

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/day1ho.pdf>

WHAT THIS COURSE IS ABOUT

(1) **How are Syntax and Semantics Connected?**

- The two most influential theories of the syntax-semantics interface are a formal one due to **Montague** and an informal one due to **Chomsky**.
- The two theories have been with us for nearly four decades.
- Over the decades, and especially within the past decade, the informal, Chomskyan, view has become the **mainstream** position, and the formal, Montagovian, view the **embattled** position.
- The premiss of this course is that **Montague's theory is better, and needs to be defended anew**, in a way that is accessible to the linguistic mainstream.

(2) Defending Montague Anew

- The defense will be presented by reformulating, in logical terms, an elaboration of the Montagovian picture that was developed during the 1970's within the **Extended Montague Grammar** (EMG) community.
- We use the **Curry-Howard correspondence** to show the direct connection between logical proofs and traditional generative grammarians' trees and labelled bracketings.
- But the heart of the argument lies in the simplicity of the linguistic analyses, especially of phenomena that mainstream linguists analyze in terms of **overt and covert movement**.

HOW THE COURSE IS ORGANIZED

- **Day One: Introduction**

Background and Formal Prerequisites

Basics of CVG Syntax

- **Day Two: Overt Movement**

Topicalization and the Empty Category Principle (ECP)

Tough-Movement, Nested Dependency Condition (NDC), Parasitic Gaps, and Across-the-Board (ATB) Condition

- **Day Three: Covert Movement**

Semantics and the Syntax-Semantic Interface

Quantifier Scope, Collapsing Overt and Covert, the Curry-Howard Scandal

- **Day Four: Interrogatives**

Ty2 Review and Interrogative Semantics

Interrogatives in Chinese and English

- **Day Five: Advanced Topics**

Inverse Scope, Parasitic Scope, and Scope Reconstruction

Scope of Comparatives and Coordinate Semantics

HISTORICAL BACKGROUND

(3) **In 1970:**

- Montague's "Universal Grammar" and "English as a Formal Language" were published, proposing that NL syntactic derivations (analysis trees) and their meanings are constructed **in parallel**.

In particular, there was nothing 'between' syntax and semantics.

- Chomsky circulated his "Conditions on Transformations" (published in 1973), introducing the **T-model**, in which interpretive rules apply between SS and LF:

Phonetics ← PF ← SS → LF → Semantics

↑

DS

↑

LEX

(4) What's the Big Difference?

- The main difference is that, on Chomsky's account, there is an inaudible level of syntactic representation—**LF**—which is (in principal, not explicitly) semantically interpreted.
- LF is nondeterministically, algorithmically derived from **SS**, the audible level of syntactic representation.
- Whereas on Montague's account, the closest analog of SS—the **analysis tree**—is directly and explicitly related to semantics by recursive rules, with no intervening syntactic representation.

(5) **And Now, almost Forty Years Later:**

- The existence of LF is still assumed within the current avatar of transformational grammar (TG), the Minimalist Program (MP).
- And the existence of LF is still rejected within the Montague-inspired research traditions such as categorial grammar (CG) and phrase structure grammar (PSG).
- On the face of it, Montague's picture seems more parsimonious and explicit, yet after all these years has not gained mainstream acceptance.

(6) **1970s Terminology**

In 1970's TG (the “**Extended Standard Theory**” or EST):

- the (essentially context-free) rules that built up DS's from lexical items were called **base** rules.
- the rules deriving SS from DS were called **transformations**.
- the rules deriving LF from SS were called **interpretive** rules.

(7) Later Transformational Terminology

- But by the end of the 1970's the latter two kinds of rules were assimilated to the one schematic transformation **move- α** .
- The movements that transform SS to LF are called **covert** because they take place after the 'branch point' and therefore have no effect on how the sentence sounds.
- By contrast, the movements that take place between DS and SS are called **overt**.
- And the rules that stick subtrees together, rather than moving a subtree within a tree (the analogs of the former base rules) are rechristened **merges**.

(8) **The Cascade**

Straightening the right arm of the T and suppressing the left arm:

Semantics

$\uparrow?$

LF

\uparrow_C

SS

\uparrow_O

DS

\uparrow_M

LEX

with the subscripts on the arrows distinguishing the three rule cycles Merge, Overt Move, and Covert Move.

