

CONVERGENT GRAMMAR

Carl Pollard
INRIA-Lorraine and Ohio State University

ESLLI 2008
Hamburg, August 4–8, 2008

The handout for this lecture is available at:

<http://www.ling.ohio-state.edu/~pollard/cvg/day3ho.pdf>

DAY THREE: SO-CALLED COVERT MOVEMENT

- The Semantic Logic
- The Syntax-Semantics Interface
- Quantifier Scope
- Should Overt and Covert Movement be Collapsed?

SEMANTICS

(1) **Syntax, Semantics, and Interface in CVG**

- Candidate syntactic proofs 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 proofs are specified by a **semantic logic** similar to lambda calculus, but with abstraction replaced by a (proof-theoretic version of) **Cooper storage and retrieval** .
- The **syntax-semantics interface** specifies which pairs of proofs go together.

(2) A Preview of the CVG Semantic Logic

- The semantic logic is broadly similar to TLC.
- The formulas/types are the **semantic types**.
- The semantic term of a sign gets semantically interpreted.
- Thus it is the closest CVG counterpart of an ‘LF’. But:
 - The semantic terms are in no way derived from syntax, and
 - there is an explicit translation into TLC, hence no indeterminacy about their interpretation.
- As in Montague semantics, basic constants denote word meanings.
- The syntax-semantics interface will ensure that free semantic variables are always paired with either (1) unbound traces, or (2) Cooper-stored semantic operators.

(3) Format for Judgments in Semantic Rules

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

- a. ‘term a is assigned type A in context Γ and **co-context** Δ .’
- b. The context lists the unbound traces.
- c. The co-context (Cooper storage, ND style) stores quantifiers, indefinites, pronouns, reflexives, *wh*-in situ, comparative and superlative operators, subdeletion gaps, topic, focus, and more.
- d. Each operator is stored together with the variable it will bind.
- d. The co-context is a set, not a list (assuming covert movement is not subject to the Nested Dependency Condition).
- e. We often omit the ‘ \dashv ’ if the co-context is empty.

(4) Semantic Types

- a. **Basic** types: for present purposes, e, t, and d (degrees).
- b. **Function** types: If A and B are types, then so is $A \rightarrow B$.
- c. **Operator** types: If A , B , and C are types, so is $G[A, B, C]$, abbreviated A_B^C .
- d. So the semantic type system is just like the syntactic category system, except
 - i. different basic types; and
 - ii. only one kind of implication (\rightarrow).

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

- Semantic operator types are used for expressions which would be analyzed in TG as undergoing (overt or covert) \bar{A} -movement.
- ‘Covertly moved’ signs: the syntax is not an operator, but the semantics (which gets Cooper-stored) is.

Example: A QNP has category NP, but its semantic type is e_t^t .

- For comparison, a typical TLG category would be $q[NP, S, S]$, with corresponding semantic type $e \rightarrow t \rightarrow t$.

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

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

- 'Overtly moved' signs: syntax and semantics are both operators; they are scoped by the semantic G-constructor working in lockstep with the syntactic G-constructor.
- Example: 'Overtly moved' *who* has category NP_S^Q and semantic type $e_t^{e \rightarrow \pi \rightarrow t}$ (where $\pi =_{\text{def}} s \rightarrow t$).
- By comparison, the standard TLG way to get the effect of NP_S^Q is $Q/(S\uparrow\text{NP})$, where \uparrow is Moortgat's (1988) **extraction** constructor. But this misses the generalization that there don't seem to be any phrases of category $S\uparrow\text{NP}$.

(7) **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.

(8) The Semantic Schemata

Basic constants and variables as in TLC, plus:

Semantic Schema M (Modus Ponens)

If $\Gamma \vdash f : A \rightarrow B \dashv \Delta$ and $\Gamma' \vdash a : A \dashv \Delta'$,
then $\vdash (f a) : B \dashv \Delta; \Delta'$

Semantic Schema C (Cooper Storage)

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

Schema R (Retrieval)

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

Schema G (Semantic Counterpart of Gazdar Schema)

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

(9) **Semantic Schema M (Modus Ponens)**

If $\Gamma \vdash f : A \rightarrow B \dashv \Delta$ and $\Gamma' \vdash a : A \dashv \Delta'$,
then $\vdash (f a) : B \dashv \Delta; \Delta'$

- a. This is the usual ND Modus Ponens, except that co-contexts have to be propagated from premisses to conclusions.
- b. Semicolons in co-contexts represent set union (necessarily disjoint, since variables are always posited fresh).

