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

The handout for this talk is available at:

`http://www.ling.ohio-state.edu/~pollard/cvg/symho.pdf`

INTRODUCTION: LIFE WITHOUT CONTINUATIONS?

(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.

- A Little History
- Back to the Future: Convergent Grammar (CVG)
  - Syntax: Bar-Hillel Meets Gazdar
  - Semantics: Curry Meets Cooper
  - The Syntax-Semantics Interface
- Quantifier Scope
- Wh-Questions
- Conclusion

A LITTLE HISTORY

(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 practicioners of:
  - PSG, e.g. Cooper, Gazdar, Pullum
  - CG, e.g. Dowty
  - switch hitters, e.g. Bach.
- CG then (essentially AB grammars) was not much different from PSG.
- Lambek's (1958) calculus didn't start to catch on with linguists until the mid-1980's.

(3) **Two Signal Achievments 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.

BACK TO THE FUTURE: CONVERGENT GRAMMAR (CVG)

(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.

CVG SYNTAX: BAR-HILLEL MEETS GAZDAR

(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 $A \multimap_{\mathrm{F}} B$, 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 $\mathrm{G}[A, B, C]$, 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 (**gramfuns**) S (subject) and C (complement). Others can be added as needed.
- The gramfuns correspond to HPSG **valence features**.
- Since the grammatical functions aren't relevant for this talk, you can just think $\multimap_S$ as Lambek's \ 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 $\mathrm{S}^{\mathrm{Q}}_{\mathrm{NP}}$.
- 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**

Words are **axioms** of the syntactic logic.

$\vdash \textsf{Chris} : \mathrm{NP}$

$\vdash \textsf{everyone} : \mathrm{NP}$

$\vdash \textsf{someone} : \mathrm{NP}$

$\vdash \textsf{who}_{\textsf{in-situ}} : \mathrm{NP}$

$\vdash \textsf{what}_{\textsf{in-situ}} : \mathrm{NP}$

$\vdash \textsf{who}_{\textsf{filler}} : \mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$

$\vdash \textsf{what}_{\textsf{filler}} : \mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$

$\vdash \textsf{barked} : \mathrm{NP} \multimap_{\mathrm{S}}$

$\vdash \textsf{liked} : (\mathrm{NP} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S})$

$\vdash \textsf{thought} : \mathrm{S} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S})$

$\vdash \textsf{wondered} : \mathrm{Q} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S})$

$\vdash \textsf{whether} : (\mathrm{S} \multimap_{\mathrm{C}} \mathrm{Q})$

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

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

Schema $\mathbf{M_s}$ (Subject Modus Ponens)

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

Schema $\mathbf{M_c}$ (Complement Modus Ponens)

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

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 $\Gamma \vdash a : A^C_B$ 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) **The CVG Syntactic Schemata**

Schema $\mathbf{M_s}$ (Subject Modus Ponens)

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

Schema $\mathbf{M_c}$ (Complement Modus Ponens)

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

Schema T (Trace)

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

Schema G (Generalized Gazdar Schema)

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

(17) **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.

(18) **A Simple Sentence**

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

(19) **An Embedded Polar Question**

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

(20) **An Embedded Constituent Question**

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

Here *what* is an operator of type $\mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$: it combines with an S containing an unbound NP trace to form a Q, while binding the trace.

(21) **A Binary Constituent Question**

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

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

(22) **Baker Ambiguity**

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

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?

(23) **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.

(24) **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'.

(25) **RC Semantic Types**

a. There are some **basic** semantic types.

b. If A and B are types, then $A \to B$ is a **functional** semantic type with **argument** type A and **result** type B.

c. If A, B, and C are types, then $\mathrm{G}[A, B, C]$, abbreviated to A^C_B, 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 ($\to$).

(26) **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 (32) 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.

