

COVERT AND OVERT MOVEMENT IN LOGICAL GRAMMAR

Carl Pollard
INRIA-Lorraine and Ohio State University

ESSLLI 2008 Workshop on Symmetric Calculi
and Ludics for Semantic Interpretation
Hamburg, August 7, 2008

1 Introduction

(1) The Reason for this Paper

- Much of the current interest in applying **continuations** to linguistics is centered on issues about **scope** of quantified NPs, *wh*-expressions, and the like.
- But the Extended Montague Grammar (EMG) of the 1970s already had good theories about these, though not always presented in the best light.
- My goal here is to tell the best EMG-style story I can about these things, as a kind of benchmark, so we can get clearer about what is gained by “continuizing”.
- In this paper I focus mostly on **wh-questions**, and next week (in the What Syntax Feeds Semantics Workshop) on some **ellipsis** and **comparative** constructions.
- The paper is about Chinese, but this talk deals with English questions, because they involve both overt and covert movement.

(2) Extended Montague Grammar (EMG)

- EMG emerged in the 1970s as an alternative to the then-current avatar of TG, Chomsky’s Revised Extended Standard Theory (REST).
- EMG sought greater simplicity, precision, and tractability.
- EMG included practitioners of:
 - PSG, e.g. Cooper, Gazdar, Pullum
 - CG, e.g. Dowty
 - switch hitters, e.g. Bach.
- CG (basically AB grammars) was not much different from PSG; Lambek’s (1958) calculus didn’t catch on with linguists until the mid-1980’s.

(3) Two Signal Achievements of EMG

- The most obvious difference between EMG and REST was that EMG eschewed **movement**, both **overt** (e.g. *Wh*-Movement ‘at Syntax’) and **covert** (e.g. Quantifier Raising (QR)).
- Two of the most significant achievements of EMG:
 - Cooper’s (1975) **storage** replaced **covert** movement.
 - Gazdar’s (1979) **linking schemata** replaced **overt** movement.
- Proof-theoretically, these two devices are almost identical.

(4) The 1980s and Beyond

- REST evolved into GB, and then into the MP, where movement is still the central explanatory device.
- Since Chomsky 1993, movement has been viewed as copying, and the covert/overt distinction as a question of which copy is audible.
- EMG spawned such frameworks as CCG, HPSG, TLG, Pregroup Grammars, ACG, etc.
- In spite of the multitude of important contributions of these frameworks, none of them quite capture the simplicity of the intuitions behind Cooper storage and the Gazdar schemata.

2 Convergent Grammar (CVG)

2.1 CVG Overview

(5) What is CVG?

- CVG is yet another post-EMG framework..
- It seeks to synthesize from the best practices of CG and PSG an approach to syntax and semantics that is simple enough to be used as a research framework by actual **linguists**, not just computational linguists and mathematical logicians.
- Logical reformulations of Cooper storage and the Gazdar schemata play a central role.
- CVG closely resembles both ACG and HPSG.

(6) CVG Compared with ACG and HPSG

- Like both ACG and HPSG, CVG has a **parallel** architecture: independent components generate candidate phonological, syntactic, and semantic entities.
- Like ACG, these candidate entities are **proofs**, each in a different logic. We call this **pure derivationality** as opposed to TG’s structural derivationality in TG that builds arboreal representations by sequences of structural operations.
- Like HPSG, the relation between syntax and semantics need **not** be a function: the same syntactic derivation can correspond to two or more distinct semantic derivations.
- This nonfunctionality of the syntax-semantics interface arises from the use of a **generalized form of Cooper storage**.

(7) Syntax, Semantics, and their Interface in CVG

- Candidate syntactic derivations are specified by a **syntactic logic** similar to ones used in CG, **plus** a (proof-theoretic version of) a **Gazdar-style linking schema**.
- Candidate semantic derivations are specified by a **semantic logic** similar to lambda calculus, but with abstraction replaced by a (proof-theoretic version of) **Cooper storage**.
- The **interface** specifies which pairs of derivations go together.

2.2 The CVG Syntactic Logic

(8) Format for CVG Syntactic Typing Judgments

$\Gamma \vdash a : A$

- ‘Term a is assigned category A in context Γ .’
- This is in the **Gentzen-sequent style of natural deduction** with **Curry-Howard proof terms**.
- The **context** is the (nonrepeating) list of **typed syntactic variables**, also called **traces**, that have been bound.
- Contexts work essentially like the HPSG SLASH.
- HPSGs are really natural-deduction systems, a fact which is obscured by the feature-structure encoding.

(9) CVG Categories

- There are some **basic** categories, such as S, NP, and N
- For present purposes we ignore morphosyntactic details such as case, agreement, verb inflection (the kinds of details handled in HPSG by **head features**). In a more detailed CVG, these would be handled by **subtyping**.
- If A and B are categories, then so are $A \multimap_F B$, where F belongs to a finite set F of **grammatical function names**. These are called **functional** categories with **argument** category A and **result** category B .
- If A , B , and C are categories, then so is $G[A, B, C]$, usually abbreviated to A_B^C . These are called **operator** categories with **binding** category A , **scope** category B , and **result** category C .