(10) **Semantic Schema C (Cooper Storage)**

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

- a. This is a straightforward ND formulation of Cooper storage.
- b. It generalizes Carpenter's (1997) Introduction rule for Moortgat's (1988) \uparrow (essentially the special case of q where the scope type and the result type are the same), but **in the semantics, not in the syntax**.

(11) **More about Schema C**

- a. The type of a committed variable always matches the binding type of the operator it is committed to.
- b. The syntax-semantics interface will guarantee that when an operator gets stored in the semantics, **no corresponding syntactic change takes place**.
- c. This is one of two reasons why the relation between syntax and semantics is **not** a function (the key difference between CVG and other kinds of CG).
- d. In this respect, our proposed architecture for the syntax-semantics interface resembles Cooper 1975/1983, Hendriks 1993, and HPSG.

(12) **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)

- a. This is a straightforward ND formulation of Cooper retrieval.
- b. It generalizes Carpenter's (1997) Elimination rule for Moortgat's \uparrow , but, again, in the semantics, **not** in the syntax.
- c. Underscoring x in Schema R is part of the term! Otherwise you can't tell whether x was bound by Schema R or Schema G.
- d. As with Storage, the syntax-semantics interface will ensure that instances of Retrieval correspond to no syntactic change.
- e. This is the other reason why the relation between syntax and semantics is **not** a function.

(13) **Where's Lambda?**

- The semantic calculus does not have λ -abstraction.
- Instead, we hypothesize that NL doesn't license free withdrawal of hypotheses.
- A variable can be bound only by scoping an internal (Schema R) or external (Schema G) operator over one of its 'continuations' (i.e. a term in which it is free).

THE SYNTAX-SEMANTICS INTERFACE

(14) **Some Lexical Entries**

⊢ Chris, Chris' : NP, e

⊢ everyone, everyone' : NP, e_t^t ⊢

⊢ someone, someone' : NP, e_t^t

⊢ liked, like' : NP \rightarrow_C NP \rightarrow_S S, $e \rightarrow e \rightarrow t$

⊢ thought, think' : S \rightarrow_C NP \rightarrow_S S, $\pi \rightarrow e \rightarrow t$

(15) **Format for Judgments in Interface Rules**

$\Gamma \vdash a, b : A : B \dashv \Delta$

- a. ‘the pair of syntactic term a of category A and semantic term B of type B is admitted in the context Γ and **co-context** Δ .’
- b. The context is a nonrepeating list of pairs each consisting of a trace and an (uncommitted) semantic variable.

Semicolons in contexts represent concatenation of nonrepeating lists.

- c. The co-context is a set of commitments (semantic operators subscripted by distinctly semantic variables).

Semicolons in co-contexts represent disjoint unions.

The co-context contains **no** syntactic information.

(16) **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 $\Gamma; \Gamma' \vdash (^s a f), (v c) : B, D \dashv \Delta; \Delta'$

- Heads combine with subjects semantically by function application.
- Semicolons in contexts represent concatenation of nonrepeating lists.
- Semicolons in co-contexts represent disjoint unions.

(17) **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 $\Gamma; \Gamma' \vdash (f \ a \ c), (v \ c) : B, D \dashv \Delta; \Delta'$

- Just like the preceding but for complements instead of subjects.
- Modus Ponens for other gramfunns follow the same pattern, **except:**
- Left Conjunct Modus Ponens is additive for contexts, but multiplicative for co-contexts (because there is no ATB Condition for in situ operators).

