MRS

LING567

2/3/2025

Outline

- Liz
  - MRS Preface
  - MRS Mechanics
- Emily
  - MRS in the Matrix
  - Lab 5 Preview

MRS Preface

MRS: Design Principles

- The design of the representations of particular linguistic phenomena follow the following general strategies/design principles
  - Represent all semantic distinctions which are syntactically or morphologically marked
  - Underspecify semantic distinctions which aren’t
  - Abstract away from non-semantic information (word order, case, ...)
  - Close paraphrases should have comparable or identical MRS representations
  - Aim for consistency across languages
  - Allow for semantic differences across languages

MRS: Goals

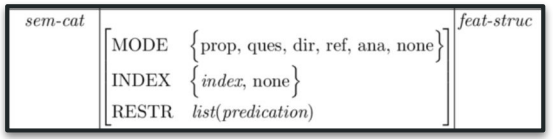
- The design of the MRS formalism answers the following four general goals:
  - *Adequate representation of NL semantics*
    - “The framework must allow linguistic meanings to be expressed correctly”
  - *Grammatical compatibility*
    - “Semantic representations must be linked clearly to other kinds of grammatical information (most notably syntax)”
  - *Computational tractability*
    - “It must be possible to process meanings and to check semantic equivalence efficiently and to express relationships between semantic representations straightforwardly”
  - *Underspecifiability*
    - “Semantic representations should allow underspecification (leaving semantic distinctions unresolved), in such a way as to allow flexible, monotonic, resolution of such partial semantic representations”

Flat Semantics

- MRS is ‘flat’ bc the EPs are never embedded in one another
    - Makes transfer rules easier because the generator can treat it as a bag (not set) of elements where order is irrelevant
    - … what else?

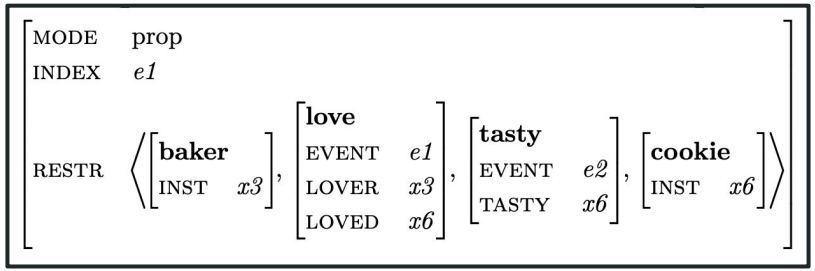
MRS Mechanics

Semantic Structure in LING566



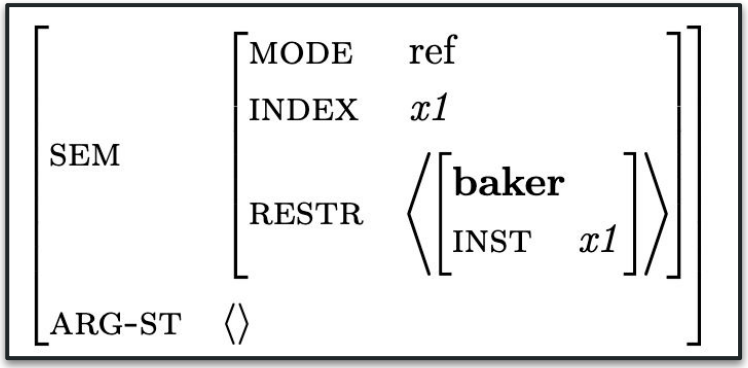
- SEM feature of type *sem-cat* with three features
  - **MODE** — semantic mode, helps “differentiate between different the kinds of meaning that are associated with various syntactic categories”
  - **INDEX** — aids in composition
  - **RESTR** — list of predications
    - predicate label
    - list of semantic arguments

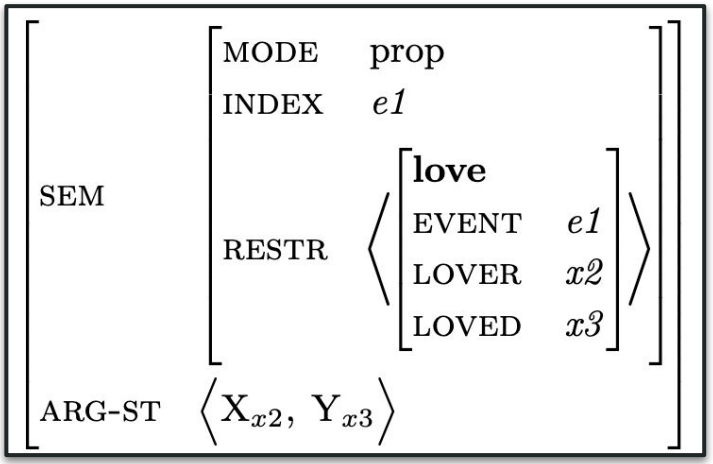
- *Bakers love tasty cookies.*



Semantic Composition in LING566

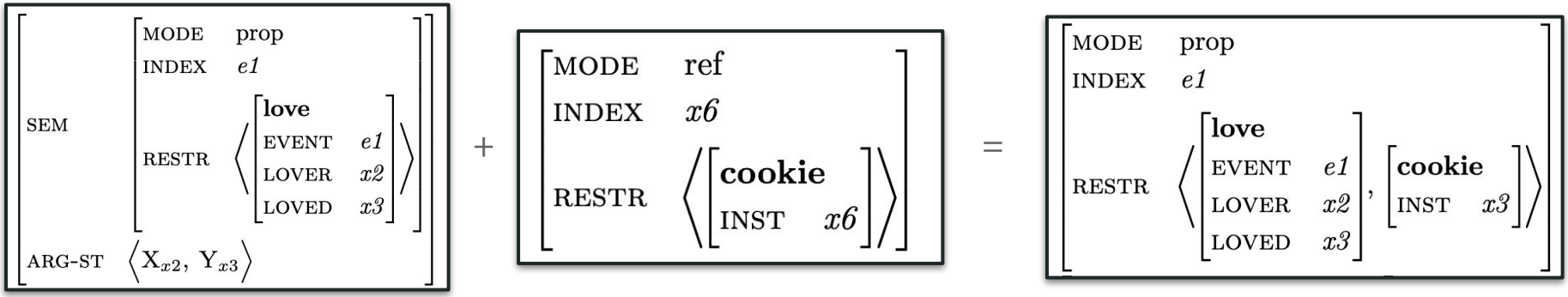
- Each lexical entry specifies the following
  - The semantic argument the INDEX corresponds to
  - The semantic argument(s) the INDEX features of the elements on the ARG-ST correspond to





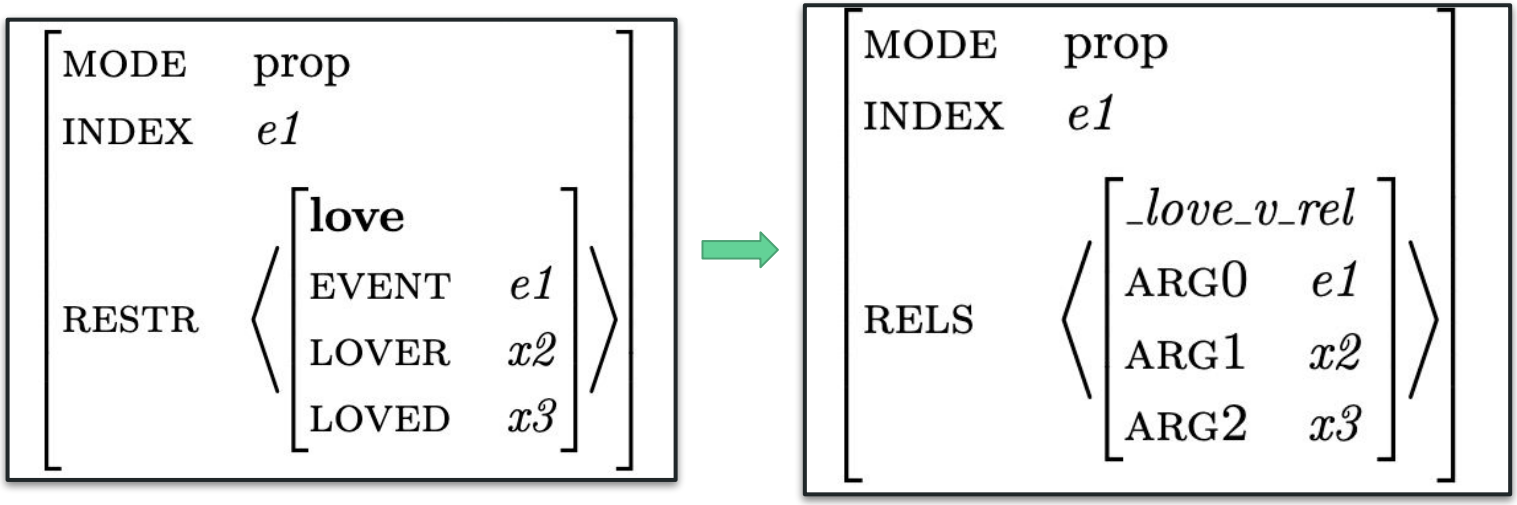
Semantic Compostion in LING566 as tuples

- Semantic object: <MODE, INDEX, RESTR>
- When composing a new semantic object for a mother node, *S*, from two child semantic objects, *H* (head) and *NH* (nonhead):
  - $S_{MODE} = H_{MODE}$
  - $S_{INDEX} = H_{INDEX}$
  - $S_{RESTR} = H_{RESTR} \oplus NH_{RESTR}$



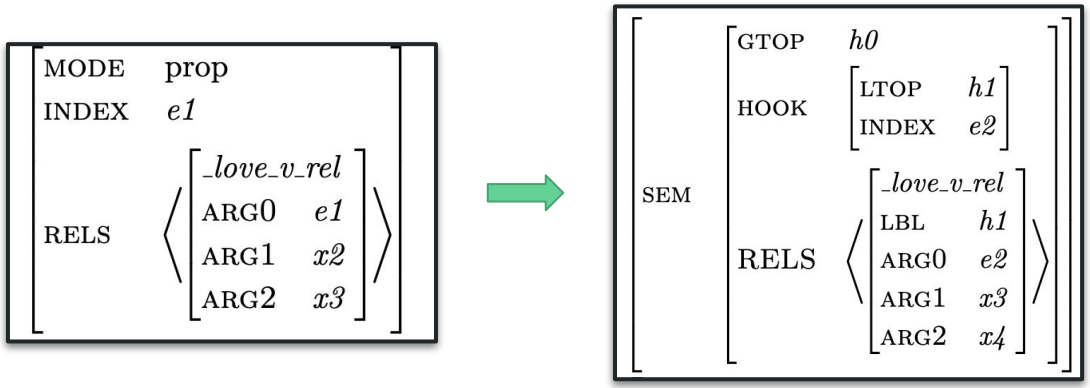
Moving to MRS: Naming changes

- Different naming convention for predicate labels
- Different naming convention for semantic arguments
- Different name for RESTR feature



Moving to MRS: Structural changes

- MODE is now a property of INDEX
- New features for outer structure:
  - GTOP — Global Top
  - HOOK
    - LTOP — Local Top
    - INDEX — moved here
  - HCONS — Handle Constraints
- New features for individual predications
  - LBL — Label



