LF-Interpretation, Compositionally

Greg Kobele

University of Chicago

Dec 02
1. Compositionality

2. Cooper Storage Laws

3. Formal Consequences

4. Interpreting Tucking-in
Plan

- Give a compositional semantics for minimalism

Main claim
LF-interpretation can be viewed directly compositionally
Semantics in Generative Grammar

- Binary branching nodes
  \[
  \left[ \begin{array}{c}
  \alpha \\
  \downarrow \\
  \beta \\
  \end{array} \right]_g = [\alpha]_g \oplus [\beta]_g
  \]

- Unary branching nodes
  \[
  \left[ \begin{array}{c}
  \bullet \\
  | \\
  \alpha \\
  \end{array} \right]_g = [\alpha]_g
  \]

- Binding
  \[
  \left[ \begin{array}{c}
  i \\
  \downarrow \\
  \alpha \\
  \end{array} \right]_g = \lambda x. [\alpha]_g[i := x]
  \]

- Traces
  \[
  [t_i]_g = g(i)
  \]
Interpreting $\lambda$ terms in type structures

- **Application**
  \[
  \llbracket(M \; N)\rrbracket^g = \llbracket M \rrbracket^g (\llbracket N \rrbracket^g)
  \]

- **Abstraction**
  \[
  \llbracket \lambda i. M \rrbracket^g = \lambda x. \llbracket M \rrbracket^g[i:=x]
  \]

- **Variables**
  \[
  \llbracket i \rrbracket^g = g(i)
  \]

- **Binary branching nodes**
  \[
  \begin{bmatrix}
  \cdot \\
  \alpha & \beta
  \end{bmatrix}^g = \llbracket \alpha \rrbracket^g \oplus \llbracket \beta \rrbracket^g
  \]

- **Unary branching nodes**
  \[
  \begin{bmatrix}
  \cdot \\
  \alpha
  \end{bmatrix}^g = \llbracket \alpha \rrbracket^g
  \]

- **Binding**
  \[
  \begin{bmatrix}
  \cdot \\
  i & \alpha
  \end{bmatrix}^g = \lambda x. \llbracket \alpha \rrbracket^g[i:=x]
  \]

- **Traces**
  \[
  \llbracket t_i \rrbracket^g = g(i)
  \]
Most expressions don’t have any meaning

\[
\left[ \begin{array}{c}
V \\
\text{praise} \\
D \\
\text{every} \\
N \\
\text{boy}
\end{array} \right]^g = \left[ \begin{array}{c}
\text{praise}
\end{array} \right]^g \oplus \left[ \begin{array}{c}
D \\
\text{every} \\
N \\
\text{boy}
\end{array} \right]^g
\]

\[
\left[ \begin{array}{c}
\text{every}
\end{array} \right]^g \oplus \left[ \begin{array}{c}
\text{boy}
\end{array} \right]^g : (et)t
\]

\[
\left[ \begin{array}{c}
\text{praise}
\end{array} \right]^g : eet
\]

these cannot be combined!

\[
\text{FA } \alpha \beta \rightarrow \alpha \rightarrow \beta
\]
\[
\text{BA } \alpha \rightarrow \alpha \beta \rightarrow \beta
\]
\[
\text{PM } \alpha t \rightarrow \alpha t \rightarrow \alpha t
\]
What is the contribution of *praise every boy* to expressions it is part of?

a quantifier part \(\text{every(boy)}(\lambda x.\ldots)\)

and a property part \(\text{praise}(x)\)

Let’s write instead:

\[
[\text{every(boy)}]_x \vdash \text{praise}(x)
\]
Notation and Operations

\[ \text{[every(boy)]}_x \vdash \text{praise}(x) \]

The general case, with multiple stored quantifiers:

\[ [Q_1]_{x_1}, \ldots, [Q_i]_{x_i} \vdash M \]

