \mathsf{e} \mid \mathsf{p} \mid \langle T_1, T_2 \rangle \mid X \mid \{x \in T : \varphi'\} \mid \Pi X.T$
where X ranges over types excluding those of the form $\Pi X.T$.

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

The language of terms is the untyped λ-calculus, enriched with logical constants. It is used to *represent* the interpretations of natural language expressions. For this reason, it includes terms to represent the usual logical connectives, together with type membership $\hat{\in} T$, identity $\hat{=}_T$ and equivalence $\hat{\cong}_T$. 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 terms 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.

All the types are term representable. 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 p.

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 ($\in \mathsf{p}$) 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 $t \notin \mathsf{p}$, 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 $t \notin \mathsf{p}$, 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 (p). Additional rules for the language of wffs govern the truth conditions of terms in p, 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

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

(5) Typing rules for λ-terms

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

(6) Truth conditions for Propositions

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

We have encoded the proof theory of PTCT in a tableau system, which we present in Fox and Lappin (2005a, Chapter 5), together with proofs of soundness and completeness. A slightly earlier version of the proof theory appears in Fox and 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 ($t, s \in \mathsf{p}$), then $t \cong_{\mathsf{p}} s$ corresponds to $t \leftrightarrow s$. In the case where two predicates of T ($t, s \in \langle T, \mathsf{p}\rangle$) are extensionally equivalent ($t \cong_{\langle T,\mathsf{p}\rangle} s$), 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 and Lappin (2005a, Chapter 5). Again, Fox and Lappin (2004) has 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). $\mathcal{D} = (D, [D \rightarrow D], \Phi, \Psi)$ where D is isomorphic to $[D \rightarrow D]$.

(7) a. D is a non-empty set,

b. $[D \to D]$ is some class of functions from D to D,

c. $\Phi : D \to [D \to D]$,

d. $\Psi : [D \to D] \to D$,

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 $\mathcal{M} = (\mathcal{D}, \mathsf{T}, \mathsf{P}, \mathsf{B}, \mathcal{B}, \mathcal{T}', \mathcal{T})$, 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 and 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 (s, t). We add the type $S \otimes T$, and a tableau rule corresponding to the following axiom.

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

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

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

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

We write $(t_1, t_2, \ldots, t_n)$ for $(t_1, (t_2, (\ldots t_n))\ldots)$, and $T_1 \otimes T_2 \otimes \ldots \otimes T_n$ for $T_1 \otimes (T_2 \otimes (\ldots \otimes T_n)\ldots)$. We will typically require that the last element t_k of the k-tuple $(t_1, \ldots, t_k) \in T_1 \otimes \ldots \otimes T_k$, is a designated object, like 0 or $\perp$. This condition insures that it is possible to recognise the final element 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 and Unger (2010) 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*" (every_student′ or in

[1]In Keenan's presentation, some generalised quantifiers can bind more than one of r's arguments, and so reduce its arity by more than 1. These GQs 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 ("*every student*", "*a different book*") in "*Every student read a different book.*"

a conventional logical notation, $\lambda Q \forall x(\mathsf{student}'(x) \rightarrow Q(x)))$ 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 $\langle\langle X, \mathsf{p}\rangle, \mathsf{p}\rangle$, which we write Quant^X for clarity (where X is typically B). Core propositional relations, such as verbs, are of type $\langle X_1, \langle \ldots, \langle X_n, \mathsf{p}\rangle \ldots\rangle\rangle$. Slightly modifying van Eijck and Unger's Haskell-based treatment of GQs (van Eijck and Unger, 2010), we define an operator R recursively to "lift" quantifiers to the appropriate level to combine with a relation.

(13) a. $R \in \langle\langle \mathsf{Quant}^X, \langle X, T\rangle\rangle, T\rangle)$

b. $Q \in \mathsf{Quant}^X \wedge r \in \langle X, \mathsf{p}\rangle \rightarrow RQr = Qr$

c. $Q \in \mathsf{Quant}^X \wedge r \in \langle X, T\rangle \wedge (T \notin \mathsf{p}) \rightarrow RQr = \lambda xRQ(rx)$

We use relation-reduction to compose representations of n quantifiers $Q_1 \ldots Q_n$ with an n-place relation r by applying R as follows.

