

COOPER STORAGE CURES THE COMMON COLD

Carl Pollard
INRIA-Lorraine and Ohio State University

NaTAL Workshop
Session on Semantics and Inference
Nancy, June 25, 2008

The handout for these talks is available at:

<http://www.ling.ohio-state.edu/~pollard/cvg/natal.pdf>

PART ONE: BACKGROUND

- Introduction: The Intuition behind Cooper Storage
- A Little History
- Back to the Future: Convergent Grammar (CVG)
- Syntax: Ajdukiewicz and Bar-Hillel Meet Gazdar
- Semantics: Curry and Gentzen Meet Cooper
- The CVG Syntax-Semantics Interface

PART TWO: CASE STUDIES

- Quantifier Scope
- Wh-Questions
- Topicalization
- Simple Comparatives
- Subdeletion
- Phrasal Comparatives

(1) A Cautionary Note about Meaning Types

- The basic meaning types I will use are ι (individual concept), π (proposition), and κ (question).
- If you're familiar with the (intensional) types used in mainstream semantic theory (such as Ty2), you can think of:

ι as $s \rightarrow e$

π as $s \rightarrow t$

κ as $(s \rightarrow t) \rightarrow s \rightarrow t$

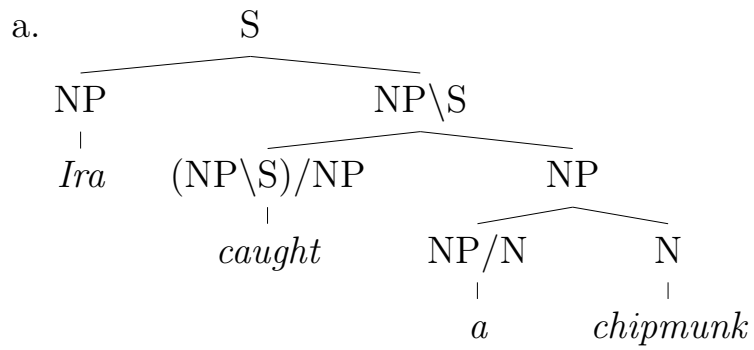
- A lot of the time, you can just think of ι as e and π as t and still get the gist.

**INTRODUCTION: THE INTUITION BEHIND
COOPER STORAGE**

(2) Cooper Storage and Retrieval (Intuitively)

- a. Imagine that while semantically interpreting a syntactic derivation bottom up, you encounter a quantified noun phrase with meaning $a : \iota \rightarrow \pi \rightarrow \pi$ in a position where a meaning of type ι is called for (say, object of a transitive verb).
- b. Then (**storage**) you can replace a by a fresh variable $x : \iota$ and place the pair (a, x) in the **store**, which is a set of such pairs.
- c. **Notation:** we'll write a_x instead of (a, x) for stored pairs.
- d. Usually, stores 'percolate upward' through derivations.
- e. But if you get to a higher node in the derivation with semantics $b : \pi$, then (**retrieval**) you can remove a_x from the store and **scope** it over $b : \pi$, obtaining $a(\lambda_x b) : \pi$.
- f. **Notation:** we'll write $a_x b$ instead of $a(\lambda_x b)$ for the result of scoping a retrieved quantifier (x will be free in b).

(3) A Simple Example: Syntax



- b. For the moment we assume the syntax is just AB (Ajdukiewicz-Bar Hillel bidirectional applicative categorial grammar).

(5) Some General Observations

- a. The semantic composition **usually** goes in lockstep with the syntactic composition:
 - Words correspond to semantic basic constants.
 - Left and right application correspond to application.
- b. But nothing happens in the syntax that corresponds to storages and retrievals in the semantics.
- c. From this it follows that semantic interpretation is not a **function** from syntactic derivations to meanings (examples soon).
- d. For a great many categorial grammarians, this is a deal breaker; in CG, the usual approach is to make the **syntax** more complicated to preserve the functionality of the syntax-semantics interface.
- e. This is a sociological fact, not a scientific problem for us.

A LITTLE HISTORY

(6) **Extended Montague Grammar (EMG)**

- In the 1970s, the EMG community emerged as an alternative to the then-current form of transformational grammar (TG), namely Chomsky's REST (Revised Extended Standard Theory).
- EMG was influenced by mathematical logic (especially model theory) and computer science.
- It sought a simpler, more precise, and more tractable alternative to REST, with an emphasis on the syntax-semantics interface.
- It consisted of practitioners of PSG (such as Robin Cooper and Gerald Gazdar), CG (such as David Dowty), and switch hitters (like Emmon Bach).
- Then-current CG (essentially AB grammars) was technically not much different from PSG (Lambek's calculus was, but it didn't start to catch on with linguistics until the mid-1980's).

