The Logics of Overt and Covert Movement in a Relational Type-Theoretic Grammar

Carl Pollard
Ohio State University and Universitat Rovira i Virgili

Fourth Workshop on Lambda Calculus and Formal Grammar
LORIA
Nancy, Sept. 18—19, 2007

These slides are available at:

`http://www.ling.ohio-state.edu/∼hana/hog/`

OVERVIEW OF THIS TALK

- Review the distinction between **syntactocentric** vs. **relational** grammar architectures
- Introduce the relational type-theoretic framework **convergent grammar** (CVG)
- Review **extraction** (**overt movement**) in CVG
- Background on **hyperintensional semantics**

OVERVIEW OF THIS TALK (CONTINUED)

- **covert movement in CVG**

  Case study 1: **quantifier raising**

  Case study 2: ***in situ* wh-construal**

SYNTACTOCENTRIC VS. RELATIONAL GRAMMAR ARCHITECTURE

4

(1) **Does Syntax Exist?**

- TLG says emphatically *no*: the things assigned to syntactic types are just prosodic/semantic *pairs*, not triples:

  "the term 'syntax' we reserve for the mediation of prosodics and semantics by sentential grammar" (Morrill et al. 2007)

- But current 'Curryesque' frameworks (ACG, λG, GF, HOG, etc.) speak of **tectostructures** (aka **abstract syntactic structures**), with their own Curry-Howard term calculus.
- The Curryesque approach is reminiscent of Lambek 1988 and 1995, where syntactic entities exist as arrows in a (biclosed monoidal) category, and semantic interpretation is a (structure-preserving) functor from the syntactic category to a (cartesian closed) semantic one (more specifically, a topos).
- Of course the Curryesque approach is also reminiscent of many non-type-theoretic architectures (P&P, LFG, HPSG, etc.).

(2) **A Red Herring**

- Whether syntax exists is a red herring, at least as far as type-theoretic grammar architectures are concerned.
- That's because TLGs can be recast into Curryesque architecture, by representing syntactic derivations using a suitable system of syntactic proof terms with constants (syntactic words), e.g. bilinear lambda calculus in the case of Lambek calculus.
- Then (analogs of) Lambek's semantic interpretation functor corresponds to a translation from the syntactic calculus to the semantic one:
  - replace syntactic types with corresponding semantic types
  - replace syntactic words with lexically specified translations
  - erase the directional subscripts on lambdas and evals

(3) **TLG vs. Curryesque Grammar Clarified**

- Recasting TLGs in the Curryesque manner shows that presence vs. absence of a distinct level of tectostructure is not a significant difference between TLG and the Curryesque grammar architecture.
- Rather, it is just a presentational difference.
- The *significant* difference between TLG and Curryesque grammar is multimodality vs. unimodality.

(4) **The "Curry-Howard Semantics" Scandal**

- There is a widespread misconception about the connection between the "Curry-Howard corresondence" and semantic interpretation, which involves two mistakes:
  - Mistake 1: thinking that typed lambda calculus (TLC) has a special connection to semantics. (It *seems* that way to many linguists because they first learn TLC as a notation for meanings.)
  - Mistake 2: thinking that the semantic terms that categorial grammarians annotate syntactic proofs with are Curry-Howard terms. (They aren't; rather they are the result of applying a 'secret' translation function to the *real* Curry-Howard terms.)
- In reality, a wide range of propositional logics correspond to term calculi with lambda-like binders; implicative intuitionistic logic and 'semantic' TLC is just a special case.

(5) **The "Curry-Howard Semantics" Clarified**

- The function from syntactic terms to semantic ones characteristic of the Curryesque architecture is the result of composing Lambek's semantic interpretation functor with *two* Curry-Howard correspondences, one syntactic and the other semantic.
- This is what is really going on when CG folk speak of a syntax-semantics 'homomorphism': it is not just a mapping from syntactic types to semantic types, but crucially involves the *inhabitants* of the types.
- This is because (categorical) functors operate both at the level of objects (= types) and arrows (= (equivalence classes of) terms).
- So TLG really does have syntax, even if TLG practicioners hide it by not writing the syntactic proof terms.

(6) **The Syntactocentric/Relational Distinction**

- We *make explicit* that our grammars license not just prosodic/semantic pairs, but rather prosodic/syntactic/semantic triples.
- Let S be the set of all entities that occur as the syntactic component of some triple licensed by the grammar.
- Then is there a function pros from S to prosodies, and a function sem from S to meanings, such that every triple licensed by the grammar is of the form $\langle \mathsf{pros}(s), s, \mathsf{sem}(s) \rangle$ for some $s \in S$?
- That is, are the sound and meaning of a licensed triple uniquely determined by its syntactic component?

(7) **Syntactocentric vs. Relational Frameworks**

- If yes, the framework is **syntactocentric**. Examples:
  - (Classical) Montague Grammar (with analysis trees as the syntactic entities)
  - The Curryesque frameworks (ACG, λG, GF, HOG, etc.), including Curryesque presentations of TLG.
- If no, the framework is **relational**. Examples:
  - Flexible Montague Grammar
  - HPSG
  - Simpler Syntax (Culicover and Jackendoff)

(8) **Some Obervations**

- Historically, the relational architecture has *mostly* been limited to *non*-type-theoretic frameworks.
- The only exception is Flexible MG.
- But from the point of view of syntax, Flexible MG (like Classical MG) is type-theoretically impoverished, in the sense that there is nothing in syntax analogous to lambda-binding (i.e. there is no hypothetical proof for syntactic derivations).

AN INTRODUCTION TO CONVERGENT GRAMMAR (CVG)

(9) **Why CVG?**

- The main idea of CVG is to generalize type-theoretic grammar from a syntactocentric to a relational architecture.
- So (as in the type-theoretic frameworks discussed above), prosody, syntax, and semantics are all represented type-theoretically, each with its system of Curry-Howard terms.
- In particular, there are binding operators in syntax. (Note that this gives CVG something in common with P&P.)
- But as in HPSG and Flexible MG, the syntax-semantics interface is relational, not a function. (And likewise for the syntax-prosody interface, which we mostly ignore here.)
- The resulting framework is a kind of type-logical HPSG.
- But it can also easily be seen as a kind of type-theoretic implementation of GB (but with syntax 'parallel to' semantics, rather than 'cascaded' into LF).