(10) Functional Categories

- We start off with the grammatical function names (**gramfun**s) s (subject) and c (complement). Others can be added as needed.
- The gramfun
s correspond to HPSG **valence features**.- Since the grammatical functions aren't relevant for this talk, you can just think \multimap_s as Lambek's \backslash and \multimap_c as Lambek's $/$.
- Example: a transitive verb has category $NP \multimap_c NP \multimap_s S$.
- There is poetic justice here, since HPSG's valence features originated as a feature-structure encoding of CG category cancellation.

(11) Syntactic Operator Categories

- These are for expressions analyzed by TG as having overtly moved.
- They correspond to **fillers** in HPSG, which in turn go back to Gazdar's (1979) analysis of **unbounded dependencies**, hence the name G for the category constructor.
- Example: if we call the category of interrogatives Q , then the interrogative pronoun *who* has category S_{NP}^Q .
- This means *who* combines with an S containing an unbound NP gap to form a Q , at the same time binding the trace.
- The G constructor is reminiscent of Moortgat's (1991) **in-situ scoping** constructor q , but in the syntactic logic G is used for **overt**—not **covert**—movement.

(12) **Some Syntactic Words, CVG Style**

$\vdash \text{Chris} : \text{NP}$
 $\vdash \text{everyone} : \text{NP}$
 $\vdash \text{someone} : \text{NP}$
 $\vdash \text{who}_{\text{in-situ}} : \text{NP}$
 $\vdash \text{what}_{\text{in-situ}} : \text{NP}$
 $\vdash \text{who}_{\text{filler}} : \text{NP}_S^Q$
 $\vdash \text{what}_{\text{filler}} : \text{NP}_S^Q$
 $\vdash \text{barked} : \text{NP} \multimap_s$
 $\vdash \text{liked} : (\text{NP} \multimap_c (\text{NP} \multimap_s S))$
 $\vdash \text{thought} : S \multimap_c (\text{NP} \multimap_s S)$
 $\vdash \text{wondered} : Q \multimap_c (\text{NP} \multimap_s S)$
 $\vdash \text{whether} : (S \multimap_c Q)$

(13) **Modus Ponens in CVG Syntax**

These are **inference schemata** of the syntactic logic.

Schema M_s (Subject Modus Ponens)

If $\Gamma \vdash a : A$ and $\Gamma' \vdash f : A \multimap_s B$,
then $\Gamma; \Gamma' \vdash ({}^s a f) : B$

Schema M_c (Complement Modus Ponens)

If $\Gamma \vdash f : A \multimap_c B$ and $\Gamma' \vdash a : A$,
then $\Gamma; \Gamma' \vdash (f a {}^c) : B$

Note: These corresponds to HPSG's valence cancellation schemata.

Also note: The proof terms are written to be mnemonic of the order in which the words are phonologically realized.

(14) **Introducing Syntactic Hypotheses in CVG**

Schema T (Trace)

$t : A \vdash t : A$ (t fresh)

Note: TGists also call traces “syntactic variables”. But in EST/GB a trace is left behind when the operator that binds it **moves** out of the argument position; and in the MP, the trace and the operator that binds it are **copies**.

Our traces are just ordinary variables.

(15) **Schema G (Generalized Gazdar Schema)**

If $\Gamma \vdash a : A_B^C$ and $t : A; \Gamma' \vdash b : B$,
then $\Gamma; \Gamma' \vdash a_t b : C$ (t not free in a)

Note: This corresponds to HPSG's Filler-Head schema, which in turn derives from Gazdar's (1979) Linking schemata.

Also note: the free occurrence of t in b is bound in $a_t b$.

(16) **CVG vs. Lambek Calculus or Lambda Calculus**

This resembles an ND presentation of lambda calculus or (Buszkowski 1987) Lambek calculus (i.e. bilinear logic). Biggest difference:

- there is no lambda-abstraction/hypothetical proof.
- Instead, the implication $A \multimap B$ that you *expect* to be introduced by binding the trace is immediately eliminated by Schema G and you go straight to a C .
- Evidently, in NL as opposed to familiar logics, **implication introduction is lexically coordinated with implication elimination** to avoid introducing implications in derivations.

(17) **A Simple Sentence**

$\vdash (^S \text{ Chris } (\text{thought } (^S \text{ Kim } (\text{liked Dana } ^C) ^C))) : S$

(18) **An Embedded Polar Question**

$\vdash (\text{whether } (^S \text{ Kim } (\text{likes Sandy } ^C) ^C)) : Q$

(19) **An Embedded Constituent Question**

$\vdash [\text{what}_{\text{filler } t} (^S \text{ Kim } (\text{likes } t ^C))] : Q$

Here *what* is an operator of type NP_S^Q : it combines with an S containing an unbound NP trace to form a Q, while binding the trace.