(18) **Schema T (Trace)**

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

- Traces are paired with semantic variables at birth.
- Compare with the MP, where traces must undergo a multistage process of ‘trace conversion’, whose details are not agreed upon, in order to become semantically interpretable.
- We simplified for expository reasons. Actually we need three interface schemas for traces (for complement, right VP-modifier, and subject traces respectively).

(19) **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)

When a semantic operator is stored, nothing happens in the syntax (because the phrase whose meaning is stored is **not** an operator syntactically).

(20) **Schema R (Retrieval)**

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

When a semantic operator is retrieved, nothing happens in the syntax.

Note: the underscoring of the bound variable is 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).

(21) **Schema G (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)

Fillers ('overtly moved' signs) are operators, both syntactically and semantically, and scope in parallel.

(22) **Comments on Schema G**

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

(23) **Subject Operator Schema (SO)**

If $\Gamma \vdash a, d : A_B^C, D_E^F \dashv \Delta$ and $\Gamma' \vdash v, f : A \dashv_{\circ_S} B, D \rightarrow E \dashv \Delta'$,
then $\Gamma; \Gamma' \vdash \text{so}(a, v), d_x(v x) : C, F \dashv \Delta, \Delta'$
(x fresh)

Recall that this schema is needed because of our assumption that in English only copmlents can have subject traces.

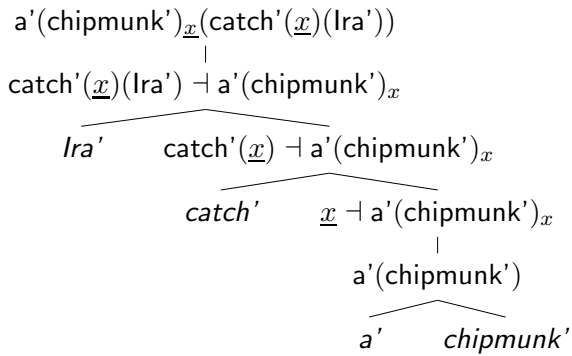
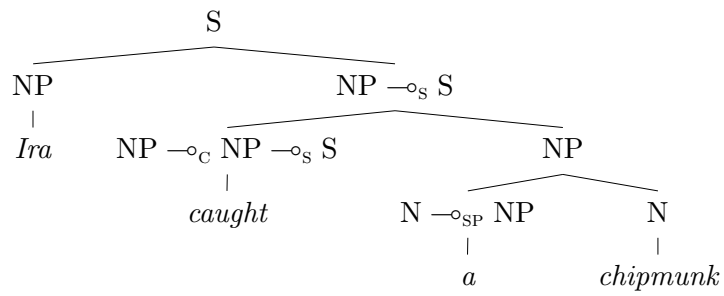
(24) **Schema TC (Trace Contraction)**

If $t, x : A, B; u, y : A, B; \Gamma \vdash c[t, u], d[x, y] : C, D \dashv \Delta$,
then $t, x : A, B; \Gamma \vdash c[t, t], d[x, x] : C, D \dashv \Delta$

Note: There is no analog of Trace Contraction for committed variables! (Stores are nonpermutably relevant; co-stores are linear.)

QUANTIFIER SCOPE

(25) Cooper Storage ('Covert Movement') Example



At the storage and retrieval nodes, nothing happens in the syntax.

(26) **The Transform τ from Semantic Terms to TLC**

Everything stays the same except:

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

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 the CVG semantic calculus or to TLC, e.g. (walk' Kim') vs. walk'(Kim').

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

Operator binding translates into abstraction immediately followed by application.

(27) **Comments on the Transform τ**

- τ makes it appear that CVG semantic proofs are ‘between’ syntax and semantics, rather reminiscent of LF.
- They **are** the closest thing to LF in CVG, but:
- They are independently generated by the semantic logic.
- They are not algorithmically derived from syntax.
- They are completely formally explicit.
- So is their translation into TLC.
- There is no CVG counterpart of ‘crashing at LF’; the closest thing would be a semantic proof that the interface did not pair with any syntactic proof.
- A closer analogy may be with the CPS transform of Bernardi and Moortgat, but τ is much simpler than that.