(10) **CVG has Three Levels**

- Prosodics: as in TLG, phonology broadly construed to include word order, external sandhi, phrasing, pitch accents, etc.
- Syntax:
  - Has its own Curry-Howard proof term system (like Curryesque grammar)
  - But it's multimodal (like TLG), and some of the modes have lambda abstraction (like Curryesque grammar)
- Semantics: possible-worlds, but of the hyperintensional persuasion, with basic types for propositions and individuals (and worlds defined as ultrafilters of propositions).

(11) **CVG is Relational**

- Neither the prosodics-syntax relation pros nor the syntax-semantics relation sem is assumed to be a function.
- So, strictly speaking, CVG is not syntactocentric.
- But the relationshop between prosodics and semantics is taken to be the *composition* sem $\circ$ pros of the syntax-semantics and the prosodics-syntax relations.

(12) **Weak Syntactocentricity**

- This architecture might still be called **weakly** syntactocentric in the following sense:
- Syntax always mediates between prosodics and semantics; there is no *direct* sound-meaning correspondence.
- So our criterion for 'putting something in syntax' is whether there is a prosodics-semantics correlation that it helps account for.
- We say s **mediates** between p and m, written $\mathsf{med}(p, s, m)$, provided p pros s and s sem m.
- So the mediation relation is the set of prosodics/syntax/semantics triples licensed by the grammar.

(13) **Formalizing the Architecture**

We have several different kinds of typing judgments (temporarily ignoring the hypotheses on the left-hand side of the turnstyle):

- $\vdash_{\mathsf{p}} a : A$ asserts that a is a prosodic entity of type A
- $\vdash_{\mathsf{s}} a : A$ asserts that a is a syntactic entity of type A
- $\vdash_{\mathsf{m}} a : A$ asserts that a is a semantic entity of type A
- $\vdash_{\mathsf{pros}} a, b : A, B$ presupposes that $\vdash_{\mathsf{p}} a : A$ and $\vdash_{\mathsf{s}} b : B$; and asserts that a pros b, i.e. that a and b are connected by the prosody/syntax interface.
- $\vdash_{\mathsf{sem}} a, b : A, B$ presupposes that $\vdash_{\mathsf{s}} a : A$ and $\vdash_{\mathsf{m}} b : B$; and asserts that a sem b, i.e. that a and b are connected by the syntax/semantics interface.
- $\vdash_{\mathsf{med}} a, b, c : A, B, C$ presupposes that $\vdash_{\mathsf{p}} a : A$, $\vdash_{\mathsf{s}} b : B$, and $\vdash_{\mathsf{m}} c : C$; and asserts that $\mathsf{med}(a, b, c)$, i.e. that b mediates between a and c.

(14) **A CVG Lexical Entry**

$$\vdash_{\mathsf{med}} /\mathsf{plg}/, \mathsf{pig}, \mathsf{pig'} : \mathrm{Pros}, \mathrm{N}, \mathrm{Ind} \supset \mathrm{Prop}$$

(15) **Narrowing the Focus**

- To keep within time limits, from here on out we will ignore prosodics, and focus on the sem relation, i.e. the syntax-semantics interface.
- Concomitantly, we'll concentrate on a fragment within which the syntax-prosodics interface actually *is* functional (even though this is not generally the case).
- For this fragment, the syntax-prosodics interface can be described informally as a translation from syntactic terms to prosodic terms that:
  - replaces each syntactic word with its lexical phonology, and
  - erases all the syntactic term constructors.

(16) **Syntax Mediates Between Prosodics and Semantics**

- So we will never syntactically encode a dependency merely to try to make sem be a function (we don't care whether it is, and it won't be).
- Rather, we syntactically encode only those dependencies that are correlated with one or more prosodic reflexes. Examples:
  - **Extraction (overt movement)**, which involves **displacement**, **sensitivity to morphosyntactic features** (such as case, verb inflection, and identity of governed prepositions), and (in many languages) **morphosyntactic scope marking**
  - **right node raising**, which involves **morphosyntactic matching** between the raised material and the 'ellipsed' material in the first conjunct
  - **phrasal comparatives**, which involve both **morphosyntactic matching** and **pitch-accentual marking** of the 'associate' phrase and the *than*-complement.

(17) **Nonsyntactic Dependencies**

- By contrast, many other kinds of dependencies will be treated as purely semantic and not encoded in syntax, e.g.
  - **covert movement**, including **quantifier scope**, ***in situ* wh-construal**, and **comparative operator construal**
  - **pronoun-antecedent dependencies** (bound anaphora)
  - construal of 'trices' (missing specifiers in **comparative subdeletion**)
  - presuppositions associated with uses of **names**
- Philosophically, this approach is akin to Flexible MG.
- Technically, it can be seen as a proof-theoretic implementation of Cooper storage that is free of syntactic (and prosodic) entanglements.

(18) **Local vs. Nonlocal Dependencies**

- Like HPSG, CVG distinguishes between

  local (or **valence**) dependencies, e.g. SU **(subject),** SP **(specifier),** C **(complement)**, and M **(marked)**, and

  nonlocal dependencies, such as SLASH (for extraction), QS (for quantifier scope), and WH (for wh-construal).

- All are treated as type constructors, but the local ones in particular are treated as flavors of implication with Modus Ponens, but without Hypothetical Proof.