(9) A Convergence of Views?

- The Cascade has long since been rejected—by all—because (in mainstream parlance) the three kinds of operations have to be **intermingled**: merges must be able to follow moves, and overt moves must be able to follow covert ones. Therefore, in current MP:
 - – There is only a single cycle of operations.
 - DS and SS do not exist.
 - There are multiple points in a derivation where the syntax connects to the interface systems.
 - This sounds pretty much like Montague grammar, especially if one takes the (typical categorial grammar) position that the analysis tree is not a “level of representation” (such as SS) but just the history of a how a string got paired with a meaning.

(10) **No Convergence Yet**

- No, TG has not merged with Montague Grammar, not yet anyway.
- Movement is still there, though decomposed (since Chomsky 1993) into Copy and (Internal) Merge.
- There are still issues about Overt vs. Covert (now framed in terms of which end of the chain is pronounced).
- There are still issues about under what conditions Merge can be 'late'.
- There are still issues about the details of how traces are converted into something semantically interpretable.
- The MP still does not accept the idea that every node in the syntactic derivation has a semantics and that syntax and semantics are built up in parallel.

(11) Moving Forward

- Even though TG and post-EMG haven't converged, the fact that they now agree on so much is a sign that progress is being made.
- How can we make headway in resolving the remaining differences?
- In 1980, Chomsky asserted that Gazdar's new nontransformational grammar was just a 'notational variant' of TG.
- Gazdar said that could only be ascertained if TG were formulated with a comparable degree of explicitness.
- Thanks to work in the past decade on formalizing the MP (e.g. Stabler, Vermaat, Retoré, Lecomte, Amblard), it is becoming easier to make scientifically meaningful comparisons between transformational and nontransformational theories.
- This course aims to retell the EMG story in a way that makes it as easy as possible to see how it differs from the 'mainstream' view.

WHAT WAS EMG?

(12) **Extended Montague Grammar (EMG)**

- EMG emerged in the mid 1970s as an alternative to Chomsky's Revised Extended Standard Theory (REST).
- It was influenced by mathematical logic (especially model theory) and computer science.
- It sought greater simplicity, precision, and tractability.
- It included practitioners of:
 - PSG, e.g. Cooper, Gazdar, Pullum
 - CG, e.g. Dowty
 - switch hitters, e.g. Bach.
- CG then (essentially AB grammars) was very similar to PSG.
- Lambek invented his calculus 20 years earlier, but linguists didn't start getting interested in it until the mid-1980's.

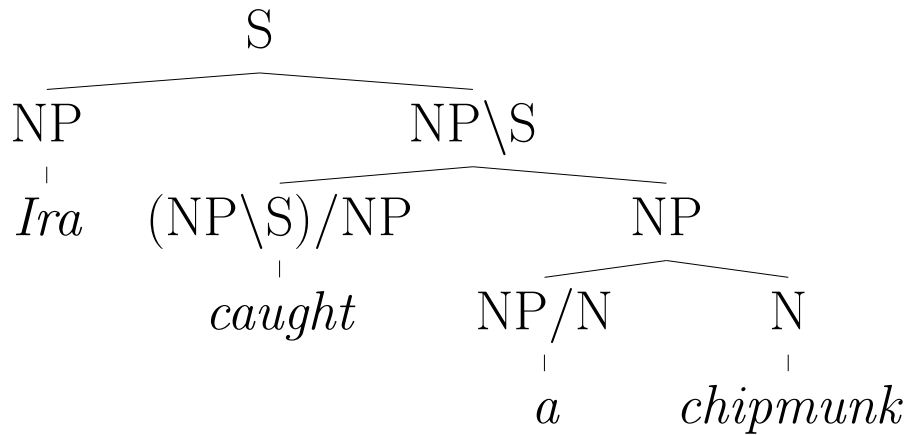
(13) **Three Signal Achievements of EMG**

- Cooper's (1975) **storage** replaced **covert** movement.
- Gazdar's (1979) **linking schemata** replaced **overt** movement.
- Bach and Partee (1980) incorporated both into a PSG-based account of (what would later be called) **binding theory** facts.