(7) Two Signal Achievements of EMG

- Technically, the most obvious trait differentiating EMG from REST was the lack of **movement**, both **overt** (e.g. Topicalization and *Wh*-Movement) and **covert** (e.g Quantifier Raising (QR)).
- Two of the most significant achievements of EMG were devices proposed for eliminating movement.
- The first, Cooper storage (1975), eliminated **covert** movement.
- The second, Gazdar's (1979) **linking schemata**, which we will discuss, eliminated **overt** movement.
- As we will see, from a logical perspective these two devices are almost identical.

(8) **The 1980s and Beyond**

- REST evolved into GB, and then into the MP, where (overt and covert) movement is still the central explanatory device.
- The EMG community fractured into numerous competing frameworks such as CCG, HPSG, TLG, Pregroup Grammars, ACG, etc.
- In spite of the significant accomplishments of these frameworks, they failed to capture the essence of Cooper's and Gazdar's contributions, therefore that they do not exert the influence they deserve to exert in contemporary linguistic theorization (though they are widely appreciated in computational linguistics).

BACK TO THE FUTURE: CONVERGENT GRAMMAR (CVG)

(9) **What is CVG?**

- CVG is yet another nontransformational framework, that I have been developing recently at the Ohio State University and INRIA-Lorraine, together with a group of students.
- 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.
- Additionally, it seeks to **reformulate** the original insights of EMG (such as Cooper storage and Gazdar's linking schemata) in a way that makes their elegance and power more evident (and therefore more influential).
- Among existing frameworks, the ones CVG most closely resembles are ACG and HPSG.

(10) CVG Compared with ACG and HPSG

- Like both ACG and HPSG, CVG has a **parallel** architecture: independent components generate candidate phonological (or phenogrammatical), syntactic (or tectogrammatical), and semantic entities.
- Like ACG, these candidate entities are **proofs**, each in a different logic. We call this **pure derivationality** as opposed to the kind of derivationality in TG that builds arboreal representations by sequences of structural operations.
- Like HPSG, CVG is (only) **weakly syntactocentric**.

(11) **Syntactocentrism, Weak vs. Strong**

- A **syntactocentric** framework is one where there is no **direct** relation between semantics and phonology.
- Instead, that relation is **mediated** by the **syntax-phonology interface** and the **syntax-semantics interface**.
- TG, CG, HPSG, and CVG are all syntactocentric.
- But CG and TG are **strongly** syntactocentric: the phonology and semantics are algorithmically **obtained from** the syntax (deterministically in CG, nondeterministically in TG).
- Whereas HPSG and CVG are **weakly** syntactocentric: the two interfaces from syntax are merely **relations**.
- In the case of the CVG syntax-semantics, this arises from the use of a **very generalized form of Cooper storage**.

(12) **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: AJDUKIEWICZ AND
BAR-HILLEL MEET GAZDAR**

(13) Format for CVG Syntactic Typing Judgments

$\Gamma \vdash a : A$

- This is in the **Gentzen-sequent style of natural deduction**.
- Informally, it says: “ a belongs to category A and has unbound traces Γ ”.
- Pedantically: “the syntactic component of the grammar assigns the term a the category A in the context Γ .”
- Γ , called the **context** of the judgment, is the list of **typed syntactic variables**, also called **traces**, that have not yet been bound.
- The context is CVG’s counterpart of the HPSG SLASH feature.
- HPSGs are really natural-deduction systems, a fact which is obscured by the feature-structure encoding.

(14) 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 $A \dashv_{\mathbf{F}} B$, where \mathbf{F} belongs to a finite set \mathbf{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 .

(15) Functional Categories

- We start off with the grammatical function names S (subject) and C (complement). Others will be added as needed.
- Logically, the $-o_F$ connectives are **implications**.
- They are like Lambek's / and \, but they encode the grammatical function of the argument, not (just) the directionality.
- They are CVG's counterparts of HPSG's **valence features**.
- Example: a transitive verb will have category $NP -o_C (NP -o_S S)$.
- There is poetic justice here, since HPSG's valence features originated as a feature-structure encoding of CG category cancellation.
- This is also in keeping with our back-to-the-future motif.

(16) Operator Categories

- These are the categories of expressions that TG analyzes 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 $\text{NP}_{\S}^{\text{Q}}$.
- This means *who* combines with an S containing an NP gap (= unbound trace) to form a Q, at the same time binding the trace.
- The G constructor is highly reminiscent of Moortgat's (1991) in-situ scoping constructor q, but we are using it for **overt** movement (e.g. English Topicalization and *wh*-questions), not just **covert** movement (like quantifier scoping).

