MRS Algebra Discussion

DELPH-IN 2025

Outline

- Questions about the MRS Algebra and how it functions
- Questions about the MRS Algebra with respect to the ERG
- Explanation of how I am using a modified version of the MRS Algebra in my work
    - looking for feedback on issues I have run into

General questions about the algebra

[QUESTION/PROMPT]

- My understanding, shown in purple

- Additional notes/corrections provided by the audience, shown in orange

In this section, each slide will present a question concerning some aspect of the MRS algebra. Then, if I have one, my answer will be included in purple, and the goal of the discussion will be to expand/correct my understanding of the various prompts and document that in orange.

What is the MRS algebra?

- Formal definitions of SEMENT (Semantic Element) objects and the operations that can be applied to two SEMENT objects to compose a larger SEMENT object

- Notes/corrections:
  - hopefully emily wrote this down
  -

What makes it an algebra?

- At a high level, my understanding is that it defines the elements to be operated on (SEMENTs) and how they can be operated on. Also, I believe it's key that the result is always another SEMENT which allows further composition with the same operations

- Notes/corrections:

What is the purpose of the MRS algebra?

- From the paper, I gather that the MRS formalism enforced “monotonic accumulation of EPs” but that the details of how composition is carried out and how variables were equated was not yet formalized and the algebra was an attempt to rectify this (???)

- Notes/corrections:

What is a SEMENT? How is it different than an MRS?

- A SEMENT is defined as a 5-tuple with the following components:
  - A “hook” consisting of a LBL and an INDEX
  - A set of slots to be plugged during composition
  - A bag of Elementary Predicates (EPs)
  - A set of equalities between variables
  - A bag of HCONs conditions
- An MRS is the same but without the set of slots to be plugged, as it is considered “finished”
- An MRS also includes a GTOP which is qeq to the highest(?) LBL in the scope tree

- Notes/corrections:

What are the possible operations used for composition?

- Non-scopal composition (e.g. composition between a pre-nominal adjective and a noun)
- Scopal composition (e.g. composition between a quantifier and a noun)

- Notes/corrections:

Composition Operations

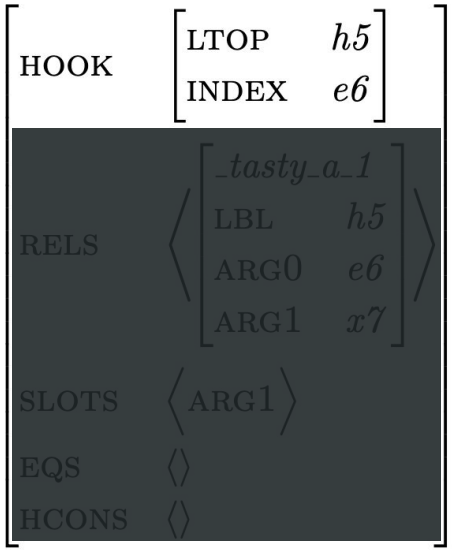
- Assume the following names for the elements of the SEMENT tuple
  - hook = the hook consisting of the LBL and INDEX
  - slots = set of slots to be plugged
  - rels = bag of EPs
  - eqs = list of equalities between variables
  - hcons = bag of handle constraints
- **Assume the following names for the SEMENTs participating in the operation:**
  - **FUNC = functor SEMENT**
  - **ARG = argument SEMENT**
  - **RES = result SEMENT**
- Assume that each element of the SEMENT can be accessed using dot notation
  - e.g. FUNC.slots refers to the slots list on the functor SEMENT

Non-scopal Composition

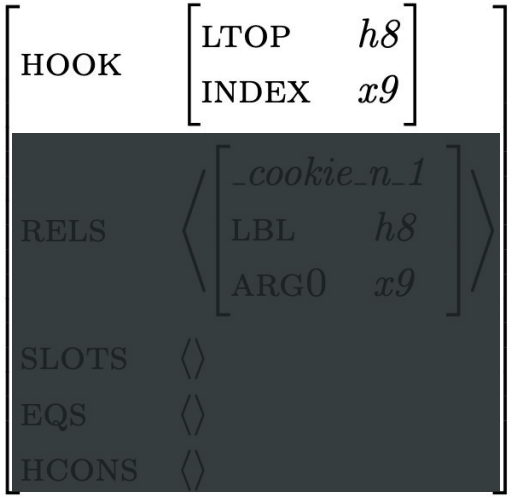
Where the hole labeled x is being plugged, composition occurs as follows:

1. RES.hook = FUNC.hook
2. RES.rels = FUNC.rels ⊕ ARG.rels
3. RES.slots = (FUNC.slots - FUNC.slots.x) ⊕ ARG.slots
4. RES.eqs = *Tr*(FUNC.eqs ∪ ARG.eqs ∪ {FUNC.slots.x = ARG.hook.index} ∪ {FUNC.hook.lbl = ARG.hook.lbl})
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons

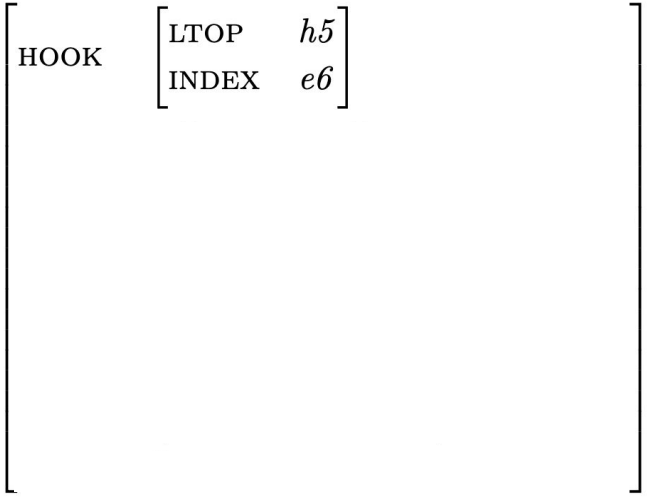
1. RES.hook = FUNC.hook



FUNC

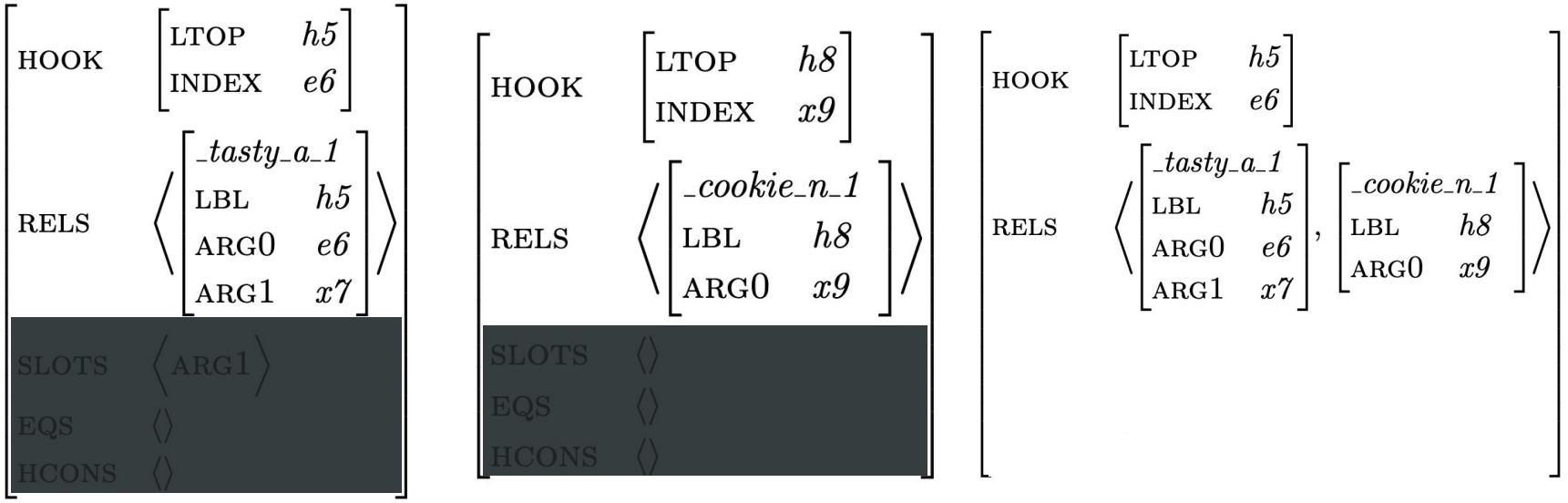


ARG



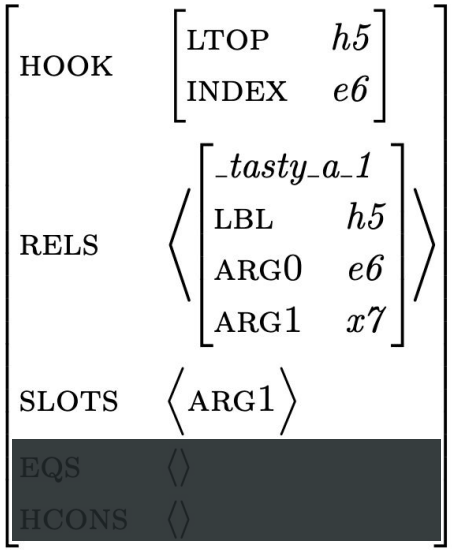
RES

2. RES.rels = FUNC.rels ⊕ ARG.rels

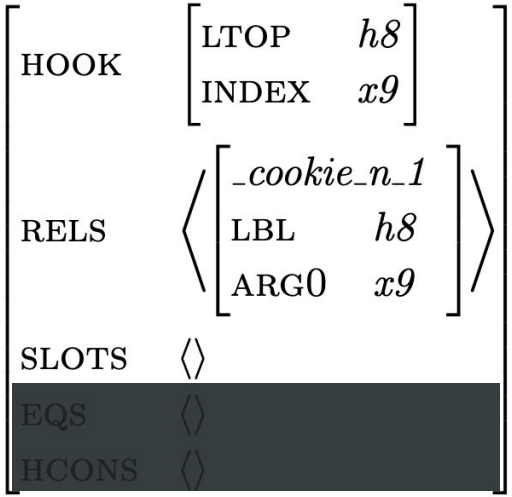


FUNC ARG RES

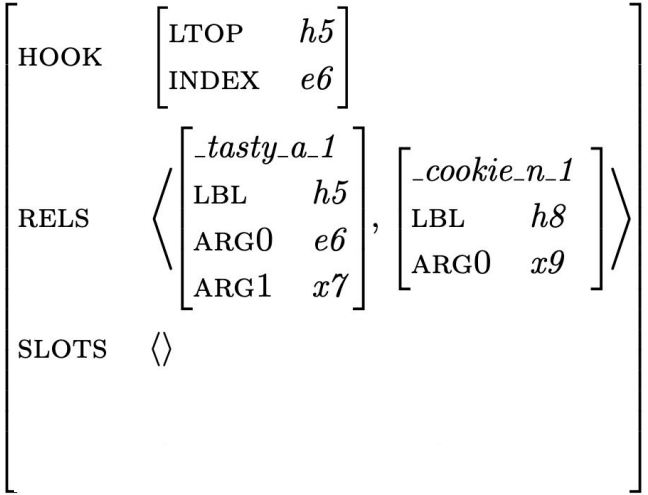
3. RES.slots = (FUNC.slots - FUNC.slots.x) ⊕ ARG.slots



FUNC

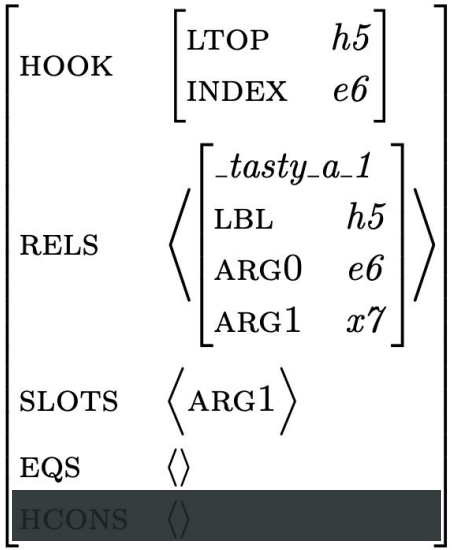


ARG

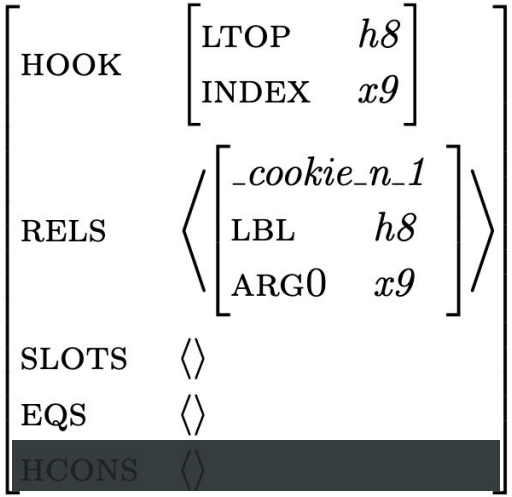


RES

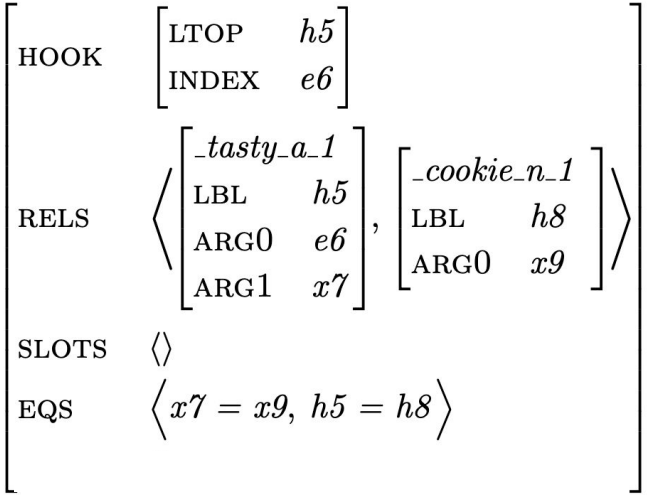
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs ∪ {FUNC.slots.x = ARG.hook.index} ∪ {FUNC.hook.lbl = ARG.hook.lbl})



FUNC



ARG



RES

5. RES.hcons = FUNC.hcons ⊕ ARG.hcons

$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h5 \\ \text{INDEX} & e6 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_tasty\_a\_1 \\ \text{LBL} & h5 \\ \text{ARG0} & e6 \\ \text{ARG1} & x7 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle \text{ARG1} \rangle \\ \text{EQS} & \langle\rangle \\ \text{HCONS} & \langle\rangle \end{bmatrix}$$

FUNC

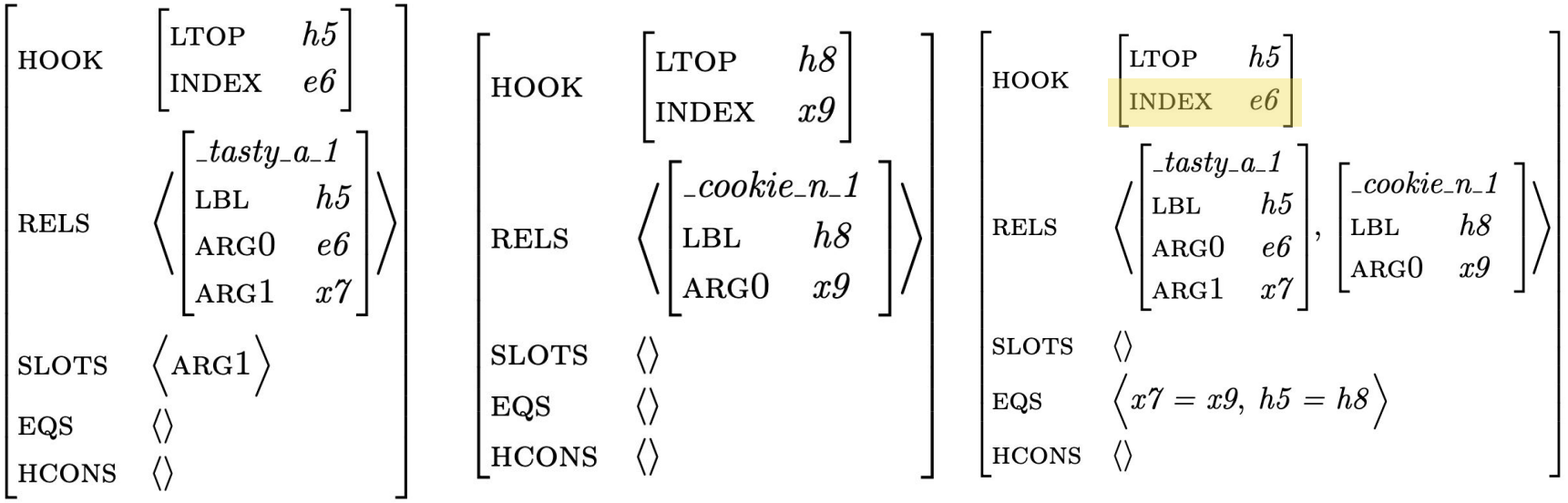
$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h8 \\ \text{INDEX} & x9 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_cookie\_n\_1 \\ \text{LBL} & h8 \\ \text{ARG0} & x9 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle\rangle \\ \text{EQS} & \langle\rangle \\ \text{HCONS} & \langle\rangle \end{bmatrix}$$

ARG

$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h5 \\ \text{INDEX} & e6 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_tasty\_a\_1 \\ \text{LBL} & h5 \\ \text{ARG0} & e6 \\ \text{ARG1} & x7 \end{bmatrix}, \begin{bmatrix} \_cookie\_n\_1 \\ \text{LBL} & h8 \\ \text{ARG0} & x9 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle\rangle \\ \text{EQS} & \langle x7 = x9,\ h5 = h8 \rangle \\ \text{HCONS} & \langle\rangle \end{bmatrix}$$

RES

but wait … wouldn't the hook coming from the adjective prevent further composition of the noun phrase?