(14) **Cooper Storage and Retrieval (Intuitively)**

- a. Imagine that while semantically interpreting a syntactic derivation bottom up, you encounter a quantified noun phrase with meaning $a : e \rightarrow e \rightarrow t$ in a position where a meaning of type e is called for (say, object of a transitive verb).
- b. Then (**storage**) you can replace a by a fresh variable $x : e$ 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 $b : t$, then (**retrieval**) you can remove a_x from the store and **scope** it over $b : t$, obtaining $a(\lambda_x b) : t$.
- 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).

(15) **A Simple Example: Syntax**



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

(17) **Notation for Cooper Storage**

- a. We use a backwards turnstile \dashv to demarcate nonempty stores.
- b. We underscore variables introduced by storage, so we can recover how the meaning was composed (without drawing the tree) just from the root meaning term

$$a'(\text{chipmunk}')_{\underline{x}}(\text{catch}'(\underline{x})(\text{lra}'))$$

- c. We'll make all of this much more precise in due course.

(18) **Observations about Cooper Storage**

- a. In most CG, semantic composition goes in in lock-step with the syntactic composition, e.g.
 - Words correspond to semantic basic constants.
 - Left/right application correspond to application.
- b. But nothing in the syntax corresponds to storages and retrievals in the semantics.
- c. So: semantic interpretation is not a **function** from syntactic derivations to meanings.

(19) More Observations about Cooper Storage

- a. For most categorial grammarians, the non-functional nature of the syntax-semantics interface puts Cooper storage beyond the pale.
- b. in CG, the usual approach is to make the **syntax** more complicated to preserve the functionality of the syntax-semantics interface.
- c. But we don't know of any scientific basis for the assumption that the syntax-semantics interface is a function.
- d. Moreover, once Cooper storage is implemented in a logical way, it is much simpler and more elegant than is generally believed.
- c. We are going to follow EMG (and HPSG) and use (a natural-deduction version of) Cooper storage in the semantics.
- d. This will keep the syntax very simple.

(20) **A Gazdar Linking Schema**

- Gazdar (1979,1981), working within CFG, used schemata like the following (for Topicalization) to analyze so-called ‘overt movement’ phenomena:

$$\langle 44, [S \alpha S/\alpha], \lambda_h((S/\alpha)'(\alpha')) \rangle$$

- The labelled bracketing allows combination of an α and an S with an α - ‘hole’, forming an S.
- The semantic recipe, in one fell swoop, binds the variable h corresponding to the hole, and applies the result to the semantics of the α .

(21) **Some Observations about Gazdar Linking**

- HPSG generalized Gazdar's / to the list-valued SLASH feature.
- HPSG's Filler-Head Schema schematized over Gazdar's linking schemata.
- But HPSG's feature-structure encoding obscured the inherent logic behind Gazdar linking.
- We will reformulate Gazdar linking natural-deduction (ND) style, with the HPSG SLASH replaced by the ND **variable context**.

(22) **EMG after 1980**

- EMG spawned CCG, HPSG, TLG, ACG, etc.
- In spite of the many important contributions made within these frameworks, none of them capture the simplicity and elegance of the intuitions behind Cooper storage and the Gazdar schemata.
- I'll present a logical reconstruction of EMG that tries to do that.
- This reconstruction is called **convergent grammar** (CVG).

A PREVIEW OF CVG

(23) CVG: a Look Ahead

- CVG is closely related to both ACG and HPSG.
- Like ACG—but unlike other frameworks descended from EMG—CVG uses **Curry-Howard proof terms** (which we will explain) to denote NL syntactic entities.
- This makes it easy to connect CVG to mainstream generative grammar because the proof terms are really just a more precise version of EST/GB-style labelled bracketings.
- Like HPSG—but unlike other frameworks descended from EMG—the relation between syntax and semantics is not a function, but rather is one-to-many.
- This is another commonality between CVG and mainstream generative grammar, where the derivation of LF from overt syntax is nondeterministic.