(19) **Some CVG Semantic Constants**

$\vdash_{\mathsf{m}}$ Fido' : Ind

$\vdash_{\mathsf{m}}$ Felix' : Ind

$\vdash_{\mathsf{m}}$ Mary' : Ind

$\vdash_{\mathsf{m}}$ rain' : Prop

$\vdash_{\mathsf{m}}$ bark' : Ind $\supset$ Prop

$\vdash_{\mathsf{m}}$ bite' : (Ind $\wedge$ Ind) $\supset$ Prop

$\vdash_{\mathsf{m}}$ give' : (Ind $\wedge$ Ind $\wedge$ Ind) $\supset$ Prop

$\vdash_{\mathsf{m}}$ believe' : (Ind $\wedge$ Prop) $\supset$ Prop

$\vdash_{\mathsf{m}}$ bother' : (Prop $\wedge$ Ind) $\supset$ Prop

(20) **Some CVG Syntactic Words**

$\vdash_{\mathsf{s}} \mathsf{it}_{\mathsf{dum}} : \mathrm{It}$

$\vdash_{\mathsf{s}} \mathsf{there}_{\mathsf{dum}} : \mathrm{There}$

$\vdash_{\mathsf{s}} \mathsf{that} : \mathrm{Fin} \multimap_{\mathrm{M}} \bar{\mathrm{S}}$

$\vdash_{\mathsf{s}} \mathsf{Fido}, \mathsf{Felix}, \mathsf{Mary} : \mathrm{NP}$

$\vdash_{\mathsf{s}} \mathsf{rained} : \mathrm{It} \multimap_{\mathrm{SU}} \mathrm{Fin}$

$\vdash_{\mathsf{s}} \mathsf{barked} : \mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}$

$\vdash_{\mathsf{s}} \mathsf{bit} : \mathrm{NP} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin})$

$\vdash_{\mathsf{s}} \mathsf{gave} : (\mathrm{NP} \circ \mathrm{NP}) \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin})$

$\vdash_{\mathsf{s}} \mathsf{believed}_1 : \mathrm{Fin} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin})$

$\vdash_{\mathsf{s}} \mathsf{believed}_2 : \bar{\mathrm{S}} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin})$

$\vdash_{\mathsf{s}} \mathsf{bothered}_1 : \mathrm{NP} \multimap_{\mathrm{C}} (\bar{\mathrm{S}} \multimap_{\mathrm{SU}} \mathrm{Fin})$

$\vdash_{\mathsf{s}} \mathsf{bothered}_2 : (\mathrm{NP} \circ \bar{\mathrm{S}}) \multimap_{\mathrm{C}} (\mathrm{It} \multimap_{\mathrm{SU}} \mathrm{Fin})$

(21) **Some CVG Lexical Entries (Prosodics Omitted)**

$\vdash_{\mathsf{sem}} \mathsf{it}_{\mathsf{dum}}, * : \mathrm{It}, \mathrm{T}$

$\vdash_{\mathsf{sem}} \mathsf{there}_{\mathsf{dum}} : \mathrm{There}, \mathrm{T}$

$\vdash_{\mathsf{sem}} \mathsf{that}, \lambda_p p : \mathrm{Fin} \multimap_{\mathrm{M}} \bar{\mathrm{S}}, \mathrm{Prop} \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{Fido}, \mathsf{Fido'} : \mathrm{NP}, \mathrm{Ind}$

$\vdash_{\mathsf{sem}} \mathsf{rained}, \lambda_u \mathsf{rain'} : \mathrm{It} \multimap_{\mathrm{SU}} \mathrm{Fin}, \mathrm{T} \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{barked}, \mathsf{bark'} : \mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}, \mathrm{Ind} \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{bit}, \lambda_y \lambda_x \mathsf{bite'}(x, y)$:
$\mathrm{NP} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Ind} \wedge \mathrm{Ind}) \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{gave}, \lambda_{y,z} \lambda_x \mathsf{give'}(x, y, z)$:
$(\mathrm{NP} \circ \mathrm{NP}) \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Ind} \wedge \mathrm{Ind}) \supset (\mathrm{Ind} \supset \mathrm{Prop})$

(22) **More CVG Lexical Entries (Prosodics Omitted)**

$\vdash_{\mathsf{sem}} \mathsf{believed}_1, \lambda_p \lambda_x \mathsf{believe'}(x, p)$:
$\mathrm{Fin} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Prop} \wedge \mathrm{Ind}) \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{believed}_2, \lambda_p \lambda_x \mathsf{believe'}(x, p)$:
$\bar{\mathrm{S}} \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Prop} \wedge \mathrm{Ind}) \supset \mathrm{Prop}$

$\vdash_{\mathsf{sem}} \mathsf{bothered}_1, \lambda_x \lambda_p \mathsf{bother'}(p, x)$:
$\mathrm{NP} \multimap_{\mathrm{C}} (\bar{\mathrm{S}} \multimap_{\mathrm{SU}} \mathrm{Fin}), \mathrm{Ind} \supset (\mathrm{Prop} \supset \mathrm{Prop})$

$\vdash_{\mathsf{sem}} \mathsf{bothered}_2 : \lambda_{x,p} \lambda_t \mathsf{bother'}(p, x)$:
$(\mathrm{NP} \circ \bar{\mathrm{S}}) \multimap_{\mathrm{C}} (\mathrm{It} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Ind} \wedge \mathrm{Prop}) \supset (\mathrm{T} \supset \mathrm{Prop})$

(23) **CVG Rules for Local Dependencies I: Fusion**
Fusion is used to form the analogs of HPSG 'COMPS-lists', for verbs that take multiple complements.
Fusion: If $\vdash_{\mathsf{sem}} a, c : A, C$ and $\vdash_{\mathsf{sem}} b, d : B, D$
then $\vdash_{\mathsf{sem}} a \cdot b, (c, d) : A \circ B, C \wedge D$

(24) **CVG Rules for Local Dependencies II: Merges**
Merges are flavors of Modus Ponens (or eval) corresponding to HPSG's VALENCE features.

Su-Merge: If $\vdash_{\mathsf{sem}} a, c : A, C$ and $\vdash_{\mathsf{sem}} f, v : A \multimap_{\mathrm{SU}} B, C \supset D$ then $\vdash_{\mathsf{sem}} ({}^{\mathrm{SU}}\ a\ f), v(c) : B, D$

Sp-Merge: If $\vdash_{\mathsf{sem}} a, c : A, C$ and $\vdash_{\mathsf{sem}} f, v : A \multimap_{\mathrm{SP}} B, C \supset D$ then $\vdash_{\mathsf{sem}} ({}^{\mathrm{SP}}\ a\ f), v(c) : B, D$

C-Merge: If $\vdash_{\mathsf{sem}} f, v : A \multimap_{\mathrm{C}} B, C \supset D$ and $\vdash_{\mathsf{sem}} a, c : A, C$ then $\vdash_{\mathsf{sem}} (f\ a\ {}^{\mathrm{C}}), v(c) : B, D$

M-Merge: If $\vdash_{\mathsf{sem}} f, v : A \multimap_{\mathrm{M}} B, C \supset D$ and $\vdash_{\mathsf{sem}} a, c : A, C$ then $\vdash_{\mathsf{sem}} (f\ a\ {}^{\mathrm{M}}), v(c) : B, D$

(25) **A Notational Sleight of Hand**

- Since we are ignoring prosodics, these syntax rules are written with the premises in the left-to-right order in which their prosodic counterparts would occur.
- This means you can figure out what what syntactic terms sound like by erasing the punctuation and term constructors, and replacing the words with their prosodies.
- This only works because English has relatively fixed word order and we are ignoring phneomena (such as extraposition and particle movement) that prevent pros from being a function.