FUNC ARG RES

Scopal Composition (e.g. “probably sleeps”)

Where the hole labeled x is being plugged, composition occurs as follows:

1. RES.hook = FUNC.hook
2. RES.rels = FUNC.rels ⊕ ARG.rels
3. RES.slots = (FUNC.slots - FUNC.slots.x) ⊕ ARG.slots
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs)
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons ⊕ [FUNC.slots.x $=_q$ ARG.hook.lbl]

Scopal Composition (e.g. “probably sleeps”)

Where the hole labeled x is being plugged, composition occurs as follows:

1. RES.hook = FUNC.hook
2. RES.rels = FUNC.rels ⊕ ARG.rels
3. RES.slots = (FUNC.slots - FUNC.slots.x) ⊕ ARG.slots
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs)
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons ⊕ [FUNC.slots.x $=_q$ ARG.hook.lbl]

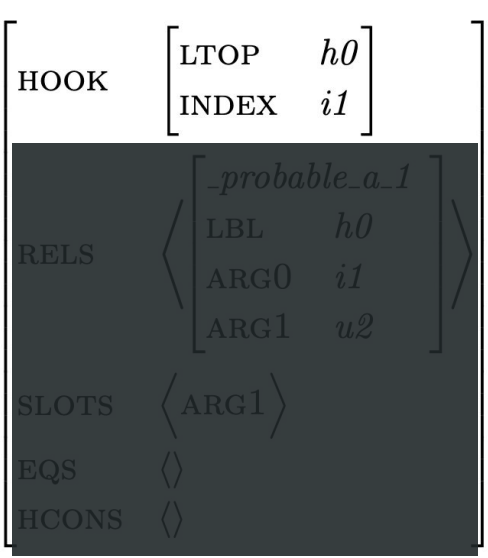
We will not repeat the full composition definition, since it is unchanged from that in §2 apart from the addition of the append operation on hcons and a slight complication of eq to deal with the handle/index pairs:

$$eq(op(a_1, a_2)) = Tr(eq(a_1) \cup eq(a_2) \cup \{hdle(hook(a_1)) = hdle(hole(a_2)), ind(hook(a_1)) = ind(hole(a_2))\})$$

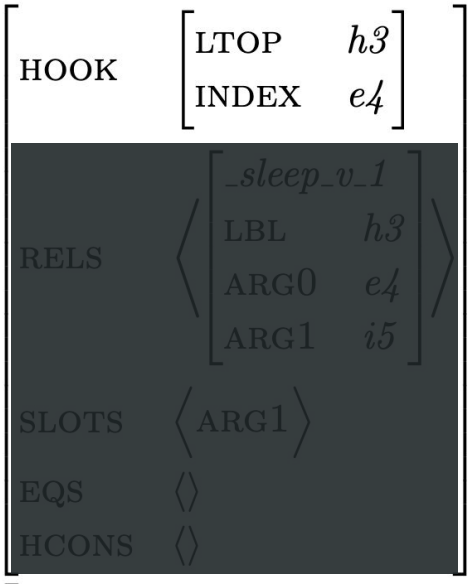
Unclear on how this affects the definition...

- ...