MRS as a tuple

- <GTOP, HOOK, RELS, HCONS>
  - **GTOP** — Global top handle
  - **HOOK** — Contains pointers used for both non-scopal and scopal composition
    - **INDEX, LTOP**
  - **RELS** — Relations, bag of EPs
  - **HCONS** — Handle Constraints, partial scope information

<MODE, INDEX, RESTR> ... <GTOP, HOOK, RELS, HCONS>

- Added HOOK
  - Added LTOP
  - Moved INDEX here ... MODE information is now a property on INDEX
- RESTR ➜ RELS
- Added GTOP feature
- Added HCONS feature

MRS as a tuple

- <GTOP, HOOK, RELS, HCONS>
  - **GTOP** — Global top handle
  - **HOOK** — Contains pointers used for both non-scopal and scopal composition
    - **INDEX** — Pointer into some EP for non-scopal composition
    - **LTOP** — Local top handle, pointer into some EP for scopal composition
  - **RELS** — Relations, bag of EPs
    - **EP** — Elementary Predications
      - **LBL** — handle
      - Relation
      - List of ordinary variable arguments
      - List of handles corresponding to scopal arguments
  - **HCONS** — Handle Constraints, partial scope information

Scope “Aside”

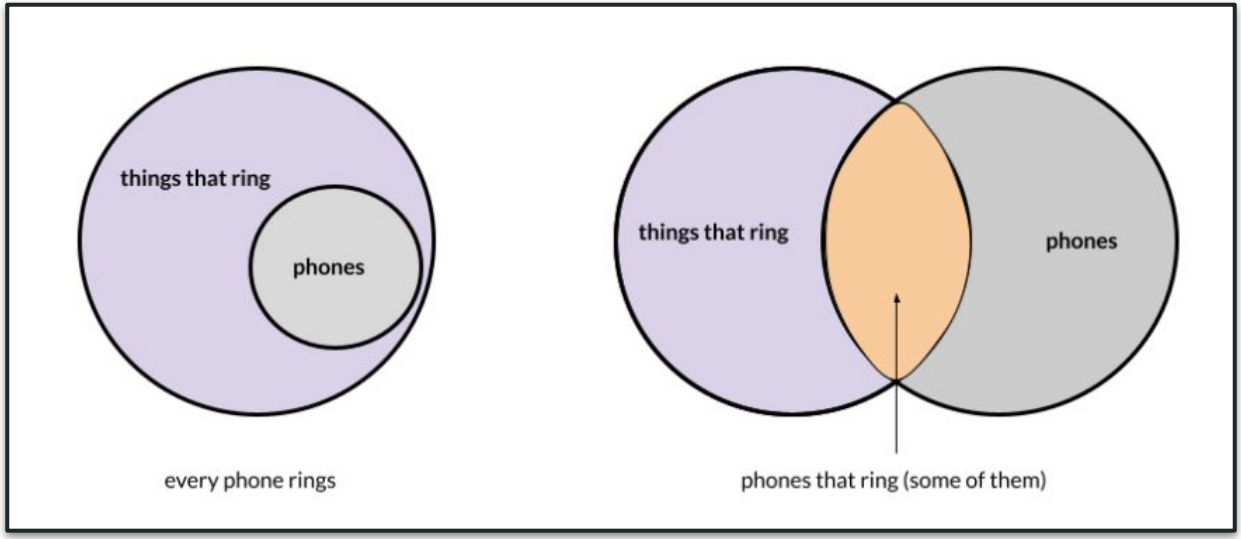
okay but what is scopal composition...

- For certain lexical items, the relationship they have to their arguments is “looser” than others.
- This can be most clearly demonstrated with sentences involving multiple quantifiers that give rise to semantic ambiguity.

- *Every dog chases some cat.*
  - Every dog chases one *particular* cat.
  - Every dog chases a cat, not necessarily the same cat.

Quantifiers as relationships between sets

- Quantifiers can be thought of as declaring a relationship between two sets

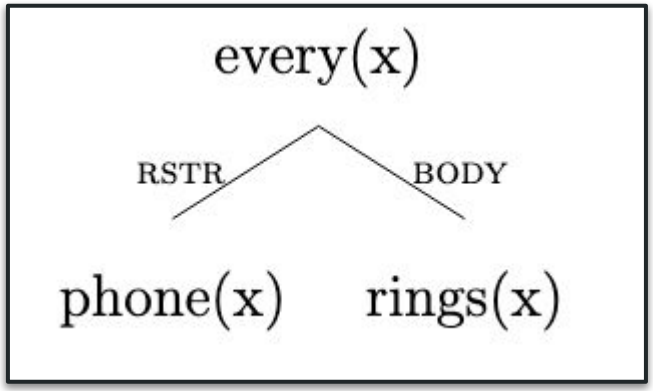


Quantifiers as operations

- *every cat* — choose every member of the set of cats
- *most cats* — choose more than half of the members of the set of cats
- *some cat* — choose one member of the set of cats
- *the cat* — choose one specific member of the set of cats
- *a cat* — choose one member of the set of cats

Representing quantifiers

- In MRS, a quantifier has four components:
  - the handle (LBL)
  - the bound variable, i.e. the entity that is being tracked across both sets (ARG0)
  - the restriction set (RSTR)
  - the body set (BODY)



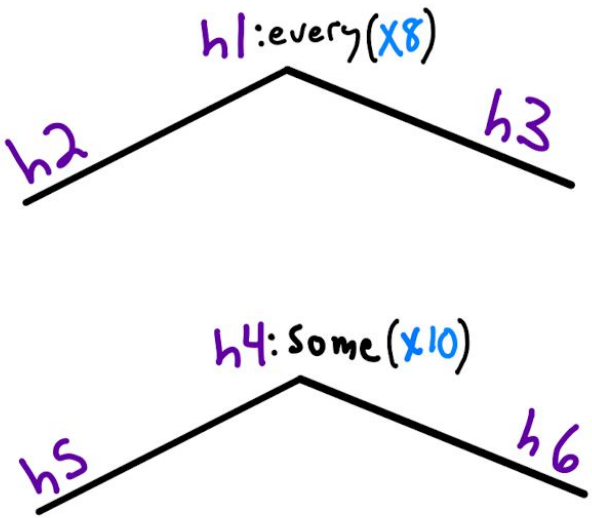
$$\begin{bmatrix} \_every\_q & \\ \text{LBL} & h0 \\ \text{ARG0} & x1 \\ \text{RSTR} & h2 \\ \text{BODY} & h3 \end{bmatrix}$$

Composing the possible scope trees

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
    - If a variable does not occur as the BV of any quantifier, or occurs as the BV but also occurs outside the RESTR or the BODY of that quantifier then it is unbound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every dog chases some cat.



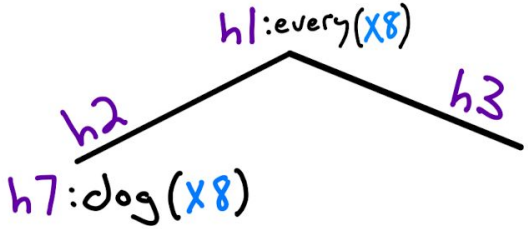
h7:dog(x8)

h9:cat(x10)

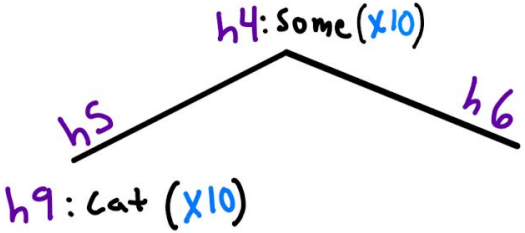
h11:chases(x8, x10)

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every dog chases some cat.

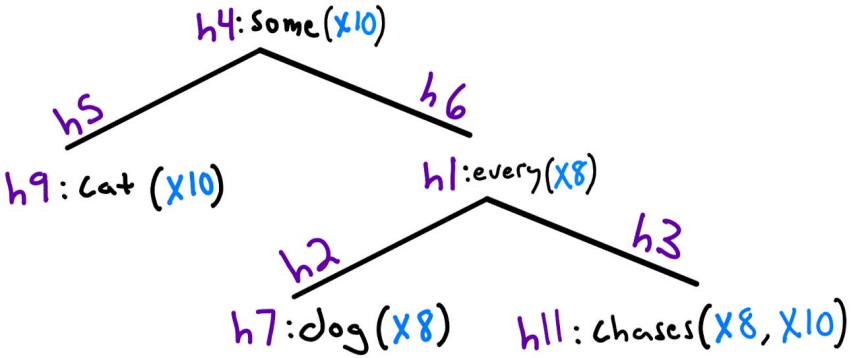


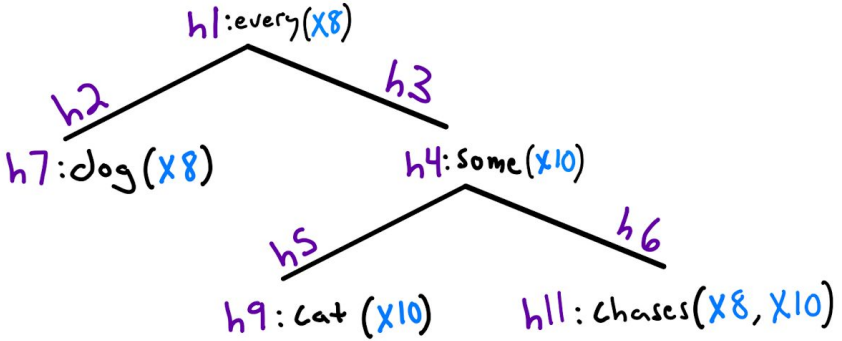
h11: chases(x8, x10)



1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

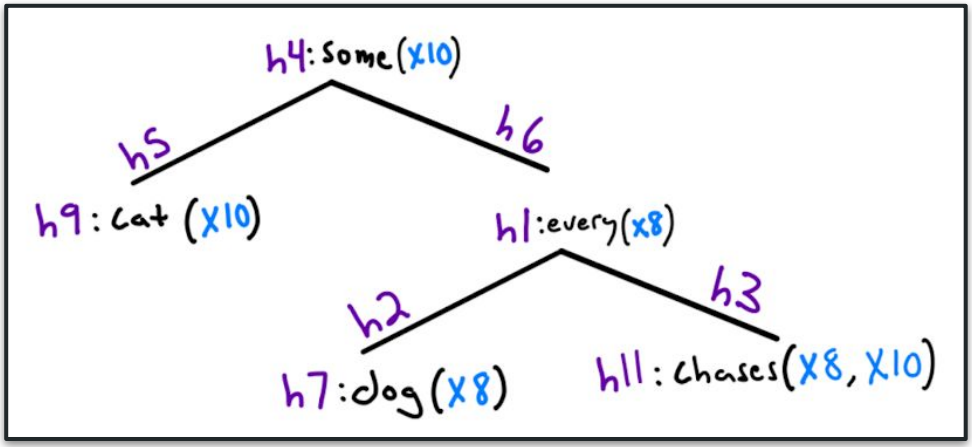
Every dog chases some cat.