(24) CVG Compared with ACG and HPSG

- Like both ACG and HPSG, CVG has a **parallel** architecture: independent components generate candidate syntactic and semantic entities (and in principle phonological ones too).
- Like ACG, these candidate entities are **proofs**, each in a different logic (not feature structures as in HPSG).
- We call this property **pure derivationality**, as opposed to the kind of derivationality in TG that builds arboreal representations by sequences of structural operations.
- Like HPSG, CVG is **weakly syntactocentric** (see below).

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

- A **syntactocentric** framework is one where there is no **direct** interface between semantics and phonology.
- Instead, that relationship 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, not algorithms that take syntax as input.
- In the case of the CVG syntax-semantics interface, this arises from the use of a form of **Cooper storage and retrieval**.

(26) **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.

CONVERGENT GRAMMAR: A LOGICAL RECONSTRUCTION OF EMG

- Background on Natural Deduction
- Syntax
- Semantics
- The Syntax-Semantics Interface

BACKGROUND: NATURAL DEDUCTION (ND)

(27) **ND Introduction**

- We review a style of ND called **Gentzen ND with Curry-Howard proof terms**, hereafter simply ND.
- We illustrate how ND works by giving a proof theory for a simple kind of propositional logic, the **minimal logic** of implication.
- Later, we'll use ND for semantic and syntactic derivations.

(28) **Minimal Logic (ML)**

- We start with some **atomic formulas** X, Y, Z, \dots , and form more formulas from them using the **implication** connective \rightarrow .
- Notational conventions:
 A, B , and C range over ML formulas.
 $A \rightarrow (B \rightarrow C)$ is abbreviated as $A \rightarrow B \rightarrow C$.

(29) **Theorems**

- There are many kinds of proof systems for ML: Hilbert-axiomatic, sequent calculus, various kinds of natural deduction, etc., but they all agree on what the **theorems** should be.

- For example, these are theorems:

$$A \rightarrow A, A \rightarrow (A \rightarrow B) \rightarrow B, (A \rightarrow B) \rightarrow (B \rightarrow C) \rightarrow (A \rightarrow C)$$

- But these are not:

$$A \rightarrow B, A \rightarrow A \rightarrow B, ((A \rightarrow B) \rightarrow A) \rightarrow A$$

(30) **Curry-Howard Correspondence (1/2)**

- Gentzen (1934) invented sequent-style ND.
- Howard (1969, published 1980), elaborating on observations of Curry (1934, 1958), showed that terms of typed lambda calculus (TLC) could be thought of as ND proofs.
- Subsequently this idea, called the **Curry-Howard correspondence** (CH) has been extended to many different kinds of logic.
- The basic ideas of CH are that, if you let the atomic formulas be the types of a TLC, then
 1. **a formula is the same thing as a type.**
 2. **A formula A has a proof iff there is a combinator (closed term containing no basic constants) of type A .**
- Hence the Curry-Howard slogan:
formulas = types, proofs = terms

(31) **Notation for ND Proof Theory**

- An ND proof theory consists of **inference rules**, which have **premisses** and a **conclusion**.
- An *n*-**ary** rule is one with *n* premisses, and a 0-ary rule is called an **axiom**.
- Premisses and conclusions have the format of a **judgment**:

$$\Gamma \vdash a : A$$

read ‘*a* is a proof of *A* with hypotheses Γ ’.

(32) **Judgments**

In a judgment

$$\Gamma \vdash a : A$$

- A is a formula/type
- a is a term/proof
- Γ , the **context** of the judgment, is a set of variable-formula pairs of the form $x : A$.

(33) **Some Rule Schemas for ML**

Hypotheses:

$x : A \vdash x : A$ (x a variable of type A)

Nonlogical Axioms:

$\vdash a : A$ (a a basic constant of type A)

Modus Ponens:

if $\Gamma \vdash f : A \rightarrow B$ and $\Gamma' \vdash a : A$,

then $\Gamma, \Gamma' \vdash f(a) : B$

Hypothetical Proof:

if $x : A, \Gamma \vdash b : B$,

then $\Gamma \vdash \lambda_x b : A \rightarrow B$

This subsystem of ML, called **implicative intuitionistic linear logic**, is all we need for present purposes. Additional (**structural**) rules are needed for full ML.