The entire point is to ignore what is stored

\[
\frac{M}{\vdash M} \uparrow \quad \frac{\Gamma \vdash M, \Delta \vdash N}{\Gamma, \Delta \vdash M \cdot N} \quad <\ast>\]
Working with Storage

\[
\frac{\text{seem} \vdash \text{Pass}}{\vdash \text{seem}} \\
\frac{\vdash \text{Pass}}{\vdash \text{Pass}} \quad \frac{\frac{[\text{every(boy)}]_x \vdash \text{praise}(x)}{[\text{every(boy)}]_x \vdash \text{Pass}([\text{praise}(x)])}}{[\text{every(boy)}]_x \vdash \text{seem}([\text{Pass}([\text{praise}(x)])])}
\]
Building *praise every boy*

\[
\frac{\text{praise}}{\vdash \text{praise}} \uparrow \frac{\text{every}}{\vdash \text{every}} \uparrow \frac{\text{boy}}{\vdash \text{boy}} \uparrow \\
\frac{\vdash \text{every boy}}{<*>}
\]

type mismatch!
Building *praise every boy*

\[
\begin{array}{c}
\text{praise} \\
\hline
\vdash \text{praise}
\end{array}
\quad \begin{array}{c}
\text{every} \\
\hline
\vdash \text{every}
\end{array}
\quad \begin{array}{c}
\text{boy} \\
\hline
\vdash \text{boy}
\end{array}
\quad \begin{array}{c}
\vdash \text{every boy}
\end{array}
\]

We want to 'insert a trace'

\[
\begin{array}{c}
\vdash M \\
\hline \\
[\![M]\!]_x \vdash x
\end{array}
\]
Building *praise every boy*

\[
\frac{}{\vdash \text{praise}}
\]

\[
\frac{}{\vdash \text{every}} \quad \frac{}{\vdash \text{boy}} \quad \frac{}{\vdash \text{every boy}}
\]

\[
\frac{}{\vdash \text{[every boy]}_x \vdash x}
\]

---

We want to 'insert a trace'

\[
\frac{}{\vdash M} \\
\frac{}{\vdash [M]_x \vdash x}
\]
Building *praise every boy*

\[
\begin{array}{c}
\text{praise} \\ \vdash \text{every} \\
\text{boy} \\ \vdash \text{every} \\
\hline
[\text{every boy}]_x \vdash x
\end{array}
\]

\[
\begin{array}{c}
\vdash \text{praise} \\
\vdash \text{every} \\
\vdash \text{boy} \\
\hline
\vdash M \\
[\text{M}]_x \vdash x
\end{array}
\]

We want to 'insert a trace'
Taking things out of storage

\[
\frac{\text{seem}}{\vdash \text{seem}} \quad \frac{\text{Pass}}{\vdash \text{Pass}} \quad \vdash \frac{[\text{every(boy)}]_x \vdash \text{praise}(x)}{[\text{every(boy)}]_x \vdash \text{Pass(praise}(x))} \quad \vdash \frac{[\text{every(boy)}]_x \vdash \text{seem(\text{Pass(praise}(x)))}}{\text{leftrightarrow}}
\]
Taking things out of storage

\[
\frac{\text{seem}}{\vdash \text{seem}} \quad \frac{\text{Pass}}{\vdash \text{Pass}} \quad \frac{\text{every(boy)}}{\vdash \text{every(boy)}} \quad \frac{\text{praise}(x)}{\vdash \text{praise}(x)} \quad \frac{\text{Pass(praise}(x))}{\vdash \text{Pass(praise}(x))} \quad \frac{\text{seem} \left( \text{Pass(praise}(x)) \right)}{\vdash \text{seem} \left( \text{Pass(praise}(x)) \right)}
\]

\[
\Gamma, [M_i]_{x_i}, \Delta \vdash N \quad \frac{\Gamma, \Delta \vdash M_i \oplus (\lambda x_i. N)}{\langle \cdot \rangle_i^\oplus}
\]