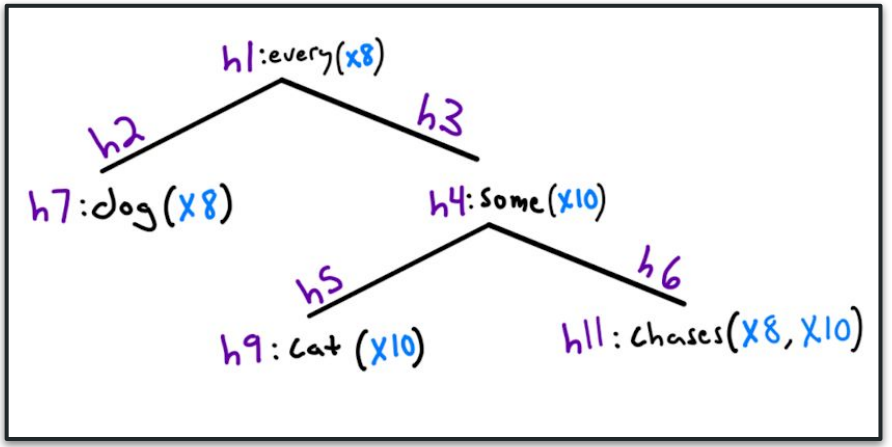


Every dog chases some cat. (Copestake et al., 2001)

Every dog chases one *particular* cat



Every dog chases a cat, not necessarily the same cat

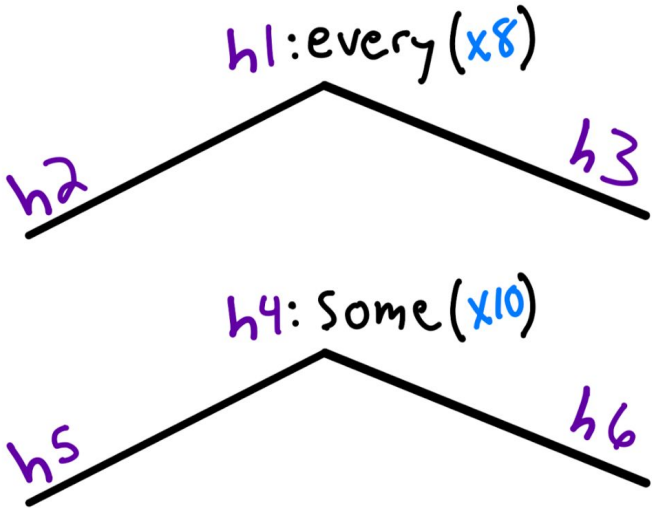


Every nephew of some famous politician runs.

- Every nephew of *one particular* famous politician runs.
- Everyone who is a nephew of a famous politician (not necessarily the same one) runs.

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every nephew of some famous politician runs.



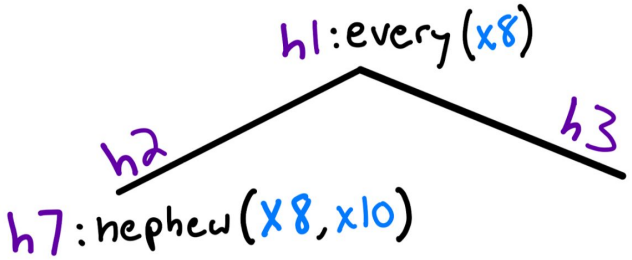
h7 : nephew (x8, x10)

h9 : famous (x10)
h9 : politician (x10)

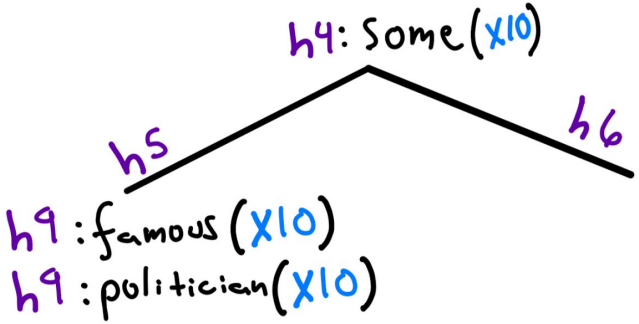
h11 : runs (x8)

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every nephew of some famous politician runs.

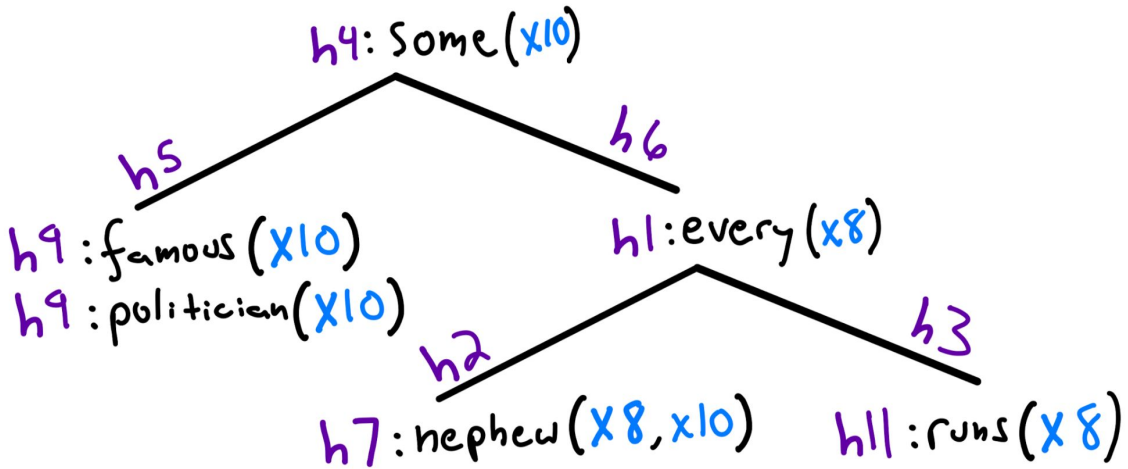


h11 : runs (x8)



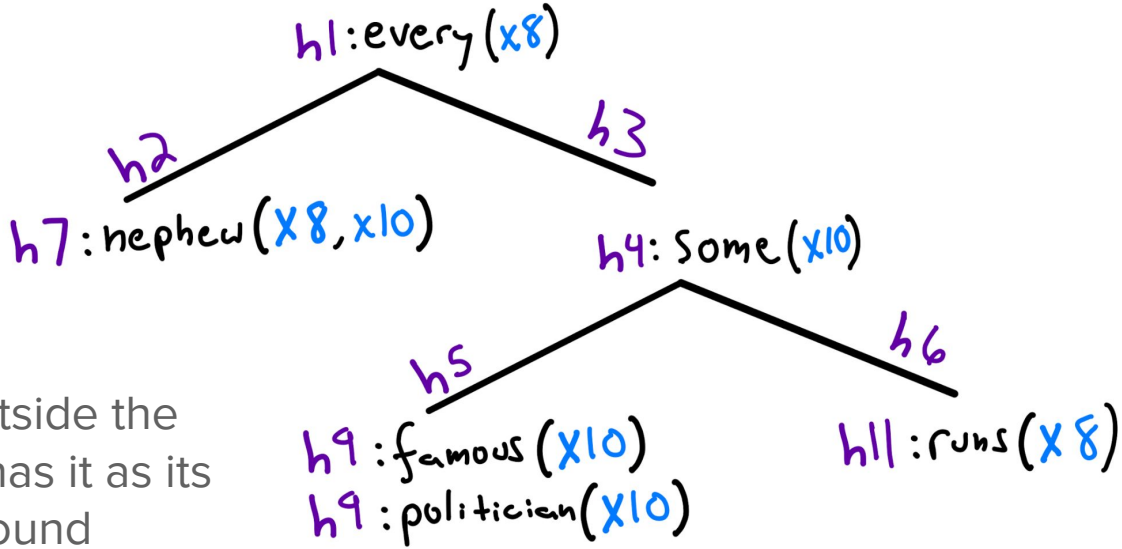
1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every nephew of some famous politician runs.



1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. **All variables must be bound**
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified

Every nephew of some famous politician runs.



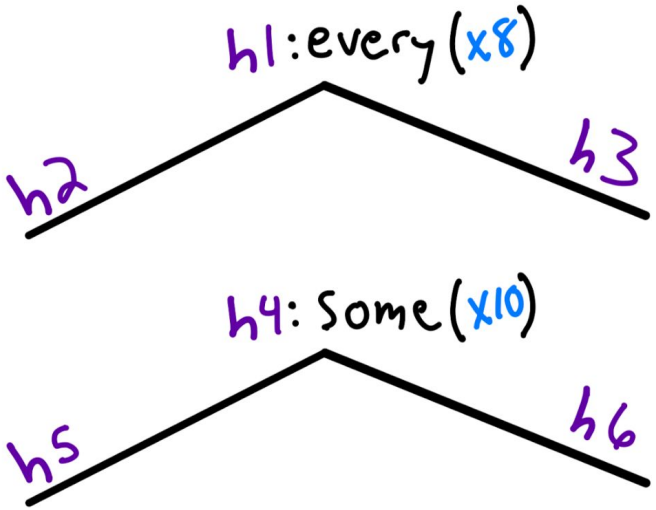
x10 appears outside the quantifier that has it as its BV, so it is unbound

What can we do?

- With our current constraints, we can only get one of the scope trees that we want.
- Let’s get rid of one of our constraints:
  1. The BV of a quantifier is the INDEX of the MRS it quantifies
  2. All variables must be bound
  3. No two quantifiers may share BVs
  4. ~~The RESTR of a quantifier is equal to the handle of the predicate being quantified~~

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. ~~The RESTR of a quantifier is equal to the handle of the predicate being quantified~~

Every nephew of some famous politician runs.



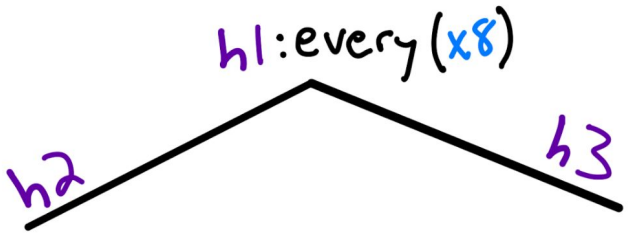
h7 : nephew (x8, x10)

h9 : famous (x10)
h9 : politician (x10)

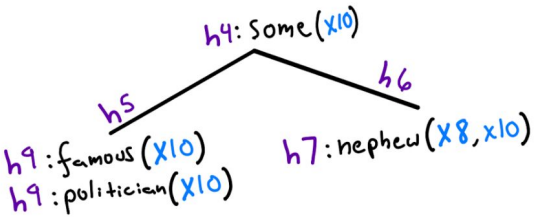
h11 : runs (x8)

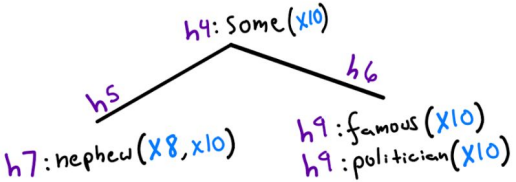
1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. ~~The RESTR of a quantifier is equal to the handle of the predicate being quantified~~

Every nephew of some famous politician runs.