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

These should be thought of as **axioms** of the syntactic logic.

⊢ Chris : NP (likewise other names)

⊢ everyone : NP

⊢ someone : NP

⊢ $\text{who}_{\text{in-situ}}$: NP

⊢ $\text{what}_{\text{in-situ}}$: NP

⊢ $\text{who}_{\text{filler}}$: NP_S^Q

⊢ $\text{what}_{\text{filler}}$: NP_S^Q

⊢ barked : NP \rightarrow_S S

⊢ liked : NP \rightarrow_C (NP \rightarrow_S S)

⊢ thought : S \rightarrow_C (NP \rightarrow_S S)

⊢ wondered : Q \rightarrow_C (NP \rightarrow_S S)

⊢ whether : (S \rightarrow_C Q)

(18) **Two CVG Syntactic Schemata**

These should be thought of as **inference rules** of the syntactic logic.

Schema M_s (Subject Modus Ponens)

If $\Gamma \vdash a : A$ and $\Gamma' \vdash f : A \multimap_s B$

then $\Gamma; \Gamma' \vdash ({}^s a f) : B$

Note: This is the CVG counterpart of HPSG's Subject-Head schema.

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: This is the CVG counterpart of HPSG's Head-Complement schema.

(19) **Another CVG Syntactic Schema**

Schema T (Trace)

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

Note: This is the CVG counterpart of HPSG's Trace schema. But it is also identical to the ND schema for introducing variables in lambda calculus.

Also 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 is an **inaudible copy** of the operator that binds it.

(20) **The Last CVG Syntactic Schema**

Schema G (Generalized Gazdar Schema)

If $\Gamma \vdash a : A_B^C$ and $t : A; \Gamma' \vdash b : B$,

then $\Gamma; \Gamma' \vdash a_t b : C$

Note: This is the CVG counterpart of HPSG's Filler-Head schema, which in turn derives from Gazdar's (1979) Linking schemata.

Also note: Schema G introduces a new kind of term of the form $a_x b$, in which the free occurrence of x in b is bound, hence the name **binding term**.

(21) **All Together Now**

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 $\Gamma; \Gamma' \vdash (f a {}^c) : B$

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 $\Gamma; \Gamma' \vdash a_t b : C$

(22) CVG vs. Lambek Calculus or Lambda Calculus

This looks a lot like an ND presentation of lambda calculus or (Buszkowski 1987) Lambek calculus (i.e. bilinear logic), except:

- the implications are labelled with grammatical function names instead of the directions left and right
- the application terms tell the grammatical function of the argument
- 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 systems of logic, **implication introduction is lexically coordinated with implication elimination** so that no implications are ever actually introduced in derivations.

(23) **A Simple Syntactic Derivation**

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

- No, I didn't forget to draw the tree. You can reconstruct it from the term.
- No, I didn't forget to put category labels on the subterms. You can reconstruct them from the categories of the words, which are easy to remember.

(24) **An Embedded Polar Question**

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

(25) **An Embedded Constituent Question**

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

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.

(26) **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.

(27) **Baker Ambiguity**

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

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

As we soon will see, operators have to scope in place, but a *wh*-in situ ‘covertly moves’ (much like a QNP) to get scope, resulting in ambiguity as to whether it scopes in the root question or the embedded one.

CURRY AND GENTZEN MEET COOPER

(28) **Toward RC Calculus**

- RC is a term calculus for NL meaning composition, also in the labelled Gentzen-sequent style of natural deduction.
- The easiest way to semantically interpret RC terms is to transform RC into TLC (typed lambda calculus), more specifically Ty2.
- Fortunately that is very easy.
- There isn't time to discuss that today, but see the handout from my Séminaire Calligramme talks at:
<http://www.ling.ohio-state.edu/pollard/cvg>.

(29) **Format for RC Typing Judgments**

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

- a. Read ‘the term a is assigned the type A in the context Γ and the **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, pied piping, comparative and superlative operators, subdeletion gaps, topic and focus, ellipsis, and more.
- d. The ‘co-’ here is mnemonic for ‘Cooper’ and ‘covert movement’.

(30) **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).

(31) **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 (37) 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.

(32) **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: $\pi_2 =_{\text{def}} \pi_{\iota\iota} =_{\text{def}} \iota \rightarrow \iota \rightarrow \pi$.

