Traces Exist (Hypothetically)!

Carl Pollard

Structure and Evidence in Linguistics
Workshop in Honor of Ivan Sag
Stanford University

April 28, 2013
Traces are usually thought to have been invented (discovered?) by linguists at MIT in the early 1970’s:

WH-fronting could be formulated so that a phonetically null copy of the WH-word is left behind in its pre-fronting position. [Wasow 1972:139, attributed to Culicover (p.c.)]

Assuming that wh-Movement leaves a trace PRO, we might then stipulate that every rule that moves an item from an obligatory category (in the sense of Emonds (1970)) leaves a trace. [Chomsky 1973:135, fn. 49]

Subsequently they became a mainstay of TG:

[D-structures] are mapped to S-structures by the rule Move-α, leaving traces coindexed with their antecedents .... [Chomsky 1981:5]
But even in TG, the ontological status of traces has not been completely straightforward:

[T]he correct LF for (32)

(32) \textit{Who did Mary say that John kissed t}

should be

(37) \textit{for which x, x a person, Mary said that John kissed [x]}

The LF (37) has a terminal symbol, x, in the position of the NP source of \textit{who}, but (32) has only a trace, i.e. only the structure $[\text{NP}_i \; e]$, where $i$ is the index of \textit{who}.

[Chomsky 1977:83-84]
In Gazdar 1981, if \(A\) and \(B\) are syntactic categories, then so is \(A/B\). Then the notion of trace is expressed as

\[
A/A \rightarrow t
\]

which is a lexical entry schema for the null string.

Pollard and Sag's (1994:161) trace schema is the same as Gazdar's, recoded as an AVM:

\[
[\text{PHON } \langle \rangle, \text{SYNSEM } [\text{LOC } [1], \text{NONLOC}|\text{SLASH } [1]]]
\]
But Pollard and Sag (1994:378–387) eliminated traces in favor of three lexical rules responsible, respectively, for extraction of complements, subjects, and adjuncts.

Sag and Fodor (1995) defended this analysis on empirical grounds, noting also the absence of (analogs of) traces in CCG and LFG.

Natural Deduction

- **Natural deduction** (Gentzen 1934, Prawitz 1965) is a style of theorem proving characterized by the presence of inference rule schemas for introducing and eliminating logical connectives (examples coming right up).

- Below we’ll focus on *implicative linear logic (ILL)*, which has just one connective $\rightarrow$ (linear implication).

- The premisses and conclusion of rules are *sequents* of the form $\Gamma \vdash A$, read ‘$A$ is deducible from the hypotheses $\Gamma$’.
  - $A$ is a formula, called the *statement* of the sequent
  - $\Gamma$ is a multiset of formulas, called the *context* of the sequent.
  - Commas in contexts represent multiset union.
Implicative Linear Logic (1/2)

- In ILL, the only rules are

  Implication Elimination, aka Modus Ponens
  \[ \Gamma, \Delta \vdash B \]
  \[ \Gamma \vdash A \rightarrow B \quad \Delta \vdash A \]

  Implication Introduction, aka Hypothetical Proof
  \[ \Gamma \vdash A \rightarrow B \]
  \[ \Gamma, A \vdash B \]

- Each rule is a local tree with the daughter(s) labelled by premisses and the mother labelled by the conclusion.
- Contexts at each node represent undischarged hypotheses.
- There is also a logical axiom schema (Hypothesize):
  \[ A \vdash A \]
With these, we can prove any ILL theorem, e.g. TR:

\[ \vdash A \rightarrow (A \rightarrow B) \rightarrow B \]

\[ \vdash (A \rightarrow B) \rightarrow B \]

\[ \vdash (A, A \rightarrow B) \rightarrow B \]

\[ \vdash A \rightarrow A, A \rightarrow B \rightarrow (A \rightarrow B) \]

\[ \vdash A \rightarrow B \rightarrow A \rightarrow B \]

A proof is a tree.

Each leaf is labelled by an axiom.

Each nonleaf and its daughters instantiates one of the rules.

The sequent labelling the root is the theorem proved.
If only the natural deduction turnstile $\vdash$ and and Gazdar’s slash / were the same thing, the Hypothesize axiom schema

$$A \vdash A$$

would be the same as Gazdar’s syntactic category for traces

$$A/A$$

That would only make sense if

- a grammar was a natural deduction system
- phrase structure trees were proof trees
- linguistic expressions were sequents
- lexical entries (not only traces) were axioms

These things are all true!

To see why, we have to reformulate PSG in terms of ILL.

ILL vs. PSG
Curry (1958) and Howard (1969) discovered a connection between implicative logic and lambda calculus: if we think of formulas as types, then a formula is a theorem iff there is a combinator (pure closed lambda term) of that type.