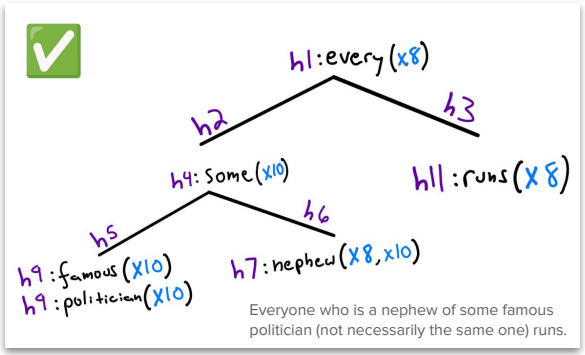
h11 : runs (x8)



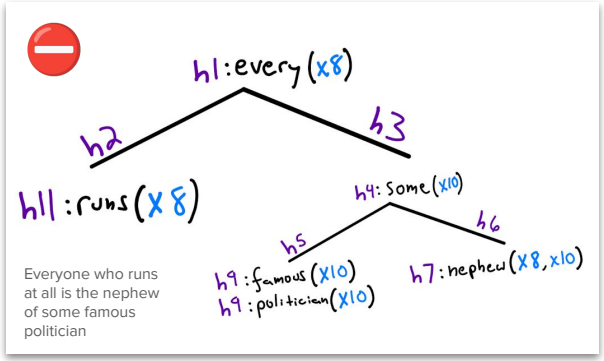


1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. ~~The RESTR of a quantifier is equal to the handle of the predicate being quantified~~

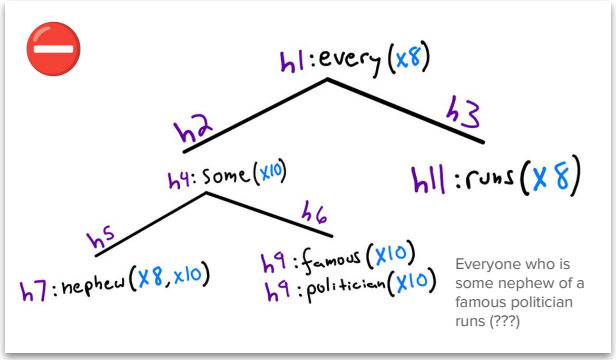
Every nephew of some famous politician runs.



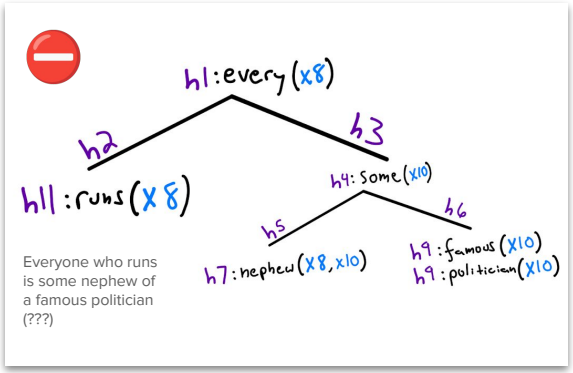
Everyone who is a nephew of some famous politician (not necessarily the same one) runs.



Everyone who runs at all is the nephew of some famous politician



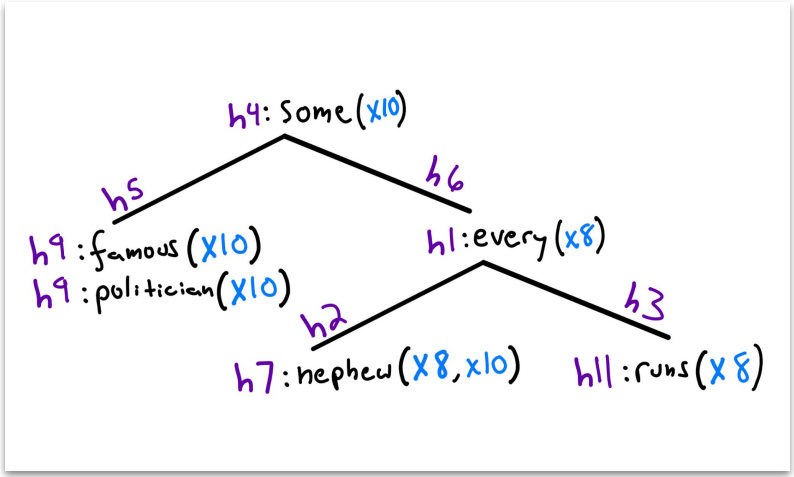
Everyone who is some nephew of a famous politician runs (???)



Everyone who runs is some nephew of a famous politician (???)

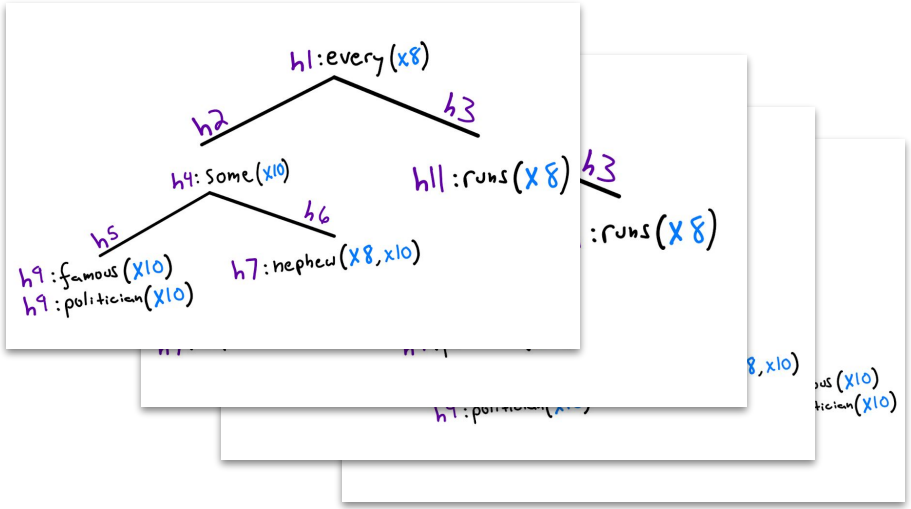
Too strict...

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is equal to the handle of the predicate being quantified



Too loose...

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. ~~The RESTR of a quantifier is equal to the handle of the predicate being quantified~~

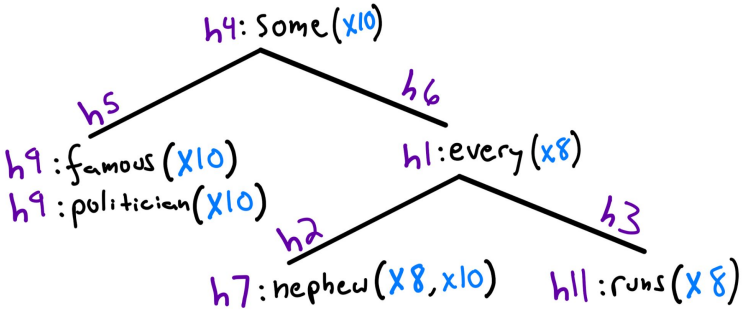


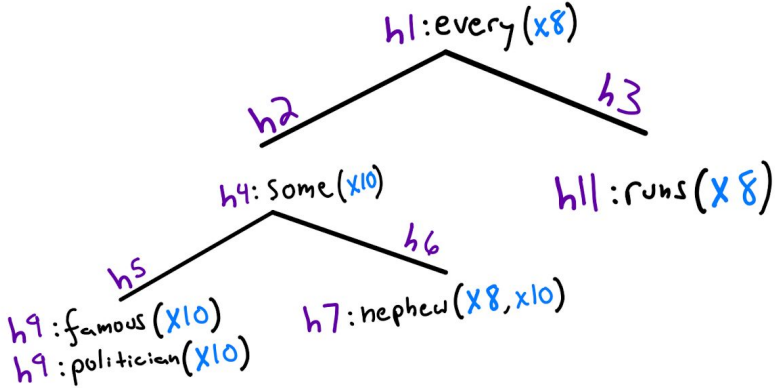
Equality Modulo Quantifiers (QEQ) (Copestake et al., 2001)

- In order to enforce this constraint, a special relationship is posited between scopal functors and their arguments
- Instead of plugging the hole directly, a QEQ (Equality Modulo Quantifiers) constraint is imposed
- If a handle is QEQ to another handle, it means they are equal with the exception that there *may* be another quantifier in between the two handles, hence “modulo quantifiers”

1. The BV of a quantifier is the INDEX of the MRS it quantifies
2. All variables must be bound
3. No two quantifiers may share BVs
4. The RESTR of a quantifier is **QEQ** to the handle of the predicate being quantified
    - i.e. it sits directly on the RESTR branch or is somewhere down inside, with only another quantifier floating in between

Every nephew of some famous politician runs.





Different kinds of scopal lexical items

- Quantifiers
  - RESTR =q LTOP of argument that it specifies
  - *a, the, some, every, many, most …*
- Scopal adverbs
  - *probably*
  - ARG1 =q whatever it modifies
- Scopal verbs
  - *think, know, believe …*
  - ARG2 =q its complement

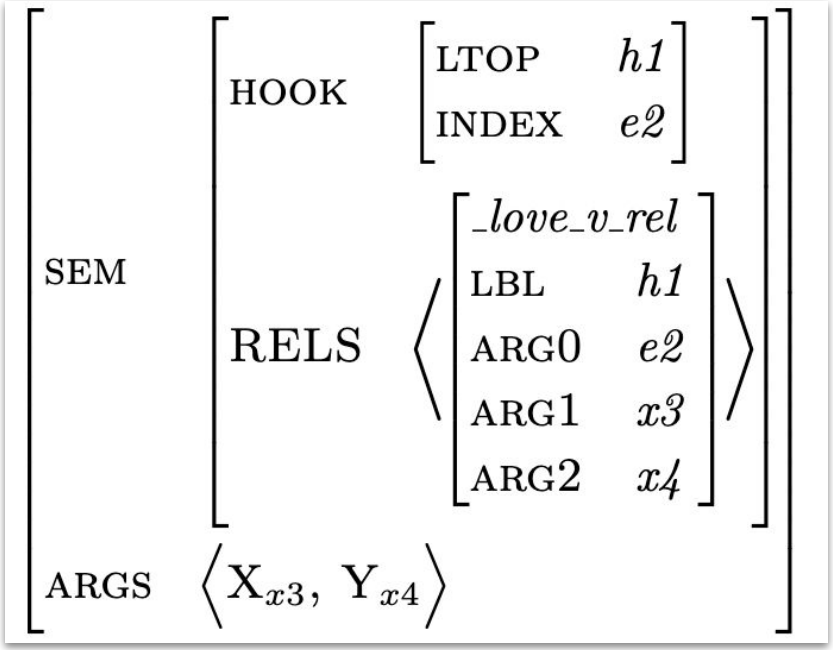
back to the big picture!