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

2.7. Indexed Permutations of GQ Scope Sequences

Many of the scope ambiguities involving GQs are most naturally treated as purely semantic in nature. So, as has been often observed, the following well-worn example (15) allows two alternative scope readings, represented in a conventional logical notation in (16) and (17).

(15) Every man loves a woman

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

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

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 $(t_1, \ldots, t_k)$ as part of the procedure for generating the set of possible scope readings of a sentence. In Fox and Lappin (2005b) we specify a standard algorithm (following Campbell, 2004) for mapping a k-tuple $(1, \ldots, k)$ into the indexed $k!$-tuple of its permutations as part of the interpretation of $perms\_scope_k$. In Section 5 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 non-worst case scenario search-space 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, for example 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 produced by the $perms\_scope_k$ function. The PTCT term consisting of the application of $perms\_scope_k$ to an appropriate input pair — 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 and Unger (2010) 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 (15) "*Every man loves a woman*". The GQs interpreting the subject NP, the object NP and the core relation are given as PTCT terms in (18), (19) and (20), respectively, and the PTCT term expressing the underspecified representation of the sentence is given in (21).

(18) $Q_1 = \lambda P \hat{\forall} x \hat{\in} \mathsf{e}(\mathsf{man}'(x) \hat{\rightarrow} P(x))$

(19) $Q_2 = \lambda Q \hat{\exists} y \hat{\in} \mathsf{e}(\mathsf{woman}'(y) \hat{\wedge} Q(y))$

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

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

The permutations of the quantifiers and the core representation produced by (21) are given by the following.

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

Applying relation-reduction (13 & 14) to each of the representations of the scope orderings gives a pair of propositions corresponding to the two readings.

(23) $(\hat{\forall} x \hat{\in} \mathsf{e}(\mathsf{man}'(x) \hat{\rightarrow} \hat{\exists} y \hat{\in} \mathsf{e}(\mathsf{woman}'(y) \hat{\wedge} \mathsf{loves}'(x, y))),$
$\hat{\exists} y \hat{\in} \mathsf{e}(\mathsf{woman}'(y) \hat{\wedge} \hat{\forall} x \hat{\in} \mathsf{e}(\mathsf{man}'(x) \rightarrow \mathsf{loves}'(x, y))))$

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. In the event that we apply $project\_scope_k$ after relation-reduction, we can define its type as

(24) $\langle(\mathsf{p}_1, \ldots, \mathsf{p}_k), \langle \mathsf{Num}, \mathsf{p}\rangle\rangle$

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 and 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, as is the case with "*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 $((Qtuple_1, Rel_1), \ldots, (Qtuple_{k!}, Rel_{k!}))$ that $perms\_scope_k$ generates from $(Qtuple_1, Rel_1)$, the initial argument pair. Each such filter is a Boolean property function that imposes a condition on the elements of the $k!$-tuple.[2]

Let $(Quants, Rel)$ be a pair of variables in which *Quants* ranges over k-tuples, and *Rel* over k-ary relations. 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).

[2]See van Eijck and Unger (2010) for examples of filters on lists specified as Boolean functions on the elements of a list.

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

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. In principle this allows the filter to be stated within the expressive resources of any first-order language that supports natural numbers.[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) $\lambda(Quants, Rel)[$
$\hat{\sim}(\hat{\exists} i, j \ \hat{\in} \ \mathsf{Num}(relcl\_embed(tuple\_element(i, Quants)$
$tuple\_element(j, Quants))$
$\hat{\wedge} \ j \ \hat{<} \ i))]$

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.

Filters (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.[4]

[3] In Fox and 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. In the case of PTCT, however, we would remain within first-order expressibility even if we were to quantify over the term representations of GQ expressions.

[4] Koller and Niehren (2000) propose a procedure for using anaphora resolution to

(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.