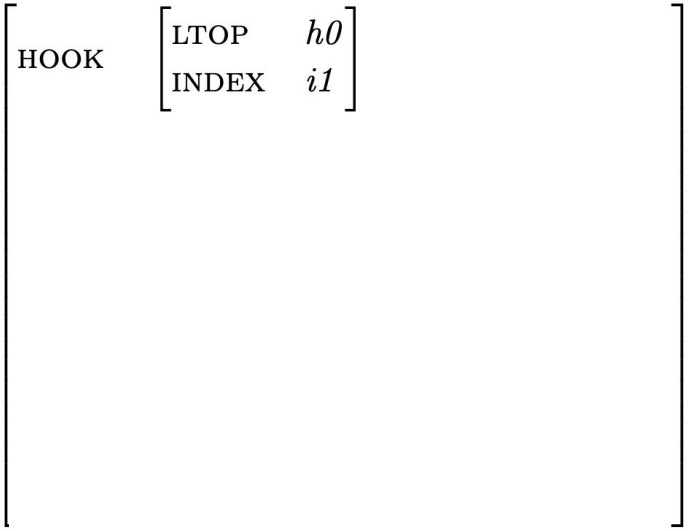
1. RES.hook = FUNC.hook



FUNC

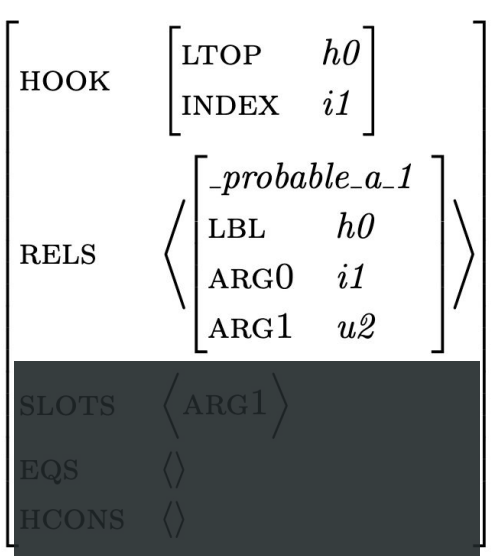


ARG

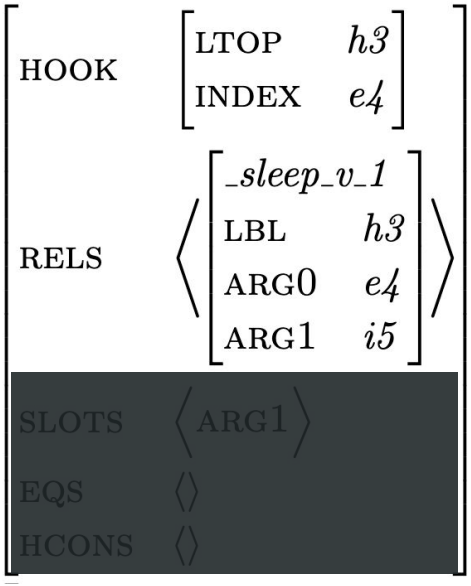


RES

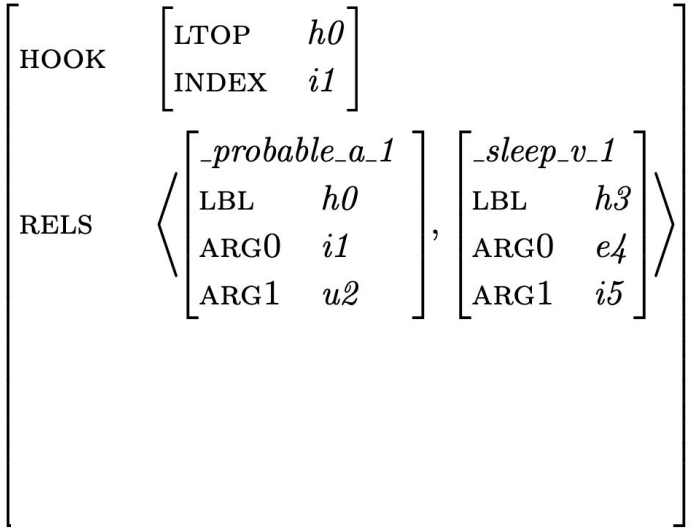
2. RES.rels = FUNC.rels ⊕ ARG.rels



FUNC

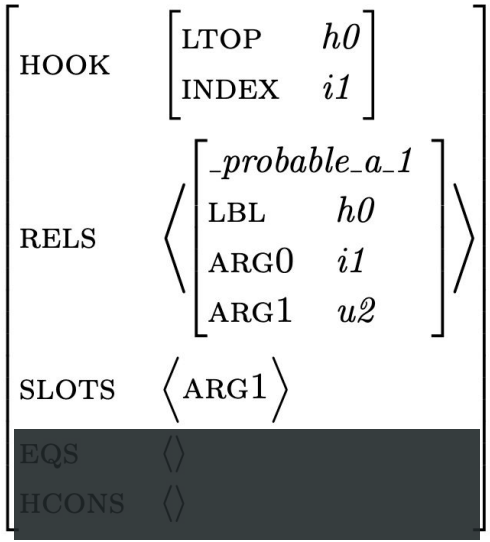


ARG

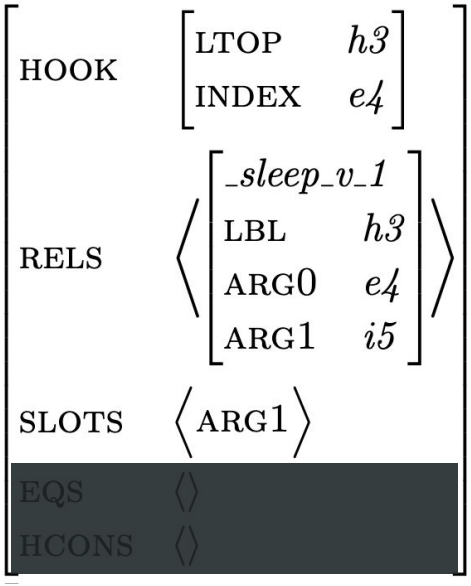


RES

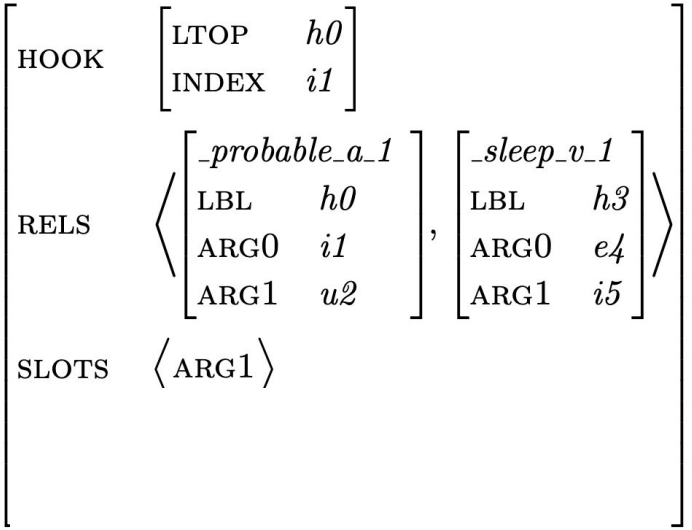
3. RES.slots = (FUNC.slots - FUNC.slots.x) ⊕ ARG.slots



FUNC

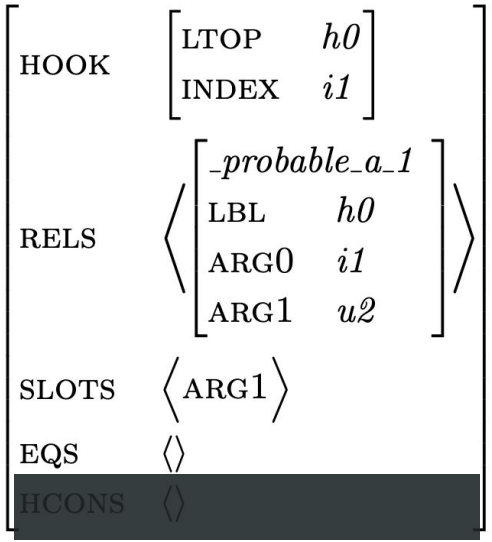


ARG

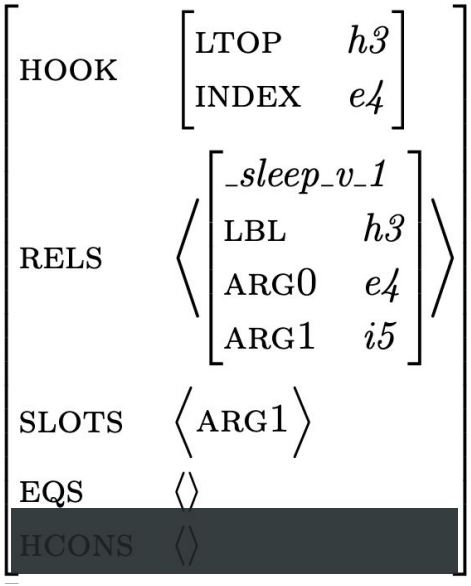


RES

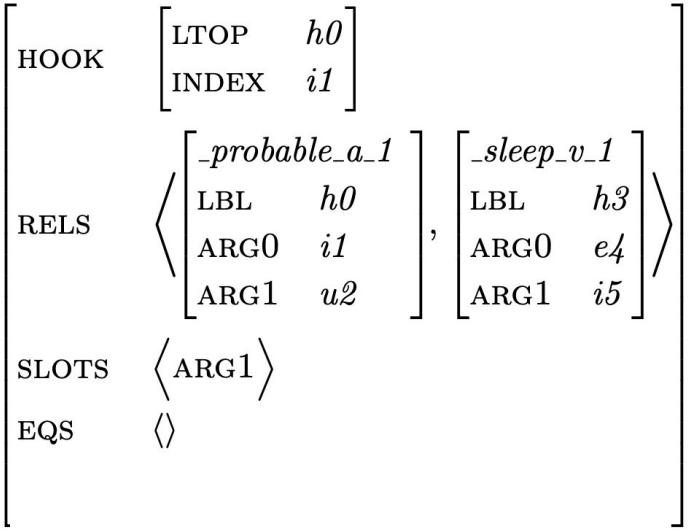
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs)



FUNC

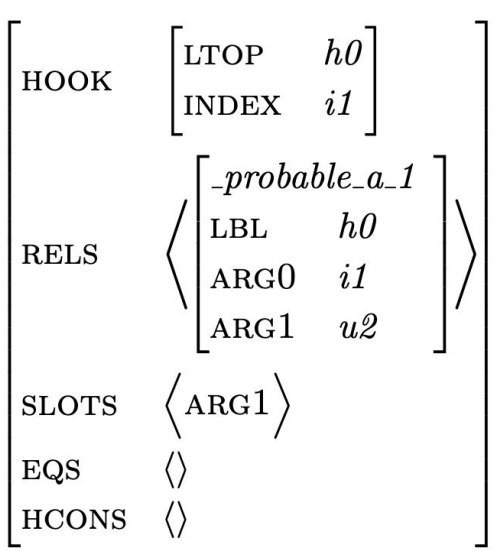


ARG

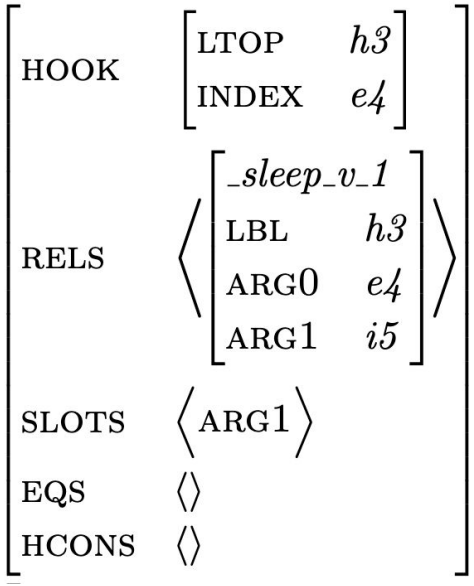


RES

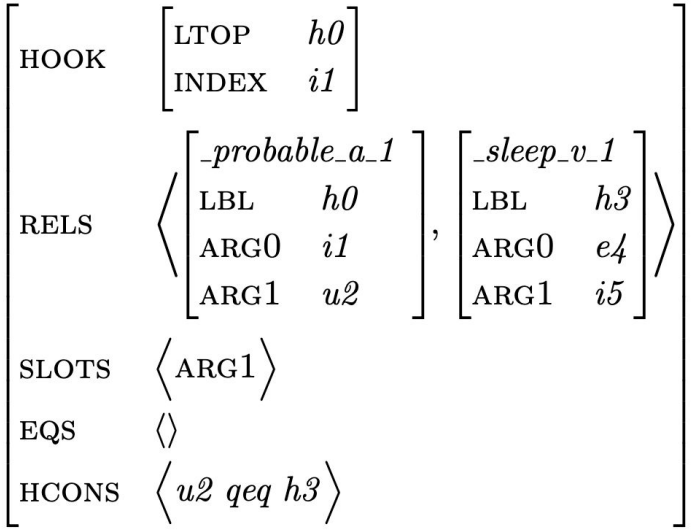
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons ⊕ [FUNC.slots.x =q ARG.hook.lbl]



FUNC



ARG



RES

Shouldn’t there be two versions of scopal composition?

- For the scopal composition in “probably sleeps,” the following must occur…
  - Establish a QEQ relationship between the scopal slot (_probable_a_1.ARG1) and the LTOP of the SEMENT for *sleeps*
  - Remove believe.ARG2 from list of slots
- But for scopal composition in “the cookie,” the following must occur…
  - Establish an equality between the _the_q.ARG0 and the INDEX of the SEMENT for *cookie*
  - Establish a QEQ relationship between the _the_q.RSTR and the LTOP of the SEMEN for *cookie*
  - Remove *both* quant.ARG0 and quant.RSTR from slots list

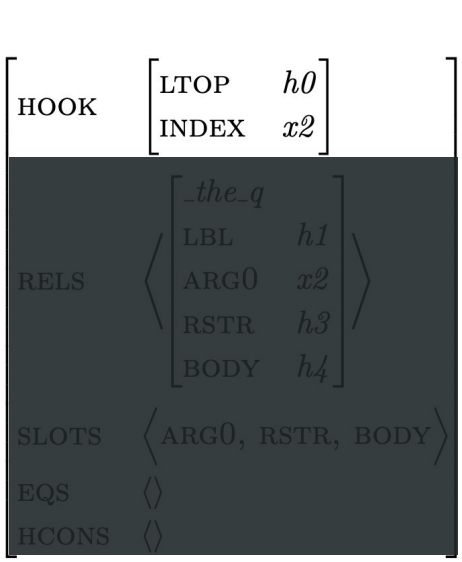
- Notes/corrections:

Scopal Composition for Quantifiers

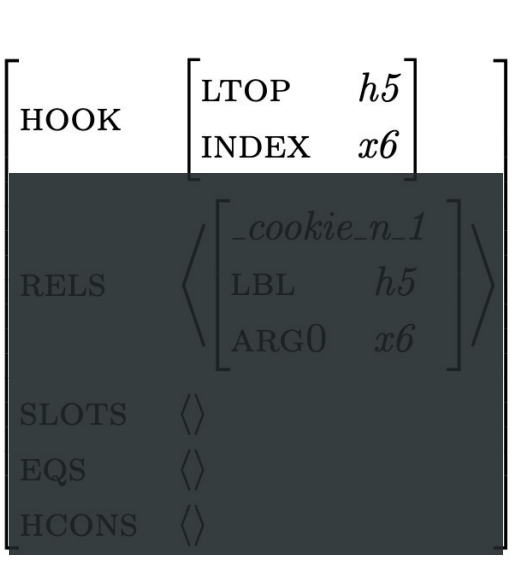
Composition occurs as follows:

1. RES.hook = FUNC.hook
2. RES.rels = FUNC.rels $\oplus$ ARG.rels
3. RES.slots = (FUNC.slots - FUNC.slots.ARG0 - FUNC.slots.RSTR) $\oplus$ ARG.slots
4. RES.eqs = Tr(FUNC.eqs $\cup$ ARG.eqs $\cup$ {FUNC.slots.ARG0 = ARG.hook.index})
5. RES.hcons = FUNC.hcons $\oplus$ ARG.hcons $\oplus$ [FUNC.slots.RSTR $=_q$ ARG.hook.lbl]

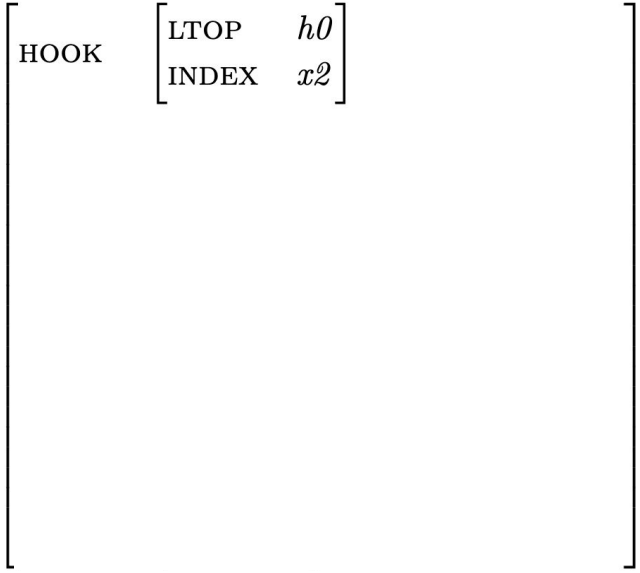
1. RES.hook = FUNC.hook



FUNC

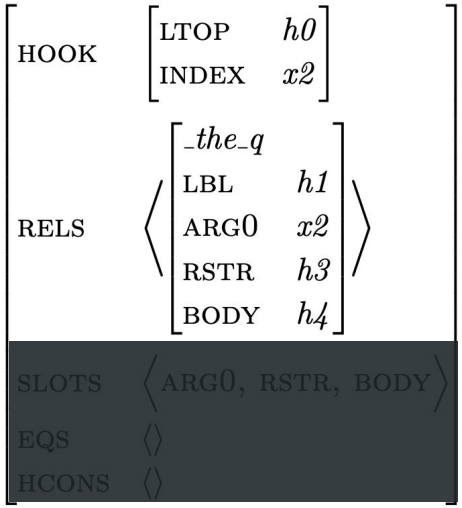


ARG

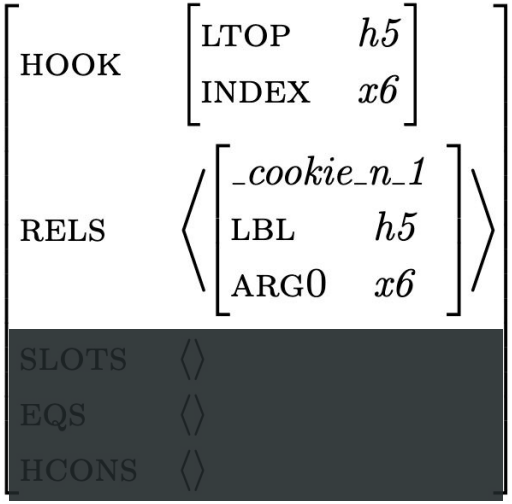


RES

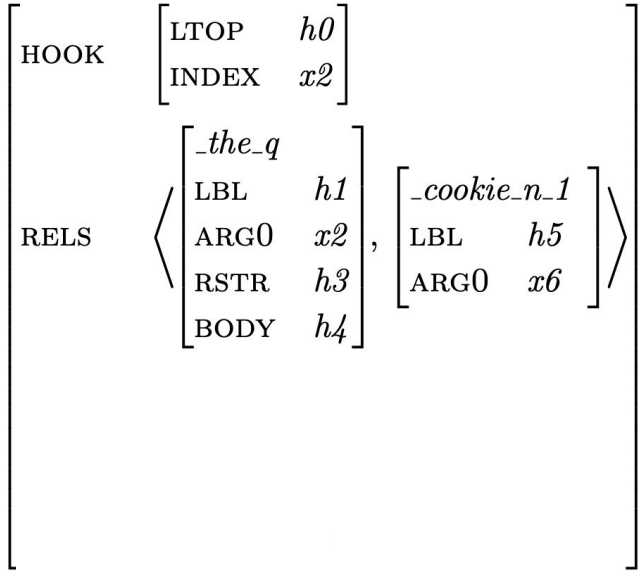
2. RES.rels = FUNC.rels ⊕ ARG.rels



FUNC

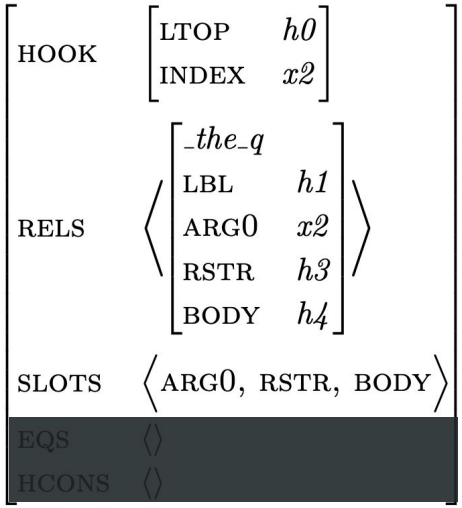


ARG

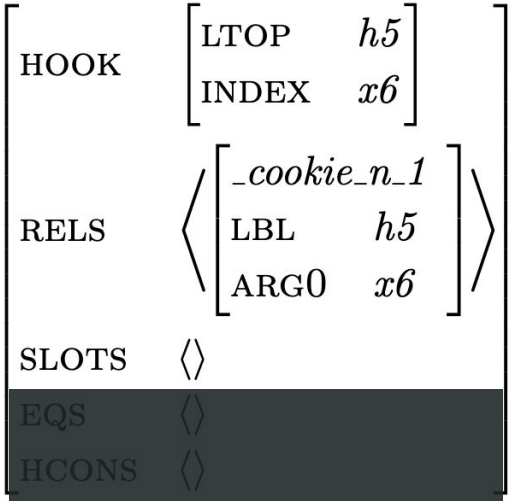


RES

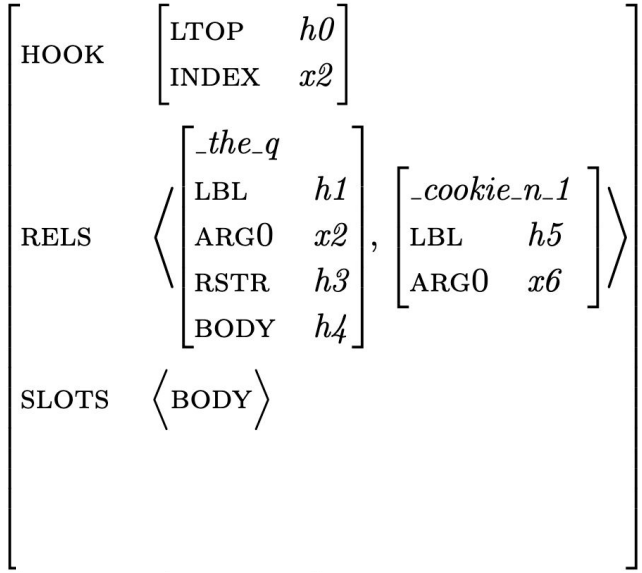
3. RES.slots = (FUNC.slots - FUNC.slots.ARG0 - FUNC.slots.RSTR) ⊕ ARG.slots