Defining Lexical Items

- We need to set the elements of HOOK (i.e. the LTOP and INDEX) for each lexical item appropriately so composition occurs properly
- We also must set the ordinary argument variables to be identified with the INDEX of the appropriate elements of the argument list
- … and set the scopal argument variables to be QEQ to the LTOP of the appropriate elements of the argument list

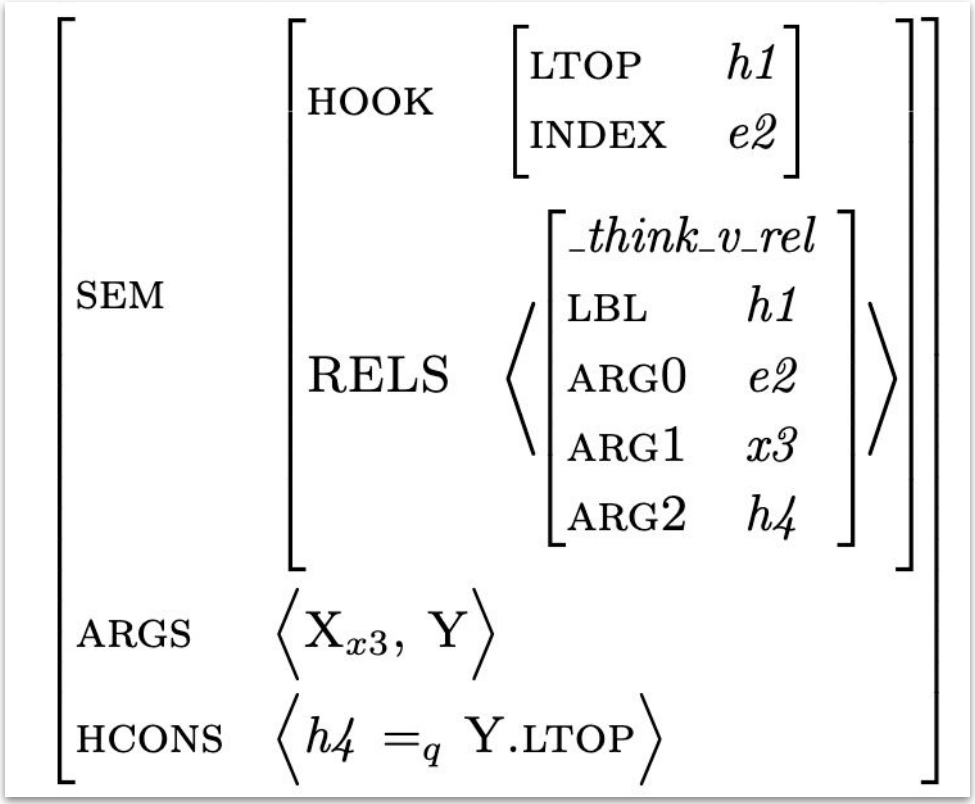
Defining Lexical Items

- Non-scopal
  - INDEX = ARG0 of key EP
  - LTOP = LBL of key EP
  - ARG variables identified appropriately



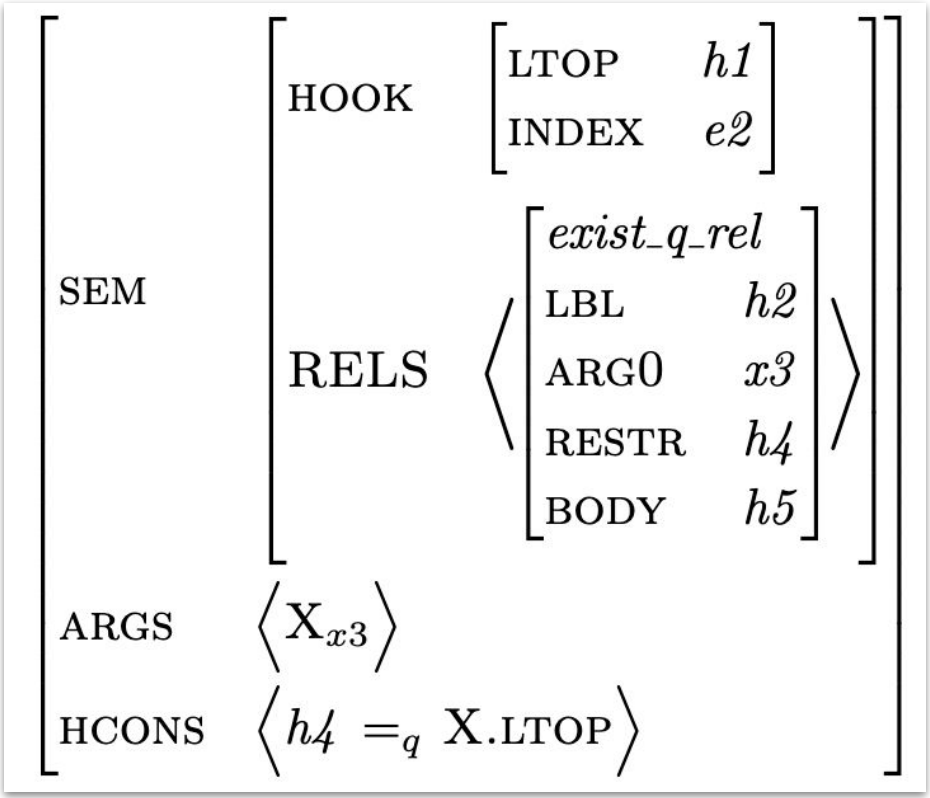
Defining Lexical Items

- Non-floating scopal
  - INDEX = ARG0 of key EP
  - LTOP = LBL of key EP
  - Ordinary ARG variables identified appropriately
  - Scopal ARG variables QEQ accordingly



Defining Lexical Items

- Floating scopal
  - INDEX = ARG0 of key EP
  - **LTOP != LBL of key EP**
  - RESTR QEQ sole element in list of arguments
  - BODY unidentified



Returning to composition

- Now we can define how the composition operations work
- Non-scopal composition
- Scopal composition

Semantic Algebra

- Copestake et al., 2001 defines an algebra where the operands are SEMENTs (Semantic Elements) and operators use these SEMENTs to compose larger SEMENTs
- A SEMENT is defined as a 5-tuple with the following components:
  1. A “hook” consisting of a LBL and an INDEX
  2. A set of holes to be plugged during composition
  3. A bag of Elementary Predicates (EPs)
  4. A set of equalities between variables
  5. A bag of HCONs conditions
- This is almost the same as an MRS except…
  1. No GTOP
  2. A set of holes
  3. A set of equalities

Composition Operations

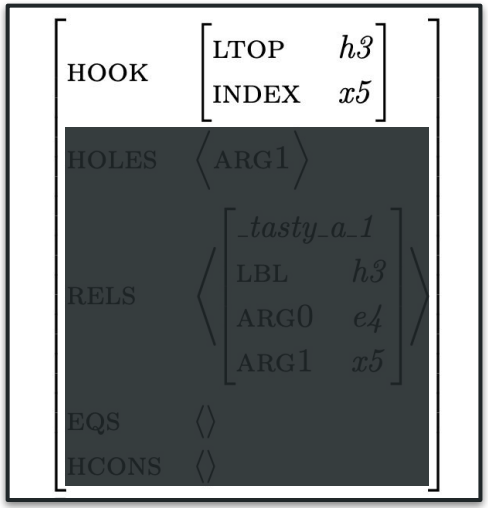
- Assume the following names for the elements of the SEMENT tuple
  - hook = the hook consisting of the LBL and INDEX
  - holes = set of holes to be plugged
  - rels = bag of EPs
  - eqs = list of equalities between variables
  - hcons = bag of handle constraints
- **Assume the following names for the 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.holes refers to the holes list on the functor SEMENT

Non-scopal Composition

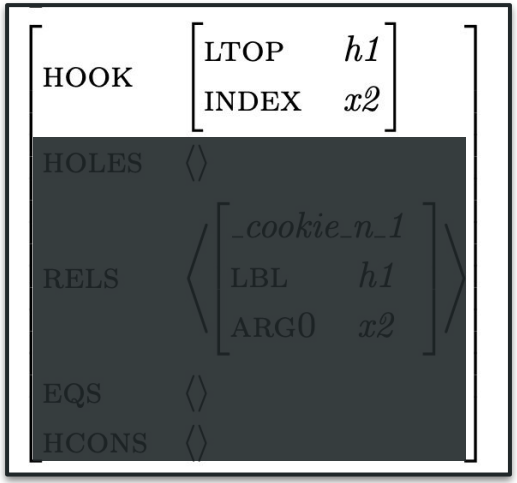
Where the hole labeled x is being plugged, composition occurs as follows:

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

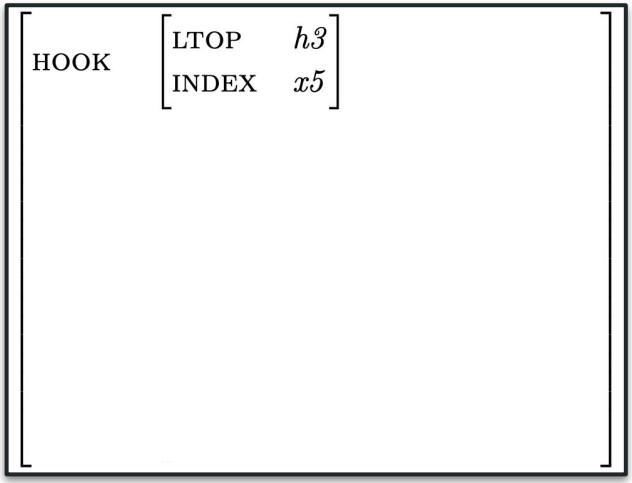
1. RES.hook = FUNC.hook



FUNC

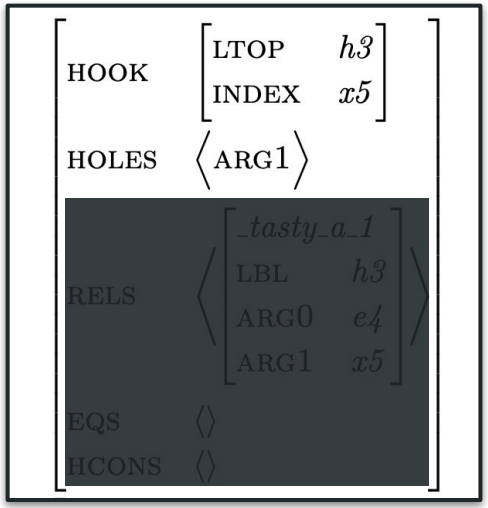


ARG

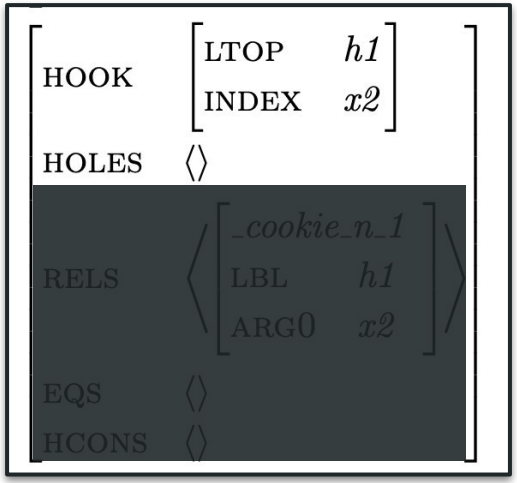


RES

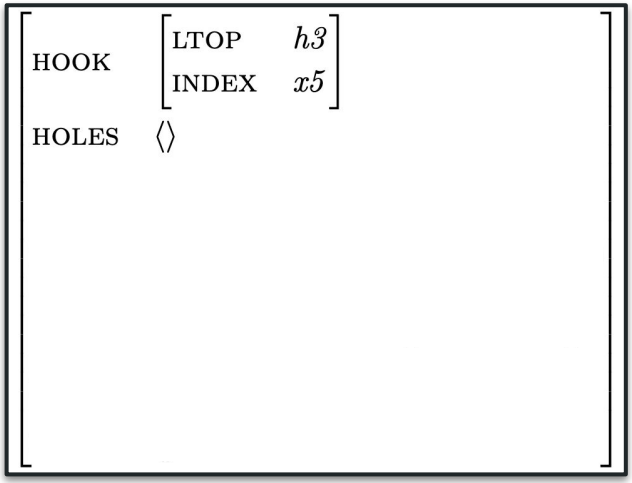
2. RES.holes = (FUNC.holes - FUNC.holes.x) ⊕ ARG.holes