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 both "*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) force on (29a) the fully resolved scope order

("*some professor*", "*every student*", "*a program*")

To simplify the presentation: let Q_1 represent "*every student*"; Q_2, "*a program*"; and Q_3, "*some professor*". 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) $\lambda(Quants, Rel)[$
$\hat{\forall} i, j \,\hat{\in}\, \mathsf{Num}((tuple\_element(i, Quants) \,\hat{=}_{GQ}\, Q_3 \,\hat{\wedge}$
$tuple\_element(j, Quants) \,\hat{=}_{GQ}\, Q_1) \,\hat{\rightarrow}$
$i \,\hat{<}\, j)]$

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

We specify the function $filter\_tuple(F, T)$ 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.[5] 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 $perms\_scope_k(Quants_k, Rel)$.

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

filter possible scope readings within a dominance constraint system for underspecified representation.

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

Expressive Completeness

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 and disjunction. 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, contextual information allows all scope permutations except the one corresponding to $(\exists, 5, \forall)$, where "*a sample*" takes wide scope, "*five sales representatives*" has an intermediate position, and "*every market manager*" takes narrow scope. He demonstrates that storage (Cooper, 1983; Pereira, 1990), hole semantics (Bos, 1995; Blackburn and 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.

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

Ebert shows that in order to achieve expressive completeness, a theory of underspecified semantic representation must be able to characterize the power set of $k!$ possible readings generated by a sentence with k interacting scope operators. PTCT is, in principle, able to achieve expressive completeness in this sense.[6] This is due

[6]Ebert (2005) makes the important observation 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 in a straightforward way.

to the fact that the terms of the λ-calculus that encode both underspecified scope representations and filters are interpreted by the type and propositional components of PTCT as having the full power of typed quantification and Boolean operations. It is important to recall, that they are part of the semantic representation language itself, and so no enrichment of this language or recourse to an external metalanguage is required to achieve expressive completeness. Moreover, these terms denote computable functions, and they remain expressions in a first-order system.

4. The Complexity Problem for Underspecified Scope Representations

4.1. The General Complexity Issue

We have already observed that generating the full set of possible resolved scope readings for a sentence with k interacting scope elements requires at least $k!$ steps. What happens when filters are added to the representation language? Assume that the set of constraints in the language is initially limited to simple linear precedence conditions of the form $Q_i < Q_j$. We will refer to such constraints as *weak filters*. When disjunction is added to the set of weak filters, then the problem of deciding whether an arbitrary reading satisfies these filters is NP-complete. It is not possible to design an algorithm for testing satisfaction of a set of filters in this theory in less than exponential time for worst cases.[7]

It is straightforward to show that this complexity result holds for underspecified scope terms with filters in PTCT, where the filters are formulated only in terms of linear precedence relations and Boolean connectives (and the quantification over indices is eliminable through reference to specified elements of a k-tuple). The λ-terms that we use to formulate such filters correspond to propositions containing Boolean connectives and state precedence conditions on scope-taking elements. These conditions can be translated into Conjunctive Normal Form (CNF), which is a conjunction of propositions, each of which is a disjunction of literals (elementary

[7]See Koller *et al.* (1998) and Duchier (2003) for results showing that the satisfiability of unrestricted dominance constraints, and dominance constraints with Boolean connectives, respectively, is NP-complete.

propositional variables or negations of them). For filters in which the number of propositional variables is k and $k \geq 3$, the satisfiability problem is NP-complete. This is the k-SAT satisfiability result.[8]

In fact, the general complexity problem for underspecified representations is even more severe than the NP-completeness result for filters limited to weak filters, conjunction, and disjunction. Let us assume that our underspecified representations can only express filters as conjunctions of weak filters, without disjunction. Such a system is expressively incomplete because, as Ebert has shown, it cannot identify the full power set of the set of possible scope readings for any given sentence, and so some scope readings are not expressible.