FUNC

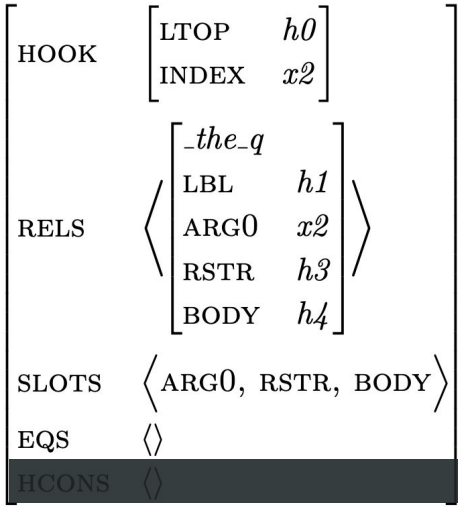


ARG

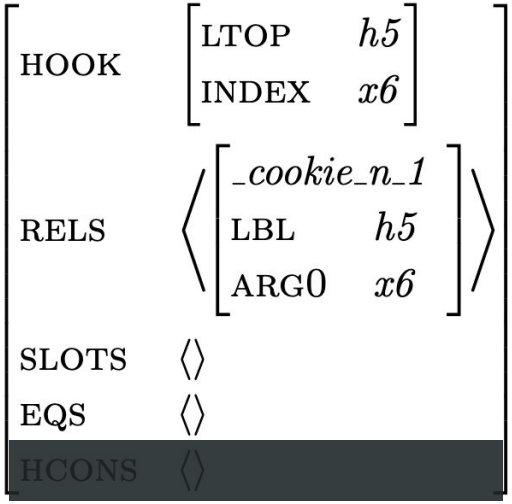


RES

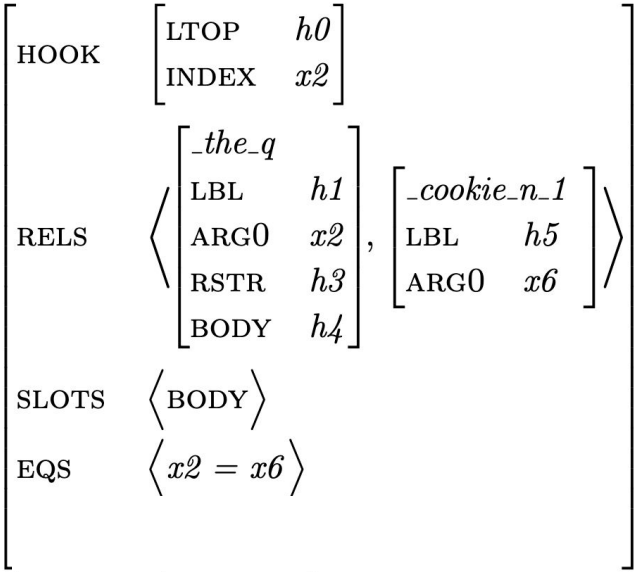
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs ∪ {FUNC.slots.ARG0 = ARG.hook.index})



FUNC



ARG



RES

5. RES.hcons = FUNC.hcons ⊕ ARG.hcons ⊕ [FUNC.slots.RSTR =q ARG.hook.lbl]

$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h0 \\ \text{INDEX} & x2 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_the\_q & \\ \text{LBL} & h1 \\ \text{ARG0} & x2 \\ \text{RSTR} & h3 \\ \text{BODY} & h4 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle \text{ARG0}, \text{RSTR}, \text{BODY} \rangle \\ \text{EQS} & \langle\rangle \\ \text{HCONS} & \langle\rangle \end{bmatrix}$$

FUNC

$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h5 \\ \text{INDEX} & x6 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_cookie\_n\_1 & \\ \text{LBL} & h5 \\ \text{ARG0} & x6 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle\rangle \\ \text{EQS} & \langle\rangle \\ \text{HCONS} & \langle\rangle \end{bmatrix}$$

ARG

$$\begin{bmatrix} \text{HOOK} & \begin{bmatrix} \text{LTOP} & h0 \\ \text{INDEX} & x2 \end{bmatrix} \\ \text{RELS} & \left\langle \begin{bmatrix} \_the\_q & \\ \text{LBL} & h1 \\ \text{ARG0} & x2 \\ \text{RSTR} & h3 \\ \text{BODY} & h4 \end{bmatrix}, \begin{bmatrix} \_cookie\_n\_1 & \\ \text{LBL} & h5 \\ \text{ARG0} & x6 \end{bmatrix} \right\rangle \\ \text{SLOTS} & \langle \text{BODY} \rangle \\ \text{EQS} & \langle x2 = x6 \rangle \\ \text{HCONS} & \langle h3 \ qeq \ h5 \rangle \end{bmatrix}$$

RES

Wrap-up

- After having gone through these operations, what doesn't match the algebra as defined?
  - ...
- When is it required to equate LTOPS?
  - ...
- Does passing the HOOK from the semantic functor always work?
  - ...
- Did we confirm/debunk the idea that there should be more than one type of scopal composition?
  - ...

The ERG and The Algebra

What is the relationship between the ERG and the algebra?

- ...

Where does the ERG adhere to the algebra?

- ...

Where does the ERG go “off algebra,” and why?

- …

For any given construction, how can I figure out if it adheres to the algebra?

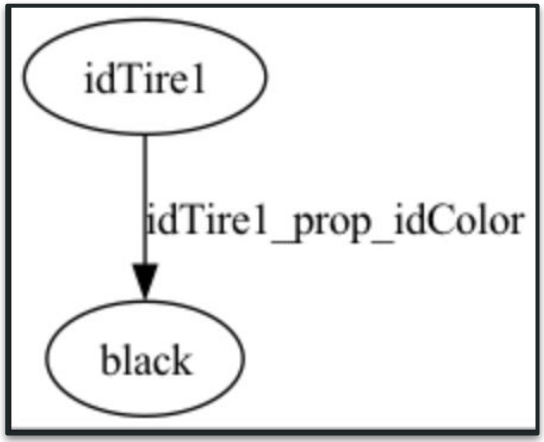
- ...

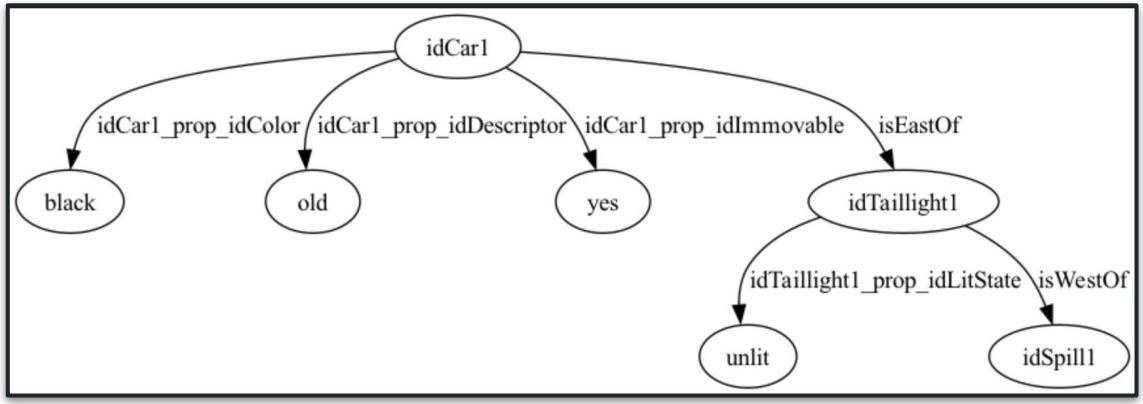
The MRS Algebra and my work

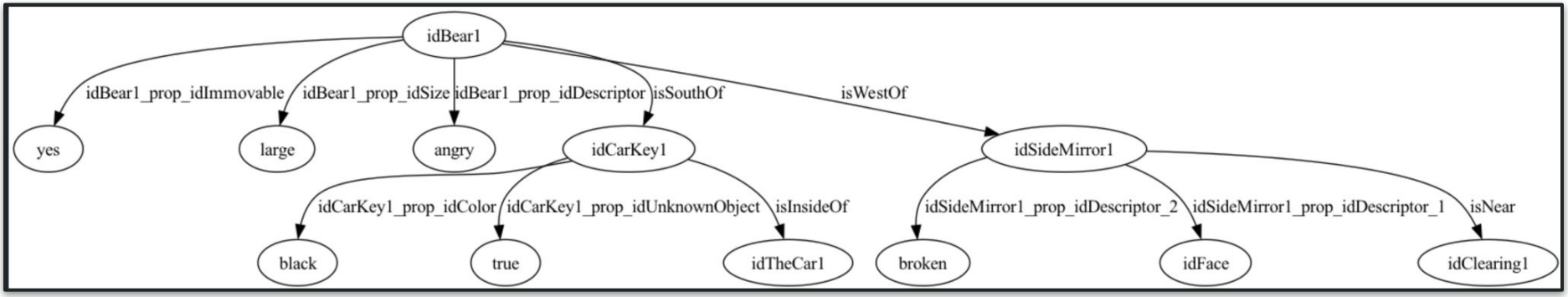
What am I doing with the MRS Algebra?

