A PARALLEL-DERIVATIONAL ARCHITECTURE FOR THE SYNTAX-SEMANTICS INTERFACE

Carl Pollard
INRIA-Lorraine and Ohio State University

ESSLLI 2008 Workshop on
What Syntax Feeds Semantics
Hamburg, August 14, 2008

1 Introduction: a Convergence of Views

(1) **Back in 1970:**

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

  In particular, there was nothing 'between' syntax and semantics.
- Chomsky's "Conditions on Transformations" (published in 1973) introduced the **T-model**, in which interpretive rules applied between SS and LF:

$$\begin{array}{c} \text{Phonetics} \leftarrow \text{PF} \leftarrow \text{SS} \rightarrow \text{LF} \rightarrow \text{Semantics} \\ \uparrow \\ \text{DS} \\ \uparrow \\ \text{LEX} \end{array}$$

(2) **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 catagorial grammar (CG) and phrase structure grammar (PSG).
- Can't we settle this?

(3) **The Cascade**

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

$$\begin{array}{c} \text{Semantics} \\ \uparrow_{?} \\ \text{LF} \\ \uparrow_{C} \\ \text{SS} \\ \uparrow_{O} \\ \text{DS} \\ \uparrow_{M} \\ \text{LEX} \end{array}$$

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

(4) **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:
-
  - 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.
- The Minimalist Program (MP) is one framework for filling in the details of this consensus view.
- This talk is about a different one, worked out within the framework of **Extended Montague Grammar** (EMG) about 30 years ago.

2 What was Extended Montague Grammar?

(5) **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 practicioners of:
  - PSG, e.g. Cooper, Gazdar, Pullum
  - CG, e.g. Dowty
  - switch hitters, e.g. Bach.

(6) **Three Signal Achievments 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.

(7) **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.
- But why?

(8) **Why Reconstruct EMG?**

- EMG had already correctly perceived many of the main defects of the T-model and had good proposals for fixing them.
- But EMG and its descendants have not presented themselves in ways that make them seem interesting or inaccessible to noninitiates, so they have often ended up "preaching to the choir".
- The case for EMG needs to be made anew, in ways that address the concerns of "mainstream generative grammarians".
- A promising approach is to reformulate the EMG ideas using an especially transparent formalism: **Gentzen natural deduction with Curry-Howard proof terms**.

3 A Logical Reconstruction of EMG

3.1 Background: Natural Deduction

(9) **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 (intuitionistic) logic of implication.
- Later, we'll use ND for semantic and syntactic derivations.

(10) **Intuitionistic Implicative Logic (IIL)**

- We start with some **atomic formulas** $\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \ldots$, and form more formulas from them using the **implication** connective $\rightarrow$.
- Notation: A, B, and C range over IIL formulas; and $A \rightarrow (B \rightarrow C)$ is abbreviated as $A \rightarrow B \rightarrow C$.
- Question: Which formulas should be considered **theorems**?
- There are many kinds of proof systems for IIL, 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, $A \rightarrow B$, $A \rightarrow A \rightarrow B$, $((A \rightarrow B) \rightarrow A) \rightarrow A$.

(11) **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

(12) **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'.
- A is a formula/type, a is a term/proof, and Γ, the **context** of the judgment, is a set of variable/formula pairs of the form $x : A$.

(13) **Some Rule Schemas for IIL**

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

Hypothetical Proof:
if $x : A, \Gamma \vdash b : B$,
then $\Gamma \vdash \lambda_x b : A \to B$

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

(14) **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**.

(15) **Reformulating EMG using ND**

- We have **two** logics, each with its own 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.
- The signs are the inputs to the interpretive interfaces:
  - the syntactic component is phonetically interpreted, and
  - the semantic component is semantically interpreted.
- We call this style of grammar **Convergent Grammar** (CVG).

(16) **Parallel-Derivational (PD) Artchitecture**

$$\begin{array}{c} \text{phonetics} \\ \uparrow \\ \text{Syn} \\ \text{Syn candidates} \rightarrow + \leftarrow \text{Sem candidates} \\ \text{Sem} \\ \downarrow \\ \text{semantics} \end{array}$$

3.2 Syntax

(17) **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 **unbound** traces.

