Locating Human Meanings:
Less Typology, More Constraint

Paul M. Pietroski, University of Maryland
Dept. of Linguistics, Dept. of Philosophy
Elizabeth, on her side, had much to do. She wanted to ascertain the feelings of each of her visitors, she wanted to compose her own, and to make herself agreeable to all; and in the latter object, where she feared most to fail, she was most sure of success, for those to whom she endeavoured to give pleasure were prepossessed in her favour.

Bingley was ready,
Georgiana was eager, and
Darcy determined to be pleased.

Jane Austen
*Pride and Prejudice*
Bingley is eager to please.

(a) Bingley is eager to be \textit{one who pleases}.

#(b) Bingley is eager to be \textit{one who is pleased}.

Bingley is easy to please.

#(a) Bingley can easily \textit{please}.

(b) Bingley can easily \textit{be pleased}.

Human children naturally acquire languages that somehow generate boundlessly many expressions that connect meanings (whatever they are) with pronunciations (whatever they are) in accord with certain constraints.
Human languages generate boundlessly many expressions that connect meanings with pronunciations in accord with certain constraints.

Do human linguistic expressions exhibit meanings of different types?

(1) Fido  (5) every cat
(2) chase  (6) chase every cat
(3) every  (7) Fido chase every cat
(4) cat  (8) Fido chased every cat.

And if so, which meaning types do they exhibit?
What are the Human Meaning Types?

- one familiar answer, via Frege’s conception of *ideal* languages
  (i) a basic type &lt;e&gt;, for *entity denoters*
  (ii) a basic type &lt;t&gt;, for *thoughts* or *truth-value denoters*
  (iii) if &lt;α&gt; and &lt;β&gt; are types, then so is &lt;α, β&gt;

Fido, Garfield, Zero, ...

Fido barked.
Fido chased Garfield.
Zero precedes every positive integer.
What are the Human Meaning Types?

• one familiar answer, via Frege’s conception of ideal languages
  (i) a basic type <e>, for entity denoters
  (ii) a basic type <t>, for thoughts or truth-value denoters
  (iii) if <α> and <β> are types, then so is <α, β>

• on the other hand, one might suspect
  (a) there are no meanings of type <e>
  (b) there are no meanings of type <t>
  (c) the recursive principle is crazy implausible
What are the Human Meaning Types?

• one familiar answer, via Frege’s conception of ideal languages
  (i) a basic type <e>, for entity denoters
  (ii) a basic type <t>, for thoughts or truth-value denoters
  (iii) if <α> and <β> are types, then so is <α, β>

That’s a lot of types
a basic type \( <e> \), for entity denoters

a basic type \( <t> \), for truth-value denoters

if \( <\alpha> \) and \( <\beta> \) are types, then so is \( <\alpha, \beta> \)

---

<table>
<thead>
<tr>
<th>( &lt;e&gt; )</th>
<th>( &lt;t&gt; )</th>
<th>(2) types at Level Zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( &lt;e, e&gt; ) ( &lt;e, t&gt; ) ( &lt;t, e&gt; ) ( &lt;t, t&gt; )</td>
<td>(4) at Level One, all ( &lt;0, 0&gt; )</td>
<td></td>
</tr>
<tr>
<td>2. eight of ( &lt;0, 1&gt; ) eight of ( &lt;1, 0&gt; ) sixteen of ( &lt;1, 1&gt; )</td>
<td>(32), including ( &lt;e, et&gt; ) and ( &lt;et, t&gt; )</td>
<td></td>
</tr>
<tr>
<td>3. 64 of ( &lt;0, 2&gt; ) 64 of ( &lt;2, 0&gt; ) 128 of ( &lt;1, 2&gt; ) 128 of ( &lt;2, 1&gt; ) 1024 of ( &lt;2, 2&gt; )</td>
<td>(1408), including ( &lt;e, &lt;e, et&gt;&gt;; &lt;et, &lt;et, t&gt;&gt;; ) and ( &lt;&lt;e, et&gt;, t&gt; )</td>
<td></td>
</tr>
<tr>
<td>4. 2816 of ( &lt;0, 3&gt; ) 2816 of ( &lt;3, 0&gt; ) 5632 of ( &lt;1, 3&gt; ) 5632 of ( &lt;1, 3&gt; ) 45,056 of ( &lt;2, 3&gt; ) 45,056 of ( &lt;3, 2&gt; ) 1,982,464 of ( &lt;3, 3&gt; )</td>
<td>(2,089,472), including ( &lt;e, &lt;e, &lt;e, et&gt;&gt; and ( &lt;&lt;e, et&gt;, &lt;&lt;e, et&gt;, t&gt; )</td>
<td></td>
</tr>
</tbody>
</table>