- In my work, I am taking structured data, such as directed graphs that describe an entity, and converting those graphs into an MRS to pass to the ERG for English text generation
- As I traverse a graph, I convert each node into a SEMENT and then perform composition between nodes using information from the edge to determine the type of composition that must be performed

Example graphs





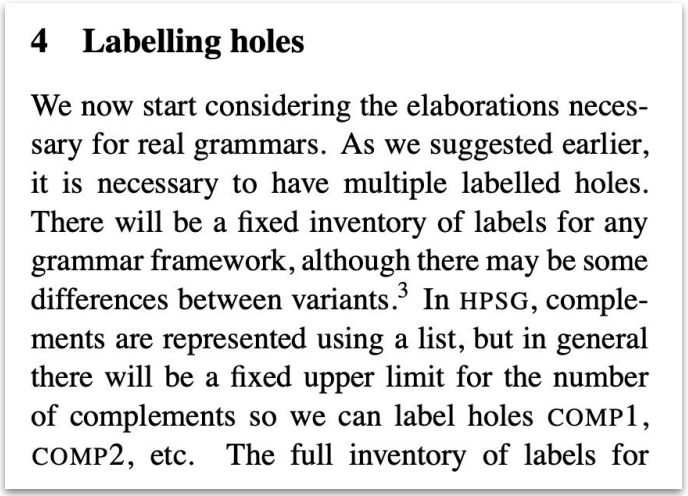


Challenges

1. Slot labeling
2. Passing slots
3. Passing hook information
4. Order of composition

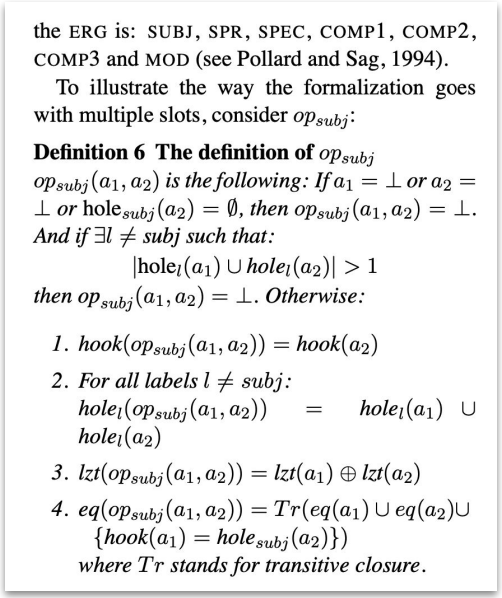
#1 Slot labeling

- Slots are not “labeled” in the way that the algebra defines
- In my case, I just use the generic argument names (ARG1, ARG2, etc)



4 Labelling holes

We now start considering the elaborations necessary for real grammars. As we suggested earlier, it is necessary to have multiple labelled holes. There will be a fixed inventory of labels for any grammar framework, although there may be some differences between variants.[3] In HPSG, complements are represented using a list, but in general there will be a fixed upper limit for the number of complements so we can label holes COMP1, COMP2, etc. The full inventory of labels for



the ERG is: SUBJ, SPR, SPEC, COMP1, COMP2, COMP3 and MOD (see Pollard and Sag, 1994).

To illustrate the way the formalization goes with multiple slots, consider op_{subj}:

Definition 6 **The definition of** op_{subj}

$op_{subj}(a_1, a_2)$ is the following: If $a_1 = \perp$ or $a_2 = \perp$ or $\text{hole}_{subj}(a_2) = \emptyset$, then $op_{subj}(a_1, a_2) = \perp$. And if $\exists l \neq subj$ such that:

$$|\text{hole}_l(a_1) \cup hole_l(a_2)| > 1$$

then $op_{subj}(a_1, a_2) = \perp$. Otherwise:

1. $hook(op_{subj}(a_1, a_2)) = hook(a_2)$
2. *For all labels $l \neq subj$:* $hole_l(op_{subj}(a_1, a_2)) = hole_l(a_1) \cup hole_l(a_2)$
3. $lzt(op_{subj}(a_1, a_2)) = lzt(a_1) \oplus lzt(a_2)$
4. $eq(op_{subj}(a_1, a_2)) = Tr(eq(a_1) \cup eq(a_2) \cup \{hook(a_1) = hole_{subj}(a_2)\})$ *where Tr stands for transitive closure.*

#2 Passing slots

- Consider “the locked box”
  - The EP for _lock_v_1 has two slots: ARG1, ARG2
  - The EP for _box_n_1 has one slot: ARG1
- While the ARG1 in each EP serve a different syntactic functions, and therefore would probably have names like SUBJ and COMP1*, when I’m working purely in MRS there is no way to “know” this
- Then, when performing non-scopal composition, both ARG1 slots will be passed up and then there will be no distinction between them in future composition

*Wouldn’t this mean there are two COMP1 slots though? So it seems the problem exists even when the slots are properly labeled

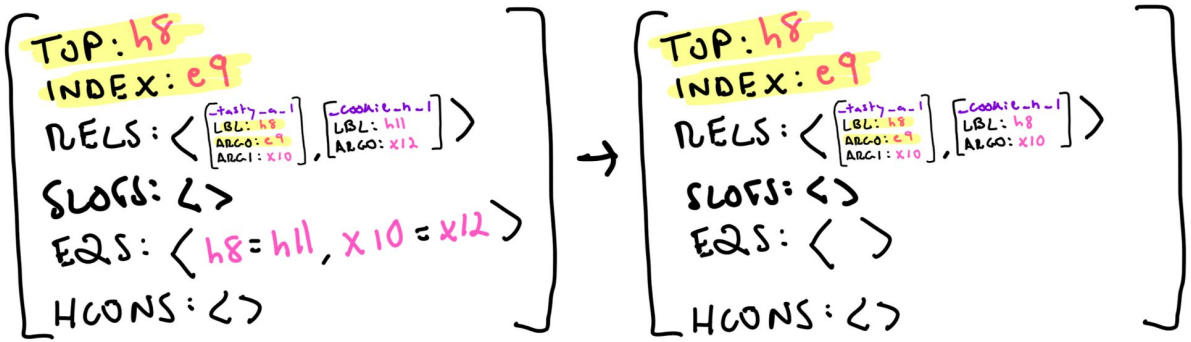
#2 Passing slots

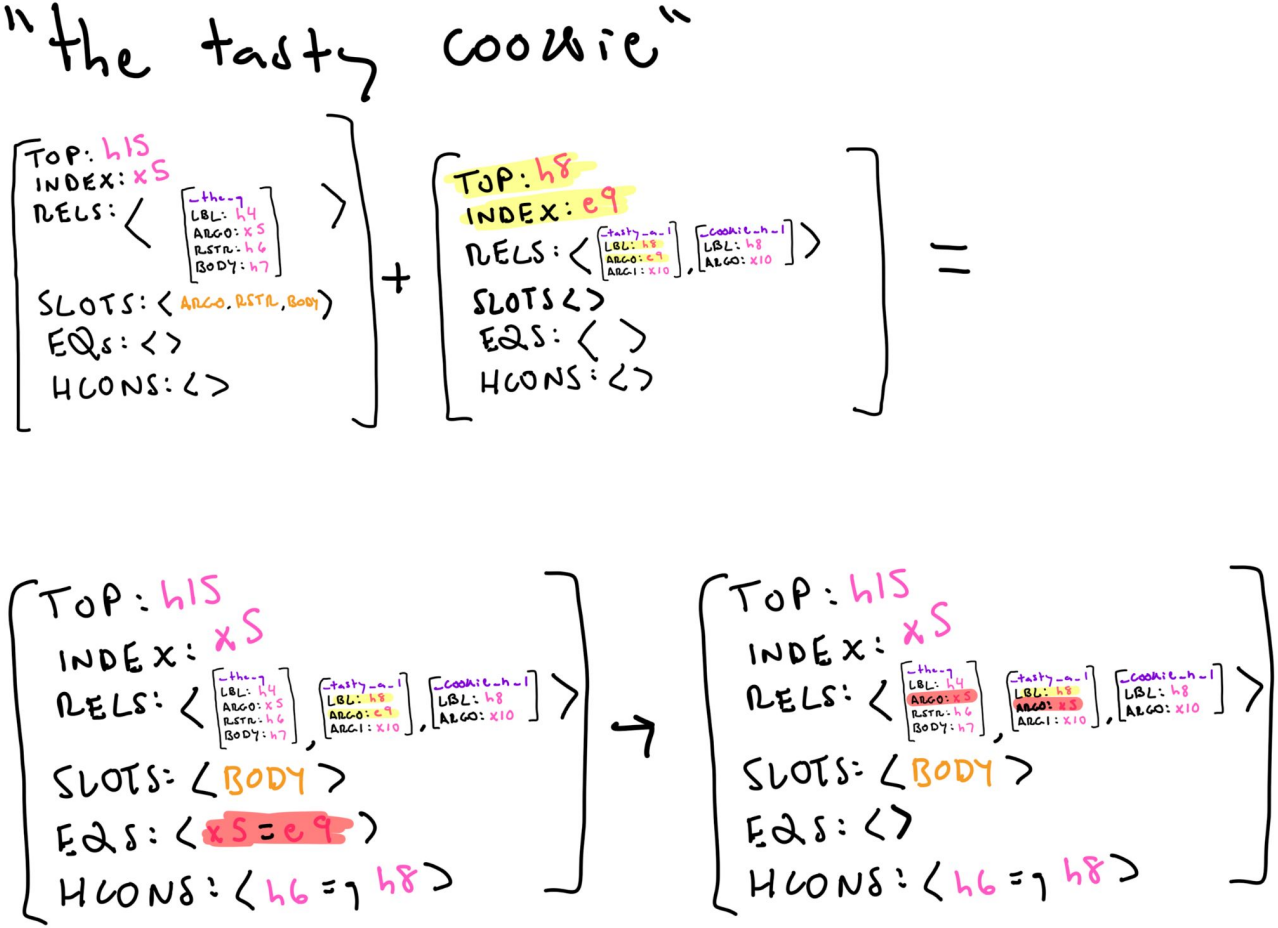
- Solution: only pass slots from one SEMENT
- This prevents such conflicts from ever coming up, but also “locks me out” from filling certain slots in the future
    - does this also make it no longer an algebra?
- So in the previous example, if I had a graph indicating there was a locked box of chocolates, if I compose “the locked box” first, and assuming I always take the slots from the semantic functor, I would then be unable to add the information that it’s a box of chocolates later