(26) **A Complex CVG Syntactic Term**

$({}^{\text{SU}}\ \mathsf{it}_{\mathsf{dum}}\ (\mathsf{bothered}_2\ (\mathsf{Mary}\ \cdot\ (\mathsf{that}\ ({}^{\text{SU}}\ \mathsf{Fido\ barked})\ {}^{\text{M}}))\ {}^{\text{C}}))$

- Terms like these are the Curry-Howard terms for the syntax logic.
- They are analogs of the traditional labelled bracketings of TG.
- They obviate the need for syntactic proofs to fill pages.
- But we needn't write type symbols on the parentheses, since we can deduce them from the types of the constants.
- Just like in that other, more familiar proof term system, TLC.

(27) **The Original Sin of Syntactic Theory**

- This started with TG, but other victims were HPSG and, more generally, model-theoretic syntax (under the construal that term has acquired in recent years).
- The mistake is thinking that labelled bracketings, or their arboreal notational variants, are 'syntactic structures' whose parts can be moved around and whose 'configurations' (dominance, c-command, government, etc.) are worthy of study.
- They are not; rather, they are proof terms that *denote* syntactic entities in a model-theoretic interpretation of the syntactic logic.

(28) **More on the Original Sin**

- The enormity of the error becomes more evident once we introduce variables and binding operators for nonlocal dependencies.
- The analogous mistake in TLC would be to say that the term $\lambda_x\ f(x)$ is formed by the rule of 'lambda movement'

  d-structure: $f(\lambda)$

  move-λ: the λ leaves behind a variable x, and then moves to a non-argument position on the left periphery to bind it:

  s-structure: $\lambda_x\ f(x)$
- Why didn't Church do it that way?

THE SLASH FLAVOR OF IMPLICATION: EXTRACTION (OVERT MOVEMENT) IN ENGLISH

(29) **The SLASH Flavor of Implication**

- $\multimap_{\text{SLASH}}$ is an implicative type constructor with Hypothetical Proof but not Modus Ponens (cf. Moortgat 1988).
- SLASH-hypotheses are syntax-semantics pairs of hypotheses (as opposed to the cascaded TG architecture where a trace (syntactic variable) eventually 'gives rise to' an LF variable).
- SLASH-hypotheses are tracked in **contexts** to the left of the $\vdash_{\mathsf{sem}}$.

(30) **Some Revised CVG Rules, with Contexts**

Fusion: If $\textsc{sl} : \Gamma \vdash_{\mathsf{sem}} a, c : A, C$ and $\textsc{sl} : \Delta \vdash_{\mathsf{sem}} b, d : B, D$
then $\textsc{sl} : \Gamma; \Delta \vdash_{\mathsf{sem}} a \cdot b, (c, d) : A \circ B, C \wedge D$

Subject Merge: If $\textsc{sl} : \Gamma \vdash_{\mathsf{sem}} a, c : A, C$
and $\textsc{sl} : \Delta \vdash_{\mathsf{sem}} f, v : A \multimap_{\textsc{su}} B, C \supset D$
then $\textsc{sl}{:}\Gamma; \Delta \vdash_{\mathsf{sem}} ({}^{\textsc{su}}\ a\ f), v(c) : B, D$

Note 1: Since (as we will see) $\multimap_{\textsc{sl}}$ is subject to Contraction but not Weakening or Permutation, the SL-field in a context should be thought of as a *multiset* of (paired) hypotheses.

Note 2: In a complete grammar, there will be be other fields in the contexts, corresponding to other kinds of nonlocal dependencies (e.g. Right Node Raising, Comparative Associate Matching, etc.).

(31) **The Trace Rule**

$\text{SL} : t, x \vdash_{\mathsf{sem}} t, x : A, B \ (B \neq \text{T})$

- For expository convenience, this and the following rules are written pretending that the SL (SLASH) field is the only field in contexts.
- This is just a SL-flavored version of the usual rule for positing a hypothesis in variable-context-style natural deduction.
- But unlike typical ND hypotheses, traces have to be constrained with respect to which other rules they can be premisses of,
- However the relevant constraints are formulated, they play the same role in CVG that the ECP played in GB (or the Trace Principle in HPSG.

(32) **Rules for (Overt) Movement**

Finite Move: If $\textsc{sl}{:}t, x : A, B; \Gamma \vdash_{\mathsf{sem}} s, p : \mathrm{Fin}, \mathrm{Prop}$
then $\textsc{sl}{:}\Gamma \vdash_{\mathsf{sem}} \lambda_t^{\mathrm{SL}} s, \lambda_x p : A \multimap_{\mathrm{SL}} \mathrm{Fin}, B \supset \mathrm{Prop}$

Infinitive Move: If $\textsc{sl}{:}t, x : A, B; \Gamma \vdash_{\mathsf{sem}} f, v : C \multimap_{\mathrm{SU}} \mathrm{Inf}, D \supset \mathrm{Prop}$
then $\textsc{sl}{:}\Gamma \vdash_{\mathsf{sem}} \lambda_t^{\mathrm{SL}} f, \lambda_x v : A \multimap_{\mathrm{SL}} (C \multimap_{\mathrm{SU}} \mathrm{Inf}), B \supset (D \supset \mathrm{Prop})$

- These are the rules for the phenomena variously known as **extraction**, **unbounded dependencies**, **overt movement**, **movement in syntax**, Ā-**dependencies**, etc.
- These are counterparts of the usual ND rule Implication Introduction (aka Hypothetical Proof).
- But unlike usual Hypothetical Proof rules, SLASH-hypotheses in English can only be withdrawn at certain types (finite sentence and infinitive verb phrase).