For ILL, lambda terms are assigned to types/formulas is as follows:

\[
\begin{align*}
x : A & \vdash x : A \\
\Gamma, \Delta & \vdash (M N) : B \\
\Gamma & \vdash M : A \rightarrow B \quad \Delta & \vdash N : A \\
\Gamma & \vdash \lambda_x M : A \rightarrow B \\
\Gamma, x : A & \vdash M : B
\end{align*}
\]
For example, the fact that TR is a theorem corresponds to the fact the combinator $\lambda_{xf}.(f \ x)$ has that type.

We can see this by adding type annotations to the proof of TR we just gave:

\[
\begin{align*}
\vdash \lambda_{xf}(f \ x) : & \ A \to (A \to B) \to B \\
& x : A \vdash \lambda_{f}(f \ x) : (A \to B) \to B \\
& x : A, f : A \to B \vdash (f \ x) : B \\
& f : A \to B \vdash f : A \to B \quad x : A \to x : A
\end{align*}
\]

This correspondence between theorems and terms is called the Curry-Howard correspondence.
In his one foray into linguistics, Curry (1961) proposed that syntax should be bifurcated into *phenogrammatical structure* (roughly, surface form) and *tectogrammatical structure* (roughly, semantically motivated combinatorics).

Curry’s idea influenced PSGians (Reape, Kathol) and CGians (Dowty, Oehrle).

In particular, Oehrle (1994) invented a kind of categorial grammar based on ILL, here called *linear grammar* (*LG*).

In the rest of this talk, I’ll sketch how to logically reconstruct the PSG theory of UDCs, by identifying Gazdar’s / with the natural deduction turnstile ⊢.
LG Basics: Phenogrammatical Types and Terms

- LG analyses consist of two simultaneous natural deduction proofs, one in the pheno dimension and one in the tecto dimension. (There is also a Montague-like semantic dimension, omitted here.)
- The only base type in the pheno logic is $s$ (string).
- If $A$ and $B$ are pheno types, so is $A \rightarrow B$.
- The pheno proof is annotated with lambda terms, called pheno terms, that encode the surface form.
- There are pheno constants of type $s$ corresponding to lexical phonologies, such as he, is, easy, etc.
- There is also a pheno constant $e$ of type $s$ corresponding to the null string.
- There is an (infix) constant $\cdot$ of type $s \rightarrow s \rightarrow s$ for concatenation.
LG Basics: Tectogrammatical Types

- The base types for the tecto logic are:
  
  $S_f$ (finite clause)
  $S_i$ (infinitive clause)
  $S_b$ (base-form clause)
  $\bar{Q}$ (embedded interrogative clause)
  PrdA (predicative adjectival clause)
  $NP_n$ (nominative NP)
  $NP_a$ (accusative NP)
  $NP_{it}$ (dummy $it$)
  $PP_{for}$ ($for$-PP)

- If $A$ and $B$ are tecto types, so is $A \rightarrow B$.

- There is no need to distinguish between (categorial) / vs. \ (as in CCG or Lambek calculus) because constituent ordering is handled in the pheno component.
LG Basics: Nonlogical Axioms (Lexical Entries)

Types of pheno terms are omitted to save space.

she = ⊢ she; NP_n
he = ⊢ he; NP_n
him = ⊢ him; NP_a
her = ⊢ her; NP_a
it = ⊢ it; NP_it
pleases = ⊢ λ_{st}.t \cdot \text{pleases} \cdot s; NP_a \rightarrow NP_n \rightarrow S_f
please = ⊢ λ_s.\text{please} \cdot s; NP_a \rightarrow NP_n \rightarrow S_b
is = ⊢ λ_{st}.t \cdot \text{is} \cdot u; (A \rightarrow \text{PrdA}) \rightarrow A \rightarrow S_f
to = ⊢ λ_s.\text{to} \cdot s; (A \rightarrow S_b) \rightarrow (A \rightarrow S_i)
for = ⊢ λ_s.\text{for} \cdot s; NP_a \rightarrow PP_{for}
easy_1 = ⊢ λ_{st}.\text{easy} \cdot s \cdot t; PP_{for} \rightarrow (NP_n \rightarrow S_i) \rightarrow NP_it \rightarrow \text{PrdA}
LG Basics: The Combine Rule

\[ \Gamma, \Delta \vdash (M \ N) ; B \]

\[ \Gamma \vdash M ; A \rightarrow B \quad \Delta \vdash N ; A \]

- This is the LG version of Modus Ponens.
- It replaces all the PSG phrasal schemas.
- It is the only rule needed for analyzing local dependencies.
- Think of a sequent \( \Gamma \vdash M ; A \rightarrow B \) as
  \[ [\text{PHON } M ; \text{HEAD } B ; \text{SUBCAT } A ; \text{SLASH } \Gamma] \]
- Combine incorporates the effect of
  - the Head Feature Principle
  - the Valence Principle (but only one argument is discharged per rule application)
  - the GAP Principle (\textit{sans} STOP-GAP, which is handled by the other rule).
to please him

Here and henceforth VP<sub>i</sub> abbreviates NP<sub>n</sub> → S<sub>b</sub>.
easy for her to please him

