MRS
Information Structure

Ling 567
January 28, 2014

Overview

- MRS
  - Goals, design principles
  - Flat semantics
  - Underspecified quantifier scope
  - Linguistic questions
  - MRS in feature structures
- Information Structure
  - What it means
  - How we represent it

MRS Preface

- Most of today's lecture covers stuff that is already implemented in the Matrix.
- The goal of this presentation is to increase your understanding of what's already there, and how to have your code interact with it.
- In the near term, you'll need to be able to look at the semantic representations and understand them.
- In later labs, you'll also be working on compositionality.

MRS: Goals

- The design of the MRS formalism answers the following four general goals:
    - Adequate representation of NL semantics
    - Grammatical compatibility
    - Computational tractability
    - Underspecifiability

MRS: Design Principles

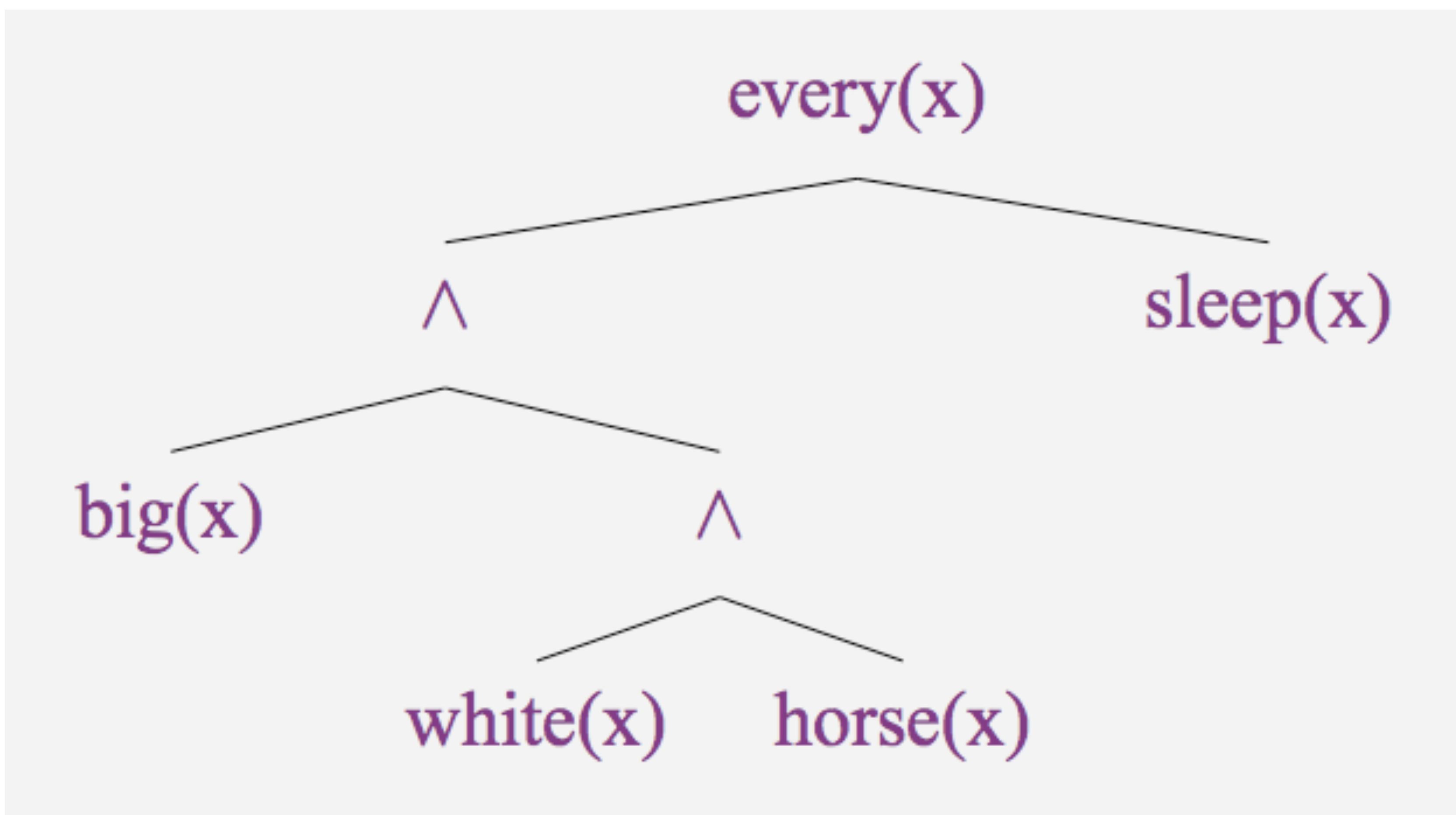
- The design of the representations of particular linguistic phenomena follow the following general strategies/design principles
  - Represent all semantic distinctions which are syntactically or morphologically marked
  - Underspecify semantic distinctions which aren't: These can be spelled-out/ambiguated if necessary in post-processing
  - 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