An underspecified representation with weak filters defines a partially ordered set (a poset) of scope operators. A fully resolved scope representation that satisfies a poset is a linear extension of this set. Brightwell and Winkler (1991) show that computing the number of linear extensions of a poset is a #P-complete problem. where #P-completeness is a degree of intractability that exceeds NP-completeness.[9] If computing the number of linear extensions of a poset is #P-complete, then generating the set of its linear extensions is, in the general case, at least exponential and so intractable. If one cannot enumerate the set of linear extensions of an arbitrary poset in polynomial time, then one cannot generate this set in polynomial time either. Therefore, no algorithm can generate the set of possible scope readings for underspecified scope representations containing only weak filters, in less than exponential time in worst cases. This result holds despite the fact that the language is expressively incomplete in Ebert's sense.

One might try to avoid the complexity problem by arguing that it is a feature of performance, and so it is not of concern to semantic theory. In fact, this manoeuvre is seriously misconceived. Underspecified semantic representations are motivated by a concern to achieve computational efficiency. They were introduced precisely in order to avoid the processing complexity involved in identifying all possible readings that can be assigned to a scope ambiguous sentence.

But any account of semantic underspecification is responsible for explaining how the representations that it provides yield the mean-

[8]We are grateful to Alexander Koller and Nati Linial for helpful discussion on this point.

[9]See Papadimitriou (1994) for the concepts of NP- and #P-completeness, and a general introduction to complexity theory.

ings that speakers associate with ambiguous sentences in discourse and dialogue. If the account does not allow these meanings to be effectively computed, then it does not offer an adequate model of how speakers are able to interpret scope ambiguous sentences. A credible theory of ambiguity cannot escape the demand for computational viability. Seeking refuge in the competence-performance distinction is a misapplication of a technique for theoretical idealisation in order to claim exemption from the question that the theory was designed to answer.

4.2. Three Non-Strategies for Dealing with the Complexity Problem

4.2.1. Underspecified Logic

One possibility is to simply avoid interpreting terms that encode underspecified scope readings, and develop a logic in which the rules of inference apply directly to underspecified premises. König and Reyle (1999) propose an underspecified logic of this kind. Using a logic that specifies valid inferences with underspecified premises and conclusions permits one to reason with underspecified representations without resolving their scope ambiguities. One could then treat the interpretation of an underspecified representation as given by the set of entailments that it supports in such a logic.

However, there are cases in which the validity of an inference depends upon a specified scope reading of the premises.

(35) a. *Every student* in the class did *not* show up yesterday.

b. No student in the class showed up yesterday.

The statement (35b) can only be inferred from the premise (35a) on the reading in which "*every student*" is given scope over "*not*".

It is also not obvious that the interpretation of underspecified representations can be entirely reduced to the set of inferences that they license. In at least some instances one needs to assign a resolved interpretation to a sentence in order to interpret it in dialogue. Imagine, for example, the following exchange on a quiz show.

(36) A: Two Toronto Maple Leafs players participated in every World Hockey Championship final for the past ten years.

B: Right.

A: Can you tell me which ones?

In order to respond to A's question in (36) B must select a relative scope ordering for "*Two Toronto Maple Leafs players*", "*every World Hockey Championship finals*", and "*for the past ten years*". Specifically, he/she must decide whether to seek a single pair of Leafs players who participated in the past ten World Hockey Championship finals, or distribute distinct pairs of Leafs players over the set of ten final games.

Finally, the task of constructing a viable logic encoding the set of entailments of underspecified representations of propositions is daunting. Such a logic must be at least sound, and completeness is a desirable property. It is not clear how to construct such a system, and the few attempts that have been made to sketch one have not yielded a full logic.

4.2.2. Approximation Algorithms

A second strategy is to construct a polynomial-time approximation algorithm for computing the full set of linear extensions of an underspecified representation within reasonable limits of correctness and likelihood. Polynomial-time randomised approximation algorithms compute a value for a given input to within a specified range of error, to a high degree of probability. They have been proposed for some #P-complete problems, like Jerrum *et al.* (2004)'s algorithm for computing the *permanent* (the number of perfect matchings) of a matrix with non-negative entries.