(18) **Categories**

- **Basic** categories, such as S, NP, and N.

  For present purposes we ignore morphosyntactic details such as case, agreement, and verb inflection.
- **Function** categories: if A and B are categories, so is $A \multimap_{\mathrm{F}} B$, for $\mathrm{F} \in \mathsf{F}$, the set of **grammatical function names** (**gramfuns**).

  A is called the **argument** type and B the **result** type.

- **Operator** categories: if A, B, and C are categories, so is $\mathrm{G}[A, B, C]$, abbreviated A^C_B.

  A, B, and C are called the **binding** category, the **scope** category B, and the **result** category C respectively.

  The G constructor is inspired by Moortgat's (1991) q-constructor, but that was for covert (not overt) movement. A standard TLG way to get the effect of A^C_B is $C/(B{\uparrow}A)$ where $\uparrow$ is Moortgat's (1988) extraction constructor.

(19) **Some Syntactic Words**

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

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

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

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

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

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

$\vdash \mathsf{liked} : \mathrm{NP} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}$

$\vdash \mathsf{thought} : \mathrm{S} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}$

$\vdash \mathsf{wondered} : \mathrm{Q} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}$

$\vdash \mathsf{whether} : \mathrm{S} \multimap_{\mathrm{C}} \mathrm{Q}$

(20) **Remarks on the Lexicon**

- QNPs are just NPs.
- *Wh*-expressions are ambiguous between NPs and operators.

(21) **The Syntactic Schemata**

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

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

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

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

Schema T (Trace)

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

Schema G (Gazdar Schema)

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

(22) **Remarks on the Syntactic Schemata**

- The Modus Ponens schemata correspond to **Merge**.
- **Traces** are just variables (no internal structure).
- The Gazdar Schema corresponds to **Overt Move**.
  It is an ND reformulation of Gazdar's (1979) linking schemata,
- *a* was not moved or copied from the position of the trace *t*.
- So there is no issue about which end of the 'chain' is pronounced.
- Merges can follow Moves because in ND you can always apply **any** rule as long as its premisses have been proved.

(23) **A Simple Sentence**

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

(24) **An Embedded Constituent Question**

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

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

(25) **A Binary Constituent Question**

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

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

(26) **A Baker Question**

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

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

For the semantics of these examples, see my paper from the Workshop on Symmetric Calculi and Ludics.

3.3 Semantics

(27) **ND-Style Semantics**

- 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.
- As we'll see, the syntax-semantics interface ensures that free semantic variables are always paired with either (1) unbound traces, or (2) Cooper-stored semantic operators.

(28) **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 Constraint).

e. We often omit the '$\dashv$' if the co-context is empty.

(29) **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 $A \to B$.

c. **Operator** types: If A, B, and C are types, so is $\mathrm{G}[A, B, C]$, 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 ($\to$).

(30) **How the Semantic Operator Types are Used**

- Semantic operator types are used for expressions which would be analyzed in TG as undergoing (overt or covert) $\bar{\mathrm{A}}$-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 $\mathrm{e}_{\mathrm{t}}^{\mathrm{t}}$.
- 'Overtly moved' signs: syntax and semantics are both operators.
  Example: 'Overtly moved' *who* has category $\mathrm{NP}_{\mathrm{S}}^{\mathrm{Q}}$ and semantic type $\mathrm{e}_{\pi}^{\mathrm{e}\to\pi\to\mathrm{t}}$ (where $\pi =_{\mathrm{def}} \mathsf{s} \to \mathrm{t}$).

(31) **The Semantic Schemata**

Constants, variables, and Modus Ponens just as in TLC, plus:

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

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

Schema G (Semantic Counterpart of Gazdar Schema)
If $\Gamma \vdash a : A_B^C \dashv \Delta$ 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)

Note: 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.

(32) **The Transform τ from Semantic Terms to TLC**
Everything stays the same except:

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

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.

This should be compared with the apparent inexplicitness about the interpretation of LF.