(20) **A Binary Constituent Question**

$\vdash [\text{who}_{\text{filler } t} (^S t (\text{likes } \text{what}_{\text{in-situ } ^C}))] : S$

Here *who* is an operator but *what* is just an NP.

(21) **Baker Ambiguity**

$\vdash [\text{who}_{\text{filler } t} (^S t (\text{wonders } [\text{who}_{\text{filler } t'} (^S t' (\text{likes } \text{what}_{\text{in-situ } ^C}))]))] : S$

Here, both *who* are operators but *what* is just an NP.

This sentence is ambiguous as to whether *what* scopes at the root question or the embedded one.

So far this is just syntax. What do these sentences mean?

2.3 The CVG Semantic Logic

(22) Toward RC Calculus

- RC is a term calculus for NL meaning composition, also in the Gentzen-sequent style of ND with Curry-Howard terms.
- The easiest way to semantically interpret RC terms is to transform RC into TLC (typed lambda calculus), more specifically Ty2.
- Fortunately that is trivially easy.

(23) Format for RC Typing Judgments

$\Gamma \vdash a : A \dashv \Delta$

- a. ‘term a is assigned type A in context Γ and **co-context** Δ .’
- b. The context is used to track semantic variables corresponding to traces (like the semantic halves of HPSG SLASH values).
- c. The co-context is a generalization of Cooper storage, not just for quantifiers, but also indefinites, names, pronouns, reflexives, *wh*-in situ, comparative and superlative operators, subdeletion gaps, topic, focus, and more.
- d. The ‘co-’ is mnemonic for ‘Cooper’ and ‘covert movement’.

(24) RC Semantic Types

- a. There are some **basic** semantic types.
- b. If A and B are types, then $A \rightarrow B$ is a **functional** semantic type with **argument** type A and **result** type B .
- c. If A , B , and C are types, then $G[A, B, C]$, abbreviated to A_B^C , is an **operator** semantic type with **binding** type A , **scope** type B , and **result** type C .
- d. To summarize: the semantic type system is just like the syntactic category system, except
 - i. different basic types; and
 - ii. only one kind of implication (\rightarrow).

(25) Basic Semantic Types

For present purposes, we use three basic semantic types:

- ι (individual concepts)
- π (propositions)
- and κ (polar questions).

Note: Here κ is mnemonic for ‘Karttunen’ because its transform (see (31) below) into Ty2 will be the Karttunen type for questions.

Also note: If you are shaky on intensional types, most of the time you can think of ι as e and π as t and still get the gist.

(26) Abbreviated Notation for Functional Types

Where σ ranges over strings of types and ϵ is the null string:

- i. $A_\epsilon =_{\text{def}} A$
- ii. $A_{B\sigma} =_{\text{def}} B \rightarrow A_\sigma$ (e.g. $\pi_{\iota\iota} = \iota \rightarrow \iota \rightarrow \pi$)
- iii. For $n \in \omega$, $A_n =_{\text{def}} A_\sigma$ where σ is the string of ι ’s of length n

Example: $\pi_2 =_{\text{def}} \pi_{\iota\iota} =_{\text{def}} \iota \rightarrow \iota \rightarrow \pi$.

Note: This clunky notation is the price we pay for not having conjunction in the type logic.

(27) How the Semantic Operator Types are Used (1/2)

- Semantic operator types are used for expressions which would be analyzed in TG as undergoing (overt or covert) \bar{A} -movement.
- ‘Covert movement’: the semantics is an operator, but the syntax is **not**.
- Example: for Moortgat (1991) a QNP has category $q[\text{NP}, \text{S}, \text{S}]$ and semantic type $(\iota \rightarrow \pi) \rightarrow \pi$.

This misses the generalization that the syntactic category of the retrieval site is irrelevant; what matters is that the semantic type be π (or, more generally, a functional type with final result π).

- Whereas for us a QNP is just an NP with semantic type ι_π^π .

(28) How the Semantic Operator Types are Used (2/2)

- ‘Overt movement’: the semantic G-constructor works in lockstep with the syntactic G-constructor.
- Example: ‘Overtly moved’ *who* has category NP_S^Q and semantic type $\iota_\pi^{\kappa_1}$, where κ_1 is the type of unary constituent questions.
- The standard TLG way to get the effect of NP_S^Q is $Q/(S \uparrow \text{NP})$, where \uparrow is Moortgat’s (1988) **extraction** constructor.
- But this misses the generalization that there don’t seem to be any phrases of category $S \uparrow \text{NP}$.