Constructing such a procedure to identify a reliable approximation of the set of linear extensions for an underspecified scope representation constrained by weak filters is a non-trivial task. For the algorithm to be useful we would need to show that the approximation-set of resolved scope readings that it returns for any underspecified term includes reasonable candidates for the interpretation of that term. In the absence of both an efficient approximation algorithm and a criterion for determining whether an approximation-set of linear extensions offers a viable interpretation of an underspecified scope term, this strategy remains problematic.

It is even less plausible for a system which allows non-weak filters containing the full range of Boolean connectives. In this case an approximation algorithm would have to approximate a set of

possible posets, and, for each one of these, a set of linear extensions, all in at most polynomial time.

4.2.3. A Single Candidate Scope Interpretation

A third possibility is to limit ourselves to underspecified scope terms with weak filters, and, for each such term, to identify a single linear extension for the poset that it specifies.

It is possible to compute a verifying linear extension of a poset in linear time as the upper complexity bound. A directed acyclic graph (DAG) encodes a poset. A generalised topological sorting procedure for a DAG yields an ordering of its edges that is equivalent to a linear extension of the poset to which it corresponds. Hagerup and Mass (1993) propose a topological sorting algorithm that yields a solution for a DAG D in $O(n + m)$ time, where n is the number of vertices in D, and m is the number of its edges.[10]

The Hagerup-Mass algorithm also decides whether a DAG has any topological sorting solution, in $O(n + m)$ time. Therefore, given a weakly filtered underspecified scope term T, it shows that determining the consistency of the filters of T is a tractable problem. In this context, T is consistent iff only if it defines a poset, and so has a linear extension. When T is represented as a DAG, the algorithm will decide the truth or falsity of the right side of this biconditional in linear time.

Moreover, Hagerup and Mass (1993) provide a second algorithm, defined in terms of the first, which decides whether a topological sorting solution for a DAG is unique, and it also runs in $O(n + m)$ time. So for any weakly filtered underspecified scope term T, we can in linear time, (i) decide whether T is consistent, (ii) if it is, compute a verifying resolved scope interpretation $\mathcal{I}$ for T, and (iii) determine if $\mathcal{I}$ is the only possible reading generated by T.

On this approach a hearer in discourse or dialogue uses a weakly filtered underspecified scope term to generate a candidate interpretation for a sentence. He/she can then recompute this interpretation in

[10]Hagerup and Mass (1993)'s algorithm generates solutions for the constrained version of the generalised topological sorting problem, where an integer corresponding to the number of vertices in a DAG is specified. Identifying a resolved scope reading for a weakly filtered underspecified scope term can be formulated as a constrained topological sorting problem for the DAG that expresses the poset defined by the term. The integer of the constrained problem would be the number of scope-taking elements in the poset.

light of new information that alters the filter set that applies to the term.

As attractive as this strategy may appear at first sight, it suffers from the restriction of underspecified scope terms to constraints that can be expressed as weak filters. Therefore, we achieve computational efficiency at the expense of expressive completeness, which undermines the viability of the approach.

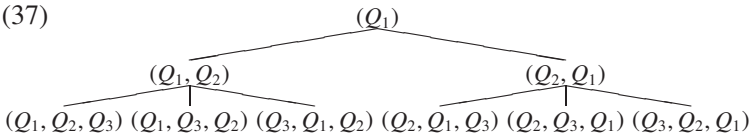
5. Using Filters to Reduce the Search Space for Possible Scope Readings

When filters contain Boolean connectives like disjunction or negation they add to the complexity involved in computing the set of possible scope readings that an underspecified scope term defines. However, weak filters can significantly reduce the search space, and hence the complexity of this problem in non-worst cases.

At first glance it might seem that it is, in general, necessary to generate the full $k!$-tuple given by $perms\_scope_k(Quants_k, Rel_k)$ 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. In fact, this is not the case.

In Fox and 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, given in Figure 1, where the indexed $k!$-tuple of possible permutations of an initial k-tuple is obtained through the construction of a tree. If this algorithm takes as its input the triple (Q_1, Q_2, Q_3), then it generates the following tree.