retrieval
Taking things out of storage

\[
\begin{align*}
\text{seem} & \uparrow \quad \text{Pass} \uparrow \\
\therefore \text{seem} & \quad \quad [\text{every}(\text{boy})]_x \vdash \text{praise}(x) \\
\therefore [\text{every}(\text{boy})]_x \vdash \text{Pass}(\text{praise}(x)) & \quad \quad \text{(**)} \\
\therefore [\text{every}(\text{boy})]_x \vdash \text{seem}(\text{Pass}(\text{praise}(x))) & \quad \quad \text{(**)} \\
\therefore \text{every}(\text{boy})(\lambda x.\text{seem}(\text{Pass}(\text{praise}(x)))) & \vdash \langle \cdot \rangle_1^{\text{FA}}
\end{align*}
\]

retrieval

\[
\Gamma, [M_i]_{x_i}, \Delta \vdash N \\
\Gamma, \Delta \vdash \langle \cdot \rangle_{i}^{\oplus}
\]

Greg Kobele (UofC)  LF-Interpretation, Compositionally  Dec 02 11 / 45
Manipulating Stores

**pure**
\[
\frac{M}{\Gamma \vdash M} \uparrow
\]

**apply**
\[
\frac{\Gamma \vdash M \quad \Delta \vdash N}{\Gamma, \Delta \vdash M \ N} \quad <*>
\]

**retrieve**
\[
\frac{\Gamma, [M_i]_{x_i}, \Delta \vdash N}{\Gamma, \Delta \vdash M_i \oplus (\lambda x_i. N)} \quad \langle \cdot \rangle_i^\oplus
\]

**store**
\[
\frac{\Gamma \vdash M}{[M]_x \vdash x} \quad \Box
\]
Understanding stores

$[M_1]_{x_1}, \ldots, [M_i]_{x_i} \vdash N$

$\Rightarrow \lambda k. k \ M_1 \ \ldots \ M_i (\lambda x_1, \ldots , x_i . N)$

Example

$[\text{every boy}]_x \vdash \text{praise } x$

$\Rightarrow \lambda k. k (\text{every boy}) (\lambda x. \text{praise } x)$
Some examples

pure

\[
\begin{align*}
&M \\
\frac{M}{\Downarrow} \\
\frac{\Downarrow}{\lambda k.k \ M \\
\frac{M}{\lambda k.k \ M}
\end{align*}
\]

\[M \uparrow \equiv \lambda k.k \ M\]

storage

\[
\begin{align*}
&M \\
\frac{\Downarrow}{[M]_x \Downarrow} \\
\frac{\Downarrow}{\lambda k.k \ M (\lambda x.x) \\
\frac{\Downarrow}{M \equiv \lambda k.k \ M (\lambda x.x)}
\end{align*}
\]

\[\square m \equiv \lambda k.m (\lambda M.k \ M (\lambda x.x))\]
More notation

idiom brackets

write $(f \ a_1 \ldots \ a_i)$

for $f^\uparrow <**> a_1 <**> \ldots <**> a_i$

application

Forward $f ▷ a := f \ a$

Backward $a ◄ f := f \ a$
Minimalist semantics

\[ \text{[merge]} \mapsto \lambda m, n. (|m \oplus n|) \]
\[ \text{[merge]} \mapsto \lambda m, n. (|m \oplus \Box n|) \]

\[ \text{[move]} \mapsto \lambda m. m \]
\[ \text{[move]} \mapsto \lambda m. \langle m \rangle^k \oplus \]

\[ \text{[ℓ]} = \mathcal{I}(ℓ)^\uparrow \]

for $\oplus \in \{\triangleright, \triangleleft\}$
Recall that

\[ \lambda m, n. (\{ m \triangleright n \}) \]

means

\[ \lambda m, n. (\triangleright)^{\uparrow} <\ast> m <\ast> n \]
Every boy laughs
Every boy laughs
Every boy laughs

⊢ every

⊢ will

⊢ laugh

⊢ move

⊢ merge

⊢ merge

⊢ merge

⊢ merge

⊢ boy

⊢ every
Every boy laughs

\[\vdash \text{move}\]
\[\vdash \text{merge}\]
\[\vdash \text{will}\]
\[\vdash \text{merge}\]
\[\vdash \text{laugh}\]
\[\vdash \lambda m, n. (m \triangleright n)\]
\[\vdash \text{every}\]
\[\vdash \text{boy}\]
Every boy laughs