at Level 5, more than \( 5 \times 10^{12} \)
a basic type $<e>$, for entity denoters
a basic type $<t>$, for truth-value denoters
if $<\alpha>$ and $<\beta>$ are types, then so is $<\alpha, \beta>$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0. | $<e>$ | $<t>$ | ziggy  
      |     |     | Number(ziggy) |
| 1. | $<e, t>$ | | $\lambda x.\text{Number}(x)$ |
| 2. | $<e, et>$ | | $\lambda y.\lambda x.\text{Predecessor}(x, y)$  
      |     |     | $\lambda y.\lambda x.\text{Precedes}(x, y)$ |
| 3. | $<e, et>, t>$ | | Transitive[$\lambda y.\lambda x.\text{Precedes}(x, y)$]  
      |     |     | Intransitive[$\lambda y.\lambda x.\text{Predecessor}(x, y)$] |
| 4. | $<e, et>, <e, et>, t>$ | | TransitiveClosure[$\lambda y.\lambda x.\text{Precedes}(x, y), \lambda y.\lambda x.\text{Predecessor}(x, y)$] |
Frege \textit{invented} a language that supported abstraction on \textit{relations}

Three precedes four.

Three is something \textit{that precedes four}. \hspace{1cm} \lambda x. \text{Precedes}(x, 4)

Four is something \textit{that three precedes}. \hspace{1cm} \lambda x. \text{Precedes}(3, x)

*Precedes is some relation \textit{that three four}. \hspace{1cm} \lambda R. R(3, 4)

The plate outweighs the knife.

The plate is something \textit{which outweighs the knife}. \hspace{1cm} \lambda \text{Outweighs}(\text{knife})

The knife is something \textit{which the plate outweighs}. \hspace{1cm} \lambda \text{Outweighs}(\text{plate})

*Outweighs is some relation \textit{which the plate the knife}. \hspace{1cm} \lambda R. R(\text{plate}, \text{knife})
a basic type \(<e>\), for entity denoters
a basic type \(<t>\), for truth-value denoters
if \(<\alpha>\) and \(<\beta>\) are types, then so is \(<\alpha, \beta>\)

3. \(<<e, et>, t>\) Transitive\([\lambda y. \lambda x. \text{Precedes}(x, y)]\)

\(\text{Precedes transits.}\)

4. \(<<e, et>, <<e, et>, t>\) TransitiveClosure\([\lambda y. \lambda x. \text{Precedes}(x, y), \lambda y. \lambda x. \text{Predecessor}(x, y)]\)

\(\text{Precedes transits predecessor.}\)
a basic type \( <e> \), for entity denoters
a basic type \( <t> \), for truth-value denoters

if \( <\alpha> \) and \( <\beta> \) are types, then so is \( <\alpha, \beta> \)