(37)



Weak 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,

(a) Given a k-tuple $(Q_1, \ldots Q_k)$, 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 (Q_1) consisting of the initial element of the k-tuple $(Q_1, \ldots, Q_k)$ 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 $(Q_1, \ldots, Q_k)$ 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 $(Q_1, \ldots, Q_k)$, added at level $(i + 1)$, one place to the 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 $(Q_1, \ldots, Q_k)$ appears in the left-most position.

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

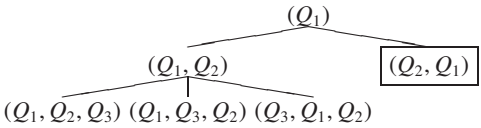
(b) To obtain the $k!$-tuple of all possible permutations of $(Q_1, \ldots, Q_k)$ concatenate the k-tuples at the leaves of the tree from left to right into a $k!$-tuple.

(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.

Figure 1: Tree Construction Algorithm.

and the subtree that it dominates is not generated. In this way weak filters can reduce the size of the tree, and so limit the search space of possible scope readings explored for a $perms\_scope_k(Quants_k, Rel_k)$ 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 (37) to give the one in (38).

(38)



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

(39) $|T| = \sum_{i=1}^{k} i!$, 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 (37) is $1! + 2! + 3! = 9$. The size of the tree in (38) is 6, which is a reduction of 30%.

The size of a subtree ST_n dominated by a node n, but not including n, is given by the formula

(40) $|ST_n| = \sum_{j=i+1}^{k} \frac{j!}{i!}$, where i is the level in the tree at which n occurs.[11]

Consider the quadruple (Q_1, Q_2, Q_3, Q_4). The algorithm in Figure 1 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 (Q_2, Q_1), 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 (Q_1, Q_2, Q_3), (Q_1, Q_3, Q_2), (Q_3, Q_1, Q_2) at level 3. If the filter $Q_2 < Q_3$ applies at this level, the 8 leaf nodes under (Q_1, Q_3, Q_2) and (Q_3, Q_1, Q_2) 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.

[11] We are grateful to Christian Ebert for supplying us with this formula.

(41) 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 (Q_1, Q_2, Q_3) and one of the filters that apply to this triple is $Q_2 < Q_3$, then the reordering operation will map the triple into (Q_2, Q_3, Q_1). We will leave the formulation of this operation for future work.

6. Other Treatments of Scope Ambiguity

6.1. 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) two 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 and 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.[12] Without the addition of this constraint language, storage suffers from expressive incompleteness in Ebert (2005)'s sense.

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.

6.2. Hole Semantics, Minimal Recursion Semantics, and Normal Dominance Constraints

Bos (1995), and Blackburn and 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).

[12] 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 and Bos (2005) for a discussion and an implementation of Keller stores.

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.[13]

An underspecified representation of a quantified first-order formula in PLU is an ordered tripe (LH, F, R). LH is a set of labels for formulas and of holes, which are (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 (42) is (43).

(42) Every student wrote a program.

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

The partial ordering constraints in (43) 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 (44). 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 (45).

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

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

These are the only two scope resolutions that satisfy the partial order conditions in (43).

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

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

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 suffers from the two 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.[14] But this requires further enrichment and complication of the theory. As we have seen, these problems do not arise on our account.

6.3. Normal Dominance Constraints and Filtering

Koller and Thater (2006) describe an algorithm for identifying equivalence classes of readings for underspecified scope representations encoded in Normal Dominance charts, which are compact representations of Normal Dominance graphs. The algorithm relies on local permutation relations between scope operators in the graphs. It eliminates redundant elements from each equivalence class. Koller and Thater (2006) show that their algorithm drastically reduces the set of scope readings for the dominance graphs of sentences in the Rondane treebank for an English corpus, while running in time polynomial on the size of the dominance graph.

This algorithm offers a very useful filtering procedure for reducing the search space for scope interpretations in corpora. Its constraints could be incorporated into PTCT as weak filters that impose unique precedence conditions on locally adjacent scope-taking elements that are permutable under semantic equivalence.