4 The Syntax-Semantics Interface

(33) **Some Lexical Entries**

$\vdash$ Chris, Chris' : NP, e

$\vdash$ everyone, everyone' : $\mathrm{NP}, \mathrm{e}_{\mathrm{t}}^{\mathrm{t}} \dashv$

$\vdash$ someone, someone' : $\mathrm{NP}, \mathrm{e}_{\mathrm{t}}^{\mathrm{t}}$

$\vdash$ liked, like' : $\mathrm{NP} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}, \mathrm{e} \rightarrow \mathrm{e} \rightarrow \mathrm{t}$

$\vdash$ thought, think' : $\mathrm{S} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}, \pi \rightarrow \mathrm{e} \rightarrow \mathrm{t}$

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

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

Heads combine with subjects semantically by function application.

Contexts (unbounded traces) and co-contexts (Cooper-stored operators) get passed up (as in old-fashioned PSG).

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

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

Just like the preceding but for complements instead of subjects.

(36) **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.

(37) **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.

(38) **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.

(39) **Schema G (Gazdar Schema)**

If $\Gamma \vdash a, d : A^C_B, D^F_E \dashv \Delta$ 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)

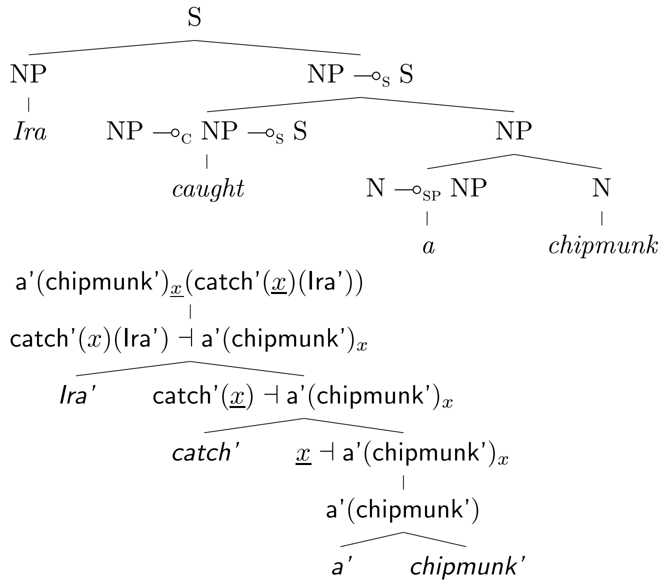
The syntactic and semantic operators scope in parallel.

Important: The operator a **binds** the trace t, but does not 'move' from the argument position t occupies, or 'copy' t.

This is just as in TLC, where there is no sense in which $\lambda_x.\mathsf{bite'}(x)(\mathsf{Fido'})$ is derived by movement or copying from $\mathsf{bite'}(\lambda)(\mathsf{Fido'})$.

5 Quantifier Scope

(40) **Cooper Storage ('Covert Movement') Example**



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

(41) **Quantifier Scope Ambiguity**

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

b. Semantics (scoped to lower clause):
((think' (everyone'$_{\underline{x}}$((like' $\underline{x}$) Kim'))) Chris')
TLC: $\mathsf{think'}(\lambda_w(\forall_x(\mathsf{person'}(x)(w) \rightarrow \mathsf{like'}(x)(\mathsf{Kim'})(w))))(\mathsf{Chris'})$

c. Semantics (scoped to upper clause):
(everyone'$_{\underline{x}}$((think' ((like' $\underline{x}$) Kim')) Chris'))
TLC: $\lambda_w(\forall_x(\mathsf{person'}(x)(w) \rightarrow \mathsf{think'}(\mathsf{like'}(x)(\mathsf{Kim'}))(\mathsf{Chris'})(w)))$

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

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

b. $\forall\exists$-reading: $(\mathsf{everyone'}_{\underline{x}}(\mathsf{someone'}_{\underline{y}}((\mathsf{like'}\ \underline{y})\ \underline{x})))$

c. $\exists\forall$-reading: $(\mathsf{someone'}_{\underline{y}}(\mathsf{everyone'}_{\underline{x}}((\mathsf{like'}\ \underline{y})\ \underline{x})))$

6 Parasitic Scope

(43) **Parasitic Scope**

- Barker (in press) introduces this term to describe quantifiers such as *the same* and *different* whose 'scope target does not exist until [another quantifier] takes its scope'.
- Other instances of this phenomenon include **superlatives** and elliptical constructions such as **phrasal comparatives**.
- Barker's analysis uses **continuations** and **choice functions**.
- We propose an account based on a notion of **focus exploitation**.