A quick reminder about quantifier scope

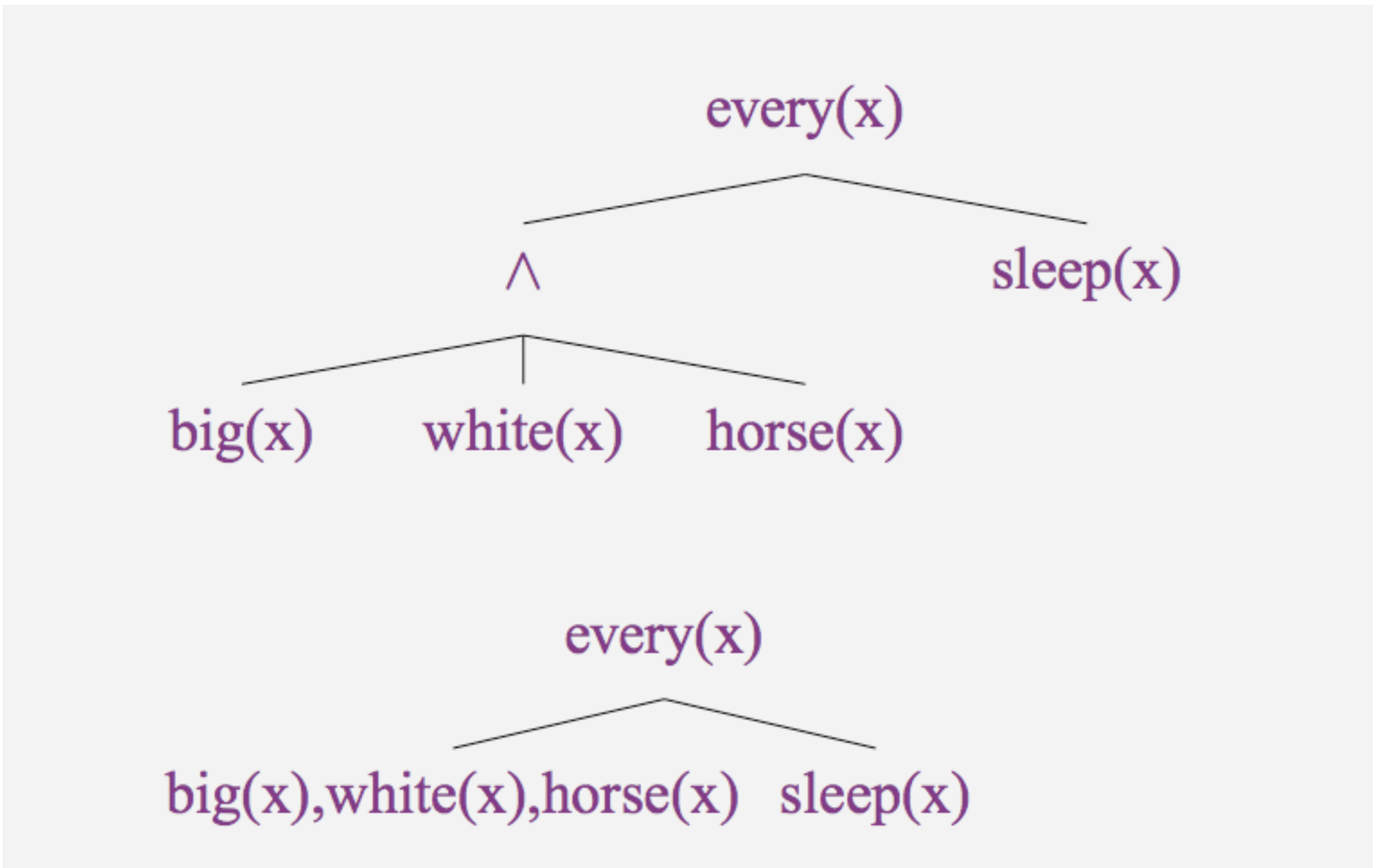
- Quantifiers (predicate logic or NL) take three arguments:
  - A variable to bind
  - A restriction
  - A body
- Every dog sleeps: $\forall x \; dog(x) sleep(x)$
- When one quantifier appears within the restriction or body of another, we say the second has wider scope

Working towards MRS (1/4)