(29) **What Goes into the Co-Context?**

- a. The co-contexts will contain semantic operators to be scoped, each paired with the variable that it will eventually bind.
- b. We call such stored pairs **commitments**, and write them in the form a_x , where the type of x is the binding type of a .
- c. Then we call x a **committed** variable, and say that a is **committed** to bind x .
- d. By contrast, the variables in the (left-of-turnstile) context are called **uncommitted** variables.

(30) **The Semantic Schemata**

Constants, variables, and Modus Ponens (Function Application) exactly as in the familiar typed lambda calculus, plus:

Semantic Schema C (Cooper Storage)

If $\Gamma \vdash a : A_B^C \dashv \Delta$, then $\Gamma \vdash x : A \dashv a_x : A_B^C; \Delta$ (x fresh)

Schema R (Retrieval)

If $\Gamma \vdash b[x] : B \dashv a_x : A_B^C; \Delta$, then $\Gamma \vdash (a_x b[x]) : C \dashv \Delta$,
(x free in b but not in Δ)

Schema G (Semantic Counterpart of Gazdar Schema)

If $\Gamma \vdash a : A_B^C \dashv \Delta$ and $x : A, \Gamma' \vdash b : B \dashv \Delta'$
then $\Gamma; \Gamma' \vdash (a_x b) : C \dashv \Delta, \Delta'$ (x not free in a)

Note: the underscoring of the bound variable in Schema R is an essential part of the proof term! Without it you can't tell whether the variable was bound by Schema R or by Schema G.

(31) **The Transform τ from RC Types to Ty2 Meaning Types**

- a. $\tau(\iota) = s \rightarrow e$
- b. $\tau(\pi) = s \rightarrow t$
- c. $\tau(\kappa) = \tau(\pi) \rightarrow \tau(\pi)$
- d. $\tau(A \rightarrow B) = \tau(A) \rightarrow \tau(B)$
- e. $\tau(A_B^C) = (\tau(A) \rightarrow \tau(B)) \rightarrow \tau(C)$

(32) **The Transform τ on Terms**

- a. Variables and basic constants are unchanged except for their types.
- b. $\tau((f a)) = \tau(f)(\tau(a))$

The change in the parenthesization has no theoretical significance. It just enables one to tell at a glance whether the term belongs to RC or to Ty2, e.g. (walk' Kim') vs. walk'(Kim').

- c. $\tau((a_x b)) = \tau(a)(\lambda_x \tau(b))$

Note: This is the important clause. It says that operator binding consists of abstraction immediately followed by application.

2.4 The CVG Syntax-Semantics Interface

(33) **Schema L (Lexicon)**

$\vdash w, c : A, B \dashv$ (for certain pairs $\langle w, c \rangle$ where w is a word of category A and c is a basic constant of type B)

This tells what words mean.

(34) **Schema M_s (Subject Modus Ponens)**

If $\Gamma \vdash a, c : A, C \dashv \Delta$ and $\Gamma' \vdash f, v : A \multimap_s B, C \rightarrow D \dashv \Delta'$,
then $\Gamma; \Gamma' \vdash ({}^s a f), (v c) : B, D \dashv \Delta; \Delta'$

This says that heads combine with subjects semantically by function application.

Note: Contexts (unbounded traces) and co-contexts (unscoped Cooper-stored operators) just get passed up, as in old-fashioned PSG.

(35) **Schema M_c (Complement Modus Ponens)**

If $\Gamma \vdash f, v : A \multimap_c B, C \rightarrow D \dashv \Delta$ and $\Gamma' \vdash a, c : A, C \dashv \Delta'$,
then $\Gamma; \Gamma' \vdash (f a {}^c), (v c) : B, D \dashv \Delta; \Delta'$

This says that heads combine with complements semantically by function application. Again, (co-)contexts are just passed up.

(36) **Schema T (Trace)**

$t, x : A, B \vdash t, x : A, B \dashv$ (t and x fresh)

This says that traces (syntactic variables) are paired with semantic variables from birth.

Note: In MP, traces must undergo a multistage process of 'trace conversion' in order to become semantically interpretable.

(37) **Schema C (Cooper Storage)**

If $\Gamma \vdash a, b : A, B_C^D \dashv \Delta$, then $\Gamma \vdash a, x : A, B \dashv b_x : B_C^D; \Delta$ (x fresh)

This says that when a semantic operator gets added to the Cooper store, nothing happens in the syntax (because the phrase whose meaning is stored is **not** an operator syntactically).

(38) **Schema R (Retrieval)**

If $\Gamma \vdash e, c[x] : E, C \dashv b_x : B_C^D; \Delta$ then $\Gamma \vdash e, (b_x c[x]) : E, D \dashv \Delta$
(x free in c but not in Δ)

This says that when a Cooper-stored semantic operator gets retrieved, again nothing happens in the syntax.

Note: the underscoring of the bound variable is an essential part of the proof term! Without it you can't tell whether the variable was bound by Schema R or by Schema G (just below).