(44) **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.

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

- ordinary determiners: type $(\mathrm{e} \rightarrow \mathrm{t}) \rightarrow \mathrm{e}_{\mathrm{t}}^{\mathrm{t}}$
- 'overtly moved' interrogative determiner *which*: type $(\mathrm{e} \rightarrow \mathrm{t}) \rightarrow \mathrm{e}_{\pi}^{\mathrm{e} \rightarrow \pi \rightarrow \mathrm{t}}$ (where $\pi =_{\mathrm{def}} \mathsf{s} \rightarrow \mathrm{t}$).
- (non-phrasal) comparative -*er*, assuming the *than*-phrase complement denotes a degree: type $\mathrm{d} \rightarrow \mathrm{d}_{\mathrm{d}}^{\mathrm{t}}$.
- Following (in spirit) Moortgat 1991, we can analyze **pragmatic focus** as an intonationally realized phrasal affix whose semantics has the (polymorphic) operizer type $B \rightarrow B_{\mathrm{t}}^{\mathrm{t}}$.

(46) **Semantic Focus as an Operizer 'Wild Card'**

- We suggest treating **semantic focus** as an operizer 'wild card' whose instantiation depends on what other sign is **exploiting** it.
- Best-known is the case of 'particles' (*only*, *even*, *too*) where the **focus instantiator** (FI) is just the semantics of the particle itself.
- Here we consider more complex cases of **parasitic scope**, where the focus exploiter (FE) 'contributes' **two** operizers: one its own semantics and the other the FI; the focused phrase is called the **asscociate**.
- In still more complex—**elliptical**—cases to be treated elsewhere, the FI takes **two** arguments: the associate and the FE's (extraposed) complement, called the **remnant**.

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

- We add to the inventory of gramfuns the name AFFIX (abbr. A), mnemonic for '(phrasal) affixation'.
- Correspondingly, we add a new 'flavor' of Modus Ponens to the syntactic (and interface) schemata ($\multimap_{\mathrm{A}}$-Elimination).
- This is used to analyze intonationally realized phrasal affixes, Japanese and Korean case markers, Chinese sentence particles, English possessive *-'s*, etc.
- Lexical entry for English semantic focus:

  $\vdash \mathsf{foc}, \mathsf{foc'} : A \multimap_{\mathrm{A}} A, B \to B^{\mathsf{t}}_{\mathsf{t}}$

(48) **An (at Least) Triply Ambiguous Superlative Sentence**

a. Kim thinks Sandy makes the most.

b. First reading: Sandy makes the most, Kim thinks.

c. Second reading: The amount Kim thinks Sandy makes exceeds the amount Kim thinks anyone else makes.

d. Third reading: The amount Kim thinks Sandy makes exceeds the amount anyone else thinks Sandy makes.

(49) **Comments on the Preceding**

- These are all **internal** readings. Examples of this kind seem to lack decitic/external readings.
- We can force the third reading by placing the focal pitch accent on **Kim**.
- We can rule out the third reading by placing the focal pitch accent on **Sandy**.

(50) **Intuitive Explanation**

- The FE *the most* and the FI have adjacent scope ('parasitic scope' or 'tucking in').
- If **Kim** is focused, then they have to scope at the root clause (because operators can raise but not lower).
- If **Sandy** is focused, then there is ambiguity as to whether it scopes in the root clause or the complement clause.