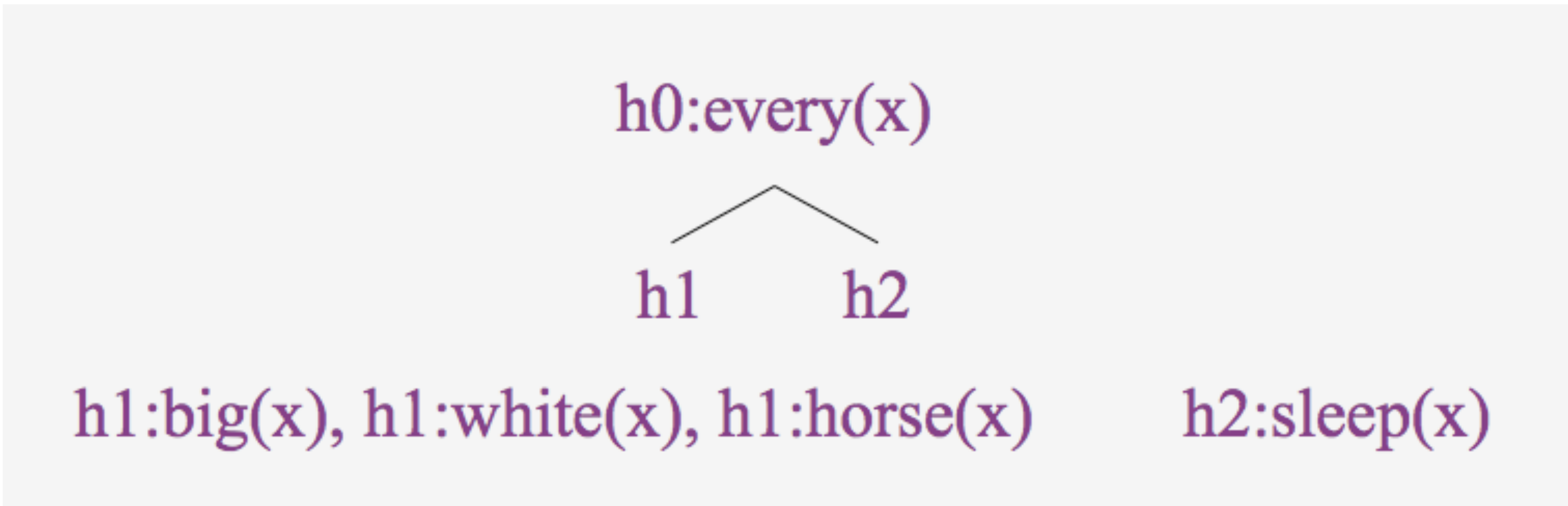
- Every big white horse sleeps
- $\text{every}(x, \wedge\text{big}(x), \wedge(\text{white}(x), \text{horse}(x))), \text{sleep}(x))$



Working towards MRS (2/4)



Working towards MRS (3/4)



- And finally:

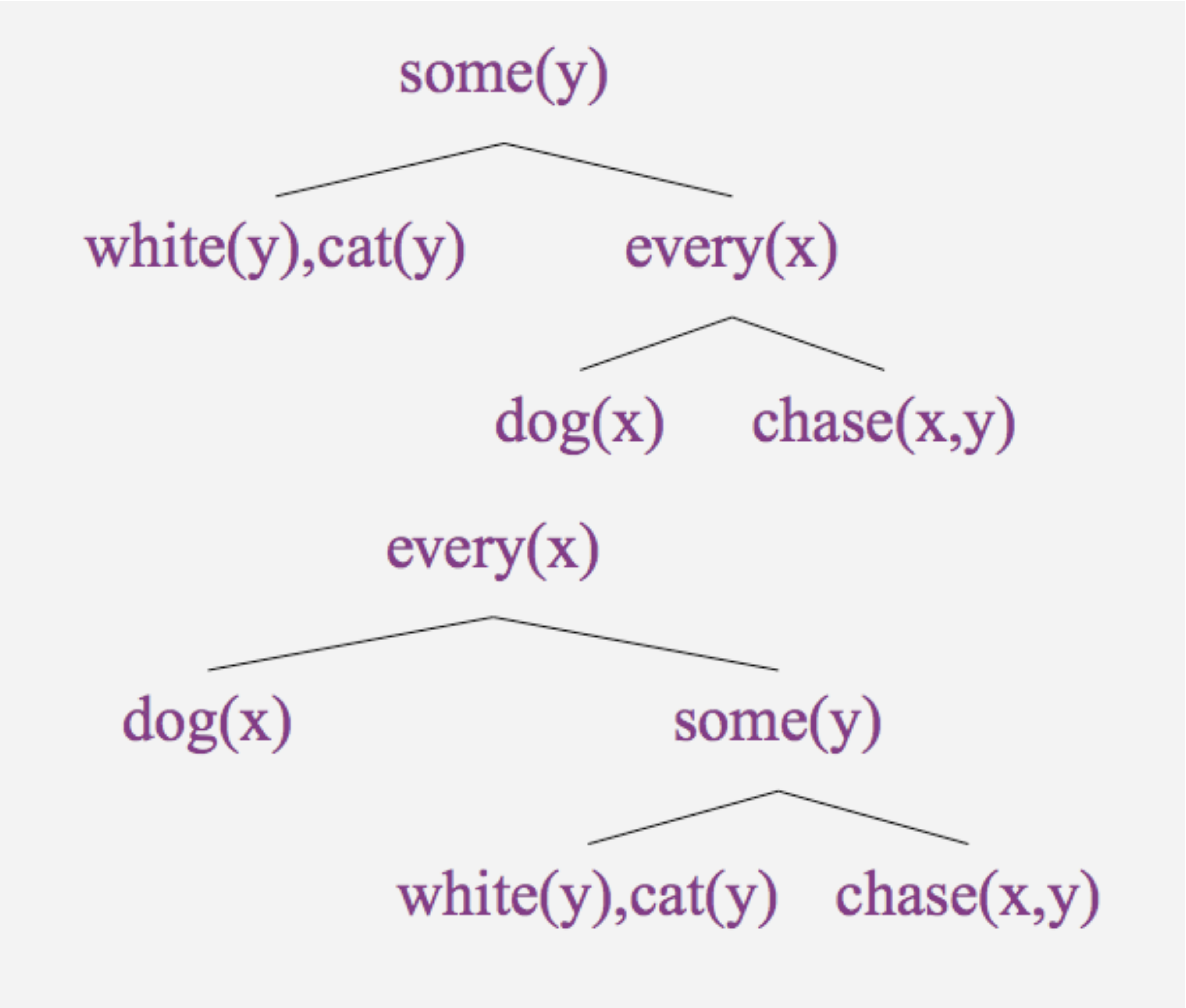
h0:every(*x, h1, h2*), *h1*:big(*x*), *h1*:white(*x*), *h1*:horse(*x*), *h2*:sleep(*x*)

Working towards MRS (4/4)

- This is a flat representation, which is a good start.
- Next we need to underspecify quantifier scope, and it’s easier to see why with multiple quantifiers.
- At the same time, we want to be able to partially specify it, since this is required for adequate representations of NL semantics.

Underspecified quantifier scope (1/2)

- Every dog chases some white cat.



Underspecified quantifier scope (2/2)

- *h1*:every(*x*,*h3*,*h4*), *h3*:dog(*x*), *h7*:white(*y*), *h7*:cat(*y*), *h5*:some(*y*,*h7*,*h1*), *h4*:chase(*x*,*y*)