(33) Remarks on Semantic Operator Types

- The operator types will be used for expressions which would be analyzed in TG as undergoing \bar{A} -movement (either overt or covert).
- In the case of covert movement, the G-constructor can be identified with Moortgat's (1991) q-constructor, but in the *semantic* logic, *not* the syntactic one.'
- Thus, for example, while for Moortgat (1996) a QNP would have category $q[\text{NP}, \text{S}, \text{S}]$ and semantic type $(\iota \rightarrow \pi) \rightarrow \pi$, for us it has category (simply) NP and semantic type ι_{π}^{π} .
- In the case of overt movement, the G-constructor will handle the semantic composition for the Gazdar syntactic schema.

(34) **RC Semantic Terms**

- a. There is a denumerable infinity of **semantic variables** of each type.
- b. There are finitely many **basic semantic constants** of each type.
- c. There are **functional** semantic terms of the form $(f a)$, where f and a are semantic terms.
- d. There are **binding** semantic terms of the form $(a_x b)$ where a and b are semantic terms and x is a semantic variable.

(35) **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.

(36) **RC 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 $\Gamma \vdash b : B \dashv a_x : A_B^C; \Delta$ then $\Gamma \vdash (a_x b) : C \dashv \Delta$
(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 $\Gamma; \Gamma' \vdash (a_x b) : C \dashv \Delta, \Delta'$

Important: the same G type constructor and binding term constructor are used in both Schema R and Schema G. But in R the operator comes from ‘within’, while in G it comes from ‘the outside’.

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

a. $\tau(l) = s \rightarrow e$

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

(38) **The Transform τ on Terms**

a. Variables and basic constants are unchanged except for their types. (We make abundant use of meaning postulates, e.g. (48) rather than giving basic constants nonbasic transforms.)

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**

(39) **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.

(40) **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.

(41) **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.

(42) **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.

(43) **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).

(44) **Schema R (Retrieval)**

If $\Gamma \vdash e, c : E, C \dashv b_x : B_C^D; \Delta$ then $\Gamma \vdash e, (b_x c) : E, D \dashv \Delta$
(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. (The side conditions will be explained later.)

(45) **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 $\lambda_x.\text{bite}'(x)(\text{Fido})$ is derived by movement or copying from $\text{bite}'(\lambda)(\text{Fido})$.

QUANTIFIER SCOPE IN ENGLISH

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

⊢ Chris, Chris' : NP, ι ⊢ (likewise other names)

⊢ everyone, everyone' : NP, ι_{π}^{π} ⊢

⊢ someone, someone' : NP, ι_{π}^{π} ⊢

⊢ liked, like' : NP \rightarrow_C (NP \rightarrow_S S), $\iota \rightarrow \iota \rightarrow \pi$ ⊢

⊢ thought, think' : S \rightarrow_C (NP \rightarrow_S S), $\pi \rightarrow (\iota \rightarrow \pi)$ ⊢

(47) **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.

(48) **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:

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

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

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

(49) **Quantifier Scope Ambiguity**

a. Syntax (both readings):

$(^S \text{Chris} (\text{thinks } (^S \text{Kim} (\text{likes everyone } ^C \text{ } ^C)))) : S$

b. Semantics (scoped to lower clause):

RC: $((\text{think}' (\text{everyone}'_{\underline{x}} ((\text{like}' \underline{x}) \text{Kim}')))) \text{Chris}') : \pi$

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: $\lambda_w(\forall_x(\text{person}'(x)(w) \rightarrow \text{think}'(\text{like}'(x)(\text{Kim}'))(\text{Chris}')(w))) :$
 $s \rightarrow t$

(50) **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 will see below.

(51) **About the Side Conditions in Schema R**

If $\Gamma \vdash e, c : E, C \dashv b_x : B_C^D; \Delta$ then $\Gamma \vdash e, (b_x c) : E, D \dashv \Delta$
(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.

***WH*-QUESTIONS**

(52) **Ty2 Meaning Types**

- 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 $A \rightarrow B$.

(53) **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. $E(A \rightarrow B) = (A \rightarrow E(B))$

(54) **Extensions of Ty2 Meanings**

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

(55) 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.

(56) **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.

(57) **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.

(58) **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 .

(59) **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.

(60) **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 (50).
- c. But (for example,) scoping one (overtly moved) *wh*-operator at a proposition produces a unary constituent 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 constituent question, then *its* type must be $\iota_{\kappa_1}^{\kappa_2}$, etc.
- d. We will return to this point presently.