(51) **Toward an Analysis of Superlatives**

a. **Fido** cost <u>the most</u>.

b. We take this to mean that Fido is the unique maximizer of the function that maps (relevant) entities to their prices.

c. We assume something's price is the maximum amount that it costs.

d. So our target semantics for this sentence is
$\mathsf{um}(\mathsf{Fido'})_{\underline{x}}.\mathsf{max}_{\underline{d}}.\mathsf{cost'}(\underline{d})(\underline{x})$
where the operizer um is subject to the meaning postulate

e. $\vdash \mathsf{um} = \lambda_x.\lambda_f.\forall_y((y \neq x) \rightarrow (f(x) > f(y))) : \mathrm{e} \rightarrow \mathrm{e}^{\mathrm{t}}_{\mathrm{d}}$

f. After normalization, (d) translates to:
$\forall_y((y \neq \mathsf{Fido'}) \rightarrow [\mathsf{max}(\lambda_d.\mathsf{cost'}(d)(\mathsf{Fido'})) > \mathsf{max}(\lambda_d.\mathsf{cost'}(d)(\mathsf{x}))])$

g. This is the semantics our theory will predict, as long as the semantics of *the most* is max and focus is instantiated as um.

g. But how?

(52) **Instantiating Focus**

a. Lexical entries:
$\vdash \mathsf{cost}, \mathsf{cost'} : \mathrm{Deg} \multimap_{\mathrm{C}} \mathrm{NP} \multimap_{\mathrm{S}} \mathrm{S}$
$\vdash \mathsf{the\_most}, \mathsf{IF}(\mathsf{um}) \cdot \mathsf{max} : \mathrm{Deg}, \mathrm{d}^{\mathrm{d}}_{\mathrm{t}} \dashv$
The semantics here means: 'max directly outscoped by the result of instantiating focus as um'.

b. Focus Instantiation Semantic Schema (FI)
If $\Gamma \vdash a \dashv \mathsf{foc'}(b)_x; \mathsf{IF}(c) \cdot d_y; \Delta$,
then $\Gamma \vdash a \dashv c(b)_x \cdot d_y; \Delta$
Note that in the corresponding interface schema, nothing happens in the syntax.

(53) **Analysis of a Superlative Sentence**

a. Syntax:
$(^{\mathrm{S}}$ (foc Fido $^{\mathrm{A}}$) (cost the_most $^{\mathrm{C}}$))

b. Semantics:

$$\mathsf{um}(\mathsf{Fido'})_{\underline{x}}.\mathsf{max}_{\underline{d}}.\mathsf{cost'}(\underline{d})(\underline{x})$$

$$\mathsf{cost'}(d)(x) \dashv \mathsf{um}(\mathsf{Fido'})_x \cdot \mathsf{max}_d$$

$$\mathsf{cost'}(d)(x) \dashv \mathsf{foc'}(\mathsf{Fido'})_x; \mathsf{IF}(\mathsf{um}) \cdot \mathsf{max}_d$$

$$x \dashv \mathsf{foc'}(\mathsf{Fido'})_x \qquad \mathsf{cost'}(d) \dashv \mathsf{IF}(\mathsf{um}) \cdot \mathsf{max}_d$$

$$\mathsf{foc'}(\mathsf{Fido'}) \qquad \textsf{\textit{cost'}} \qquad d \dashv \mathsf{IF}(\mathsf{um}) \cdot \mathsf{max}_d$$

$$\textsf{\textit{foc'}} \quad \textsf{\textit{Fido'}} \qquad \textsf{\textit{IF}}(\textsf{\textit{um}}) \cdot \textsf{\textit{max}}$$

c. Normalized TLC translation:
$\forall_y((y \neq \mathsf{Fido'}) \rightarrow [\mathsf{max}(\lambda_d.\mathsf{cost'}(d)(\mathsf{Fido'})) > \mathsf{max}(\lambda_d.\mathsf{cost'}(d)(\mathsf{x}))])$

(54) ***The Same***

a. Plural-focus *the same*:
Fido and Felix got <u>the same present</u>.

$\exists_y(\mathsf{present'}(y) \wedge \forall_x[(x <_{\mathsf{a}} \mathsf{Fido'} + \mathsf{Felix'}) \rightarrow \mathsf{get'}(y)(x)])$

Here + denotes Link join (plural formation), and $<_{\mathsf{a}}$ denotes the part-of relation between an atom and a plural.

b. Elliptical (associate-remnant) *the same*:
Fido got <u>the same present</u> *as* **Felix**.