⊢ will
    \[\text{move}\]
    \[\text{merge}\]

⊢ laugh
    \[\text{merge}\]

⊢ every boy
Every boy laughs

\[
\begin{align*}
\llbracket \text{move} \rrbracket \quad &\quad \vdash \text{will} \quad \lambda m, n. (m \triangleright \square n) \\
\llbracket \text{merge} \rrbracket \quad &\quad \vdash \text{laugh} \quad \vdash \text{every boy}
\end{align*}
\]
Every boy laughs

⊢ will [move]

⊢ will [merge]

[move]

⊢ will [every boy] ⊨ laugh x
Every boy laughs

\[ \llbracket \text{move} \rrbracket \]

\[
\lambda m, n. (|m \triangleright n|)
\]

\[ \vdash \text{will} \quad [\text{every boy}]_x \vdash \text{laugh} \ x \]
Every boy laughs

\[\text{move}\]

\[\text{[every boy]}_x \vdash \text{will (laugh } x\text{)}\]
Every boy laughs

\[ \lambda m. \langle m \rangle^1 \]

\[
\left[ \text{every boy} \right]_x \vdash \text{will (laugh } x) \]
Every boy laughs

⊢ every boy (λx. will (laugh x))
1. Compositionality

2. Cooper Storage Laws

3. Formal Consequences

4. Interpreting Tucking-in
Plan

- Present algebraic laws of cooper storage
- Introduce delimited continuations

Main claim
LF-interpretation à la H&K is based on delimited continuations
Applicative Functor Laws

identity

\[ \text{id} \uparrow <\star> u = u \]

composition

\[ ((\circ \uparrow <\star> u) <\star> v) <\star> w = u <\star> (v <\star> w) \]

homomorphism

\[ f \uparrow <\star> x \uparrow = (f \times) \uparrow \]

interchange

\[ u <\star> x \uparrow = (\lambda P. Px) \uparrow <\star> u \]
Deriving homomorphism

\[
f^\uparrow \ <\ast\> \ x^\uparrow = (f \ x)^\uparrow
\]

\[
\frac{f}{\vdash f}
\quad \frac{x}{\vdash x}
\quad \frac{f \ x}{\vdash f \ x}
\]

\[
\vdash f \ x
\]
Laws Particular to Cooper Storage

1. That which is stored, can be retrieved

\[ \langle E[\square_k M] \rangle^k_N = N^\uparrow \mathrlap{<*> } M \mathrlap{<*> } \langle E[\square_k \text{id}^\uparrow] \rangle^k \]

2. Storing and retrieving vacuity is vacuous

\[ \langle u \mathrlap{<*> } \square_k \text{id}^\uparrow \rangle^k = u \]
Deriving Law 1

 Retrieval from storage

\[
\langle E[\Box_k M] \rangle_N^k = N^\uparrow \oplus M \oplus \langle E[\Box_k \text{id}^\uparrow] \rangle_{\triangleright}
\]

\[
\Gamma, \Delta \vdash \lambda x_k. O \quad \Gamma, \Delta \vdash \text{id} \triangleright (\lambda x_k. O)
\]

\[
\Gamma, \Delta \vdash N \circ M (\lambda x_k. O)
\]
shift0 and reset

- $x$
- $(M \ N)$
- $\lambda x. M$
- $\square_k M$
- $\langle M \rangle^k$

$\beta$ $(\lambda x. M) \ N \rightsquigarrow M[x := N]$

$\eta$ $\lambda x. (M \ x) \rightsquigarrow M$

shift0 $\langle E[\square_k M]\rangle^k \rightsquigarrow M \ (\lambda x. \langle E[x]\rangle^k)$

reset $\langle V \rangle^k \rightsquigarrow V$
shift0 and reset

- $x$
- $(M \; N)$
- $\lambda x. M$
- $\Box_k M$
- $\langle M \rangle^k$

\( \beta \quad \lambda x. M \; N \rightsquigarrow M[x := N] \)

\( \eta \quad \lambda x. (M \; x) \rightsquigarrow M \)