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

- a. $\vdash \text{id}_A = \lambda_x x (Z : \tau(\kappa_n))$
- 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. $\vdash \text{equals}'_A = \lambda_x \lambda_y \lambda_w (x = y)$

(62) **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. $\vdash \text{what}^n = \text{which}^n(\text{thing}')$

(63) **Lexicon for Interrogative Fragment**

⊢ Kim, Kim' : NP, ι ⊢

⊢ liked, like' : (NP \rightarrow_C (NP \rightarrow_S S), $\iota \rightarrow (\iota \rightarrow \pi)$ ⊢

⊢ whether, whether' : (S \rightarrow_C S, $\pi \rightarrow \kappa$) ⊢

⊢ wondered, wonder'_n : S \rightarrow_C (NP \rightarrow_S S), $\kappa_n \rightarrow (\iota \rightarrow \pi)$ ⊢

⊢ who_{filler}, who⁰ : NP_S^Q, $\iota_{\pi}^{\kappa_1}$ ⊢

⊢ who_{in-situ}, whoⁿ : NP, $\iota_{\kappa_n}^{\kappa_{n+1}}$ ⊢ (for $n > 0$)

⊢ what_{filler}, what⁰ : NP_S^Q, $\iota_{\pi}^{\kappa_1}$ ⊢

⊢ what_{in-situ}, whatⁿ : NP, $\iota_{\kappa_n}^{\kappa_{n+1}}$ ⊢ (for $n > 0$)

(64) **Observations about Interrogative *who***

- The interrogative 'pronoun' *who* is **syntactically** ambiguous between a syntactic operator $\mathbf{who}_{\text{filler}}$ and an NP, $\mathbf{who}_{\text{in-situ}}$.
- $\mathbf{who}_{\text{filler}}$ 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 \mathbf{who}^0 over an open proposition), is a unary constituent question (type κ_1).
- $\mathbf{who}_{\text{in-situ}}$ cannot scope syntactically, but its stored meaning (any of \mathbf{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}).

(65) **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?*

- 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;)

(66) **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.

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

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

b. RC: $\vdash (\text{whether}' (\text{like}' \text{Sandy}' \text{Kim}')) : \kappa_0$

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

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

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

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

(70) **Baker Ambiguity**

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

b. RC: $\vdash (\text{who}_x^0((\text{wonder}'_2 (\text{what}_y^1(\text{who}_z^0((\text{like}' y z)))) x)) : \pi \dashv$

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

c. RC: $\vdash (\text{what}_y^1(\text{who}_x^0((\text{wonder}'_1 (\text{who}_z^0((\text{like}' y z)))) x))) : \pi \dashv$

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

TOPICALIZATION

(71) **A New Grammatical Function for Phrasal Affixation**

- We add to the inventory of grammatical function names the name AFFIX (abbr. A), mnemonic for ‘(phrasal) affixation’.
- As with other grammatical functions, we add a new flavor of Modus Ponens to the syntactic (and interface) schemata (the Elimination rule for $\neg\circ_A$).
- This schema will be used to analyze (e.g) intonationally realized information-structural affixes, Japanese and Korean case markers, Chinese sentence particles, English possessive -’s, etc.

(72) Syntactic Analysis of Topicalization in CVG

- We add to the lexicon a topicalization phrasal-affix schema
 $\vdash \text{top, top}' : A \dashv\circ_A A_S^T, B \rightarrow B_\pi^\pi \dashv$
- The syntax-phonology interface specifies that **top** is optionally realized as a contrastive-topic pitch accent on the realization of the *A*.
- **top** is not an operator, but it is an **operizer**: it changes the phrase it affixes to into a syntactic operator.
- This is a CVG counterpart of the notion of a ‘movement trigger’ in TG.
- We remain agnostic for now whether the category T of topicalized sentences is distinct from S.

(73) Toward a Semantic Analysis of Topicalization

- The assumption that the meaning of a topicalized sentence is just a proposition is a simplification in the absence of a concrete type-theoretic embodiment of information-structural concepts such as contrastive topic.
- Under the RC-toTy2 transform, the semantic operator **top'** is mapped to a family (parametrized by Ty2 meaning types B) of constants of type $B \rightarrow (B \rightarrow (s \rightarrow t)) \rightarrow (s \rightarrow t)$.
- Intuitively, **top'**(b)(P) means something like ' $P(b)$, but for certain other B 's, x , contextually associated with b , whether ' $P(x)$ ' remains unresolved.'
- This approach (I think) gets at the intuition that 'structured propositions' try to get at, but as far as its type goes, **top'**(b)(P) is just a proposition.