(39) **Schema G (Generalized Gazdar Schema)**

If $\Gamma \vdash a, d : A_B^C, D_E^F \dashv \Delta$ and $t, x : B, D; \Gamma' \vdash b, e : B, E \dashv \Delta'$,
then $\Gamma; \Gamma' \vdash (a_t b), (d_x e) : C, F \dashv \Delta, \Delta'$
(t free in b , x free in e)

This says that fillers are operators, both syntactically and semantically.

Important: although co-contexts are **sets** of committed semantic variables, contexts are **lists** of **pairs** of a trace and a semantic variable. This captures the **Prohibition on Crossed Dependencies**.

Important: The operator a **binds** the trace t , but there is **absolutely** no construal of the words 'move' or 'copy' under which a moved from the argument position t occupies, or copied t , just as in lambda calculus there is no sense in which $\lambda_x.\text{bite}'(x)(\text{Fido})$ is derived by movement or copying from $\text{bite}'(\lambda)(\text{Fido})$.

3 Quantifier Scope

(40) **Lexicon for Quantifier Scope Fragment**

- $\vdash \text{Chris, Chris}' : \text{NP}, \iota \dashv$ (likewise other names)
- $\vdash \text{everyone, everyone}' : \text{NP}, \iota_\pi^\pi \dashv$
- $\vdash \text{someone, someone}' : \text{NP}, \iota_\pi^\pi \dashv$
- $\vdash \text{liked, like}' : (\text{NP} \dashv_C (\text{NP} \dashv_S \text{S})), \iota \rightarrow \iota \rightarrow \pi \dashv$
- $\vdash \text{thought, think}' : \text{S} \dashv_C (\text{NP} \dashv_S \text{S}), \pi \rightarrow (\iota \rightarrow \pi) \dashv$

(41) **A Refinement**

- Actually the QNP meanings have to be polymorphically typed to $\iota_{\pi\sigma}^{\pi\sigma}$ where σ ranges over strings of types, since quantifiers can be retrieved not just at proposition nodes, but also at nodes with functional types whose final result type is proposition.
- An important case is $\sigma = \iota$: quantifiers can be retrieved at nodes which are semantically individual properties ($\pi_{\iota} = \iota \rightarrow \pi$), such as VPs and Ns:
 - a. [Campaigning in every state] is prohibitively expensive.
 - b. Most [people with few interests] are uninteresting.

(42) **Ty2 Meaning Postulates for Generalized Quantifiers**

- $\vdash \text{every}' = \lambda_Q \lambda_P \lambda_w \forall_x (Q(x)(w) \rightarrow P(x)(w))$
- $\vdash \text{some}' = \lambda_Q \lambda_P \lambda_w \exists_x (Q(x)(w) \wedge P(x)(w))$
- $\vdash \text{everyone}' = \text{every}'(\text{person}')$
- $\vdash \text{someone}' = \text{some}'(\text{person}')$

Types for Ty2 variables are as follows:

- $x, y, z : s \rightarrow e$ (individual concepts)
- $p, q : s \rightarrow t$ (propositions); $w : s$ (worlds)
- $P, Q : ((s \rightarrow e) \rightarrow (s \rightarrow t))$ (properties of individual concepts).

(43) **Quantifier Scope Ambiguity**

- a. Syntax (both readings):
 $(^s \text{Chris} (\text{thinks } (^s \text{Kim} (\text{likes everyone } ^c) ^c))) : S$
- b. Semantics (scoped to lower clause):
RC: $((\text{think}'_{\underline{x}}((\text{like}'_{\underline{x}} \text{Kim}')) \text{Chris}')) : \pi$
Ty2: $\text{think}'(\lambda_w(\forall_x(\text{person}'(x)(w) \rightarrow \text{like}'(x)(\text{Kim}'))(w)))(\text{Chris}') : s \rightarrow t$
- c. Semantics (scoped to upper clause):
RC: $(\text{everyone}'_{\underline{x}}((\text{think}'((\text{like}'_{\underline{x}} \text{Kim}')) \text{Chris}')) : \pi$
Ty2: $\lambda_w(\forall_x(\text{person}'(x)(w) \rightarrow \text{think}'(\text{like}'(x)(\text{Kim}'))(\text{Chris}'))(w)) : s \rightarrow t$

(44) **Raising of Two Quantifiers to Same Clause**

Note: from now on we omit the Ty2 transform.

- a. Syntax (both readings): (^s everyone (likes someone ^c)) : S
- b. $\forall\exists$ -reading: (everyone' x(someone' y((like' y) x))) : π
- c. $\exists\forall$ -reading: (someone' y(everyone' x((like' y) x))) : π
- d. These are possible because for generalized quantifiers, the result type is the same as the scope type.
- e. Things are not so straightforward in the case of multiple in-situ wh-operators, as we soon will see.