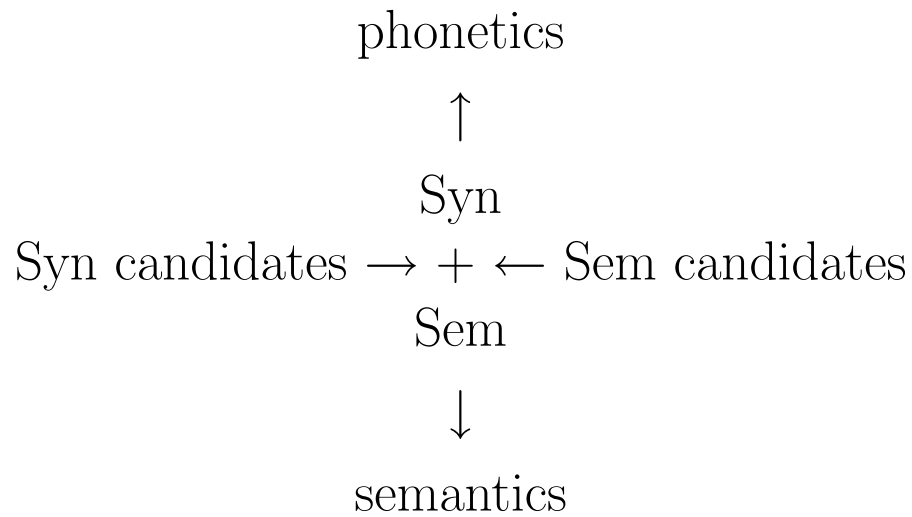
(34) **Curry-Howard Correspondence (2/2)**

- Variables correspond to **hypotheses**.
- Basic constants correspond to **nonlogical axioms**.
- Derivability of $\Gamma \vdash a : A$ corresponds to A being **provable** from the hypotheses in Γ .
- Application corresponds to **Modus Ponens**.
- Abstraction corresponds to **Hypothetical Proof**.

(36) Reformulating EMG using ND

- We have **two** logics, each with an ND proof theory.
- The **syntax-semantics interface** recursively defines the the set of syntax-semantics proof-pairs that belong to the NL in question.
- We call those pairs the **signs** of the NL.
- Signs are inputs to the interpretive interfaces:
 - the syntax is phonetically interpreted, and
 - the semantics is semantically interpreted.

(37) **Parallel-Derivational (PD) Architecture**



SYNTAX

(38) **ND-Style Syntax**

- The inference rules are the **syntax rules**.
- The formulas/types are the **syntactic categories**.
- The proofs/terms are the **syntactic expressions**.
- The basic constants are the **syntactic words**;
- The variables are **traces**.
- The context of a judgment is the list of traces still **unbound** at that point in the proof.

(39) Categories

- **Basic** categories, such as S , NP , and N .
- **Function** categories: if A and B are categories, so is $A \multimap_F B$, for F a **grammatical function name (gramfun)**.
 A is called the **argument** category and B the **result** category.
- **Operator** categories: if A , B , and C are categories, so is $G[A, B, C]$, abbreviated A_B^C .
 A , B , and C are called the **binding** category, the **scope** category, and the **result** category.

(40) **Basic Categories**

- To get started: S, NP, and N. Others will be added as needed.
- Here we ignore morphosyntactic details such as case, agreement, and verb inflection.
- In a more detailed CVG, these would be handled (much as in pregroup grammars) by **subtyping**.

(41) **Function Categories**

- As in many frameworks (RG, HPSG, LFG, DG, traditional grammar) grammatical functions (gram-funs) like **subject** and **complement** are treated as theoretical primitives.
- To start we just assume the gramfuns SUBJECT (S) and COMPLEMENT (C). Others will be added as needed.

(42) More on Function Categories

- At first the identities of the gramfunns will not be important, so if you are used to Lambek calculus you can just think of $A \multimap_S B$ as $A \setminus B$ and $A \multimap_C B$ as A/B .
- Or if you are used to ACG you can think of both as $A \multimap B$.

(43) Operator Categories

- Operator categories, such as NP_S^Q for ‘overtly moved’ interrogative *who*, are a type-theoretic reconstruction of Gazdar’s (1979/1981) analysis of unbounded dependencies.
- The G constructor is inspired by Moortgat’s (1991) **q**-constructor, which was used for covert (not overt) movement.
- A standard TLG way to get the effect of A_B^C is $C/(B\uparrow A)$ where \uparrow is Moortgat’s (1988) extraction constructor.