(33) **The Topicalization Rule**
If $\textsc{sl}{:}\Gamma \vdash_{\mathsf{sem}} a, b : A, B$
and $\textsc{sl}{:}\Delta \vdash_{\mathsf{sem}} c, d : A \multimap_{\textsc{sl}} \mathrm{Fin}, B \supset \mathrm{Prop}$
then $\textsc{sl}{:}\Gamma, \Delta \vdash_{\mathsf{sem}} \tau(a, c), d(b) : \mathrm{Top}, \mathrm{Prop}$

- Felix, Fido bit.
- $\vdash_{\mathsf{sem}} \tau(\mathsf{Felix}, \lambda_t^{\textsc{sl}}({}^{\textsc{su}}\ \mathsf{Fido}_n\ (\mathsf{bit}\ t\ {}^{\textsc{c}}))), \mathsf{bite'}(\mathsf{Fido'}, \mathsf{Felix'}) : \mathrm{Top}, \mathrm{Prop}$
- Note that the topicalization syntactic term constructor is nonlogical; in particular it is *not* Modus Ponens.
- However, its semantic counterpart *is* Modus Ponens.

(34) **More Lexical Entries**

$\vdash_{\mathsf{sem}} \mathsf{is}, \lambda_v v$:
$(\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{S}_{\mathsf{adj}}) \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Fin}), (\mathrm{Ind} \supset \mathrm{Prop}) \supset (\mathrm{Ind} \supset \mathrm{Prop})$

$\vdash_{\mathsf{sem}} \mathsf{easy}, \lambda_r \lambda_x \mathsf{easy'}(r(x)) :$
$(\mathrm{NP} \multimap_{\mathrm{SL}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Inf})) \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{S}_{\mathsf{adj}}),$
$(\mathrm{Ind} \supset (\mathrm{Ind} \supset \mathrm{Prop})) \supset (\mathrm{Ind} \supset \mathrm{Prop})$

$\vdash_{\mathsf{sem}} \mathsf{to}, \lambda_v v : (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Bse}) \multimap_{\mathrm{C}} (\mathrm{NP} \multimap_{\mathrm{SU}} \mathrm{Inf}),$
$(\mathrm{Ind} \supset \mathrm{Prop}) \supset (\mathrm{Ind} \supset \mathrm{Prop})$

(35) **Tough Movement**

- Felix is easy to bite.
- $\vdash_{\mathsf{sem}}$ $({}^{\mathrm{SU}}$ Felix (is (easy λ_t^{SL}(to (bite t $^{\mathrm{C}}$) $^{\mathrm{C}}$) $^{\mathrm{C}}$) $^{\mathrm{C}}$)),
  $\mathsf{easy}'(\lambda_z \mathsf{bite}'(z, \mathsf{Felix}'))$: Fin, Prop
- Note that this analysis of *tough*-constructions is a logical reconstruction of the one in Chomsky 1977.

(36) **Violins and Sonatas**

- My violin, your sonata is easy to play on.
- $\vdash_{\mathsf{sem}} \tau((^{\mathrm{SP}}\ \mathsf{my\ violin}), \lambda_t^{\mathrm{SL}}(^{\mathrm{SU}}\ (^{\mathrm{SP}}\ \mathsf{your\ sonata})$
  $(\mathsf{is}\ (\mathsf{easy}\ \lambda_{t'}^{\mathrm{SL}}\ (\mathsf{to}\ (\mathsf{play}\ (t'\ \circ (\mathsf{on}\ t\ ^{\mathrm{C}}))\ ^{\mathrm{C}})\ ^{\mathrm{C}})\ ^{\mathrm{C}})\ ^{\mathrm{C}}))),$
  $\mathsf{easy'}(\mathsf{play\text{-}on'}(\mathsf{your'}(\mathsf{sonata'}), \mathsf{my'}(\mathsf{violin'}))) : \mathrm{Fin}, \mathrm{Prop}$

(37) **So-Called Parasitic Gaps are Licensed by Contraction**

If $\textsc{sl}{:}\Gamma; t, x : A, B; t', y : A, B; \Delta \vdash_{\mathsf{sem}} c, d : C, D$
then $\textsc{sl}{:}\Gamma; t, x : A, B; \Delta \vdash_{\mathsf{sem}} c[t'/t], d[y/x] : C, D$

a. This book is hard to start **t** without finishing **t**.
b. These are the reports we filed **t** without reading **t**.
c. Which rebel leader did rivals of **t** assassinate **t**?

(38) **The Prohibition on 'Movement from Nowhere'**

a. *Bagels, Kim likes muffins.
b. *Who did you see Kim?
c. *The student who I talked to Sandy just arrived.
d. *What this country needs a good five-cent cigar is a good 20-cent draft beer.
e. *It's Kim that Sandy likes Dana.
f. *John is easy to please Mary.

Such examples are not licensed since there is no Weakening for $\multimap_{\text{SLASH}}$.

(39) **The Prohibition on Crossed Dependencies**

a. i. This violin$_j$, even the most difficult sonatas are easy λ_{t_i} [to play $\mathbf{t}_i$ on $\mathbf{t}_j$].

ii. *This sonata$_i$, even the most exquisitely crafted violins are difficult λ_{t_j} [to play $\mathbf{t}_i$ on $\mathbf{t}_j$].

b. i. Which problems$_i$ don't you know who$_j$ [to talk to $\mathbf{t}_j$ about $\mathbf{t}_i$]?

ii. *Which people$_j$ don't you know what$_i$ [to talk to $\mathbf{t}_j$ about $\mathbf{t}_i$]?

The crossed dependencies are not licensed because there is no Permutation for $\multimap_{\text{SLASH}}$.

HYPERINTENSIONAL SEMANTICS OVERVIEW

(40) **What is Hyperintensional Semantics?**

- A semantic theory written in HOL with **subtypes**.
- Meanings are **hyperintensions**.
- The set HYPER of **hyperintensional types** is closed under the cartesian-closed constructors (T, $\wedge$, $\supset$) and subtyping (i.e. meanings form a **subtopos**).
- Basic hyperintensional types are Ind (individual concepts) and Prop (propositions).
- Basic **extensional** types are Ent (entities) and Bool (truth values).
- World is not a basic type, but is *defined* as the subtype of Prop $\supset$ Bool whose members are the **ultrafilters** of propositions.