(27) **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. $A_{B\sigma} =_{\text{def}} B \to A_\sigma$ (e.g. $\pi_{\iota\iota} = \iota \to \iota \to \pi$)

iii. For $n \in \omega$, $A_n =_{\text{def}} A_\sigma$ where σ is the string of ι's of length n

Example: $\pi_2 =_{\text{def}} \pi_{\iota\iota} =_{\text{def}} \iota \to \iota \to \pi$.
Note: This clunky notation is the price we pay for not having conjunction in the type logic.

(28) **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) $\bar{\text{A}}$-movement.
- 'Covert movement': the semantics is an operator, but the syntax is **not**.
- Example: for Moortgat (1991) a QNP has category $\mathrm{q}[\mathrm{NP}, \mathrm{S}, \mathrm{S}]$ and semantic type $(\iota \to \pi) \to \pi$.

  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}.

(29) **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 $\mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$ 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 $\mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$ is $\mathrm{Q}/(\mathrm{S} \uparrow \mathrm{NP})$, where $\uparrow$ is Moortgat's (1988) **extraction** constructor.
- But this misses the generalization that there don't seem to be any phrases of category $\mathrm{S} \uparrow \mathrm{NP}$.

(30) **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 **uncommited** variables.

(31) **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 $\Gamma \vdash a : A^C_B \dashv \Delta$, then $\Gamma \vdash x : A \dashv a_x : A^C_B; \Delta$ (x fresh)

Schema R (Retrieval)

If $\Gamma \vdash b[x] : B \dashv a_x : A^C_B; \Delta$, then $\Gamma \vdash (a_{\underline{x}} b[\underline{x}]) : C \dashv \Delta$,
(x free in b but not in Δ)

Schema G (Semantic Counterpart of Gazdar Schema)

If $\Gamma \vdash a : A^C_B \dashv \Delta$ 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.

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

a. $\tau(\iota) = \mathrm{s} \rightarrow \mathrm{e}$

b. $\tau(\pi) = \mathrm{s} \rightarrow \mathrm{t}$

c. $\tau(\kappa) = \tau(\pi) \rightarrow \tau(\pi)$

d. $\tau(A \rightarrow B) = \tau(A) \rightarrow \tau(B)$

e. $\tau(A^C_B) = (\tau(A) \rightarrow \tau(B)) \rightarrow \tau(C)$

(33) **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.

THE CVG SYNTAX-SEMANTICS INTERFACE:

SEVEN SCHEMATA

(34) **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.

(35) **Schema $\mathbf{M_s}$ (Subject Modus Ponens)**

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

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.

(36) **Schema $\mathbf{M_c}$ (Complement Modus Ponens)**

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

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

(37) **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.

(38) **Schema C (Cooper Storage)**

If $\Gamma \vdash a, b : A, B^D_C \dashv \Delta$, then $\Gamma \vdash a, x : A, B \dashv b_x : B^D_c ; \Delta$ (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).

(39) **Schema R (Retrieval)**

If $\Gamma \vdash e, c[x] : E, C \dashv b_x : B^D_C ; \Delta$ then $\Gamma \vdash e, (b_{\underline{x}} c[\underline{x}]) : E, D \dashv \Delta$
(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 (next slide).

(40) **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 ('overtly moved' phrases) 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 $\lambda_x.\mathsf{bite'}(x)(\mathsf{Fido'})$ is derived by movement or copying from $\mathsf{bite'}(\lambda)(\mathsf{Fido'})$.

QUANTIFIER SCOPE

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

$\vdash \textsf{Chris}, \textsf{Chris'} : \mathrm{NP}, \iota \dashv$ (likewise other names)

$\vdash \textsf{everyone}, \textsf{everyone'} : \mathrm{NP}, \iota^{\pi}_{\pi} \dashv$

$\vdash \textsf{someone}, \textsf{someone'} : \mathrm{NP}, \iota^{\pi}_{\pi} \dashv$

$\vdash \textsf{liked}, \textsf{like'} : (\mathrm{NP} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}), \iota \rightarrow \iota \rightarrow \pi \dashv$

$\vdash \textsf{thought}, \textsf{think'} : \mathrm{S} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}), \pi \rightarrow (\iota \rightarrow \pi) \dashv$

(42) **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 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.