<table>
<thead>
<tr>
<th></th>
<th>( &lt;e&gt; )</th>
<th>( &lt;t&gt; )</th>
<th>( &lt;e, e&gt; )</th>
<th>( &lt;e, t&gt; )</th>
<th>( &lt;t, e&gt; )</th>
<th>( &lt;t, t&gt; )</th>
<th>( &lt;e, e, e&gt; )</th>
<th>( &lt;e, e, t&gt; )</th>
<th>( &lt;e, t, e&gt; )</th>
<th>( &lt;e, t, t&gt; )</th>
<th>( &lt;t, e, e&gt; )</th>
<th>( &lt;t, e, t&gt; )</th>
<th>( &lt;t, t, e&gt; )</th>
<th>( &lt;t, t, t&gt; )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( &lt;e&gt; )</td>
<td>( &lt;t&gt; )</td>
<td>( &lt;e, e&gt; )</td>
<td>( &lt;e, t&gt; )</td>
<td>( &lt;t, e&gt; )</td>
<td>( &lt;t, t&gt; )</td>
<td>( &lt;0, 0&gt; )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>eight of ( &lt;0, 1&gt; )</td>
<td>eight of ( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
<td>( &lt;0, 1&gt; )</td>
<td>( &lt;1, 0&gt; )</td>
</tr>
<tr>
<td>2</td>
<td>sixteenth of ( &lt;1, 1&gt; )</td>
<td>( &lt;1, 1&gt; )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>64 of ( &lt;0, 2&gt; )</td>
<td>64 of ( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
<td>( &lt;0, 2&gt; )</td>
<td>( &lt;2, 0&gt; )</td>
</tr>
<tr>
<td>4</td>
<td>128 of ( &lt;1, 2&gt; )</td>
<td>128 of ( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
<td>( &lt;1, 2&gt; )</td>
<td>( &lt;2, 1&gt; )</td>
</tr>
<tr>
<td></td>
<td>1024 of ( &lt;2, 2&gt; )</td>
<td>( &lt;2, 2&gt; )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2816 of ( &lt;0, 3&gt; )</td>
<td>2816 of ( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
<td>( &lt;0, 3&gt; )</td>
<td>( &lt;3, 0&gt; )</td>
</tr>
<tr>
<td>6</td>
<td>5632 of ( &lt;1, 3&gt; )</td>
<td>5632 of ( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
<td>( &lt;1, 3&gt; )</td>
</tr>
<tr>
<td>7</td>
<td>45,056 of ( &lt;2, 3&gt; )</td>
<td>45,056 of ( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
<td>( &lt;2, 3&gt; )</td>
<td>( &lt;3, 2&gt; )</td>
</tr>
<tr>
<td>8</td>
<td>1,982,464 of ( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
<td>( &lt;3, 3&gt; )</td>
</tr>
</tbody>
</table>

(2) types at Level Zero
(32), including \( <e, et> \) and \( <et, t> \)
(1408), including \( <e, e, et>, <et, et, t> \)
(2,089,472), including \( <e, e, e, et>, <e, et>, <et, e, et>, t> \)
What are the Human Meaning Types?

- one familiar answer, via Frege’s conception of **ideal** languages
  (i) a basic type \(<e>\), for **entity denoters**
  (ii) a basic type \(<t>\), for **thoughts** or **truth-value denoters**
  (iii) if \(<\alpha>\) and \(<\beta>\) are types, then so is \(<\alpha, \beta>\)

- a suggestion in the footnotes of “On Semantics”

*Filter Functionals:*

no \(<\alpha, \beta>\) types where \(\alpha\) is **non-basic**

\(<et, t>\) \(<e, e, e, e, e, t>\)
What are the Human Meaning Types?

- one familiar answer, via Frege’s conception of \textit{ideal} languages
  (i) a basic type $<e>$, for \textit{entity denoters}
  (ii) a basic type $<t>$, for \textit{thoughts} or \textit{truth-value denoters}
  (iii) if $<\alpha>$ and $<\beta>$ are types, then so is $<\alpha, \beta>$

- a suggestion less permissive than “\textit{Filter Functionals}”

  \textit{No Recursion}: no $<\alpha, \beta>$ types

  (1) a basic type $<M>$, for \textit{monadic predicates}
  (2) a basic type $<D>$, for \textit{dyadic predicates}
  ...
  (n) a basic type $<N>$, for \textit{N-adic predicates}
What are the Human Meaning Types?