(28) **Lexicon for QR Fragment**

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

⊢ everyone, everyone' : NP, e_t^t ⊣

⊢ someone, someone' : NP, e_t^t ⊣

⊢ likes, like' : (NP \rightarrow_C (NP \rightarrow_S S), e \rightarrow e \rightarrow t ⊣

⊢ thinks, think' : S \rightarrow_C (NP \rightarrow_S S), $\pi \rightarrow$ e \rightarrow t) ⊣

(29) **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:

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

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

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

(30) **Quantifier Scope Ambiguity**

a. Syntax (both readings):

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

b. Semantics (scoped to lower clause):

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

TLC: $\text{think}'(\lambda_w(\forall_x(\text{person}'(x)(w) \rightarrow \text{like}'(x)(\text{Kim}') (w))))(\text{Chris}')$

c. Semantics (scoped to upper clause):

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

TLC: $\lambda_w(\forall_x(\text{person}'(x)(w) \rightarrow \text{think}'(\text{like}'(x)(\text{Kim}'))(\text{Chris}') (w)))$

Note: The TLC translations are obtained by applying the τ transform to the CVG semantics, applying meaning postulates, and normalizing.

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

- a. Syntax (both readings): (^S everyone (likes someone ^C)) : S
- b. $\forall\exists$ -reading: (everyone' x (someone' y ((like' y) x)))
- c. $\exists\forall$ -reading: (someone' y (everyone' x ((like' y) x)))
- 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 later.

(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. $t_{ee} = e \rightarrow e \rightarrow t$)
- iii. For $n \in \omega$, $A_n =_{\text{def}} A_\sigma$ where σ is the string of e 's of length n

Example: $t_2 =_{\text{def}} t_{ee} =_{\text{def}} e \rightarrow e \rightarrow t$.

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

(33) **A Refinement**

- Actually the QNP meanings have to be polymorphically typed to $e_{t\sigma}^t$ 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 $\sigma = e$: quantifiers can be retrieved at nodes which are semantically individual properties ($t_e = e \rightarrow t$), such as VPs and Ns:
 - a. [Campaigning in every state] is prohibitively expensive.
 - b. Most [people with few interests] are uninteresting.

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

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

(35) Operizers

- Recall that an **operator** is a (syntactic or semantic) term whose type is of the form A_B^C .
- We define an **operizer** to be a functional term whose result type is an operator type.
- An operator can be thought of as a 0-ary operizer.
- Intuitively, an operizer is a ‘movement trigger’: it converts its argument into something that ‘has to move’ to take scope.

(36) **Some Signs with Operizer Semantics**

- ordinary determiners: type $(e \rightarrow t) \rightarrow e_t^t$
- ‘overtly moved’ interrogative determiner *which*: type $(e \rightarrow t) \rightarrow e_{\pi}^{e \rightarrow \pi \rightarrow t}$ (where $\pi =_{\text{def}} \mathbf{s} \rightarrow t$).
- (non-phrasal) comparative *-er*, assuming the *than*-phrase complement denotes a degree: type $d \rightarrow d_d^t$.
- Following (in spirit) Moortgat 1991, we can analyze **pragmatic focus** as an intonationally realized phrasal affix whose semantics has the (polymorphic) operizer type $B \rightarrow B_t^t$.
- And likewise for the *in situ* counterpart of topicalization.
- But first recall how we analyzed overt movement’ topicalization.

(37) Review of Topicalization Syntax (1/2)

- Syntactically, we analyze topicalization as a phrasal affix **top**, which is phonologically realized as a ‘B’ (L+H*) pitch accent on (the phonological realization of) its argument.
- Categorially, **top** is a operizer that converts the phrase it affixes to, of category A , into an operator of category A_S^T :

$$\vdash \mathbf{top} : A \multimap_A A_S^T$$

(38) **Review of Topicalization Syntax (2/2)**

$\vdash ((\text{Sandy top}^A)_t(\text{Kim (likes } t^C))) : T$

- a. In no respect was the topicalized phrase moved or copied from the trace position.
- b. The topicalized phrase is not “adjoined”, nor does it occupy a position analogous to ‘Spec of CP’.
- c. Instead, **top** converts **Sandy** into an operator that scopes over the ‘gappy’ S, binding its NP trace and producing a T.
- d. This is typical of how ‘overt movement’ works in CVG.