(41) **Entailment and Worlds**

- The constant $\models$: (Prop $\wedge$ Prop) $\supset$ Bool denotes the **entailment** relation on propositions.
- $\models$ is axiomatized to be a **strict boolean preorder**.
- The boolean operations on Prop are the meanings of the English 'logic words' *not, and, or*, and *implies*.
- For a proposition p to be **true** at a world w (written $p@w$) is to be a set-theoretic member of it.
- Theorem: p entails q iff q is true at every world where p is true. [This is an internal version of the the principal lemma for the Stone Representation Theorem.]
- The preceding enables us to use Stone duality to clarify basic issues in the semantics of modals, questions, and counterfactuals.]

(42) **The Extensional Type Corresponding to a Hyperintensional Type**

This is recursively defined as follows:

a. $\mathrm{E}(\mathrm{Prop}) =_{\mathrm{def}} \mathrm{Bool}$

b. $\mathrm{E}(\mathrm{Ind}) =_{\mathrm{def}} \mathrm{Ent}$

c. $\mathrm{E}(1) =_{\mathrm{def}} 1$

d. $\mathrm{E}(A \wedge B) =_{\mathrm{def}} \mathrm{E}(A) \wedge \mathrm{E}(B)$

e. $\mathrm{E}(A \supset B) =_{\mathrm{def}} A \supset \mathrm{E}(B)$

f. $\mathrm{E}(A_a) =_{\mathrm{def}} \mathrm{E}(A)$

(43) **The Functor from Hyperintensions to Intensions**
We define a (cartesian-closed) functor Ext from hyperintensions to Carnapian intensions (= functions from worlds to extensions).

- At types: for $A \in \text{HYPER}$, $\text{Ext}(A) =_{\text{def}} \text{World} \supset \text{E}(A)$.
- At terms: for details, see Pollard 2007a,b. The key cases for now are given by the axioms:

  $\vdash \forall_{p,w}[\mathsf{ext}_{\text{Prop}}(p)(w) = p@w]$
  $\vdash \forall_{w,z}[\mathsf{ext}_{A\wedge B}(z)(w) = (\mathsf{ext}_A(\pi(z))(w), \mathsf{ext}_B(\pi'(z))(w))]$
  $\vdash \forall_{w,f}[\mathsf{ext}_{A\supset B}(f)(w) = \lambda_{x\in A}\mathsf{ext}_B(f(x))(w)]$
- Note that $\mathsf{ext}_{\text{Prop}}$ denotes the Stone embedding that maps each proposition to the set of ultrafilters of which it is a member.
- So the Carnapian propositions are the clopens (open-and-closed sets) in the Stone topology.

(44) **Extensionality**

a. In standard PWS, a property is called **extensional** iff whether or not an intension has the property depends only on the intension's extension. Examples:

b. Asssuming Zog is one of the Ancients, being seen by Zog is an extensional property. So if Zog sees Hesperus, then Zog must also see Phosphorus (since they have the same extension).

c. But being worshipped by Zog is not an extensional property: Zog might have worshipped Hesperus but not Phosphorus.

d. Being a groundhog is an extensional property. So (assuming the meaning of *Miss America* is a nonrigid individual), in a world where Miss America and Dick Cheney are coextensive, Dick Cheney is a groundhog in that world iff Miss America is a groundhog in that world.

(45) **Extensionality in Hyperintensional Semantics**

a. By an A-**property**, we mean something of type $A \supset$ Prop. E.g.

i. groundhog' : Ind $\supset$ Prop is interpreted as an **individual property**

ii. obvious : Prop $\supset$ Prop is intepreted as a **propositional property**.

b. If $A \in$ HYPER, we say an A-property f is **extensional** iff, at every world w, for any two A-hyperintensions a and b, if a and b are coextensive at w, then $f(a)$ and $f(b)$ have the same truth value.

c. More generally, we say a functional hyperintension $f : A \supset B$ ($A, B \in$ HYPER) is **extensional** iff, at every world w, for any two A-hyperintensions a and b, if a and b are coextensive at w, then so are $f(a)$ and $f(b)$.

(46) **Definitions (Quantifiers and Determiners)**
For $A \in \text{HYPER}$,

- an A-**quantifier** is an extensional function of type $(A \supset \text{Prop}) \supset \text{Prop}$, i.e. an extensional property of A-properties.
- an A-**determiner** is an extensional function of type $((A \supset \text{Prop}) \wedge (A \supset \text{Prop})) \supset \text{Prop}$, i.e. an extensional property of pairs of A-properties.

(47) **Examples of Determiners**

a. We introduce families of constants (parametrized by $A \in \text{HYPER}$) interpreted as the meanings of the English determiners *every*, *some*, and *no* , e.g.:

$\vdash \mathsf{every'}_A : ((A \supset \text{Prop}) \wedge (A \supset \text{Prop})) \supset \text{Prop}$

b. The expected truth-conditional behavior of these determiners is given by the following meaning postulates:

$\vdash \forall_{w,P,Q}[\mathsf{every'}(P,Q)@w = \forall_x(P(x)@w \supset Q(x)@w)]$

$\vdash \forall_{w,P,Q}[\mathsf{some'}(P,Q)@w = \exists_x(P(x)@w \wedge Q(x)@w)]$

$\vdash \forall_{w,P,Q}[\mathsf{no'}(P,Q)@w = \sim\exists_x(P(x)@w \wedge Q(x)@w)]$

COVERT MOVEMENT

(48) **Covert Movement Overview**

- By **covert movement**, we mean phenomena traditionally handled by movement at LF in TG.
- These include (*inter alia*), **quantifier scope**, **construal of *in situ* wh-expressions**, and **construal of comparatives**.
- Covert movement differs from extraction because:
  - the syntactic entity whose intepretation is at issue is not prosodically displaced or deleted.
  - the 'morphosyntactic features' of the entity is do not have to be remembered/checked/matched at any 'higher' point in the derivation
  - there is no morphosyntactic registration (e.g. on verbs or complementizers) of the 'pathway' between the entity and the point in the derivation where it is construed/takes scope.