• one familiar answer, via Frege’s conception of ideal languages
  (i) a basic type <e>, for entity denoters
  (ii) a basic type <t>, for thoughts or truth-value denoters
  (iii) if <α> and <β> are types, then so is <α, β>

• a suggestion much less permissive than “Filter Functionals”

  No Recursion: no <α, β> types

  (1) a basic type <M>, for monadic predicates
  (2) a basic type <D>, for dyadic predicates

  Minimal Relationality
Degrees of “Semantic Relationality”

• **None**: *e.g.*, Monadic Predicate Calculi
  – some \( M \) is (also) \( P \)

• **Unbounded**: *e.g.*, Tarski-style Predicate Calculi
  – \( Mx \ & \ Py \ & \ Syz \ & \ Rxw \ & \ Bzuv \ & \ldots \)
a Tarski-style Predicate Calculus permits Unbounded Adicity

<table>
<thead>
<tr>
<th>Expression</th>
<th>Adicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown(x)</td>
<td>1</td>
</tr>
<tr>
<td>Brown(x) &amp; Dog(x)</td>
<td>1</td>
</tr>
<tr>
<td>Saw(x, y)</td>
<td>2</td>
</tr>
<tr>
<td>Dog(x) &amp; Saw(x, y)</td>
<td>2</td>
</tr>
<tr>
<td>Dog(x) &amp; Saw(x, y) &amp; Cat(z)</td>
<td>3</td>
</tr>
<tr>
<td>Dog(x) &amp; Saw(x, y) &amp; Cat(z) &amp; Saw(z, w)</td>
<td>4</td>
</tr>
<tr>
<td>Dog(Fido) &amp; Saw(Fido, Garfield)</td>
<td>0</td>
</tr>
<tr>
<td>Between(x, y, z)</td>
<td>3</td>
</tr>
<tr>
<td>Quartet(x, y, z, w)</td>
<td>4</td>
</tr>
<tr>
<td>Between(x, y, z) &amp; Quartet(w, x, y, x)</td>
<td>4</td>
</tr>
<tr>
<td>Between(x, y, z) &amp; Quartet(w, v, y, x)</td>
<td>5</td>
</tr>
<tr>
<td>Between(x, y, z) &amp; Quartet(w, v, u, y)</td>
<td>6</td>
</tr>
<tr>
<td>Between(x, y, z) &amp; Quartet(w, v, u, t)</td>
<td>7</td>
</tr>
</tbody>
</table>

unbounded adicity, but no typology

... each expression (wff) is a *sentence*

... and each *sentence* is *satisfied* by all/some/no *sequences* of domain entities
Degrees of “Semantic Relationality”

• **None**: *e.g.*, Monadic Predicate Calculi
  – some $M$ is (also) $P$

• **Some, but Less Than Unbounded**
  – Minimally Relational (maximally limited)
  – “Mildly” Relational (severely limited)
  – Bounded, but still “pretty permissive”

• **Unbounded**: *e.g.*, Tarski-style Predicate Calculi
  – $Mx \& Py \& Syz \& Rxw \& Bzuv \& \ldots$
Plan for Rest of the Talk

• Characterize a notion of “Minimally Relational”

• Describe a Possible Language that is Minimally Relational and (correlatively) “Minimally Interesting” in this respect

• Suggest that while Human Meanings may be a little more interesting, they approximate Minimal Relationality

• End with reminders of some other respects in which Human Languages seem to be Minimally Interesting, and suggest that semantic typology is yet another case
Minimally Relational

• admit *dyadic* predicates, but no predicates of higher adicity
  – ABOVE(_, _) and CAUSE(_, _) are OK; so is AGENT(_, _)
  – SELL(_, _, _, _) and BETWEEN(_, _, _) are not-OK

• admit relational notions only in the *lexicon*
  – BETWEEN(_, _, JIM) is not-OK
  – ON(_, _) & HORSE(_) is not-OK

• correspondingly limited *combinatorial operations*
  – if ON(_, _) and HORSE(_) combine, the result is *monadic*
  – combining lexical items *cannot* yield relational notions
We can imagine a language whose expressions are limited to...

(1) finitely many *atomic monadic* predicates: $M_1(\_)$ ... $M_k(\_)$

(2) finitely many *atomic dyadic* predicates: $D_1(\_, \_)$ ... $D_j(\_, \_)$

(3) boundlessly many *complex monadic* predicates

Monad + Monad $\Rightarrow$ Monad

\[
\begin{align*}
\text{BROWN(\_)} + \text{HORSE(\_)} & \Rightarrow \text{BROWN(\_)}^\text{HORSE(\_)} \\
\text{FAST(\_)} + \text{BROWN(\_)}^\text{HORSE(\_)} & \Rightarrow \text{FAST(\_)}^\text{BROWN(\_)}^\text{HORSE(\_)}
\end{align*}
\]
We can imagine a language whose expressions are limited to...