(39) Rough-and-Ready Topicalization Semantics

- lexical entry schema for topicalization phrasal affix:
 $\vdash \text{top}, \text{top}' : A \multimap_A A_S^T, B \rightarrow B_t^t \dashv$
- The τ transform maps the semantic operator top' to a family (parametrized by Ty2 meaning types B) of constants of type $B \rightarrow (B \rightarrow t \rightarrow t)$.
- Intuitively, $\text{top}'(b)(P)$ means something like ‘ $P(b)$, but for certain other B ’s, x , contextually associated with b , $\text{whether}'(P(x))$ remains unresolved.’
- $\vdash ((\text{Sandy top}^A)_t^S \text{Kim} (\text{likes } t^C))$,
 $((\text{top}' \text{Sandy}')_x((\text{like}' x) \text{Kim}')) : T, t \dashv$

(40) A Case of Lexical Ambiguity

- 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:

$$\vdash \text{top}_{\text{in-situ}}, \text{top}' : A \text{---}_{\text{A}} A, B \rightarrow B_{\text{t}}^{\text{t}} \dashv$$

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

$$\vdash ({}^{\text{S}} \text{Kim} (\text{likes} (\text{Sandy } \text{top}_{\text{in-situ}}^{\text{A}} \text{C}))), \\ ((\text{top}' \text{Sandy}')_{\underline{x}}((\text{like}' \underline{x}) \text{Kim}')) : \text{S}, \text{t} \dashv$$

- In short, we analyze ‘optional movement’ as lexical ambiguity (of a prosodically realized affix).

**SHOULD OVERT AND COVERT MOVEMENT BE
COLLAPSED?**

(41) **Overt Movement = and Covert Movement?**

- The existence of covert-overt movement ‘twins’ like the English topicalization operators:

$$\vdash \text{top}, \text{top}' : A \multimap_A A_S^T, B \rightarrow B_t^t \dashv$$

$$\vdash \text{top}_{\text{in-situ}}, \text{top}' : A \multimap_A A, B \rightarrow B_t^t \dashv$$

suggests that the two kinds of ‘movement’ should be viewed as subcases of the same general phenomenon, as is often proposed within both TG and CG.

- This is also suggested by the difficulty of adjudicating some cases; for example, comparative subselection gaps (Day 5) can be analyzed either as traces bound by *than* or inaudible Cooper-stored **max** operators that are retrieved within the *than*-phrase.
- But there are many reasons to keep covert and overt movement analytically distinct.

(42) **Reasons not to Collapse Covert and Overt Movement (1/2)**

- The obvious one: if we assume, e.g. QNP's are syntactically just ordinary NPs in argument positions, we don't have to explain why the 'tail of the chain is audible'.
- Overt and covert phenomena do not seem to observe the same island conditions.
- With covert phenomena, a good case can be made that the morphosyntactic features of the operator (such as case or identity of governed preposition) are never checked at the retrieval site.
- With covert phenomena, a good case can be made that only the semantic type of the scope, not its syntactic category, is relevant for licensing retrieval.

(43) **Reasons not to Collapse Covert and Overt Movement (2/2)**

- We explain the Across-the-Board (ATB) condition by assuming the Modus Ponens schema for left conjuncts is additive on contexts but multiplicative on co-contexts.
- We explain many cases of quantifier scope ambiguity by assuming co-contexts are permutable, but we explain the Nested Dependency Condition (NDC) by assuming contexts are not permutable.
- We explain ‘parasitic’ gaps by assuming contexts are contractible, but assuming the same thing about co-contexts makes wrong predictions (e.g. that *every boy saw every boy* has a reading paraphrasable as *every boy saw himself*).
- Overt movement has been demonstrated to exhibit ‘scope marking’ in a wide range of typologically distinct languages (Kikuyu, Chamorro, Irish, Yiddish, etc.), but no such phenomena seem to have been attested for covert movement.