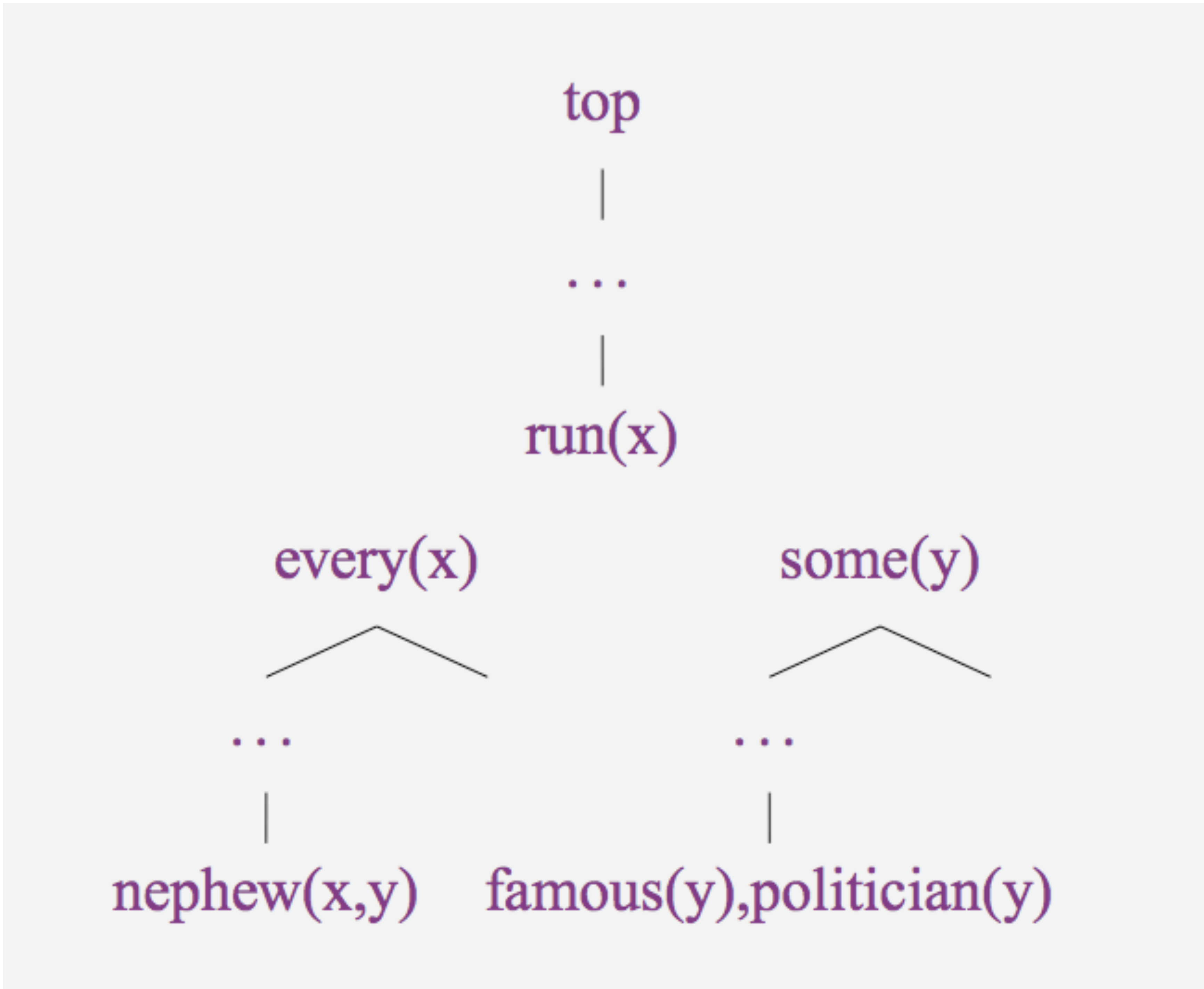
- *h1*:every(*x*,*h3*,*h5*), *h3*:dog(*x*), *h7*:white(*y*), *h7*:cat(*y*), *h5*:some(*y*,*h7*,*h4*), *h4*:chase(*x*,*y*)

- *h1*:every(*x*,*h3*,*hA*), *h3*:dog(*x*), *h7*:white(*y*), *h7*:cat(*y*), *h5*:some(*y*,*h7*,*hB*), *h4*:chase(*x*,*y*)

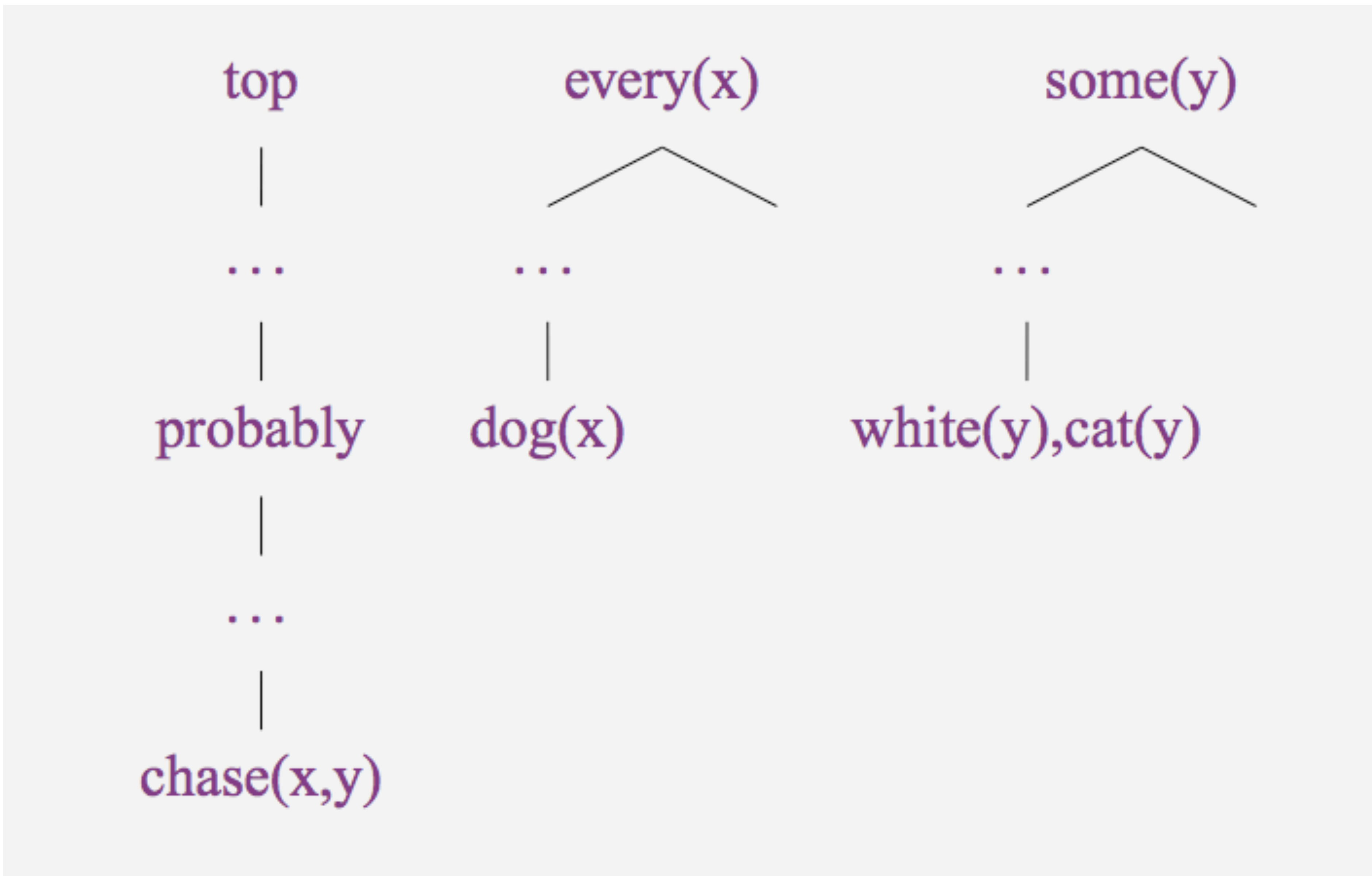
Partially constrained quantifier scope (1/4)