(49) **Keeping the Syntax Simple**

- Our approach to covert movement is in the spirit of Cooper 1983 and Hendriks 1993: keep the syntax simple.
- Thus, syntactically, quantificational expressions like *everyone* and *in situ* wh-expressions like *whom* will just be NPs.
- Such approaches seem to have been widely discredited in the categorial grammar community as 'baroque', 'ad hoc', and 'inelegant', to mention some of the more charitable characterizations.
- Of course these approaches are inconsistent with the syntactocentrism assumed by nearly all categorial grammarians.
- Our version uses a ternary semantic type-constructor q analogous to Moortgat's (1991) *in situ* binder.
- But since it is only used in the semantic logic, issues of its prosodic interpretation do not arise (it has none).

(50) **Co-Contexts**

- We now add to our typing judgments, in addition to the context to the left of the turnstile, a **co-context** (or **store**) to the right of the **co-turnstile**: $\Gamma \vdash a, b : A, B \dashv \Sigma$
- Analogously to the context, the co-context Σ contains fields for different 'flavors' of stored elements, e.g. QS (quantifer store), WH (*in situ wh*), ER (comparative operator), etc.
- The QS field is a multiset of pairs of the form (x,q) where x is a variable (call its type C) and $q : \mathsf{q}(C, \text{Prop}, \text{Prop})$.
- It might help some of you to think of $\mathsf{q}(C, D, E)$ as analogous to $(D \uparrow C) \downarrow E$ where $D \uparrow C$ is a delimited continuation and $\downarrow$ corresponds to scope-taking.
- You might also want to think of variables in the co-context as co-variables.
- If you don't understand the last two bullets, don't worry.

(51) **Quantified NPs**

Note: from now on we abbreviate the hyperintensional types Ind and Prop to I and P respectively.

$\vdash$ everyone, everyone' : NP, $\mathsf{q}(\mathrm{I}, \mathrm{P}, \mathrm{P}) \dashv$

Notice the funny semantic type instead of the expected $(\mathrm{I} \supset \mathrm{P}) \supset \mathrm{P}$—we'll come back to this.

(52) **The Quantifier Storage Rule**
If $\Gamma \vdash a, q : A, \mathsf{q}(C, \mathrm{P}, \mathrm{P}) \dashv \Sigma[\mathrm{Qs}{:}\Delta]$

then $\Gamma \vdash a, x : A, C \dashv \Sigma[\mathrm{Qs}{:}\Delta; (x, q)]$

- x is fresh
- If you are thinking of $\mathsf{q}(C, \mathrm{P}, \mathrm{P})$ as $(\mathrm{P} \uparrow C) \downarrow \mathrm{P}$, and of P as a result type, then this rule might seem analogous to Double Negation.
- I *think* not, but I am interested to hear opposing views.

(53) **The Quantifier Retrieval Rule**
If $\Gamma \vdash b, p : B, \mathrm{P} \dashv \Sigma[\mathsf{QS}{:}\Delta; (x, q)]$

then $\Gamma \vdash b, \rho_x(q, b) : B, \mathrm{P} \dashv \Sigma[\mathsf{QS}{:}\Delta]$

- The variable $x : C$ is not free in q.
- $\rho_{C,D,E}$ is a polymorphic binary term constructor whose first argument has type $\mathsf{q}(C, D, E)$, whose second argument has type D, and whose result type is E.
- ρ_x binds all free occurrences of x in its second argument.

(54) **An Example of Quantifer Scope Ambiguity**

a. John thinks everyone sings.

b. Both readings have have the same syntax:
($^{\text{SU}}$ John (thinks ($^{\text{SU}}$ everyone sings) $^{\text{C}}$))

c. In both cases, everyone' is stored.

d. The ambiguity depends on whether everyone' is retrieved at the complement clause or the main clause:

   i. think'(John', ρ_x(everyone', walk'(x)))

   ii. ρ_x(everyone', think'(John', walk'(x)))

e. These don't quite look like familiar semantic lambda-terms, do they?

   i. For one thing, they contain the unfamiliar ρ-binder.

   ii. For another, they contain the subterm everyone' with the unfamiliar type q(I, P, P).

f. What are they, anyway?

(55) **Source Language and Target Language**

- The odd-looking expressions in (54d) are terms of type Prop in a HOL enriched with the q type constructor and the associated ρ-binding term constructor. We can think of this logic as our **semantic source language**.
- To get familiar-looking lambda terms, we have to translate into the **semantic target language**, ordinary HOL.
- The translation consists of
  a. replacing all types of the form $\mathsf{q}(C, D, E)$ by $(C \supset D) \supset E$
  b. replacing terms of the form $\rho_x(q, b)$ by $q(\lambda_x b)$
- Once this is done, we get:
  a. $\mathsf{think'}(\mathsf{John'}, \mathsf{everyone'}(\lambda_x \mathsf{walk'}(x)))$
  b. $\mathsf{everyone'}(\lambda_x \mathsf{think'}(\mathsf{John'}, \mathsf{walk'}(x)))$
- This may seem a bit complicated, but not in comparison with recent continuation-based approaches.

(56) **Hyperintensional Semantics of Questions**

- We analyze questions as **realistic properties of propositions**.
- The type is $\mathrm{P} \supset \mathrm{P}$.
- The corresponding extensional type is $\mathrm{P} \supset \mathrm{Bool}$.
- Here **realistic** means that the extension at each world w is a subset of w (i.e. a set of w-facts).
- Specifically, the extension of a question at a world is the complete set of all factual 'atomic' answers (both positive and negative).
- There is a categorical equivalence between this semantics and the Groenendijk-Stokhof partition semantics (this can be proved by Stone duality).
- We analyze root (direct) and embedded (indirect) questions in the same way.