(1) finitely many *atomic monadic* predicates: \( M_1(\_ ) \) ... \( M_k(\_ ) \)

(2) finitely many *atomic dyadic* predicates: \( D_1(\_, \_ ) \) ... \( D_j(\_, \_ ) \)

(3) boundlessly many *complex monadic* predicates

**Monad + Monad \( \rightarrow \) Monad**

for each entity:

\[ \Phi(\_ )^\land \Psi(\_ ) \text{ applies to it} \]

if and only if

\[ \Phi(\_ ) \text{ applies to it, } and \]
\[ \Psi(\_ ) \text{ applies to it} \]
We can imagine a language whose expressions are limited to...

(1) finitely many \textit{atomic monadic} predicates: \( M_1(\_ \_ \_) \ldots \ M_k(\_ \_ \_) \)

(2) finitely many \textit{atomic dyadic} predicates: \( D_1(\_, \_ \_ \_ \_) \ldots \ D_j(\_, \_ \_ \_ \_) \)

(3) boundlessly many \textit{complex monadic} predicates

Monad + Monad \( \rightarrow \) Monad

Dyad + Monad \( \rightarrow \) Monad

for each entity:

\( \Phi(\_) \\wedge \Psi(\_) \) applies to it if and only if

\( \Phi(\_) \) applies to it, \textit{and}

\( \Psi(\_) \) applies to it

\( \exists[\text{ON}(\_, \_ \_ \_ \_ \_ \_) \wedge \text{HORSE}(\_ \_ \)] \)

(thing that is) on a horse
We can imagine a language whose expressions are limited to...

(1) finitely many *atomic monadic* predicates: \( M_1(\_ ) \ldots M_k(\_ ) \)

(2) finitely many *atomic dyadic* predicates: \( D_1(\_ ,\_ ) \ldots D_j(\_ ,\_ ) \)

(3) boundlessly many *complex monadic* predicates

\[
\text{Monad + Monad } \rightarrow \text{ Monad} \quad \quad \quad \text{Dyad + Monad } \rightarrow \text{ Monad}
\]

for each entity:

\[
\Phi(\_ )^\Psi(\_ ) \text{ applies to it if and only if } \Phi(\_ ) \text{ applies to it, and } \Psi(\_ ) \text{ applies to it}
\]

\[
\text{ON}(\_ ,\_ ) + \text{HORSE}(\_ ) \quad \downarrow \quad \exists[\text{ON}(\_ ,\_ )^\text{HORSE}(\_ )]
\]

(thing that is) on a horse

\# thing that a horse is on
We can imagine a language whose expressions are limited to...

(1) finitely many *atomic monadic* predicates: $M_1(\_)$ ... $M_k(\_)$

(2) finitely many *atomic dyadic* predicates: $D_1(\_, \_)$ ... $D_j(\_, \_)$

(3) boundlessly many *complex monadic* predicates

For each entity: $\Phi(\_)^\Psi(\_)$ applies to it if and only if $\Phi(\_)$ applies to it, and $\Psi(\_)$ applies to it.

For each entity: $\exists[\Delta(\_, \_)^\Psi(\_)]$ applies to it if and only if it bears $\Delta$ to *something* that $\Psi(\_)$ applies to.
\[\exists [\text{AGENT}(\_ , \_ )^\wedge \text{HORSE}(\_ )]^\wedge \text{EAT}(\_ )^\wedge \text{FAST}(\_ )\]

is like

\[\exists e [\text{AGENT}(e' , e ) \& \text{HORSE}(e)] \& \text{EAT}(e') \& \text{FAST}(e')\]

\[\exists [\text{AGENT}(\_ , \_ )^\wedge \text{FAST}(\_ )^\wedge \text{HORSE}(\_ )]^\wedge \text{EAT}(\_ )\]

is like

\[\exists e [\text{AGENT}(e' , e ) \& \text{FAST}(e) \& \text{HORSE}(e)] \& \text{EAT}(e')]\]