Such filters, like those discussed in Section 5, do not solve the general complexity problem. In worst case scenarios no scope op-

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

erators in a sentence are permutable under semantic equivalence, and so there is no redundancy to eliminate. Moreover, as the set of theorems for first-order logic is not decidable, semantic equivalence among quantified sentences in a first-order representation language cannot be effectively tested.

Koller *et al.* (2008) use regular tree grammars (RTGs) to encode Normal Dominance Charts. They present an algorithm for the elimination of redundancy through the reduction of semantic equivalence classes that improves on the coverage and performance of the procedure suggested in Koller and Thater (2006). They also observe that, as RTGs can generate any finite subset of possible scope readings, they provide an underspecified representation system that can achieve expressive completeness.

It is important to recall that Normal Dominance charts and the filtering algorithms that apply to them are generated outside of the semantic representation language through operations on the expressions of this language. By contrast the underspecified scope terms in PTCT and their filters are lambda terms of the representation language.

6.4. Glue Language Semantics and Packed Scope Representations

Dalrymple *et al.* (1999) and Crouch and 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, 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 and 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. Therefore it makes no attempt to avoid the complexity problem even in cases where filtering would greatly reduce the search space of possible scope readings.

On the approach that we are suggesting, the interpretation of underspecified scope terms would be delayed as long as possible in a discourse or dialogue to permit the accumulation of a maximal set of filters, which would, in many instances, greatly reduce the set of possible scope interpretations.

6.5. Relation Reduction

van Eijck and Unger (2010) develop an approach to underspecified representations, in the functional programming language Haskell, which uses relation-reduction and arbitrary arity relations. This inspired important elements of our account, which we have developed within a more restrictive formal theory.

We give a fully general treatment of scope and generalise van Eijck and Unger'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.

7. 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. $project\_scope_k(perms\_scope_k(Q_k, R_k), i)$ projects the ith scope reading in the $k!$-tuple of the scope readings generated by $perms\_scope_k(Q_k, R_k)$.[15]

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 specified a tree generation algorithm to characterise (the permutation part of) the computable function that $perms\_scope_k$ denotes. When weak 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. While they do not solve the general complexity problem of underspecified scope readings, they do permit a significant improvement in efficiency for non-worst cases. In this respect, computing the interpretation of filtered underspecified scope representations is analogous to theorem proving in propositional logic. In both cases the general problem is intractable (NP complete), but efficient computation in non-worst cases is possible.

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 the possibility of relative efficiency.

[15] Putting aside the details of relation-reduction.

Works Cited

1. Blackburn, P. and J. Bos. 2005. *Representation and Inference for Natural Language*. Stanford: CSLI.
2. Bos, J. 1995. Predicate Logic Unplugged. In *Proceedings of the Tenth Amsterdam Colloquium*. Amsterdam, Holland.
3. Brightwell, G. and P. Winkler. 1991. Counting Linear Extensions is #P-Complete. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, 175–181. New Orleans, LA.
4. Campbell, W. H. 2004. Indexing Permutations. *Journal of Computing in Small Colleges* 19: 296–300.
5. Cooper, Robin. 1983. *Quantification and Syntactic Theory*. Synthese Language Library. Dordrecht: D. Reidel.
6. Copestake, A., D. Flickinger, C. Pollard, and I. A. Sag. 2006. Minimal Recursion Semantics. *Research on Language and Computation* 3: 281–332.
7. Crouch, D. 2005. Packed Rewriting for Mapping Semantics to KR. In *Proceedings of the Sixth International Workshop on Computational Semantics*, 103–114. Tilburg.
8. Crouch, D. and J. van Genabith. 1999. Context Change, Underspecification, and Structure of Glue Language Derivations. In *Semantics and Syntax in Lexical Functional Grammar*, edited by M. Dalrymple, 117–189. Cambridge, MA: MIT.
9. Curry, H. B. and R. Feys. 1958. *Combinatory Logic*, *Studies in Logic*, vol. 1. North Holland.
10. Dalrymple, M., J. Lamping, F. Pereira, and V. Saraswat. 1999. Quantification, Anaphora, and Intensionality. In *Semantics and Syntax in Lexical Functional Grammar*, edited by M. Dalrymple, 39–89. Cambridge, MA: MIT.
11. Duchier, D. 2003. Dominance Constraints with Boolean Connectives: A Model Eliminative Treatment. *Theoretical Computer Science* 293: 321–343.
12. Ebert, C. 2005. *Formal Investigation of Underspecified Representations*. Ph.D. thesis, Department of Computer Science, King's College London. Unpublished.
13. van Eijck, J. and C. Unger. 2010. *Computational Semantics with Functional Programming*. Cambridge: Cambridge Univesity Press.
14. Fox, C. and 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.