(74) **Sandy, Kim likes**

$\vdash ((\text{Sandy top}^A)_t(\text{Kim (likes } t^C)))$,
 $((\text{top}' \text{Sandy}')_x(\text{like}' x \text{ Kim}')) : \mathbb{T}, \pi \dashv$

(75) **What's the Connection with Cooper Storage?**

- The connection is that in English, contrastive topics can also **stay in situ**, with (seemingly) the same information-structural import.
- We can account for this by adding to the the lexicon a second phrasal affix $\text{top}_{\text{in-situ}}$ with the same (operizer) meaning and the same phonology as **top**, but instead of being an syntactic operizer, it leaves the category of the phrase it affixes to unchanged:

$$\vdash \text{top}_{\text{in-situ}}, \text{top}' : A \multimap_A A, B \rightarrow B_\pi^\pi \dashv$$

- We then get cases of **in situ topicalization** such as:

$$\vdash (\text{S Kim (likes (Sandy } \text{top}_{\text{in-situ}}^A \text{ C)})), \\ ((\text{top}' \text{ Sandy}')_{\underline{x}}((\text{like}' \underline{x}) \text{ Kim}')) : \mathcal{S}, \pi \dashv$$

- Accentually marked focus can perhaps be analyzed in an analogous fashion.

SIMPLE COMPARATIVES

(76) **Three Kinds of Comparatives**

We illustrate the applicability of Cooper storage to the analysis of three kinds of English comparative construction

- a. Jo owes more than five dollars to Bo. (simple comparative)
- b. Jo owes more than Yo spent d on Fido to Bo. (comparative sub-deletion)
- c. JO owes more to Bo than KIM. (phrasal comparative)
- d. Jo owes more to BO than to KIM. (phrasal comparative)

Note: In the simple and subdeletion examples, the *than*-phrase can be extraposed; while in phrasal comparatives, the *than* phrase *must* extrapose to the right of both the comparative morpheme and the pitch-accented phrase with which it is compared (the **associate**). We will not analyse such facts today though.

(77) **Preliminary Observations**

- a. All comparatives contain a **comparative morpheme** (here, *more*), and a phrase introduced by either *than* or *as* (here the former).
- b. In **simple** comparatives, the complement of *than* is a **degree** phrase (Deg).
- c. In **subdeletion** comparatives, the *than*-complement is a sentence that appears to have a Deg missing (in traditional terminology, **subdeleted**), indicated by *d*.
- d. Each **phrasal comparative** has two phrases, optionally bearing focal pitch accents, which, intuitively, denote the two things being compared. One of them, called the **remnant**, appears to be the *than* complement; and the other is called the **associate**.
- e. Semantically, each sentence expresses a comparison between two **degrees** belonging to the same **scale** (here, the scale whose members are monetary values, such as the one denoted by *five dollars*).

(78) **Informal Meaning Analysis**

- a. The (maximum) amount that Jo owes to Bo exceeds five dollars.
- b. The (maximum) amount that Jo owes to Bo exceeds the (maximum) amount that Yo spent on Fido.
- c. The (maximum) amount that Jo owes to Bo exceeds the (maximum) amount that Kim owes to Bo.
- d. The (maximum) amount that Jo owes to Bo exceeds the (maximum) amount that Jo owes to Kim.

(79) **Slightly More Formal Meaning Analysis**

- a. Jo owes more than five dollars to Bo. (simple comparative)
 $\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \5
- b. Jo owes more than Yo spent d on Fido to Bo. (comparative sub-deletion)
 $\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{spend}'(\text{fido}')(d)(\text{yo}'))$
- c. JO owes more to Bo than KIM. (phrasal comparative)
 $\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{kim}'))$
- d. Jo owes more to BO than to KIM. (phrasal comparative)
 $\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{owe}'(\text{kim}')(d)(\text{jo}'))$

(80) **Fine Points**

- For simplicity we stick to extensional analysis, using basic types e, t, and d (degrees).
- We would need intensional types, e.g. degree concepts, for non-rigid degree expressions like *Fido's price*.
- We pretend the only scale is degrees of monetary value.
- In a richer fragment, we would need different types for degrees in different scales, e.g. linear extents, areas, volumes, masses, velocities, etc.

(81) **Some Even Finer Points**

- We use least upper bounds (**lub**) rather than the usually assumed maximums. (If you don't know the difference, don't worry unless you plan to work on comparatives.)
- Maximums work fine as long as the scales are all linearly ordered and all the sets of degrees that arise actually *have* one.
- Additionally we assume the set of degrees is endowed with a strict partial order $<$, such that every set of degrees has a **lub**.
- I.e., the set of degrees ordered by $<$ is a complete join-semilattice.
- In the present case, this requires adjoining a top element that represents having infinite monetary value.
- Without these assumptions, analyses of subdeletion tend to fail for unexpected, nonlinguistic, reasons.