- For the BODY of quantifiers, we have no particular constraints to add.
- In turns out that the RESTRICTION needs to have partially underconstrained scope:
  - Every nephew of some famous politician runs.
    - every(x,some(y,famous(y) $\wedge$ politician(y), nephew(x,y)) run(x))
    - some(y,famous(y) $\wedge$ politician(y), every(x, nephew(x,y),run(x)))
  - But not:
    - every(x,run(x),some(y,famous(y) $\wedge$ politician(y), nephew(x,y)))
    - ‘Everyone who runs is a nephew of a famous politician.’

Partially constrained quantifier scope (2/4)



Partially constrained quantifier scope (3/4)



Partially constrained quantifier scope (4/4)

$$\langle h0, \{h2 : \text{every}(x, h3, h4), h5 : \text{nephew}(x, y),$$
$$h6 : \text{some}(y, h7, h8), h9 : \text{politician}(y), h9 : \text{famous}(y),$$
$$h10 : \text{run}(x)\},$$
$$\{h0 =_q h10, h7 =_q h9, h3 =_q h5\}\rangle$$

$$\langle h0, \{h1 : \text{every}(x, h2, h3), h4 : \text{dog}(x),$$
$$h5 : \text{probably}(h6), h7 : \text{chase}(x, y),$$
$$h8 : \text{some}(y, h9, h10), h11 : \text{white}(y), h11 : \text{cat}(y)\},$$
$$\{h0 =_q h5, h2 =_q h4, h6 =_q h7, h9 =_q h11\}\rangle$$

We’ve arrived at MRS!

- Flat structure
- Underspecification & partial specification of quantifier scope are possible

Linguistic Questions

- How do we build MRS representations compositionally?
- Is it linguistically adequate to insist that no process suppress relations?
- Under what circumstances do NLs (partially) constrain scope?
- Is it linguistically adequate to give scopal elements (esp. quantifiers, but also scopal modifiers) center-stage?

MRS in feature structures

- RELS: List (diff-list) of relations
- HCONS: List (diff-list) of handle constraints
- HOOK: Collection of features ‘published’ for further compisition: INDEX, LTOP, XARG
- ARGn: Roles within relations

Anatomy of an MRS

- An MRS consists of:
  - A top handle
  - A list of relations, each labeled by a handle A list of handle constraints
  - An (underspecified) MRS is well-formed iff the constraints can be resolved to form one or more trees (singly-rooted, connected, directed acyclic graphs).