We don’t need variables to capture the meanings of ‘horse eat fast’ and ‘fast horse eat’.
$\text{SEE}(\_)^\exists [\text{THEME}(\_, \_)^\text{HORSE}(\_) ]$

is like

$\text{SEE}(e^\prime) \& \exists e [\text{THEME}(e^\prime, e) \& \text{HORSE}(e) ]$

$\text{SEE}(\_)^\exists [\text{THEME}(\_, \_)^\exists [\text{AGENT}(\_, \_)^\text{HORSE}(\_) ]^\text{EAT}(\_) ]$

is like

$\text{SEE}(e^{\prime\prime}) \& \exists e^\prime [\text{THEME}(e^{\prime\prime}, e^\prime) \& \exists e [\text{AGENT}(e^\prime, e)^\text{HORSE}(e) ] \& \text{EAT}(e^\prime) ]$

We don’t need variables to capture the meanings of ‘see a horse’ and ‘see a horse eat’.
What are the Human Meaning Types?

--two basic types, <e> and <t>
--endlessly many derived types of the form <α, β>
-- <α> can combine with <α, β> to form <β>

--a monadic type <M>
--a dyadic type <D>, for finitely many atomic expressions
-- <M> + <M> ➔ <M>
   <M> + <D> ➔ <M>
Can Human Lexical Items have “Level Four Meanings”?

(sold a friend a car for a dollar)

whatever the order of arguments,
the concept SOLD, which differs from GAVE,
is plausibly (at least) tetradic
Can Human Lexical Items have “Level Four Meanings”?

So why not...

\[ \lambda y. \lambda z . \lambda w. \lambda x . x \text{ sold } y \text{ to } z \text{ for } w \]
Can Human Lexical Items have “Level Four Meanings”? 

$$\lambda Z \cdot \lambda Y \cdot \lambda X \cdot \text{GLONK}(X, Y, Z)$$

$$\forall x [X(x) \lor Y(x) \lor Z(x)]$$

$$\exists x [X(x) \land Y(x)] \land \exists x [Y(x) \land Z(x)]$$
Can Human Lexical Items have Level Three Meanings?

```
<e, t>
├── FIDO<e>
├── CHASED(, )<e, e, t> GARFIELD<e>
└── <t>
    ├── ROMEO<e>
    │   └── GAVE(, )<e, e, e, t> GARFIELD<e>
    └── <e, et>
        └── <e, t> JULIET<e>
```
but double-object *constructions* do not show that verbs can have Level Three Meanings
a thief jimmied a lock with a knife
Why not instead…

The concept JIMMIED is plausibly (at least) triadic. So why isn’t the verb of type <e, <e, <et>>>?
Why not…

`between` \( \mapsto \lambda z. \lambda y. \lambda x. x \text{ is between } y \text{ and } z \)
Still, one might think that many verbs do have Level Three Meanings...
Can Human Lexical Items have Level Three Meanings?

Saying that expressions of type \(<e, \_et>\) can be modified by expressions of type \(<\_et>\) is like positing a covert Level 4 \textit{element}.

And why does the modifier skip over the thing chased, applying instead to the chase?
if the meaning of ‘chase’ is at Level Three, then a “passivizer” would also be at Level Four:

\[<<e, <e, _et>, <e, et>>\]

Kratzer and others “sever” agent-variables from verb meanings:

‘chase’ ➔

\[\lambda y. \lambda e . e \text{ is a chase of } y\]
But if the posited verb meaning is below Level Three, do we really need the covert Level Three element?
What are the Human Meaning Types?

• one familiar answer, via Frege’s conception of *ideal* languages
  (i) a basic type $<e>$, for *entity denoters*
  (ii) a basic type $<t>$, for *thoughts* or *truth-value denoters*
  (iii) if $<\alpha>$ and $<\beta>$ are types, then so is $<\alpha, \beta>$

• but is it *independently* plausible that some of our *human* linguistic expressions have meanings of type $<e>$?
  -- proper nouns like ‘Tyler’, ‘Burge’, and ‘Pegasus’?
  -- pronouns like ‘he’, ‘she’, ‘it’, ‘this’, ‘that’?