(82) **The Hard Parts**

The hard work is going to be assigning meanings to:

- a. *than*
- b. *more*
- c. whatever it is that 'happened' to the associate phrase, that can be reflected phonologically as a focal pitch accent.
- d. *d* (the inaudible thing where subdeletion 'happened')

(83) **Where We are Going with This**

We will analyze these expressions as follows:

- a. *than'* is $>$, the dual (= reverse) of the $<$ order on the scale.
- b. *more'* is a semantic operizer (“covert movement trigger”) that maps a set of degrees to a generalized quantifier over degrees.
- c. The focal pitch accent on the associate is the phonological realization of another semantic operizer, which converts the phrase it marks into a semantic operator, thereby forcing it to (in TG parlance) “undergo covert focus raising”.
- d. The inaudible subdeletion operator is precisely the **lub** operator that maps a set of degrees to a degree.

(84) **Lexicon for Simple Comparatives (Boring Part)**

⊢ *fido*, *fido'* : NP, e ⊣ (likewise other names)

⊢ \$5, \$5' : Deg, d ⊣

⊢ *owes*, *owe'* : Deg \rightarrow_C To \rightarrow_C NP \rightarrow_S S,

d \rightarrow e \rightarrow e \rightarrow t ⊣

⊢ *spent*, *spend'* : Deg \rightarrow_C On \rightarrow_C NP \rightarrow_S S,

d \rightarrow e \rightarrow e \rightarrow t ⊣

⊢ *to*, *id* : NP \rightarrow_C To, e \rightarrow e ⊣

⊢ *on*, *id* : NP \rightarrow_C On, e \rightarrow , e ⊣

Note: New syntactic categories: Deg (degree phrase); On (nonpredicative PP headed by *on*); To (nonpredicative PP headed by *to*); Th (*than*-phrase).

(85) **Lexicon for Simple Comparatives (Interesting Part)**

$\vdash \text{than}_{\text{Deg}}, > : \text{Deg} \multimap_{\text{C}} \text{Th}, d \rightarrow d \rightarrow t \dashv$

This *than* takes a Deg complement; it denotes $>$, which is a binary relation on degrees.

$\vdash \text{more}, \text{more}' : \text{Th} \multimap_{\text{C}} \text{Deg}, (d \rightarrow t) \rightarrow d_t^t \dashv$

This *more* takes a *than*-phrase complement to form a Deg. The Deg so formed denotes **not** a degree (type d), but rather a **generalized quantifier over degrees** (type d_t^t).

This means *more* is a **semantic operizor** (covert movement trigger).

The RC constant *more'* translates into Ty2 as $\mathbf{B}'(\text{lub})$, where *lub* is the least upper bound function on sets of degrees, and \mathbf{B}' is the combinator $\lambda_f \lambda_Q \lambda_P Q(f(P))$ (function composition, but with its first two arguments permuted).

(86) **Simple Comparative Example**

Syn: (^s jo ((owes (more (than \$5^c)^c)^c)(to bo^c)^c))

RC: more'(> (\$5'))_d(owe' bo' d jo')

Ty2: lub(λ_d owe'(bo')(d)(jo')) > \$5

SUBDELETION

(87) Additional Lexicon for Comparative Subdeletion

$\vdash \text{than}_S, < : S \multimap_C \text{Th}, d \rightarrow d \rightarrow t$

This *than* takes a degree-denoting sentential complement. (Yes, there are such things, as we'll see!)

$\vdash \text{trice}, \text{lub} : \text{Deg}, d_t^d$

- This is the inaudible Deg (nicknamed “trice” in HPSG) involved in Comparative Subdeletion.
- Semantically it is an operator that binds a degree variable in a boolean expression, returning a degree (type d_t^d).
- In lambda-calculus terms, it maps a set of degrees to a degree (by taking the least upper bound).
- Alternatively, we could analyze *trice* as a Deg trace, and make than_S a syntactic operator (category $S_{\text{Deg}}^{\text{Th}}$) that binds it.

(88) **A Subdeletion Example**

We only show the analysis of the *than*-clause, since otherwise subdeletion comparatives work just like simple comparatives.

Syn: $(\text{than}_S (^S \text{yo } ((\text{spent } \text{trice}^C)(\text{on } \text{fido}^C)^C))^C) : \text{Th}$

RC: $(> \text{lub}_{\underline{d}}(\text{spend}' \underline{d} \text{fido}' \text{yo}')) : d \rightarrow t$

Note: Now you know where degree-denoting sentences come from.