⊢ easy · for · her · to · please · him; NP_{it} → PrdA

⊢ λt easy · for · her · t; VP_i → NP_{it} → PrdA ⊢ to · please · him; VP_i

easy_1 ⊢ for · her; PP_{for}

for her
It is easy for her to please him

:\[ \vdash \text{it} \cdot \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please} \cdot \text{him}; \text{S}_f \]

\[ \vdash \lambda_t t \cdot \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please} \cdot \text{him}; \text{NP}_{\text{it}} \rightarrow \text{S}_f \]

\[ \vdash \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please} \cdot \text{him}; \text{NP}_{\text{it}} \rightarrow \text{PrdA} \]
LG Basics: The Stop-Gap Rule

\[ \Gamma \vdash \lambda_t M; A \rightarrow B \]
\[ \Gamma, t; A \vdash M; B \]

- This is the LG version of Hypothetical Proof.
- There is no PSG rule corresponding to this rule.
- Instead, the PSG counterpart is the STOP-GAP (or TO-BIND) feature on the lexical head of the Head-Filler Rule and lexical entries like *easy*.
- Stop-Gap discharges a hypothesis (trace) and lambda-binds the string variable \( t \) that it introduced.
LG Basics: Trace

\[ t; A \vdash t; A \]

- This is the LG counterpart of the Hypothesize schema
- Here \( t \) is a variable of type \( s \) (string)
- \( A \) can be instantiated by any tecto type, e.g.
  \[ t; \text{NP}_a \vdash t; \text{NP}_a \]
- Think of \( \text{NP}_a \vdash \text{NP}_a \) as LG-ese for \( \text{NP}_a[\text{SLASH} \langle \text{NP}_a \rangle] \).
- Equipped with Stop-Gap and Trace, we can analyze UDCs as soon as we add suitable lexical entries.
whom = \lambda f.\text{whom} \cdot (f \ e); (\text{NP}_a \rightarrow S_f) \rightarrow Q

easy_2 = \lambda s_f.\text{easy} \cdot s \cdot (f \ e); PP_{for} \rightarrow (\text{NP}_a \rightarrow VP_i) \rightarrow \text{NP}_n \rightarrow \text{PrdA}

In both of these lexical entries:

■ one of the arguments has an NP\_a gap (which will have been discharged by an application of Stop-Gap)

■ The bound variable \( f \) is of type \( s \rightarrow s \) (functions from strings to strings), corresponding to the gappy argument

■ when the lexical entry combines with that argument, the null string \( e \) is lambda-converted into the gap position!

■ As much as I would like to take credit for it, this bit of pheno-technology was invented by Muskens (2007).
whom she pleases

\[ \vdash \text{whom} \cdot \text{she} \cdot \text{pleases} \cdot e; \bar{Q} \]

\[ \text{whom} \vdash \lambda_s \text{she} \cdot \text{pleases} \cdot s; \text{NP}_a \rightarrow S_f \]

\[ s; \text{NP}_a \vdash \text{she} \cdot \text{pleases} \cdot s; S_f \]

\[ \text{she} \vdash \lambda_t \text{t} \cdot \text{pleases} \cdot s; \text{NP}_n \rightarrow S_f \]

\[ \text{pleases} \vdash s; \text{NP}_a \vdash s; \text{NP}_a \]

- The non-branching node is the instance of Stop-Gap that binds the \text{NP}_a trace.
- That together with the instance of Combine just above it capture the effect of HPSG’s Filler-Head rule.
Again, the nonbranching node is the instance of Stop-Gap that binds the $NP_a$ trace.
Here AP abbreviates $NP_n \to \text{PrdA}$.

By the time $easy$ combines with the infinitive VP, its $NP_a$ gap has already been bound.

So there is no need for $easy$ to have a STOP-GAP feature.
He is easy for her to please $t$

\[ \vdash \text{he} \cdot \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please}; S_f \]

\[ \vdash \lambda_t \text{he} \cdot \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please}; \text{NP}_{it} \rightarrow S_f \]

\[ \vdash \text{is} \cdot \text{easy} \cdot \text{for} \cdot \text{her} \cdot \text{to} \cdot \text{please} \cdot e; \text{AP} \]
Summary

We logically reconstructed PSG inside of linear grammar.

- Phrase structure trees become *natural-deduction proof trees*.
- Node labels become *sequents*.
- SLASH becomes the *turnstile* ($\vdash$) in sequents.
- SLASH values become the *contexts* in sequents.
- The valence features all become *linear implication* ($\rightarrow$).
- Traces become *hypotheses (logical axioms)*.
- Other lexical entries become *nonlogical axioms*.
- The phrasal schemas collapse into *Modus Ponens (Combine)*.
- The only other rule is *Hypothetical Proof (Stop-Gap)*, which does the work of PSG’s STOP-GAP feature.

I wish we had known about natural deduction 30 years ago!