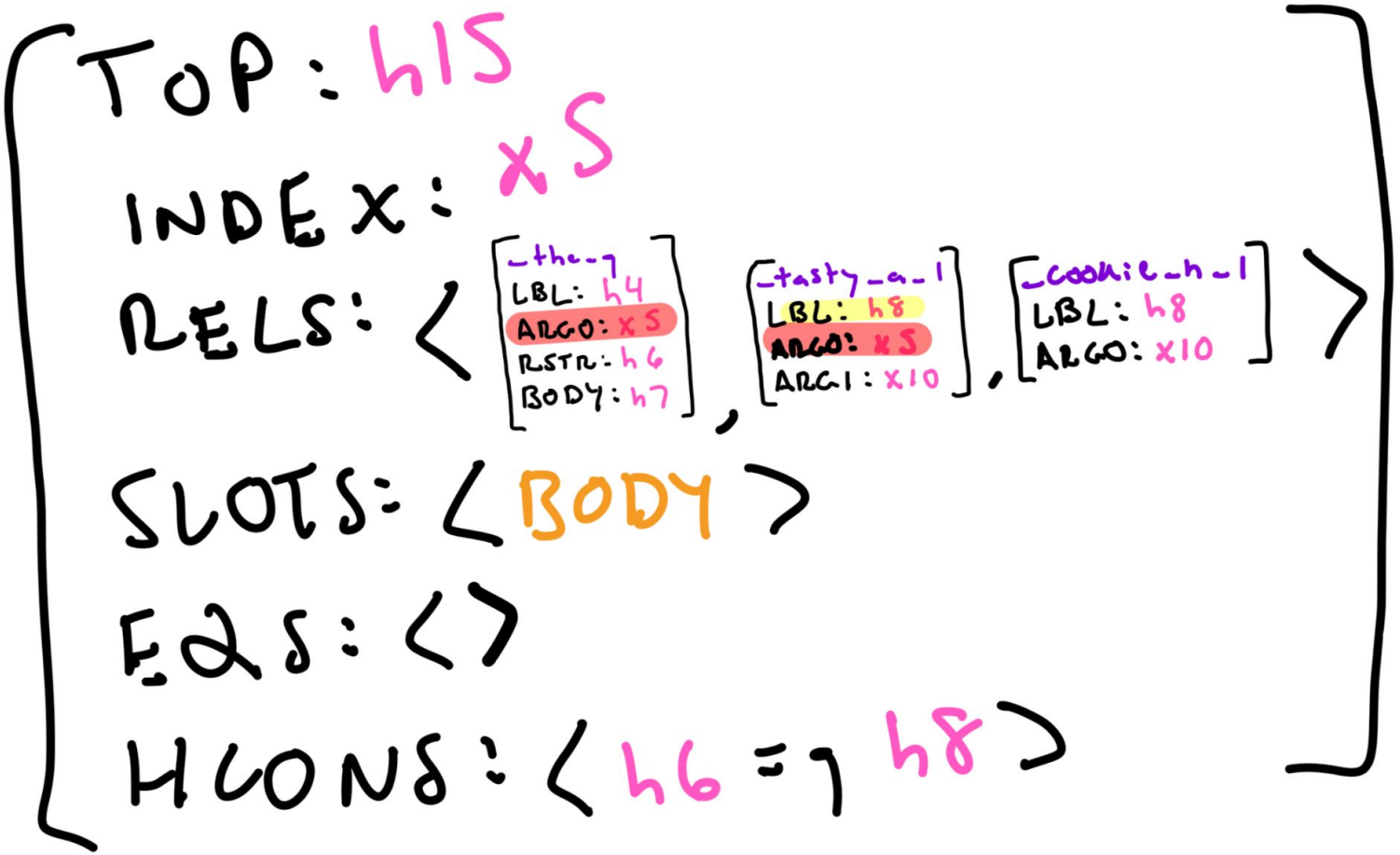
#3 Passing hook information

- The algebra’s definitions of semantic operators states that the hook information should always come from the semantic functor
- … but does this work?
- Consider the NP “the tasty cookie”

"tasty cookie"




"the tasty cookie"
TOP: h15
INDEX: x5
RELS: < [_the_1 LBL: h4 ARG0: x5 RSTR: h6 BODY: h7] >
SLOTS: < ARG0, RSTR, BODY >
EQs: < >
HCONS: < >
+
TOP: h8
INDEX: e9
RELS: < [_tasty_a_1 LBL: h8 ARG0: e9 ARG1: x10], [_cookie_n_1 LBL: h8 ARG0: x10] >
SLOTS < >
EQs: < >
HCONS: < >
=
TOP: h15
INDEX: x5
RELS: < [_the_1 LBL: h4 ARG0: x5 RSTR: h6 BODY: h7], [_tasty_a_1 LBL: h8 ARG0: e9 ARG1: x10], [_cookie_n_1 LBL: h8 ARG0: x10] >
SLOTS: < BODY >
EQs: < x5 = e9 >
HCONS: < h6 =q h8 >
TOP: h15
INDEX: x5
RELS: < [_the_1 LBL: h4 ARG0: x5 RSTR: h6 BODY: h7], [_tasty_a_1 LBL: h8 ARG0: x5 ARG1: x10], [_cookie_n_1 LBL: h8 ARG0: x10] >
SLOTS: < BODY >
EQs: < >
HCONS: < h6 =q h8 >


TOP: h15
INDEX: x5
RELS: <
[_the_7
LBL: h4
ARG0: x5
RSTR: h6
BODY: h7]
[_tasty_a_1
LBL: h8
ARG0: x5
ARG1: x10]
[_cookie_n_1
LBL: h8
ARG0: x10]
>
SLOTS: <BODY>
EQS: <>
HCONS: <h6 =q h8>

#3 Passing the hook information

- Solution: pass the hook information from the “syntactic” head
- Additionally, pass the slots from the syntactic head as well
  - since I am only passing slots from one SEMENT, always pass the slots from the same SEMENT that contributes the hook information
- This results in me having two versions of the non-scopal operation:
  - op_non_scopal_functor_hook
    - passes hook and slots from semantic functor
  - op_non_scopal_argument_hook
    - passes hook and slots from semantic argument

#4 Order of composition

- Because of these decisions, it essentially means that I must perform my composition roughly in the order of syntactic composition, which I don’t love
- For example, I wish I could compose “extremely tasty cookie” either by composing “extremely tasty” first or “tasty cookie” first, but it looks like I can’t do that
- I have the sense that a fully constructed MRS is “agnostic” to syntactic construction, so I would expect to be able to compose it without considering this, but I don’t think that I can

“extremely tasty cookie”

“tasty cookie” + “extremely” vs. “extremely tasty” + “cookie”

“tasty cookie”	HOOK from *cookie*
“extremely tasty cookie”	“extremely” ARG1 plugged with cookie.ARG0 🚫

“extremely tasty”	HOOK from *tasty*, ARG1 slot from *tasty*
“extremely tasty cookie”	“tasty” ARG1 plugged with cookie.ARG0 ✅

- Is there a solution that doesn’t require performing composition in a certain order?
- …

Challenges Recap

1. Slot labeling
    - Use the generic argument names (ARG1, ARG2, etc)
2. Passing slots
    - Only pass slots from the “syntactic” head
3. Passing hook information
    - Pass the hook information from the “syntactic” head
4. Order of composition
    - Order of composition must occur roughly in the order of the anticipated order of syntactic composition

Summary of my implementation

- SEMENT object consists of
  - hook (LTOP and INDEX)
  - bag of EPs
  - set of slots
  - set of EQs
  - bag of HCONS
- Operations:
  - op_non_scopal_functor_hook
  - op_non_scopal_argument_hook
  - op_scopal
  - op_scopal_quantifier

Final thoughts/feedback?

- ...