15. —. 2005a. *Foundations of Intensional Semantics*. Oxford: Blackwell.
16. —. 2005b. Underspecified Interpretations in a Curry-Typed Representation Language. *The Journal of Logic and Computation* 15: 131–143.
17. Hagerup, T. and M. Mass. 1993. Generalized Topological Sorting in Linear Time. In *Fundamentals of Computing Theory*, 279–288. Berlin and New York: Springer.
18. Jerrum, M., A. Sinclair, and E. Vigoda. 2004. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Nonnegative Entries. *Journal of the ACM* 52: 671–697.
19. Keenan, E. 1992. Beyond the Fregean Boundary. *Linguistics and Philosophy* 15: 199–221.
20. Keller, W. 1988. Nested Cooper Storage: The Proper Treatment of Quantification in Ordinary Noun Phrases. In *Natural Language Parsing and Linguistic Theories*, edited by U. Reyle and C. Rohrer. Dordrecht: Reidel.
21. Koller, A. and J. Niehren. 2000. On Underspecified Processing of Dynamic Semantics. In *Proceedings of COLING 18*, 460–466. Saarbrücken: International Conference On Computational Linguistics.
22. Koller, A., J. Niehren, and S. Thater. 2003. Bridging the Gap between Underspecified Formalisms: Hole Semantics as Dominance Constraints. In *Proceedings of 11th EACL*. Budapest.
23. Koller, A., J. Niehren, and R. Treinen. 1998. Dominance Constraints: Algorithms and Complexity. In *Selected Papers from the Third International Conference on Logical Aspects of Computational Linguistics*, 106–125. Berlin: LNCS, Springer.
24. Koller, A., M. Regneri, and S. Thater. 2008. Regular Tree Grammars as a Formalism for Scope Underspecification. In *Proceedings the 46th Annual Meeting of the ACL*. Columbus, OH.
25. Koller, A. and S. Thater. 2006. An Improved Redundancy Elimination Algorithm for Underspecified Representations. In *Proceedings of COLING 21 and the 44th Annual Meeting of the ACL*, 409–416. Sydney, Australia.
26. König, E. and U. Reyle. 1999. A General Reasoning Scheme for

Underspecified Representations. In *Logic, Language, and Reasoning: Essays in Honour of Dov Gabbay*, edited by Ohlbach H.J and U. Reyle. Kluwer.

27. Maxwell, J. and R. Kaplan. 1995. A Method for Disjunctive Constraint Satisfaction. In *Formal Issues in Lexical Functional Grammar*, edited by M. Dalrymple, R. Kaplan, J. Maxwell, and A. Zaenen. Stanford, CA: CSLI.
28. Meyer, A. 1982. What is a model of the lambda calculus? *Information and Control* 52: 87–122.
29. Montague, R. 1974. *Formal Philosophy: Selected Papers of Richard Montague*. New Haven, CT/London, UK: Yale University Press. Edited with an introduction by R. H. Thomason.
30. Papadimitriou, C., ed. 1994. *Computational Complexity*. Reading, MA: Addison-Wesley.
31. Pereira, F. 1990. Categorial Semantics and Scoping. *Computational Linguistics* 16: 1–10.
32. Presburger, M. 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.
33. Reyle, U. 1993. Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics* 10: 123–179.