(44) **Some Syntactic Words**

- ⊢ Chris : NP
- ⊢ everyone : NP
- ⊢ someone : NP
- ⊢ $\text{who}_{\text{in-situ}}$: NP
- ⊢ $\text{what}_{\text{in-situ}}$: NP
- ⊢ who_{fill} : NP_S^Q
- ⊢ $\text{what}_{\text{fill}}$: 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

(45) **Quantified NPs (QNPs)**

- e.g. *everyone, someone*
- They are simply NPs, as in PSG.
- It's their **meanings** (generalized quantifiers) that take scope.
- The scoping of those GQ's will be handled over in the **semantic** logic.

(46) ***Wh*-Expressions**

- These are **syntactically** ambiguous.
- As NPs, they occur “in situ”.

The scoping of the semantic operators that are the **meanings** of in situ *wh*-expressions are handled over in the **semantic** logic.

- As syntactic operators (NP_S^Q), they occur in nonargument (“filler”) positions and bind traces in argument positions.

These correspond to ‘overtly moved’ expressions in TG, but they were not moved or copied.

(47) **Four Syntactic Schemata**

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 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 T (Trace)

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

Schema G (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)

(48) **Modus Ponens in the Syntax (1/2)**

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

(49) **Modus Ponens in the Syntax (2/2)**

- These schemata are the CVG counterparts of
 - HPSG’s Subject-Head and Head-Complement schemata
 - CG’s /- and \-Elimination schemata
 - TG’s Merges.
- The ‘passing up’ of the contexts is like SLASH inheritance in HPSG.
- The proof terms mnemonically reflect the temporal order in which the phonologies of the premisses are realized.

(50) **Schema T (Trace Schema)**

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

- This is just the usual ND schema for positing hypotheses/variables.
- It is the CVG counterpart of HPSG's Trace schema.

(51) **More about Traces**

- 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, a trace is an **inaudible copy** of the operator that binds it.
- So the TG sense of ‘syntactic variable’ is something different from the usual logical notion.

(52) **Schema G (Generalized Gazdar Schema)**

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

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

- This is the CVG counterpart of HPSG's Filler-Head schema, which in turn derives from Gazdar's (1979/1981) Linking schemata.
- Schema G introduces a proof term of the form $a_t b$, in which the free occurrence of t in b is bound.
- The binding of the trace and the scoping of the syntactic operator are compressed into one step.

(53) **More about Schema G**

- There is no schema of Hypothetical Proof as such: the effect is as if the t hypothesis was withdrawn to prove $A \rightarrow B$, and then an $(A \rightarrow B) \rightarrow C$ was immediately applied to that to prove C .
- The operator was not moved or copied from the trace position, so there is no issue about ‘which end of the chain’ is pronounced.

(54) **A Simple Sentence**

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

- No, I didn't forget to draw the tree. You can reconstruct it from the proof term.
- No, I didn't forget the category labels on the sub-terms. You can reconstruct them from the categories of the words, which are easy to remember.
- The proof term is essentially similar to an EST/GB labelled bracketing, except that it also tells the grammatical function of the syntactic arguments.

(55) **An Embedded Polar Question**

$\vdash (\text{whether } (^s \text{ Kim (likes Sandy } ^c)) ^c) : Q$

We introduce a new category Q for embedded interrogative sentences.

(56) **An Embedded Constituent Question**

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

- Here *what* is an operator of type $\text{NP}_{\text{S}}^{\text{Q}}$: it combines with an S containing an unbound NP trace to form a Q, while binding the trace.
- Notice that the is **not** analyzed as a “projection” of a ‘functional category’: there is no null complementizer with respect to which the operator is a “specifier”.

(57) **A Binary Constituent Question**

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

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

(58) **A Baker Question**

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

- Here, both *who*'s are operators, but *what* is just an NP.
- Sentences like this, discussed by Baker (1970), are (famously) ambiguous with respect to the scope of the in situ *wh*- expression.
- We'll deal with the semantics in due course.