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

$\vdash \mathsf{every'} = \lambda_Q \lambda_P \lambda_w \forall_x (Q(x)(w) \rightarrow P(x)(w))$

$\vdash \mathsf{some'} = \lambda_Q \lambda_P \lambda_w \exists_x (Q(x)(w) \wedge P(x)(w))$

$\vdash \mathsf{everyone'} = \mathsf{every'}(\mathsf{person'})$

$\vdash \mathsf{someone'} = \mathsf{some'}(\mathsf{person'})$

Types for Ty2 variables are as follows:

$x, y, z : \mathrm{s} \rightarrow \mathrm{e}$ (individual concepts)

$p, q : \mathrm{s} \rightarrow \mathrm{t}$ (propositions); $w : \mathrm{s}$ (worlds)

$P, Q : ((\mathrm{s} \rightarrow \mathrm{e}) \rightarrow (\mathrm{s} \rightarrow \mathrm{t}))$ (properties of individual concepts).

(44) **Quantifier Scope Ambiguity**

a. Syntax (both readings):
(^{S} Chris (thinks (^{S} Kim ($\mathsf{likes\ everyone}$ ^{C}) ^{C}))) : S

b. Semantics (scoped to lower clause):
RC: $((\mathsf{think'}\ (\mathsf{everyone'}_{\underline{x}}((\mathsf{like'}\ \underline{x})\ \mathsf{Kim'})))\ \mathsf{Chris'}) : \pi$
Ty2: $\mathsf{think'}(\lambda_w(\forall_x(\mathsf{person'}(x)(w) \rightarrow \mathsf{like'}(x)(\mathsf{Kim'})(w))))(\mathsf{Chris'})$: $\mathrm{s} \rightarrow \mathrm{t}$

c. Semantics (scoped to upper clause):
RC: $(\mathsf{everyone'}_{\underline{x}}((\mathsf{think'}\ ((\mathsf{like'}\ \underline{x})\ \mathsf{Kim'}))\ \mathsf{Chris'})) : \pi$
Ty2: $\lambda_w(\forall_x(\mathsf{person'}(x)(w) \rightarrow \mathsf{think'}(\mathsf{like'}(x)(\mathsf{Kim'}))(\mathsf{Chris'})(w)))$: $\mathrm{s} \rightarrow \mathrm{t}$

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

Note: from now on we omit the Ty2 transform.

a. Syntax (both readings): ($^{\mathrm{S}}$ everyone (likes someone $^{\mathrm{C}}$)) : S

b. ∀∃-reading: (everyone'$_{\underline{x}}$(someone'$_{\underline{y}}$((like' $\underline{y}$) $\underline{x}$))) : π

c. ∃∀-reading: (someone'$_{\underline{y}}$(everyone'$_{\underline{x}}$((like' $\underline{y}$) $\underline{x}$))) : 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.

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

If $\Gamma \vdash e, c[x] : E, C \dashv b_x : B^D_C; \Delta$ then $\Gamma \vdash e, (b_{\underline{x}} c[\underline{x}]) : E, D \dashv \Delta$
(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 occurence.

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.

WH-QUESTIONS

(47) **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 $A \rightarrow B$.

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

These are defined as follows:

a. $\mathsf{E}(\mathrm{s} \rightarrow \mathrm{e}) = \mathrm{e}$

b. $\mathsf{E}(\mathrm{s} \rightarrow \mathrm{t}) = \mathrm{t}$

c. $\mathsf{E}(A \rightarrow B) = (A \rightarrow \mathsf{E}(B))$

(49) **Extensions of Ty2 Meanings**

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

a. $\vdash \forall_x \forall_w (\mathsf{ext}_w(x) = x(w)$ (for $x : \mathrm{s} \rightarrow \mathrm{e}$)

b. $\vdash \forall_p \forall_w (\mathsf{ext}_w(p) = p(w)$ (for $p : \mathrm{s} \rightarrow \mathrm{t}$)

c. $\vdash \forall_f \forall_w (\mathsf{ext}_w(f) = \lambda_x \mathsf{ext}_w(f(x))$ (for $f : A \rightarrow B$, A and B Ty2 meaning types.