• we know how to Pegasize, and
  treat names as special cases of monadic predicates
What are the Human Meaning Types?

• one familiar answer, via Frege’s conception of *ideal* languages
  (i) a basic type <e>, for *entity denoters*
  (ii) a basic type <t>, for *thoughts* or *truth-value denoters*
  (iii) if <α> and <β> are types, then so is <α, β>

• but is it *independently* plausible that some of our *human* linguistic expressions have meanings of type <t>?
  -- which ones? VPs, TPs, CPs?
  -- pronouns like ‘he’, ‘she’, ‘it’, ‘this’, ‘that’?

• we know (via Tarski) how to treat “sentences” as special cases of monadic predicates
Do Human i-Languages have expressions of type $<t>$?

$S \Rightarrow NP \text{ aux } VP$

Why think *tensed* phrases denote truth values?

\[
T(P) \quad \text{Why think } \text{tensed phrases denote truth values?}
\]

\[
/ \quad \backslash
\]

\[
T \\
V(P) \Rightarrow \lambda e . e \text{ is (tenselessly) a John-see-Mary event}
\]

\[
past \\
/ \quad \backslash
\]

\[
D(P) \\
V(P)
\]

\[
\text{past} \\
/ \quad \backslash
\]

\[
D(P) \\
V(P)
\]

\[
\text{past} \\
/ \quad \backslash
\]

\[
D(P) \\
V(P)
\]

\[
\text{past} \\
/ \quad \backslash
\]

\[
D(P) \\
V(P)
\]

Why think the *tense* morpheme

is of type $<et, t>$

$\lambda E . \exists e [\text{Past}(e) \land E(e)]$

as opposed to $<et>$ or $<M>$

$\lambda e . \text{Past}(e)$
Do Human i-Languages have expressions of type <t>?

Why think the *tense* morpheme
is of type <et, t>

\[
\lambda e . e \text{ is (tenselessly) a John-see-Mary event}
\]
a quantifier...
\[
\lambda E . \exists e [\text{Past}(e) \land E(e)]
\]
...that is also a conjunctive adjunct to V?
Kinds of Quantifiers

Propositional Calculus

0 (monadic) Mx & Px

1 (dyadic) Rxy

2

3

4 ...

unbounded adicity

& Syz & Rwx & Bzuv & ...

Kinds of Predicates:
Propositional Calculus: complete sentences (truth-table conjunction)

Kinds of Quantifiers:
- ... 
- Second-Order
- First-Order

Kinds of Predicates:
- 0 (monadic)
- 1 (dyadic)
- 2
- 3
- 4 ...
- unbounded adicity

“Mildly Relational” Second-Order Systems

Quantification over Properties

Aristotelian Syllogisms

“Marily Relational” Second-Order Systems

Quantification over Relations

Church’s λ-Calculus (maybe typed a la Frege, and limited to a few “Lower Levels”)

Second-Order Systems

“Minimally Relational” Second-Order Systems

Tarskian Predicate Calculus

Between(x, y, z)

Sold(x, y, z, w)

Cause(x, y)
Plan for Rest of the Talk

• Characterize a notion of “Minimally Relational”

• Describe a Possible Language that is Minimally Relational and (correlatively) “Minimally Interesting” in this respect

• Suggest that while Human Meanings may be a little more interesting, they approximate Minimal Relationality

• End with reminders of some other respects in which Human Languages seem to be Minimally Interesting, and suggest that semantic typology is yet another case
Flavors of Recursion

• Some recursive procedures are very, very, …, very boring

• Others generate more interesting
  [phrases [within [phrases [within [phrases … ]]]]]

• And some allow for displacement of a sort
  that permits construction of relative clauses
  like ‘who saw Juliet’ and ‘who Romeo saw’,
  whose elements can be systematically recombined
  to form boundlessly many expressions
  that allow for displacement…
Some recursive procedures are very boring.