\( \text{shift0} \quad \langle E[\Box_k M]\rangle^k \rightsquigarrow M \; (\lambda x. \langle E[x]\rangle^k) \)

\( \text{reset} \quad \langle V\rangle^k \rightsquigarrow V \)

\( \langle E[\Box_k M]\rangle^k \rightsquigarrow M \; (\lambda x. \langle E[x]\rangle^k) \)

\( \langle E[\Box_k M]\rangle^k \; \triangleright \; = \; M \; <\ast\rangle \; \langle E[\Box_k \text{id}^\uparrow]\rangle^k \)
Minimalist semantics (via delimited continuations)

\[ [\text{merge}] \mapsto \lambda m, n. (|m \oplus n|) \]
\[ [\text{merge}] \mapsto \lambda m, n. (|m \oplus \Box n|) \]

\[ [\text{move}] \mapsto \lambda m. m \]
\[ [\text{move}] \mapsto \lambda m. \langle m \rangle^k \]

\[ [\ell] = I(\ell)^\uparrow \]

for \( \oplus \in \{\triangleright, \triangleleft\} \)
Minimalist semantics (via delimited continuations)

\[
[\text{merge}] \mapsto \lambda m, n. m \oplus n \\
[\text{move}] \mapsto \lambda m. m \\
k \oplus [\ell] = I(\ell)
\]

for \( \oplus \in \{\triangledown, \blacklozenge\} \)
Minimalist semantics (via delimited continuations)

\[ \text{[merge]} \mapsto \lambda m, n. m \oplus n \]
\[ \text{[merge]} \mapsto \lambda m, n. (|m \oplus \Box n|) \]

\[ \text{[move]} \mapsto \lambda m.m \]
\[ \text{[move]} \mapsto \lambda m.\langle m \rangle^k \]

\[ [\ell] = I(\ell) \]

for \( \oplus \in \{\triangleright, \triangleleft\} \)
Minimalist semantics (via delimited continuations)

\[
\begin{align*}
\text{[[merge]]} & \mapsto \lambda m, n. m \oplus n \\
\text{[[merge]]} & \mapsto \lambda m, n. m \oplus \Box_k (\lambda o. n \oplus o)
\end{align*}
\]

\[
\begin{align*}
\text{[[move]]} & \mapsto \lambda m. m \\
\text{[[move]]} & \mapsto \lambda m. \langle m \rangle^k
\end{align*}
\]

\[
[[\ell]] = \mathcal{I}(\ell)
\]

for \( \oplus \in \{\triangleright, \triangleleft\} \)
1. Compositionality

2. Cooper Storage Laws

3. Formal Consequences

4. Interpreting Tucking-in
give examples of the benefits of viewing LF-interpretation compositionally

Main claim
LF-interpretation not only *can* be viewed directly compositionally, it *should* be.
The set of meanings assigned to sentences of a grammar is
- a linear higher order IO language
and thus the set of structures which yield a given $\lambda$-term
- can be found in polynomial time
Decompositionality

any part of a derivation has a normal semantic value

\[
\begin{bmatrix}
\text{[ ]}
\end{bmatrix}
= \begin{bmatrix}
\lambda x.
\end{bmatrix}
\begin{bmatrix}
\text{[ ]}
\end{bmatrix}
\begin{bmatrix}
\text{[ ]}
\end{bmatrix}
\]
Top-down parsing

\[ \lambda m. \]

\[ m \]
Top-down parsing

\[ \lambda m. \langle m \rangle^k \]
Top-down parsing

\[
\lambda t, v. \\
\langle \uparrow \leftrightarrow t \leftrightarrow v \rangle^k
\]
Top-down parsing

\[ \lambda t, v. \langle \uparrow \& t \& v \rangle^k \]
Top-down parsing

\[ \lambda t, v. \langle t \leftrightarrow v \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \langle t \triangleright^* (\triangleright^* V \triangleright^* \square^k_s) \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \left< t \right> \left< V \right> \left< s \right> \]

\[ \left< t \right> \left< (\uparrow \downarrow \left< V \right> \downarrow \right> \left< k \right> \]
Top-down parsing