Anatomy of a relation

- A relation has:
  - A predicate (string or type)
  - A label (handle)
  - One or more arguments: ARG0-n (ARG0 canonically being the event or individual introduced by the relation)
- The value of each ARGn is either:
  - An index, canonically identified with the ARG0 of
  - another relation
  - A handle: identified with the label of another relation, the HARG of a handle constraint, or not identified with anything

Anatomy of a handle constraint

- Current sole handle constraint type: qeq
- ‘Equal modulo quantifiers’
- Features: HARG, LARG
- → Unless some quantifier scopes in between, the value of this ARGn is the same as the label of that relation.
- When the label of a relation is the value of an ARGn, this corresponds to a branch in an MRS tree.
- When the value of an ARGn is qeq the label of a relation, this corresponds to a ‘dotted’ branch – i.e., a dominance relation.

When else are handles identified?

- Relations with the same handle value share the same scope.
- Typically, we see this with intersective modifiers (adverbs, adjectives, PPs) which share their handles with their modifies.

Composition: Overview

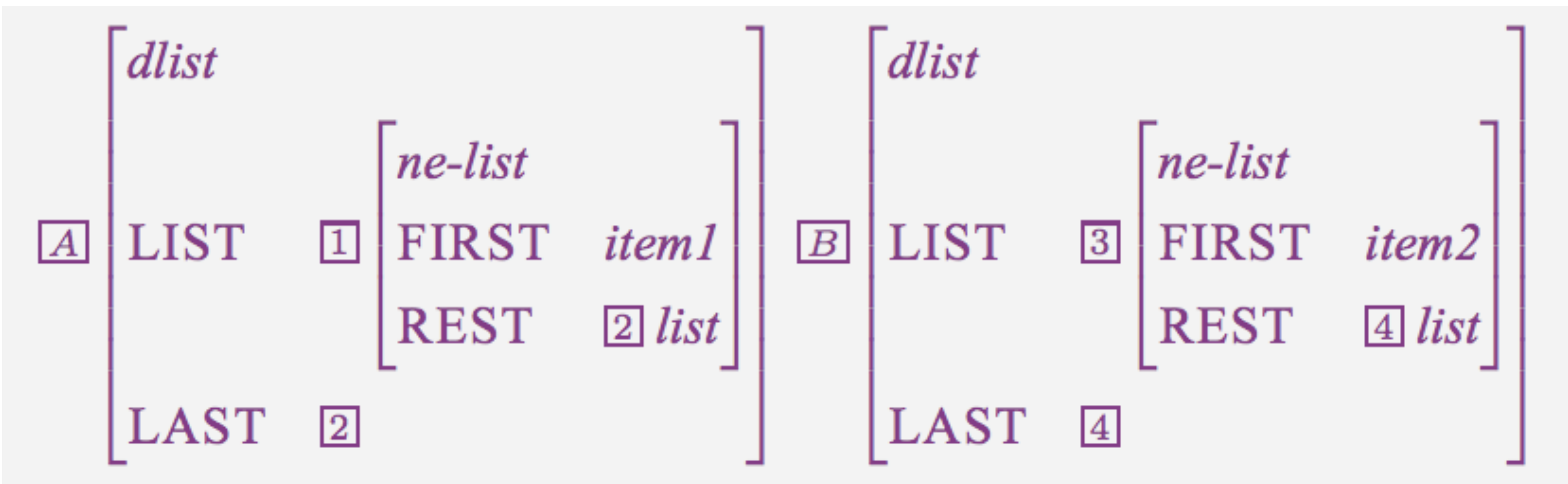
- RELS and HCONS on mother nodes
- HOOK, LKEYS
- ARGn <> indices
- ARGn <> handles
- LBL <> LBL
- Building *qeq*s

RELS and HCONS on mother nodes

- The RELS and HCONS value of the mother is the append of the values from the daughter(s) and the C-CONT of the mother.
- C-CONT is the ‘constructional content’: allows phrase structure rules to introduce relations.
- Examples?
- From a semantic point of view, the C-CONT is just another daughter.