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

(50) **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.

(51) **Types for Polar Questions**

a. RC meaning type: κ

b. Meaning type of Ty2 transform: $(\mathrm{s} \to \mathrm{t}) \to (\mathrm{s} \to \mathrm{t})$ (property of propositions)

c. Type of Ty2 denotation: $(\mathrm{s} \to \mathrm{t}) \to \mathrm{t}$ (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.

(52) **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 and p is whichever proposition is a w-fact, that x walks or that x does not walk.

(53) **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.

(54) **Types for Interrogatives (Summary)**

a. The RC type for polar interrogatives (*whether Fido barked*) is $\kappa_0 = \kappa$, whose Ty2 transform is $(\mathrm{s} \to \mathrm{t}) \to \mathrm{s} \to \mathrm{t}$ (property of propositions).

b. The RC type for unary constituent interrogatives (*who barked*) is $\kappa_1 = \iota \to \kappa$, whose Ty2 transform is $\iota \to (\mathrm{s} \to \mathrm{t}) \to \mathrm{s} \to \mathrm{t}$ (function from individuals to properties of propositions).

c. The RC type for binary consituent interrogatives (*who bit who*) is $\kappa_2 = \iota \to \iota \to \kappa$, whose Ty2 transform is $\iota \to \iota \to (\mathrm{s} \to \mathrm{t}) \to \mathrm{s} \to \mathrm{t}$ (curried function from pairs of individuals to properties of propositions), etc.

(55) **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 s 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 (45).

c. But (for example,) scoping one (overtly moved) *wh*-operator at a proposition produces a unary consituent question, so its type must be $\iota_{\pi}^{\kappa_1}$.

d. So if we want to scope a second (in-situ) *wh*-operator over that unary constituent question to form a binary consituent question, then *its* type must be $\iota_{\kappa_1}^{\kappa_2}$, etc.

d. We will return to this point presently.

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

a. $\vdash \mathsf{id}_A = \lambda_x x \ (Z : \tau(\kappa_n))$

b. $\vdash \mathsf{and'} = \lambda_p \lambda_q \lambda_w (p(w) \wedge q(w))$

c. $\vdash \mathsf{or'} = \lambda_p \lambda_q \lambda_w (p(w) \vee q(w))$

d. $\vdash \mathsf{not'} = \lambda_p \lambda_w \neg p(w)$

e. $\vdash \mathsf{equals'}_A = \lambda_x \lambda_y \lambda_w (x = y)$

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

a. $\vdash \mathsf{whether'} = \lambda_q \lambda_p (p \ \mathsf{and'} \ ((p \ \mathsf{equals'} \ q) \ \mathsf{or'} \ (p \ \mathsf{equals'} \ \mathsf{not'}(q))))$

b. $\vdash \mathsf{which}^0 = \lambda_Q \lambda_P \lambda_x \lambda_p (Q(x) \ \mathsf{and'} \ \mathsf{whether'}(P(x))(p))$

c. $\vdash \mathsf{which}^n = \lambda_Q \lambda_Z \lambda_{x_0} \ldots \lambda_{x_n} \lambda_p (Q(x) \ \mathsf{and'} \ Z(x_0) \ldots (x_n)(p)) \ (n > 0)$

d. $\vdash \mathsf{who}^n = \mathsf{which}^n(\mathsf{person'})$

e. $\vdash \mathsf{what}^n = \mathsf{which}^n(\mathsf{thing'})$

(58) **Lexicon for Interrogative Fragment**

$\vdash \textsf{Kim}, \textsf{Kim'} : \mathrm{NP}, \iota \dashv$

$\vdash \textsf{liked}, \textsf{like'} : (\mathrm{NP} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}), \iota \rightarrow (\iota \rightarrow \pi) \dashv$