\[ \lambda t, V, s. \langle t \leftrightarrow (V \leftrightarrow \Box_k s) \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \langle t \leftrightarrow (V \leftrightarrow \Box k s) \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \]

\[ s \leftrightarrow \langle t \leftrightarrow (V \leftrightarrow \Box kid^\uparrow) \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \]

\[ s <\ast> (t <\ast> (V <\ast> \Box \text{kid}^\uparrow))^{k} \]
Top-down parsing

\[ \lambda t, V, s. \]

\[ s \leftrightarrow \langle (\circ \uparrow \leftrightarrow t \leftrightarrow V) \leftrightarrow \Box_k \text{id} \uparrow \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \]
\[ s \leftrightarrow \langle (\circ \uparrow \leftrightarrow t \leftrightarrow V) \leftrightarrow \square_k \text{id} \rangle^k \]
Top-down parsing

\[ \lambda t, V, s. \\ s \leftrightarrow (\circ \uparrow \leftrightarrow t \leftrightarrow V) \]
Top-down parsing

\[ \lambda t, V, d, n. \]

\[
(\triangleright \uparrow <\ast> d <\ast> n) <\ast> (\circ \uparrow <\ast> t <\ast> V)
\]
Top-down parsing

\[ \lambda t, V, d, n. \ (\rhd^\uparrow <**> d <**> n) <**> (\circ^\uparrow <**> t <**> V) \]
Top-down parsing

\[ \lambda t, V, d, n. \ (d \ll out \ n) \ll out (\circ \uparrow \ll out t \ll out V) \]
Top-down parsing

\[ \lambda t, V, n. \ (\text{every}^\uparrow \leftrightarrow n) \leftrightarrow (\circ^\uparrow \leftrightarrow t \leftrightarrow V) \]
Top-down parsing

\[ \lambda t, V. \quad (\text{every} \uparrow \triangleleft \text{boy} \uparrow) \triangleleft (\circ \uparrow \triangleleft t \triangleleft V) \]
Top-down parsing

\[ \lambda t, V. (\textit{every} ^ \uparrow \textit{<*> } \textit{boy} ^ \uparrow) \textit{<*>} (\circ ^ \uparrow \textit{<*> } t \textit{<*> } V) \]
Top-down parsing

\[ \lambda t, V. (every \; boy)^\uparrow <*\> (\circ^\uparrow <*\> t <*\> V) \]
Top-down parsing

\[ \lambda V. \]
\[ (\text{every boy})^{\uparrow} <**> (\circ^{\uparrow} <**> \text{will}^{\uparrow} <**> V) \]
Top-down parsing

\[ \lambda V. \quad (\text{every boy})^\uparrow \triangleleft \rangle (\circ^\uparrow \triangleleft \rangle \text{will}^\uparrow \triangleleft \rangle V) \]
Top-down parsing

\[ \lambda V. \]

\[ (every \ boy)^\uparrow \ <*> \ ((\lambda f. will \circ f)^\uparrow \ <*> \ V) \]
Top-down parsing

\[ \lambda V. (every \ boy)^\uparrow \left<\ast\ast\right> ((\lambda f. will \circ f)^\uparrow \left<\ast\ast\right> V) \]
Top-down parsing

\[ \lambda V. \quad (\lambda P. \text{every boy}(\lambda x. \text{will}(P \ x)))^\uparrow <*> V \]
Top-down parsing

\[(\lambda P. \text{every boy}(\lambda x. \text{will } (P\ x))))^{\uparrow} \leftrightarrow \text{laugh}^{\uparrow}\]
(λP.every boy(λx.will (P x)))↑ <> laugh↑
Top-down parsing

\[
\text{move} \\
\text{merge} \\
\text{will} \quad \text{merge} \\
\text{laugh} \quad \text{merge} \\
\text{every} \quad \text{boy}
\]

\[
(\text{every \ boy}(\lambda x. \text{will} \ (\text{laugh} \ x)))^\uparrow
\]
Ellipsis

\[ \lambda m. (\text{Pass} \circ \text{praise})^\uparrow <\star> \Box m \]