FUNC

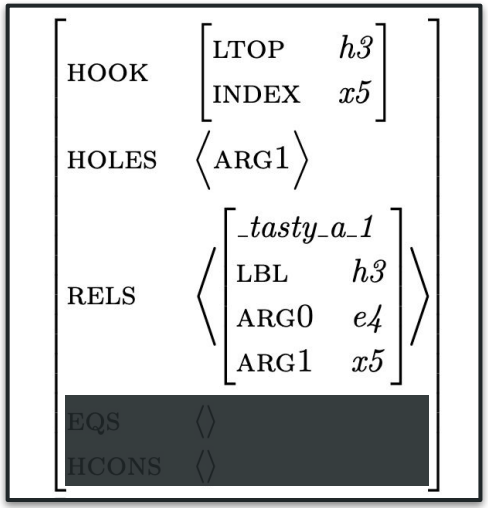


ARG

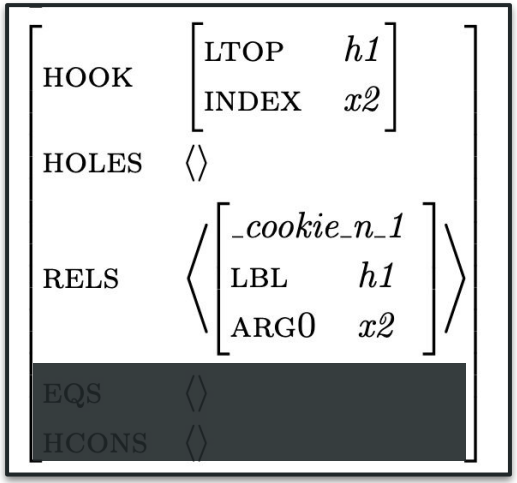


RES

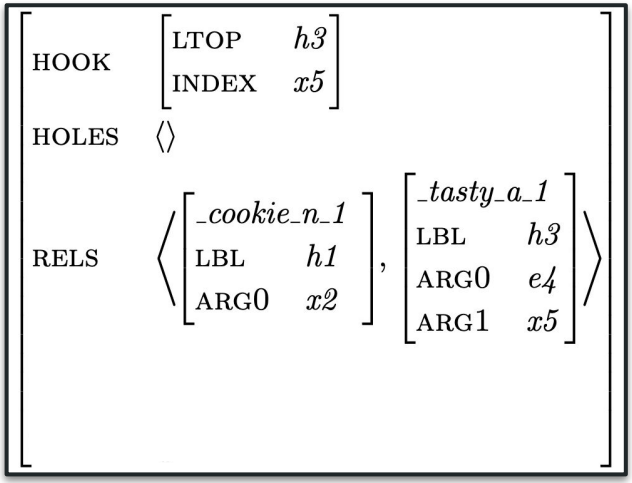
3. RES.rels = FUNC.rels ⊕ ARG.rels



FUNC

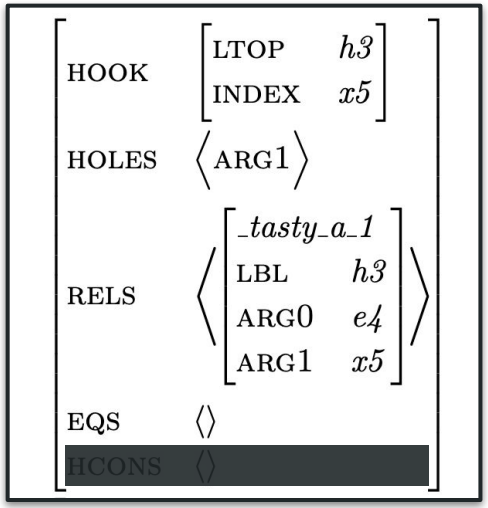


ARG

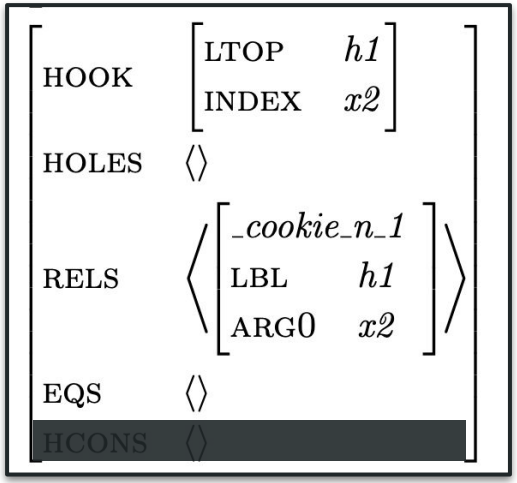


RES

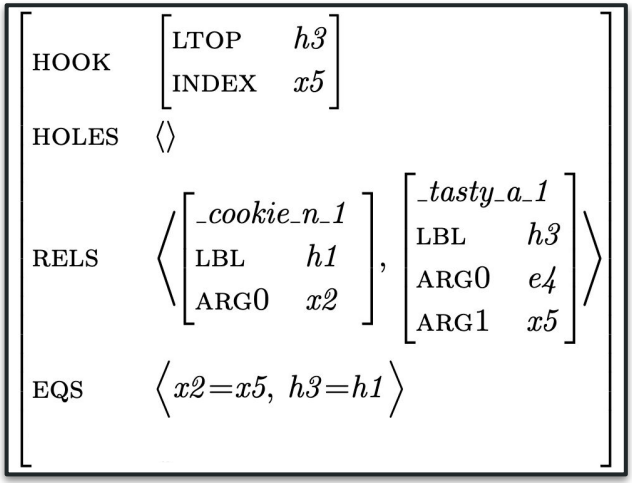
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs ∪ {FUNC.holes.x = ARG.hook.index} ∪ {FUNC.hook.lbl = ARG.hook.lbl})



FUNC

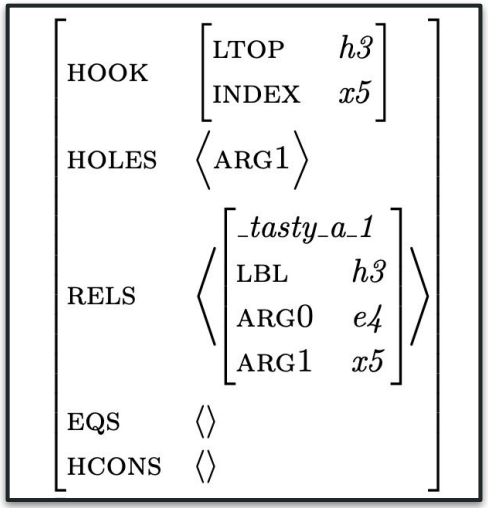


ARG

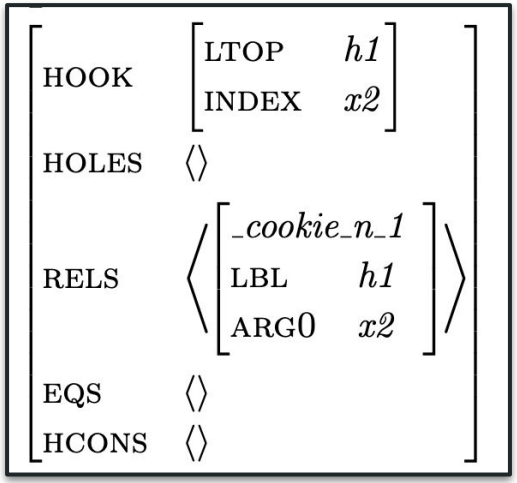


RES

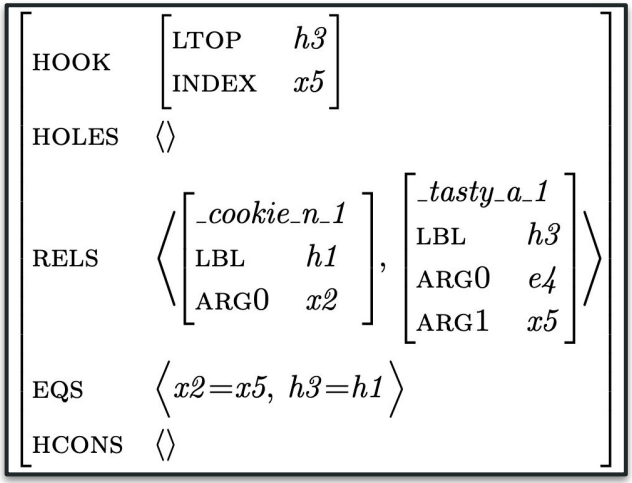
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons



FUNC



ARG



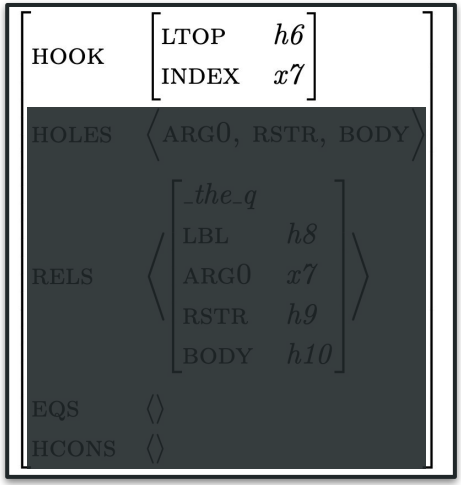
RES

Scopal Composition for Quantifiers

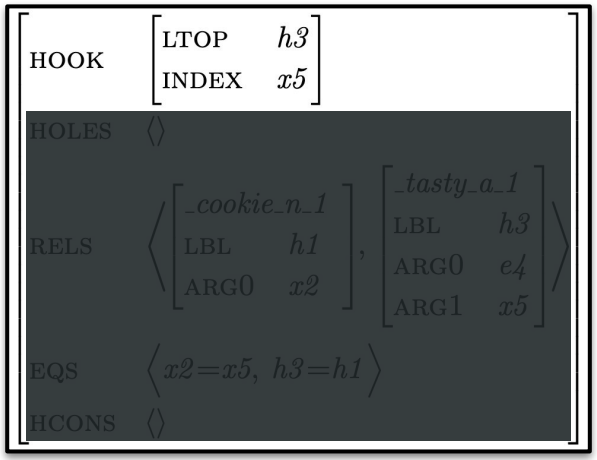
Where the hole labeled x is being plugged, composition occurs as follows:

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

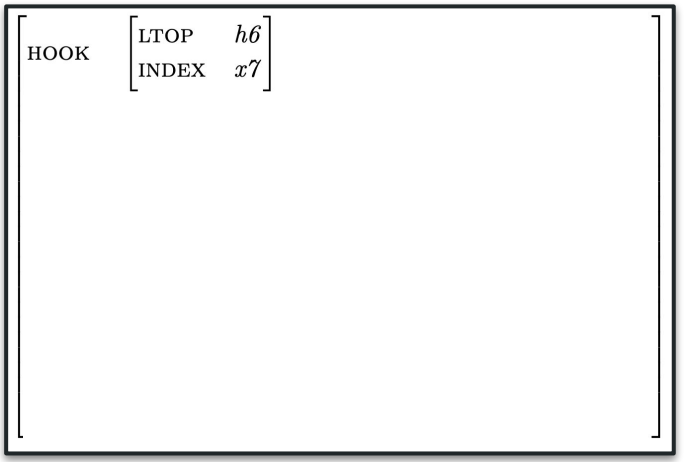
1. RES.hook = FUNC.hook



FUNC

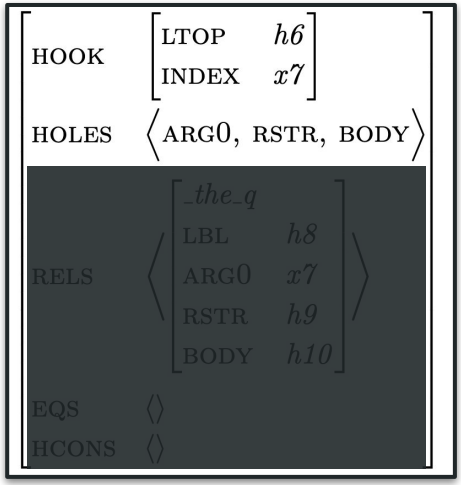


ARG

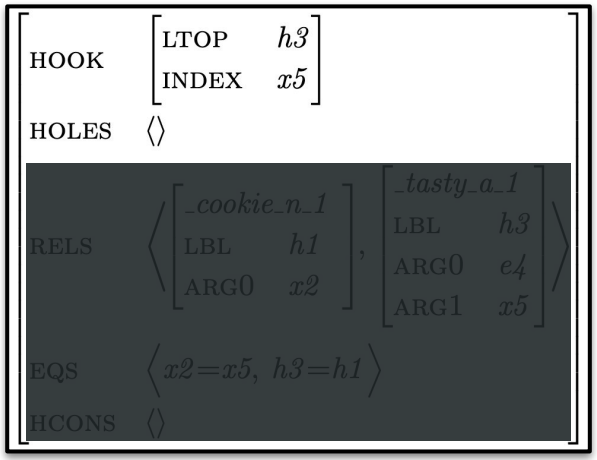


RES

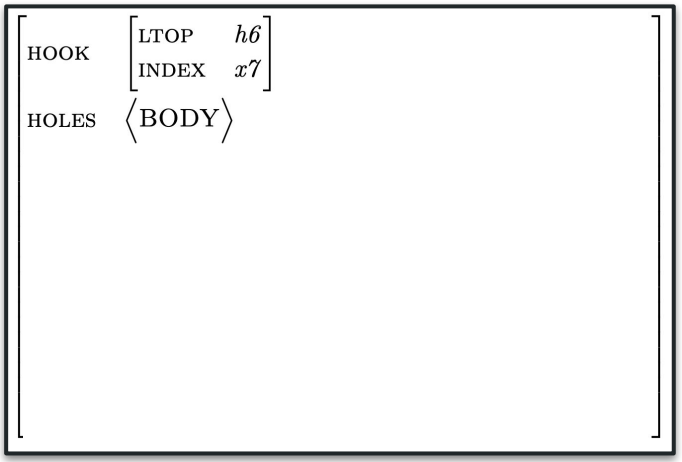
2. RES.holes = (FUNC.holes - FUNC.holes.x - FUNC.holes.RSTR) ⊕ ARG.holes



FUNC

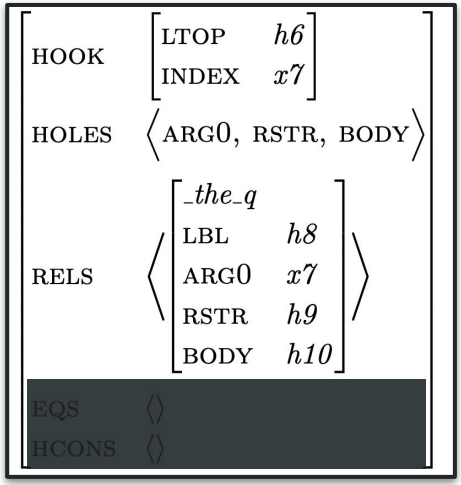


ARG

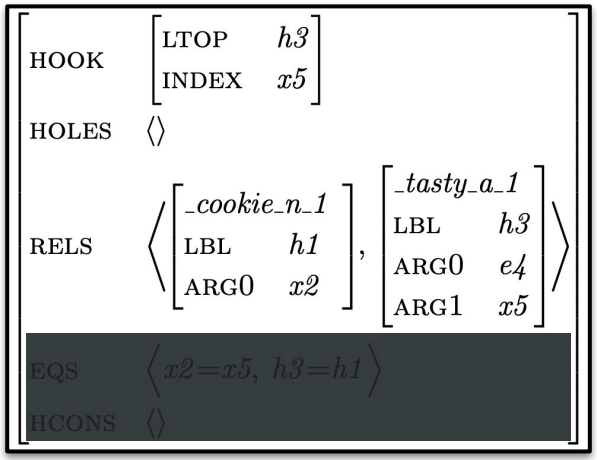


RES

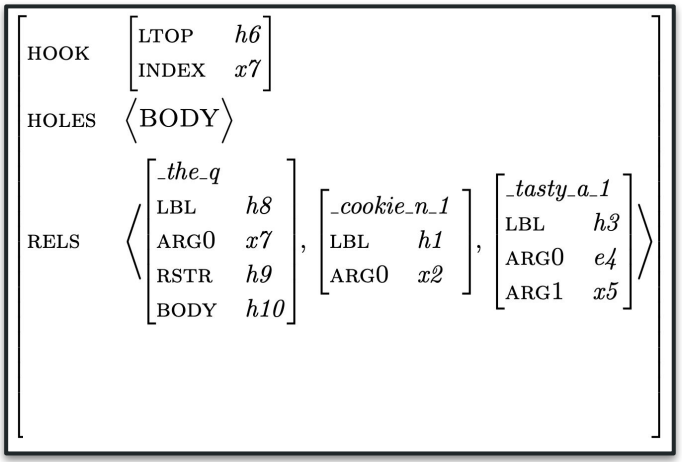
3. RES.rels = FUNC.rels ⊕ ARG.rels



FUNC

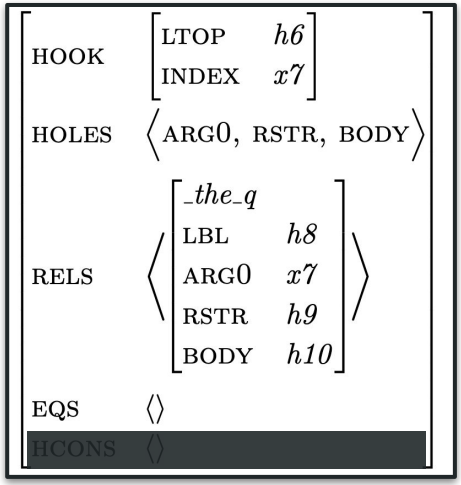


ARG

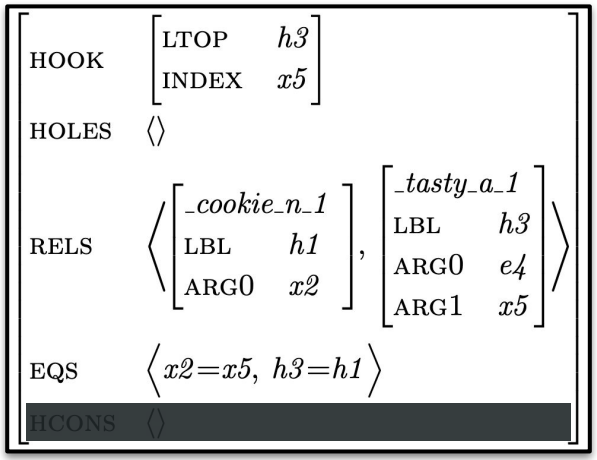


RES

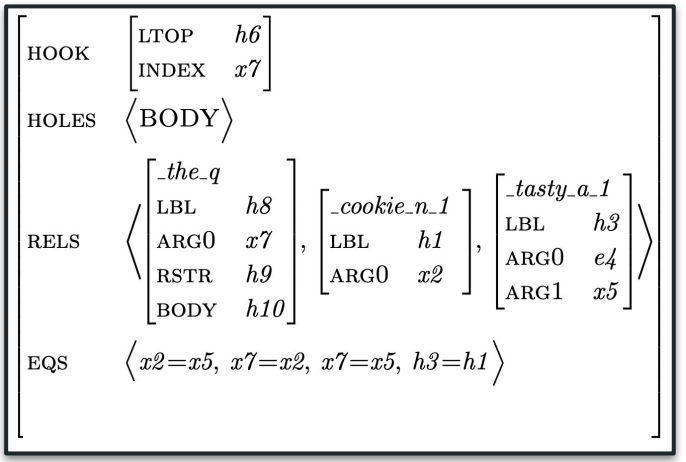
4. RES.eqs = Tr(FUNC.eqs ∪ ARG.eqs ∪ {FUNC.holes.x = ARG.hook.index})



FUNC

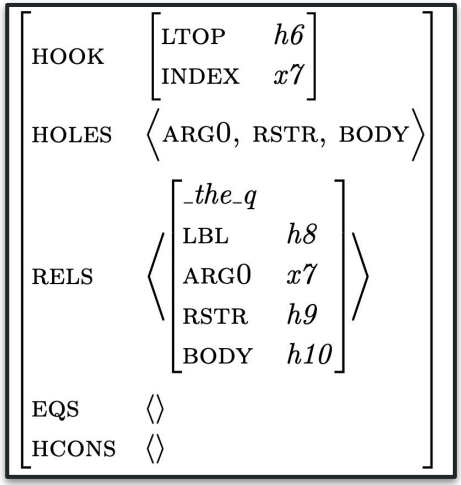


ARG

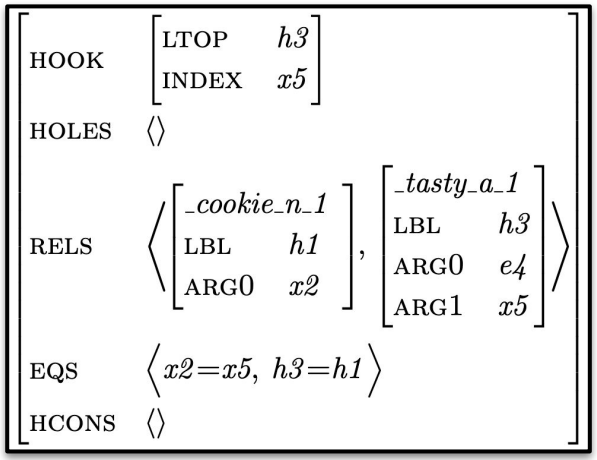


RES

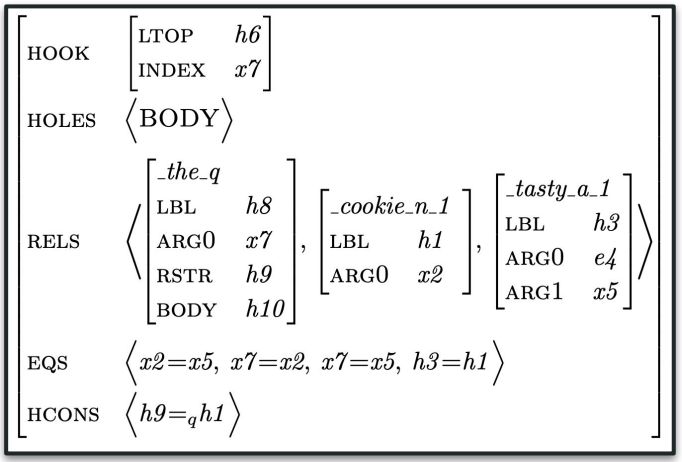
5. RES.hcons = FUNC.hcons ⊕ ARG.hcons ⊕ [FUNC.holes.RSTR =q ARG.hook.lbl]