N ➔ phrases

NP ➔ N

P ➔ within

PP ➔ P NP

PP ➔ within NP ➔ within N ➔ within phrases

NP ➔ N PP

NP ➔ N within phrases ➔ phrases within phrases

S ➔ NP aux VP ➔ Romeo did see Juliet ➔

Romeo saw Juliet ➔ Romeo saw who ➔

who Romeo saw t ➔ CP
Ways of Generating Lots of Expressions

• Finite State (Markovian)

• Phrase Structure ("Context Free")

• Transformational
  – but humanly constrained ("mildly" context sensitive)
  – not so constrained ("pret-ty" context sensitive)
  – computable but otherwise unconstrained
PushDown Automata are not very, ..., very boring. (A stack is a fine thing.)

But Turing Machines (with limited tape) can do a lot more.
Caveat: distinguish *sets* of generable expressions (E-languages) from expression-generating *procedures* (I-languages).

the *power* relations reflect the available operations: with regard to *generative capacity*, CS-grammars > PS-grammars > FS-grammars.
Finite State Phrase Structure

Context Sensitive

Mildly Context Sensitive

Human Grammars (I-Languages) seem to have a bit more generative power than PS-grammars.

This locates Human Languages in a “Computational Space.” Can they be located in a “Semantic Space”? 
Propositional Calculus: complete sentences (truth-table conjunction)

Kinds of Predicates:

0 1 2 3 4 ... unbounded adicity

(monadic) (dyadic) ...

First-Order

Second-Order

Kinds of Quantifiers:

... Church’s λ-Calculus (maybe typed a la Frege, and limited to a few “Lower Levels”)

“Mildly Relational” Second-Order Systems

“Minimally Relational” Second-Order Systems

Aristotelian Syllogisms

Quantification over Relations

Between(x, y, z)

Sold(x, y, z, w)

cause(x, y)

Tarskian Predicate Calculus

Second-Order Systems (Minimal Typology)
a basic type $<$e$>$, for entity denoters
a basic type $<$t$>$, for truth-value denoters
if $<$α$>$ and $<$β$>$ are types, then so is $<$α, β$>$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>$&lt;$e$&gt;$</td>
<td>$&lt;$t$&gt;$</td>
<td></td>
<td>(2) types at Level Zero</td>
</tr>
<tr>
<td>1.</td>
<td>$&lt;$e, e$&gt;$</td>
<td>$&lt;$e, t$&gt;$</td>
<td>$&lt;$t, e$&gt;$</td>
<td>$&lt;$t, t$&gt;$</td>
</tr>
<tr>
<td></td>
<td>eight of $&lt;$0, 1$&gt;$</td>
<td>eight of $&lt;$1, 0$&gt;$</td>
<td>sixteen of $&lt;$1, 1$&gt;$</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>64 of $&lt;$0, 2$&gt;$</td>
<td>64 of $&lt;$2, 0$&gt;$</td>
<td></td>
<td>(1408), including $&lt;$e, $&lt;$e, et$&gt;&gt;$; $&lt;$et, $&lt;$et, t$&gt;&gt;$; and $&lt;&lt;$e, et$&gt;$, t$&gt;$</td>
</tr>
<tr>
<td></td>
<td>128 of $&lt;$1, 2$&gt;$</td>
<td>128 of $&lt;$2, 1$&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>1024 of $&lt;$2, 2$&gt;$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>2816 of $&lt;$0, 3$&gt;$</td>
<td>2816 of $&lt;$3, 0$&gt;$</td>
<td></td>
<td>(2,089,472), including $&lt;$e, $&lt;$e, $&lt;$e, et$&gt;&gt;$ and $&lt;&lt;$e, et$&gt;$, $&lt;&lt;$e, et$&gt;$, t$&gt;$</td>
</tr>
<tr>
<td></td>
<td>5632 of $&lt;$1, 3$&gt;$</td>
<td>5632 of $&lt;$1, 3$&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>45,056 of $&lt;$2, 3$&gt;$</td>
<td>45,056 of $&lt;$3, 2$&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,982,464 of $&lt;$3, 3$&gt;$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thanks,
and thanks to Jim

James Higginbotham

On Semantics

In this article I will formulate and develop one conception of semantic inquiry in generative linguistics. In conjunction with specific applications, I will address questions about domains of investigation, the data in those domains that ought to be accounted