PHRASAL COMPARATIVES

(89) **Phrasal Comparative Examples Recalled**

a. JO owes more to Bo than KIM.

$$\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{kim}'))$$

b. Jo owes more to BO than to KIM.

$$\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{owe}'(\text{kim}')(d)(\text{jo}'))$$

c. Remember the *than*-complement is called the **remnant**, and the (potentially focus-accented) phrase in the main clause (everything except the *than*-phrase) is called the **associate**.

d. The truth conditions of the sentence depend on which phrase in the main clause is the associate.

e. Roughly, the associate and the remnant are being compared.

f. But more precisely, two **degrees** are being compared.

g. One of those degrees relates to the associate, and the other to the remnant.

(90) **The Intuition behind Phrasal Comparatives**

- Phrasal comparatives are a kind of **ellipsis** construction.
- In order to find the degree related to the remnant *KIM* in *JO owes more to Bo than KIM* we must:
 - make a copy of the “ellipsed” material “*x* owes *d* to Bo”, which resulted from covertly moving the associate *JO* and the comparative operator *more* out of the main clause
 - lambda-abstract on *d*, then on *x*
 - apply the resulting abstract to the remnant
 - take the lub of the result.
- Then we do exactly the same thing again, but with the associate instead of the remnant, and compare the two degrees.

(91) **Toward a Semantics of Phrasal Comparatives**

- JO owes more to Bo than KIM.
- We need a new *more*, more_{pc} , that does not combine directly with a *than*-phrase, because the contribution of *than* will be brought in only at the very end.
- So we don't need to apply that combinator \mathbf{B}' ; we can just take the meaning to be the **lub** operator:
 $\vdash \text{more}_{\text{pc}}, \text{lub} : \text{Deg}, d_t^d \dashv$
- Since we also need to covertly “evict” *JO*, we need to assume an **associate semantic operizer**
 $\vdash \text{assoc}, \text{assoc}' : A \multimap_A A_S^S, B \rightarrow B_t^t \dashv$
that turns (the meaning of) *Jo* into an operator.
- We'll worry about what assoc' means in a moment.

(92) **The Main Clause of a Phrasal Comparative**

Syn: (^S (jo assoc ^A) ((owes more_{pc} (to bo ^C) ^C)) : S

RC: $\text{assoc}'(\text{jo}')_x \text{lub}_d(\text{owe}' d \text{bo}' x) : e \rightarrow (d \rightarrow d \rightarrow t) \rightarrow t$

Ty2: $\text{assoc}(\text{jo}')(\lambda_x.\text{lub}(\lambda_d.\text{owe}'(d)(\text{bo}')(x))) : (d \rightarrow d \rightarrow t) \rightarrow e \rightarrow t$

We know the type (up to permutation of the arguments) because we know it still has to apply to the ‘main connective of the ellipsis’ (namely $>$), and then finally to the remnant.

- We also know the target semantics for the whole sentence:
 $\text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{jo}')) > \text{lub}(\lambda_d \text{owe}'(\text{bo}')(d)(\text{kim}'))$
- This is enough to calculate the meaning of the associate operizer.

(93) The Meaning of the Associate Opererizer

- Remember that τ is the transform from RC to Ty2, and **assoc'** is the RC constant for the meaning of the associate opererizer.
- Then $\tau(\text{assoc}')$ is
 $\lambda_x \lambda_P \lambda_r \lambda_y . r(P(y))(P(x)) : e \rightarrow (e \rightarrow d) \rightarrow (d \rightarrow d \rightarrow t) \rightarrow e \rightarrow t$
- For our current example, the values fed to this opererizer are
 - a. the associate **jo**
 - b. the ellipsed environment $\lambda_x . \text{lub}(\lambda_d . \text{owe}'(\text{bo}')(d))(x)$
 - c. the main connective of the ellipsis, $>$
 - d. the remnant **kim'**
- In short, the associate opererizer feeds both the associate and the remnant to the ellipsed environment, obtaining (in this case) two degrees, which are then compared using the main connective.
- But actually, the opererizer is polymorphically typed and has application to a whole range of ellipsis constructions.

CONCLUSION

(94) **Summary**

- CVG is a synthesis of (largely old) ideas, developed within the PSG and CG communities, and reformulated in natural deduction.
- The syntax-semantics interface recursively specifies a set of pairs consisting of a syntactic proof and a semantic proof.
- The syntactic logic is simple: (multi-)applicative CG plus an ND reformulation of Gazdar's (1979) machinery for handling 'overt movement' (traces and a linking schema).
- The semantic logic (RC) is simple too: like TLC but with lambda-abstraction replaced by an ND reformulation of Cooper's (1975) storage and retrieval.
- Cooper storage, suitably generalized, is much more elegant, and much more broadly applicable, than is generally believed.
- But I lied about the common cold.