$\vdash \textsf{whether}, \textsf{whether'} : (\mathrm{S} \multimap_{\mathrm{C}} \mathrm{S}, \pi \rightarrow \kappa) \dashv$

$\vdash \textsf{wondered}, \textsf{wonder'}_n : \mathrm{S} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}), \kappa_n \rightarrow (\iota \rightarrow \pi) \dashv$

$\vdash \textsf{who}_{\textsf{filler}}, \textsf{who}^0 : \mathrm{NP}^{\mathrm{Q}}_{\mathrm{S}}, \iota^{\kappa_1}_{\pi} \dashv$

$\vdash \textsf{who}_{\textsf{in-situ}}, \textsf{who}^n : \mathrm{NP}, \iota^{\kappa_{n+1}}_{\kappa_n} \dashv$ (for $n > 0$)

$\vdash \textsf{what}_{\textsf{filler}}, \textsf{what}^0 : \mathrm{NP}^{\mathrm{Q}}_{\mathrm{S}}, \iota^{\kappa_1}_{\pi} \dashv$

$\vdash \textsf{what}_{\textsf{in-situ}}, \textsf{what}^n : \mathrm{NP}, \iota^{\kappa_{n+1}}_{\kappa_n} \dashv$ (for $n > 0$)

(59) **Observations about Interrogative *who***

- The interrogative 'pronoun' *who* is **syntactically** ambiguous between a syntactic operator $\mathsf{who}_{\mathsf{filler}}$ and an NP, $\mathsf{who}_{\mathsf{in\text{-}situ}}$.
- $\mathsf{who}_{\mathsf{filler}}$ 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).
- $\mathsf{who}_{\mathsf{in\text{-}situ}}$ 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' consituent question (type κ_{n+1}).

(60) **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?*

(61) **More Consequences**

- Arbitrarily many in-situ *wh*-expressions can take their semantic scope at a given consituent 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 impossssible embedded questions as in:

  **Kim wondered Chris thought who barked*?

  (Intended meaning: Kim wondered who Chris thought barked.)

(62) **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.

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

a. Syntax: $\vdash$ (whether ($^{\mathrm{S}}$ Kim (likes Sandy $^{\mathrm{C}}$)) $^{\mathrm{C}}$) : S

b. Semantics: $\vdash$ (whether' (like' Sandy' Kim')) : κ_0

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

a. Syntax: $\vdash$ [what$_{\mathsf{filler}}$ $_t$($^{\mathrm{S}}$ Kim (likes t $^{\mathrm{C}}$))] : S

b. Semantics: $\vdash$ (what^0_y((like' y) (Kim')) : κ_1

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

a. Syntax: $\vdash$ [who$_{\mathsf{filler}}$ $_t$($^{\mathrm{S}}$ t (likes what$_{\mathsf{in\text{-}situ}}$ $^{\mathrm{C}}$))] : S

b. Semantics: $\vdash$ (what$^1_{\underline{y}}$(who^0_x((like' $\underline{y}$) (x))) : κ_2

(66) **Baker Ambiguity**

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

b. $\vdash (\textsf{who}^{0}_{x}((\textsf{wonder'}_{2}\ (\textsf{what}^{1}_{\underline{y}}(\textsf{who}^{0}_{z}((\textsf{like'}\ \underline{y})\ z))))\ x)) : \pi$

(E.g. Chris wonders who likes what.)

c. $\vdash (\textsf{what}^{1}_{\underline{y}}(\textsf{who}^{0}_{x}((\textsf{wonder'}_{1}\ (\textsf{who}^{0}_{z}((\textsf{like'}\ \underline{y})\ z)))\ x)))) : \pi$

(E.g. Chris wonders who likes the books, and Kim wonders who likes the records.)

CONCLUSIONS

(67) **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.

(68) **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.