FUNC

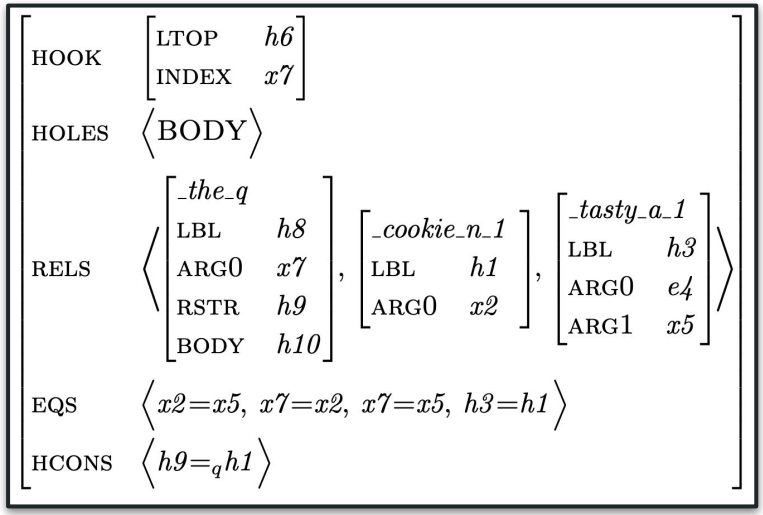


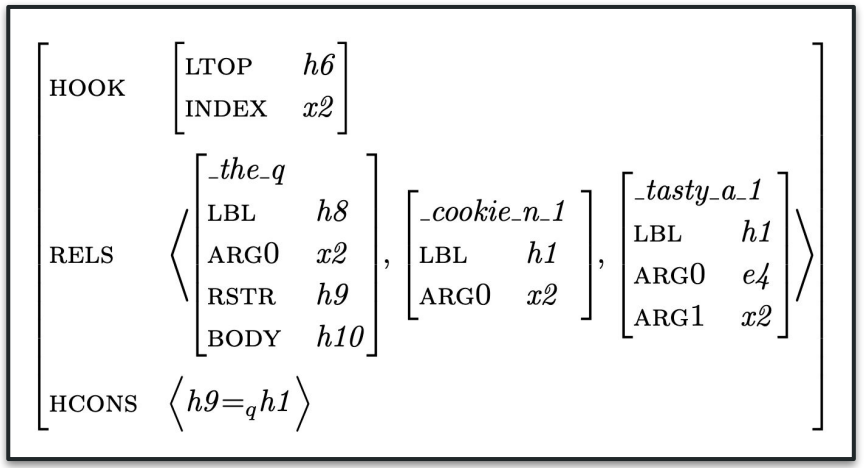
ARG



RES

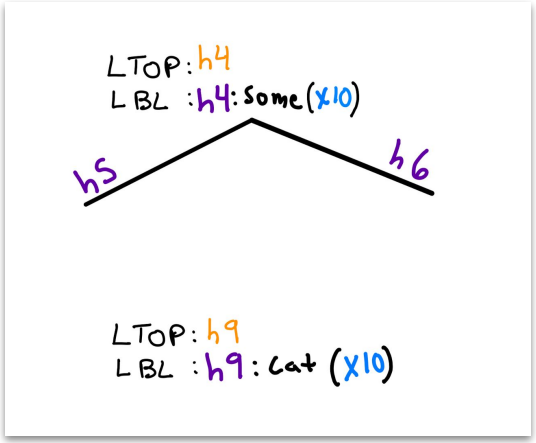
Dropping holes & Collapsing equalities

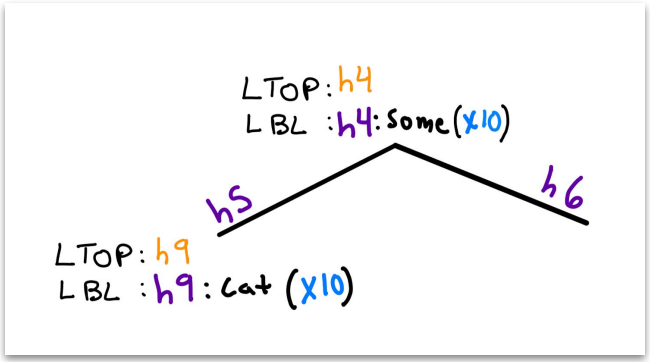


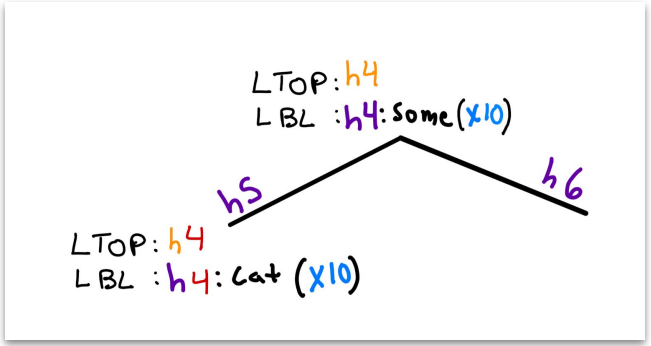


Why can’t the LTOP of a quantifier be identified with its LBL?

- Because our composition operations insist that the LTOP of the result of composition is equal to the LTOP of the semantic functor, we will end up with a tangled scope tree

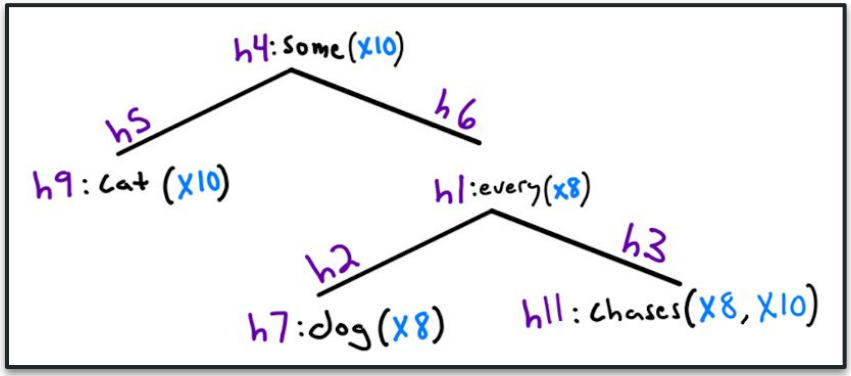


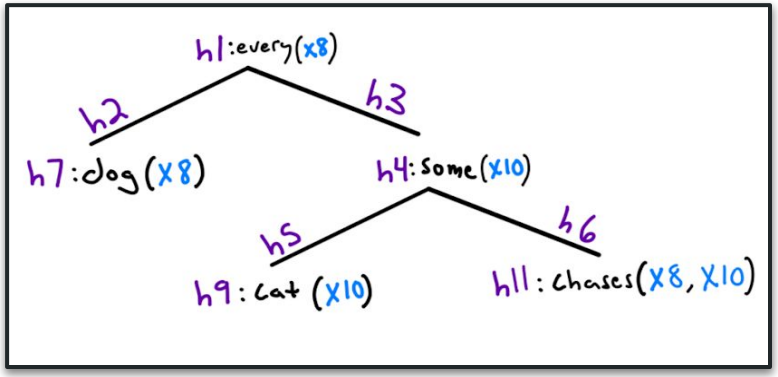




What's up with GTOP...?

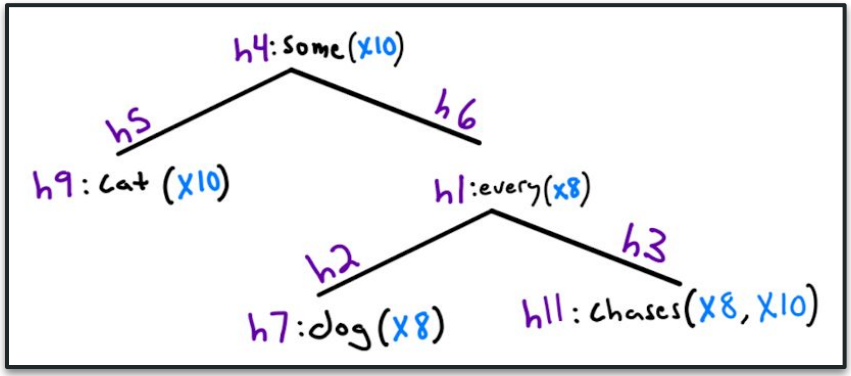
- Setting the GTOP correctly is part of the requirements for a well-formed MRS
- **Root condition**: the root condition stipulates that the GTOP is QEQ to the LTOP of the root phrase
- GTOP =q h11
    - So GTOP is equal to the LTOP of "Every dog chases some cat" (which ultimately originates from "chases") with the exception of the quantifiers floating in between

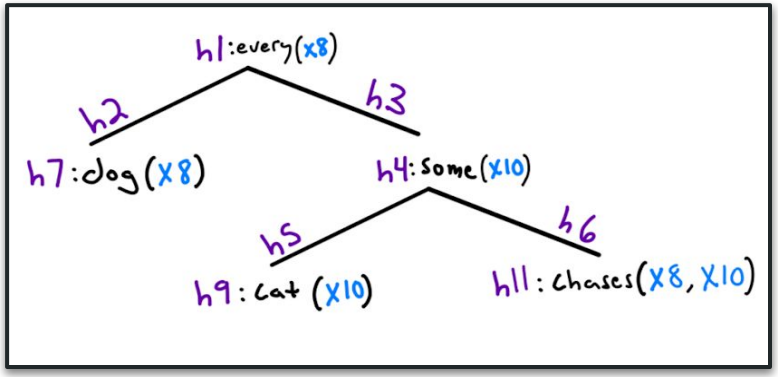




AnnC Jun 2024

We originally wanted GTOP as a location that could be the equivalence of the “top box” in DRT. I’m not sure whether or not there are potentially constraints that ought to be associated with GTOP given the various restrictions we’ve imposed on ourselves as to what gets done in the MRS, but at least in principle I think we should allow for richer semantics. In any case, having GTOP in the formalism is still a good idea even if there are no such constraints, because it allows us to use the single formal device of a qeq - otherwise we’d need a separate type of scoping relationship to express the idea that the quantifiers were floating “above” other things. All GTOP is doing is providing a way of talking about the root node in the tree so it doesn’t formally add significant complexity, it seems to me.





Summary

- MRS is the semantic framework that is used in the Grammar Matrix
- Each MRS object is a tuple, <GTOP, HOOK, RELS, HCONS>
- We have different types of variables to account for the different types of equalities we want to be able to set
  - Variables of type *e* and *x* are set to be directly equal to each other
  - Variables of type *h* (handles) are in QEQ relationships to one another to allow for underspecification of scope resolution
- We have two different kinds of semantic composition
  - Non-scopal
  - Scopal

Your turn Emily :)