(57) **Example (Direct Polar Question)**

- Is Bush crazy?
- $\lambda_p[p$ and' $((p$ equals' crazy'(Bush')) or' $(p$ equals' not'(crazy'(Bush'))))]$
- $\lambda_p[p@w \wedge ((p =$ crazy'(Bush')) $\vee (p =$ not'(crazy'(Bush'))))]$

(58) **Example (Indirect Direct Polar Question)**

- whether Bush is crazy
- whether'(crazy'(Bush'))
- Meaning Postulate:
  $\vdash$ whether' $= \lambda_{p'}\lambda_p[p$ and' $((p$ equals' $p')$ or' $(p$ equals' not'$(p'))))]$

(59) **Example (Single WH-Question)**

- Who sings?
- $\lambda_p[p \,\mathsf{and'}\, \mathsf{exists'}(\mathsf{person'}, \lambda_x((p \,\mathsf{equals'}\, \mathsf{sing'}(x)) \,\mathsf{or'}\, (p \,\mathsf{equals'}\, \mathsf{not'}(\mathsf{sing'}(x)))))]$
- $\lambda_p[p@w \wedge \exists_x[\mathsf{person'}(x)@w \wedge ((p = \mathsf{sing'}(x)) \vee (p = \mathsf{not'}(\mathsf{sing'}(x)))))]$

(60) **A Useful Abbreviation**

- If p', p'' are propositional terms, then $\mathsf{which'}_{x_1,\ldots x_n}(p', p'')$ abbreviates:
  $\lambda_p(p \,\mathsf{and'}\, \mathsf{exists'}(\lambda_{x_1,\ldots,x_n} p', \lambda_{x_1,\ldots,x_n}((p \,\mathsf{equals'}\, p'') \,\mathsf{or'}\, (p \,\mathsf{equals'}\, \mathsf{not'}(p'')))))$
- The, for example, the above meaning for *Who sings?* becomes:
  $\mathsf{which}'_x(\mathsf{person'}(x), \mathsf{sing'}(x))$

(61) **Example (Multiple WH-Question)**

- Who likes what?
- $\lambda_p[p$ and' exists'$(\lambda_{x,y}($person'(x) and' thing'$(y)),$
  $\lambda_{x,y}((p$ equals' like"$(x,y))$ or' $(p$ equals' not'(like'$(x,y)))))]$
- which'$_{x,y}($person$'(x)$ and' thing'$(y),$ like'$(x,y))$
- $\lambda_p[p@w \wedge \exists_{x,y}[($person'$(x)@w \wedge$ thing'$(y)@w) \wedge ((p =$ like'$(x,y)) \vee$
  $(p =$ not'(like'$(x,y)))))]$

(62) **Chinese WH-Questions**

- Chinese is a wh-*in situ* language.
- Wh-expressions occur in the same position as their non-wh counterparts.
- Wh-construal in Chinese is generally considered not to be subject to island constraints.
- Chinese wh-words are often analyzed as variables, not operators, since they can be indefinite as well as interrogative.

(63) **Chinese Single-WH-Questions**

a. i. Shei xihuan Zhangsan?
   ii. who like Zhangsan
   iii. 'Who likes Zhangsan?'

b. i. Zhangsan xihuan shei?
   ii. 'Whom does Zhangsan like?'

(64) **An Unambiguous Chinese Multiple-WH-Question**

a. Shei xihuan shenme?
b. who like what
c. 'Who likes what?'

(65) **An Ambiguous Chinese Multiple-WH-Question**

a. Zhangsan xiang-zhidao shei mai-le shenme.
b. Zhangsan wonder who buy-ASP what
c. i. 'Zhangsan wonders who bought what.'
ii. 'Who does Zhangsan wonder what (that person) bought?'
iii. 'What does Zhangsan wonder who bought?'

(66) **Analysis of Chinese WH-Expressions**

$\vdash$ shei, $x : \mathrm{NP}, \mathrm{I} \dashv \textsc{wh}{:}(x, \textsf{person'})$

$\vdash$ shenme, $y : \mathrm{NP}, \mathrm{I} \dashv \textsc{wh}{:}(y, \textsf{thing'})$

(67) **The Rule of WH-*in situ* Construal**

If $\Gamma \vdash s, p : \mathrm{S}, \mathrm{P} \dashv \Sigma[\textsc{wh}{:}\Delta; (x_1, f_1), \ldots, (x_n, f_n)]$, then

$\Gamma \vdash s, \mathsf{which'}_{x_1,\ldots,x_n}(f_1(x_1)\ \mathsf{and'}\ \ldots \mathsf{and'}\ f_n(x_n), p) : \mathrm{S}, \mathrm{P} \supset \mathrm{P} \dashv \Sigma[\textsc{wh}{:}\Delta]$

(68) **The Analysis of Chinese WH-Questions**

a. Zhangsan xiang-zhidao shei mai-le shenme.

b. Zhangsan wonder who buy-ASP what

c. i. 'Zhangsan wonders who bought what.'

ii. 'Who does Zhangsan wonder what (that person) bought?'

iii. 'What does Zhangsan wonder who bought?'

d. i. $\mathsf{wonder'}(\mathsf{Zhangsan'}, \mathsf{which'}_{x,y}(\mathsf{person'}(x)\ \mathsf{and'}\ \mathsf{thing'}(y), \mathsf{buy'}(x, y)))$

ii. $\mathsf{which'}_x(\mathsf{person'}(x), \mathsf{wonder'}(\mathsf{Zhangsan'}, \mathsf{which'}_y(\mathsf{thing'}(y), \mathsf{buy'}(x, y))))$

iii. $\mathsf{which'}_y(\mathsf{thing'}(y), \mathsf{wonder'}(\mathsf{Zhangsan'}, \mathsf{which'}_x(\mathsf{person'}(x),\ \mathsf{buy'}(x, y))))$

REFERENCES

Jónsson, B. and A. Tarski. 1951. Boolean algebras with operators, part 1. *American Journal of Mathematics 73.4*, 891-939.

Pollard, C. 2007. *Hyperintensional Semantics*. Slides for course at ESSLLI 2007, Dublin. `http://www.ling.ohio-state.edu/~hana/hog/`

Pollard, C. 2007. Hyperintensions. *Journal of Logic and Computation*.

Pollard, C. 2007. Nonlocal dependencies via variable contexts. In R. Muskens, ed., *Workshop on New Directions in Type-Theoretic Grammar*. ESSLLI07, Dublin. Slides available at `http://www.ling.ohio-state.edu/~hana/hog/`.

Pollard, C. 2007. Stone dual semantics for natural language. To be presented at Semantics in Paris 2, October 2007.

Stone, M. 1936. The theory of representation for boolean algebras. *Transactions of the American Mathematical Society 40*, 37-111.

Stone, M. 1937. Topological representations of distributive lattices and Brouwerian lattices. *Casopis Pest. Math. 67*, 1-25