(45) **The Side Conditions in Schema R**

If $\Gamma \vdash e, c[x] : E, C \dashv b_x : B_C^D; \Delta$ then $\Gamma \vdash e, (b_x c[\underline{x}]) : E, D \dashv \Delta$
(x free in c but not in Δ)

- a. The first conjunct prohibits vacuous quantification. For example, there is no reading of
Every owner of a donkey has regrets.
where the existential is in the scope (as opposed to the restrictor) of the universal, since *a donkey* binds no variable occurrence.
- b. The second conjunct makes sure that an operator binds every occurrence of 'its' variable. For example, there is no reading of
A rumor about him upset every boy.
where the universal is the antecedent of the pronoun but is outscoped by the existential, since then *every boy* fails to bind the occurrence of 'its' variable coming from the pronoun.
- c. The side conditions obviate the need for *nested* storage.

4 Constituent Questions

(46) **Ty2 Meaning Types**

These are defined as follows:

- a. $s \rightarrow e$ (individual concepts) is a Ty2 meaning type.
- b. $s \rightarrow t$ (propositions) is a Ty2 meaning type.
- c. If A and B are Ty2 meaning types, then so is $A \rightarrow B$.

(47) **Extensional Types Corresponding to Ty2 Meaning Types**

These are defined as follows:

- a. $E(s \rightarrow e) = e$
- b. $E(s \rightarrow t) = t$
- c. $E(A \rightarrow B) = (A \rightarrow E(B))$

(48) **Extensions of Ty2 Meanings**

The relationship between Ty2 meanings and their extensions is axiomatized as follows, where the family of constants $\text{ext}_A : s \rightarrow (A \rightarrow E(A))$ is parametrized by the Ty2 meaning types:

- a. $\vdash \forall_x \forall_w (\text{ext}_w(x) = x(w))$ (for $x : s \rightarrow e$)
- b. $\vdash \forall_p \forall_w (\text{ext}_w(p) = p(w))$ (for $p : s \rightarrow t$)
- c. $\vdash \forall_f \forall_w (\text{ext}_w(f) = \lambda_x \text{ext}_w(f(x)))$ (for $f : A \rightarrow B$, A and B Ty2 meaning types).

Note: we suppress the type parameter, and write ext_w for $\text{ext}(w)$.

(49) **Overall Approach to Interrogative Semantics**

The approach is described in detail in Pollard 2008. Key ideas:

- The analysis of polar questions (after transformation into Ty2) is that of Karttunen 1977: at each world w , an interrogative sentence denotes a set of w -facts (in this case, a singleton).
- For n -ary constituent interrogatives, the denotation at w is a (curried) n -ary function to w -facts. The **range** of that function is similar to the Karttunen semantics, except that it contains both positive and negative ‘true atomic answers.’
- An interrogative meaning of this kind induces an equivalence relation on worlds which is a **refinement** of the Groenendijk-Stokhof (1984) partition semantics.

(50) **Types for Polar Questions**

- a. RC meaning type: κ
- b. Meaning type of Ty2 transform: $(s \rightarrow t) \rightarrow (s \rightarrow t)$ (property of propositions)
- c. Type of Ty2 denotation: $(s \rightarrow t) \rightarrow t$ (characteristic function of) a (singleton) set of propositions)
- d. Example: at w , *Does Chris walk* (or *whether Chris walks*) denotes the singleton set whose member is whichever is true at w , the proposition that Chris walks or the proposition that s/he doesn’t.

(51) **Types for Unary Constituent Questions**

- a. RC meaning type: κ_1
- b. Meaning type of Ty2 transform: $(s \rightarrow e) \rightarrow ((s \rightarrow t) \rightarrow (s \rightarrow t))$
(function from individual concepts to properties of propositions).
- c. Type of Ty2 denotation: $(s \rightarrow e) \rightarrow ((s \rightarrow t) \rightarrow t)$ (function from individual concepts to sets of propositions). Technically, the curried version of the characteristic function of a certain binary relation between individual concepts and propositions.
- d. Example: at w , *who walks* denotes the (functional) binary relation between individual concepts x and propositions p that obtains just in case x is a w -person and p is whichever proposition is a w -fact, that x walks or that x does not walk.

(52) **Types for Binary Constituent Questions**

- a. RC meaning type: κ_2
- b. Meaning type of Ty2 transform: $(s \rightarrow e) \rightarrow ((s \rightarrow e) \rightarrow ((s \rightarrow t) \rightarrow (s \rightarrow t)))$ (curried function from pairs of individual concepts to properties of propositions).
- c. Type of Ty2 denotation: $(s \rightarrow e) \rightarrow ((s \rightarrow e) \rightarrow ((s \rightarrow t) \rightarrow t))$ (curried function from pairs of individual concepts to sets of propositions). Technically, the curried version of the characteristic function of a certain ternary relation between individual concepts, individual concepts, and propositions.
- d. Example: at w , *who likes what* denotes the (functional) ternary relation between individual concepts x and y and propositions p that obtains just in case x is a w -person, y is a w -thing, and p is whichever proposition is a w -fact, that x likes y or that x does not like y .