\[
\begin{array}{c}
\text{move} \\
/ \\
\text{merge} \\
/ \\
T_{\text{past}} \\
\end{array}
\quad
\begin{array}{c}
\text{move} \\
/ \\
\text{merge} \\
/ \\
\text{merge} \\
/ \\
v_{\text{pass}} \\
\end{array}
\quad
\begin{array}{c}
\text{move} \\
/ \\
\text{merge} \\
/ \\
T_{\text{past}} \\
\end{array}
\quad
\begin{array}{c}
\text{move} \\
/ \\
\text{merge} \\
/ \\
\text{merge} \\
/ \\
\text{NP} \rightarrow \text{VP} \\
\end{array}
\]

\[
\text{praise} \quad \text{Mary} \\
\text{Harry} \\
\]

\[
\llbracket \text{move} \rrbracket (\llbracket \text{merge} \rrbracket (\llbracket T_{\text{past}} \rrbracket)(\llbracket e \rrbracket(\llbracket \text{Harry} \rrbracket)))
\]
1. Compositionality

2. Cooper Storage Laws

3. Formal Consequences

4. Interpreting Tucking-in
Plan

- give compositional semantics for parasitic scope
- ibid. for tucking-in movement

Main claim
Tucking-in type movement involves complex quantifier formation
Parasitic Scope:

```
S
  N
    everyone

S
  N/(Adj\N)
    same

S
  N
    served

S
  N/(Adj\N)
    same

S
  N
    served
```

Parasitic Scope:
Before all movement

\[ \text{[same]}_Q, \text{[everyone]}_x \vdash \text{serve } x (\text{the (Q waiter)}) \]

The *types* of the stored elements are as follows:

- **same** \( (\text{A}et)et \)
- **everyone** \( (et)t \)

\( \text{A} = (et)et \)
Before all movement

\[[\text{same}]_Q, [\text{everyone}]_x \vdash \text{serve } x \ (\text{the } (Q \ \text{waiter}))\]

The *types* of the stored elements are as follows:

- same  \( (Aet)et \)
- everyone  \( (et)t \)

\[A = (et)et\]

What we want:

\[\vdash \text{everyone } (\text{same } (\lambda Q, x.\text{serve } x \ (\text{the } (Q \ \text{waiter}))))\]
Before all movement

\[ \text{same}_Q, [\text{everyone}]_x \vdash \text{serve } x (\text{the } (Q \text{ waiter})) \]

The *types* of the stored elements are as follows:

\[
\begin{align*}
\text{same} & \quad (A et) et \\
\text{everyone} & \quad (et) t \\
A & \equiv (et) et
\end{align*}
\]

**What we want:**

\[ \vdash (\text{everyone } \circ \text{same}) (\lambda Q, x. \text{serve } x (\text{the } (Q \text{ waiter}))) \]
Parasitic Storage (Sketch)

\[
\frac{[same]_Q, [everyone]_x \vdash serve x (the (Q \ waiter))}{[everyone \circ same]_{Q; x} \vdash serve x (the (Q \ waiter))}
\]

\[\vdash (everyone \circ same) (\lambda Q, x. serve x (the (Q \ waiter)))\]
Parasitic Storage

\[
[same]_Q, [everyone]_x \vdash serve x (the (Q waiter))
\]

\[
[\prod everyone same]_z \vdash let (x, Q) be z in serve x (the (Q waiter))
\]

\[
\Gamma, [M : (\alpha\beta)\gamma]_x, [N : (\delta\alpha\eta)\alpha\beta]_y, \Delta \vdash O : \zeta
\]

\[
\Gamma, [\prod M N : ((\delta \otimes \alpha)\eta)\gamma]_z, \Delta \vdash let (x, y) be z in O : \zeta
\]

where \( \prod M N := M \circ N \circ \text{flip} \circ \text{curry} = \lambda R. M (N (\lambda y, x. R (x, y))) \)
Tucking-in Movement

who \[ t_x \]
\[ t_x \]
bought what

Greg Kobele (UofC)
Before all movement

\[ [\text{who}]_x, [\text{what}]_y \vdash \text{bought } y \ x \]
Before all movement

\[ [\text{who}]_x, [\text{what}]_y \vdash \text{bought } y \ x \]