$\exists_y(\mathsf{present'}(y) \wedge \mathsf{get'}(y)(\mathsf{Fido'}) \wedge \mathsf{get'}(y)(\mathsf{Felix'}))$

c. These sentences have equivalent truth conditions.

d. Here we only analyze plural-focus *the same*.

e. Elliptical *the same* and other associate-remnant constructions are analyzed in work in progress.

(55) **Analysis of Plural-Focus *The Same***

a. We cannot escape from positing a special coordination rule with semantics corresponding to Link join (plural formation).

b. We also need a new basic semantic type e' for plural entities.

c. Syntactically, plural-focus *the same* is just a determiner.

d. But semantically, it is an FE operizer:

1. Its own semantics is the existential generalized determiner $\mathsf{a'}$.
2. The FI is the distributive operizer dist that converts a plural to a universal quantifier, characterized by the meaning postulate $\vdash \mathsf{dist} = \lambda_{x'}.\lambda_P.\forall_x((x <_{\mathsf{a}} x') \to P(x)) : \mathrm{e}' \to \mathrm{e}^{\mathrm{t}}_{\mathrm{t}}$
3. Unlike *the most*, in this case the FE outscopes the FI.

e. So the lexical entry for *the same* is:
$\vdash \mathsf{the\_same}, \mathsf{a'} \cdot \mathsf{FI}(\mathsf{dist}) : \mathrm{N} \multimap_{\mathrm{SP}} \mathrm{NP}, \mathrm{et} \to \mathrm{e}^{\mathrm{t}}_{\mathrm{t}}$

(56) **Analysis of a Plural-Focus *The Same* Sentence**

a. Syntax: $(^{\mathrm{S}}$ (foc (Fido and Felix) $^{\mathrm{A}}$) (got (the_same present $^{\mathrm{SP}}$) $^{\mathrm{C}}$))

b. Semantics:

$$\mathsf{a'(present')}_{\underline{y}}.\mathsf{dist(Fido' + Felix)}_{\underline{x}}.\mathsf{get'}(\underline{y})(\underline{x})$$

$$\mathsf{get'}(y)(x) \dashv \mathsf{a'(present')}_y \cdot \mathsf{dist(Fido' + Felix')}_x$$

$$\mathsf{get'}(y)(x) \dashv \mathsf{a'(present')}_y \cdot \mathsf{FI(dist)}; \mathsf{foc'(Fido' + Felix')}_x$$

$$x \dashv \mathsf{foc'(Fido' + Felix')}_x \qquad \mathsf{get'}(y) \dashv \mathsf{a'(present')}_y \cdot \mathsf{FI(dist)}$$

$$\mathsf{foc'(Fido' + Felix')} \qquad \mathsf{a'(present')} \cdot \mathsf{FI(dist)}$$

$$\mathit{foc'} \quad \mathsf{(Fido' + Felix')} \qquad \mathit{a'} \cdot \mathit{FI(dist)} \quad \mathit{present'}$$

$$\mathit{Fido'} \quad + \quad \mathit{Felix'}$$

c. Normalized TLC translation:
$\exists_y(\mathsf{present'}(y) \wedge \forall_x[(x <_{\mathsf{a}} \mathsf{Fido' + Felix'}) \to \mathsf{get'}(y)(x)])$

7 Conclusions

(57) **Summing Up**

- EMG had a good theory of how to repair the T-model.
- But so far the story has not been told in a way that has gained it mainstream acceptance.
- Howard (1980) provided the technology to retell the EMG story simply and clearly.

(58) **The EMG Story Retold**

- Syntactic and semantic derivations are **parallel**, not cascaded.
- Derivations are **proofs**, not sequences of tree operations.
- **All** signs have a semantics ('it's phases all the way down').
- Traces are **ordinary logical variables**, not copies of their binders.
- **There is no 'Trace Conversion'**: traces are paired with semantic variables from birth.
- Merge is **Modus Ponens**.
- 'Overt Move' works as **Gazdar** said.
- 'Covert Move' works as **Cooper** said.
- Rules can intermingle because that's always the case in proofs.
- Interpretation of the semantic proof is **simple** and **explicit**.
- **There is no 'LF'** between syntax and semantics.