(53) **Types for Interrogatives (Summary)**

- a. The RC type for polar interrogatives (*whether Fido barked*) is $\kappa_0 = \kappa$, whose Ty2 transform is $(s \rightarrow t) \rightarrow s \rightarrow t$ (property of propositions).
- b. The RC type for unary constituent interrogatives (*who barked*) is $\kappa_1 = \iota \rightarrow \kappa$, whose Ty2 transform is $\iota \rightarrow (s \rightarrow t) \rightarrow s \rightarrow t$ (function from individuals to properties of propositions).
- c. The RC type for binary constituent interrogatives (*who bit who*) is $\kappa_2 = \iota \rightarrow \iota \rightarrow \kappa$, whose Ty2 transform is $\iota \rightarrow \iota \rightarrow (s \rightarrow t) \rightarrow s \rightarrow t$ (curried function from pairs of individuals to properties of propositions), etc.

(54) **Multiple *Wh*-In Situ vs. Multiple Quantifier Raising**

- a. The fact that not all questions have the same type introduces a complexity that does not arise with quantifier scope.
- b. But as we'll see, it also explains a lot.
- b. Since the result type of a quantifier is the same as its scope type, we can scope multiple quantifiers one after the other (44).
- c. But scoping one (overtly moved) *wh*-operator at a proposition produces a unary constituent question, so its type must be $t_{\pi}^{\kappa_1}$.
- d. So if we want to scope a second (in-situ) *wh*-operator over that unary constituent question to form a binary constituent question, then *its* type must be $t_{\kappa_1}^{\kappa_2}$, etc.
- d. We will return to this point presently.

(55) **Ty2 Meaning Postulates for Some Standard Logical Constants**

- b. $\vdash \text{and}' = \lambda_p \lambda_q \lambda_w (p(w) \wedge q(w))$
- c. $\vdash \text{or}' = \lambda_p \lambda_q \lambda_w (p(w) \vee q(w))$
- d. $\vdash \text{not}' = \lambda_p \lambda_w \neg p(w)$
- e. $\vdash \text{equals}'_A = \lambda_x \lambda_y \lambda_w (x = y)$

(56) **Ty2 MPs for Some Less Standard Logical Constants**

- a. $\vdash \text{whether}' = \lambda_q \lambda_p (p \text{ and}' ((p \text{ equals}' q) \text{ or}' (p \text{ equals}' \text{not}'(q))))$
- b. $\vdash \text{which}^0 = \lambda_Q \lambda_P \lambda_x \lambda_p (Q(x) \text{ and}' \text{whether}'(P(x))(p))$
- c. $\vdash \text{which}^n = \lambda_Q \lambda_Z \lambda_{x_0} \dots \lambda_{x_n} \lambda_p (Q(x) \text{ and}' Z(x_0) \dots (x_n)(p))$ ($n > 0$)
- d. $\vdash \text{who}^n = \text{which}^n(\text{person}')$
- e. $\vdash \text{what}^n = \text{which}^n(\text{thing}')$

(57) **Lexicon for Interrogative Fragment**

- $\vdash \text{Kim}, \text{Kim}' : \text{NP}, \iota \dashv$
 $\vdash \text{liked}, \text{like}' : (\text{NP} \multimap_c (\text{NP} \multimap_s \text{S}), \iota \rightarrow (\iota \rightarrow \pi) \dashv$
 $\vdash \text{whether}, \text{whether}' : (\text{S} \multimap_c \text{S}, \pi \rightarrow \kappa) \dashv$
 $\vdash \text{wondered}, \text{wonder}'_n : \text{S} \multimap_c (\text{NP} \multimap_s \text{S}), \kappa_n \rightarrow (\iota \rightarrow \pi) \dashv$
 $\vdash \text{who}_{\text{filler}}, \text{who}^0 : \text{NP}_S^Q, \iota_{\pi}^{\kappa_1} \dashv$
 $\vdash \text{who}_{\text{in-situ}}, \text{who}^n : \text{NP}, \iota_{\kappa_n}^{\kappa_{n+1}} \dashv$ (for $n > 0$)
 $\vdash \text{what}_{\text{filler}}, \text{what}^0 : \text{NP}_S^Q, \iota_{\pi}^{\kappa_1} \dashv$
 $\vdash \text{what}_{\text{in-situ}}, \text{what}^n : \text{NP}, \iota_{\kappa_n}^{\kappa_{n+1}} \dashv$ (for $n > 0$)

(58) **Observations about Interrogative *who***

- The interrogative 'pronoun' *who* is **syntactically** ambiguous between a syntactic operator $\text{who}_{\text{filler}}$ and an NP, $\text{who}_{\text{in-situ}}$.
- $\text{who}_{\text{filler}}$ can only form an interrogative (Q) by scoping syntactically over a 'declarative' (i.e. semantically propositional) S containing at least one unbound NP trace, and the semantic result (formed by scoping who^0 over an open proposition), is a unary constituent question (type κ_1).
- $\text{who}_{\text{in-situ}}$ cannot scope syntactically, but its stored meaning (any of who^n , $n > 0$) can be retrieved at a constituent question (type κ_n , $n > 0$) to form a 'higher' constituent question (type κ_{n+1}).