What we want:

\[ \vdash \text{who } (\lambda x. \text{what } (\lambda y. \text{bought } y \ x)) \]
Before all movement

\[ \text{[who]}_x, \text{[what]}_y \vdash \text{bought } y \times \]

What we want:

\[ \vdash (\text{who} \circ (\lambda R, x.\text{what} (\lambda y. R y x))) \text{ bought} \]
Before all movement

\([\text{who}]_x, [\text{what}]_y \vdash \text{bought } y \times\]

What we want:

\[\vdash (\text{who} \circ (L \text{~what})) \text{ bought}\]

\[L := B(CBC)B\]
\[= \lambda D, R, x. D(\lambda y. Ryx)\]
GQs as arity reducers

A DP denotes a function of type $\forall n.(e^{n+1}t)e^n t$

For $D : (et)t$

$$\mathbb{L}^n D : (e^{n+1}t)e^n t$$

$$= \lambda R, x_1, \ldots, x_n. D (\lambda y. R y x_1 \ldots x_n)$$
Tucked Storage (Sketch)

\[
\begin{align*}
[\text{who}]_x, [\text{what}]_y \vdash \text{bought } y \times \\
[\text{who} \circ (\mathbb{L} \text{ what})]_{x,y} \vdash \text{bought } y \times \\
\vdash (\text{who} \circ (\mathbb{L} \text{ what})) (\lambda x, y.\text{bought } y \times)
\end{align*}
\]
Tucked Storage

\[
[\text{who}]_x, [\text{what}]_y \vdash \text{bought} \ y \ x \\
[\prod \text{who} \ (\llbracket \text{what} \rrbracket)]_z \vdash \text{let} \ (x, y) \ \text{be} \ z \ \text{in} \ \text{bought} \ y \ x
\]

\[
\Gamma, [M : (et)t]_x, [N : (et)t]_y, \Delta \vdash O : \zeta \\
\Gamma, [\prod M \ (\llbracket N \rrbracket) : ((e \otimes e)t)t]_z, \Delta \vdash \text{let} \ (x, y) \ \text{be} \ z \ \text{in} \ O : \zeta^{\text{tuck}}
\]

where \( \prod M \ N := M \circ N \circ \text{flip} \circ \text{curry} = \lambda R. M (N (\lambda y, x. R (x, y))) \)
Parasitic vs Tucked Storage

\[
[\text{everyone}]_x, [\text{same}]_Q \vdash \text{serve } x \ (\text{the } (Q \ \text{waiter})) \\
[\prod \text{everyone same}]_z \vdash \text{let } (x, Q) \text{ be } z \text{ in serve } x \ (\text{the } (Q \ \text{waiter}))
\]

\[
[\text{who}]_x, [\text{what}]_y \vdash \text{bought } y \ x \\
[\prod \text{who (L what)}]_z \vdash \text{let } (x, y) \text{ be } z \text{ in bought } y \ x
\]
\[
[\mathcal{M}]_x, [\mathcal{N}]_y \vdash O \\
[\prod \mathcal{M} \mathcal{N}]_z \vdash \text{let } (x, y) \text{ be } z \text{ in } O
\]

\[
[\mathcal{M}]_x, [\mathcal{N}]_y \vdash O \\
[\prod \mathcal{M} (\bot \mathcal{N})]_z \vdash \text{let } (x, y) \text{ be } z \text{ in } O
\]
Parasitic vs Tucked Storage

\[ \mathcal{M}, \mathcal{N}_y \vdash O \]

\[ \prod M (\mathbb{L}^n N)_z \vdash \text{let } (x, y) \text{ be } z \text{ in } O \]
LF-interpretation is in fact directly compositional

- This gives us access to a wealth of results and tools
  - generative capacity
  - generation
  - delimited continuations
- gives new questions to explore
  - semantic bootstrapping?!
  - conditioning parsing on meaning?!
  - connex to cogsci via probabilistic programming?!
- and addresses old ones