(59) **Consequences**

- There can be no purely in-situ interrogatives (leaving aside pragmatically restricted, intonationally marked ones which we cannot go into here):
**I wonder Fido bit who?*
- A *wh*-expression cannot scope, either overtly or covertly, over a polar interrogative:
**I wonder whether Fido bit who?*
**I wonder who whether Fido bit?*
- In each constituent interrogative, only one 'overtly moved' *wh*-expression can take scope there:
**I wonder who who(m) bit?*

(60) **More Consequences**

- Arbitrarily many in-situ *wh*-expressions can take their semantic scope at a given constituent interrogative:
Who gave what to who when?
- There are (Baker) ambiguities that hinge on how high an in-situ *wh*-expression scopes:
Who wondered who bit who?
- Even though subject *wh*-expressions might *look* in situ:
Who barked?
they aren't really; if they were, they could also scope higher to form impossible embedded questions as in:
**Kim wondered Chris thought who barked?*
(Intended meaning: Kim wondered who Chris thought barked.)

(61) **Wh-In Situ Languages**

In languages without overt *wh*-movement, the counterpart of *who* is just an NP with **all** the meanings who^n ($n \geq 0$), **including** who^0 .

That is: the difference between overt and covert *wh*-movement languages is in the lexicon.

(62) **An English Embedded Polar Question**

- a. Syntax: $\vdash (\text{whether } (^s \text{ Kim (likes Sandy } ^c)) ^c) : S$
- b. Semantics: $\vdash (\text{whether}' (\text{like}' \text{ Sandy}' \text{ Kim}')) : \kappa_0$

(63) **An English Embedded Constituent Question**

- a. Syntax: $\vdash [\text{what}_{\text{filler } t} (^s \text{ Kim (likes } t ^c))] : S$
- b. Semantics: $\vdash (\text{what}_y^0 ((\text{like}' y) (\text{Kim}')) : \kappa_1$

(64) **A English Binary Constituent Question**

- a. Syntax: $\vdash [\text{who}_{\text{filler } t} (^s t (\text{likes what}_{\text{in-situ } ^c}))] : S$
- b. Semantics: $\vdash (\text{what}_y^1 (\text{who}_x^0 ((\text{like}' y) (x)))) : \kappa_2$

(65) **Baker Ambiguity**

- a. $\vdash [\text{who}_{\text{filler } t} (^s t (\text{wonders } [\text{who}_{\text{filler } t'} (^s t' (\text{likes what}_{\text{in-situ } ^c}))] ^c))] : S$
- b. $\vdash (\text{who}_x^0 ((\text{wonder}'_2 (\text{what}_y^1 (\text{who}_z^0 ((\text{like}' y) z)))) x) : \pi$
(E.g. Chris wonders who likes what.)
- c. $\vdash (\text{what}_y^1 (\text{who}_x^0 ((\text{wonder}'_1 (\text{who}_z^0 ((\text{like}' y) z)))) x) : \pi$
(E.g. Chris wonders who likes the books, and Kim wonders who likes the records.)

5 Conclusion

(66) What this Talk was About

- On the intuitive level, EMG had a simple and elegant theory of so-called movement phenomena.
- But it was hard to present the theory persuasively using 1970's technology (no substructural logic or Curry-Howard).
- HPSG complicated the story line by
 - coding everything up in feature logic
 - abandoning Montague semantics for situation semantics.
- Later CG complicated things in different ways:
 - insisting (as per Montague) that the syntax-semantics interface be a function, no matter how much it complicates the syntax
 - Not picking up on Gazdar's insight that 'overt movement' binds a trace without ever introducing an implication.
- We re-told the EMG movement story, in Curry-Howard terms.

(67) The EMG Story about Movement, Retold

- The syntax-semantics interface recursively specifies a set of pairs, each consisting of a syntactic proof and a semantic proof.
- The syntactic logic is like familiar CG, but with \uparrow replaced by an ND reformulation of Gazdar's machinery for overt movement (traces and a linking schema).
- The semantic logic is like familiar lambda calculus, but with abstraction replaced by an ND reformulation of Cooper's machinery for covert movement (storage and retrieval).
- As far as it goes, this account seems simpler and more straightforward than continuation-based accounts.
- But it remains to be seen whether the full range of phenomena treated in terms of continuations will yield to such elementary methods.