Appending lists with unification

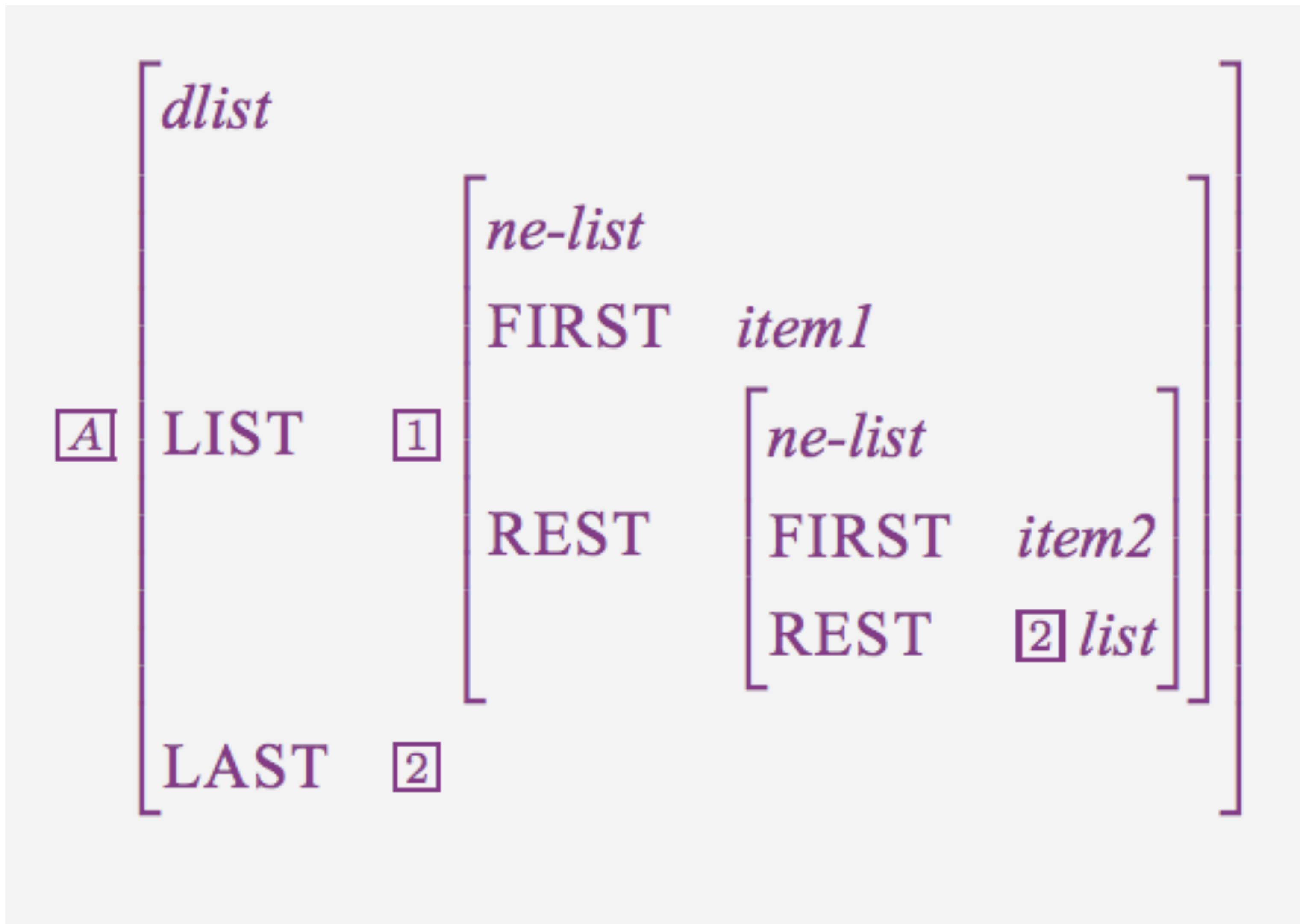
- A diff-list embeds an open-ended list into a container structure providing a ‘pointer’ to the end of the ordinary list.



- To append : (i) unify the front of [B] (i.e. the value of its LIST feature) into the tail of [A] (its LAST value) and

- (ii) use the tail of difference list [B] as the new tail for the result of the concatenation.

Result of appending lists



Matrix type: dl-append

- NB: **Not** for direct use in the grammar; this type is just meant as reference

```
dl-append := avm & [APPARG1 [LIST #first,
                             LAST #between],
                    APPARG2 [LIST #between,
                             LAST #last],
                    RESULT  [LIST #first,
                             LAST #last]].
```

Diff-lists: practicalities

- Typically errors with diff-lists involve circularity and not direct unification failure.
- If the LKB complains about circular feature structures, check your difference lists.
- Don’t try to constrain the length of a difference list.
- Unifying structures which include diff lists in an append relation can result in diff lists constrained to be empty.

Returning to our regularly scheduled programming...

- Why do we need diff-lists?
- Why do we need append?

Semantic compositionality in action

```
basic-unary-phrase := phrase &
  [ SYNSEM.LOCAL.CONT [ RELS [ LIST #first,
                                LAST #last ]],
    C-CONT [ RELS [ LIST #mid,
                     LAST #last ]],
    ARGS < sign & [ SYNSEM.LOCAL
                [ CONT [ RELS [ LIST #first,
                                 LAST #mid ]]]]>].
```

Now what?

- Phrase structure rules (and lexical rules) gather up RELS and HCONS from daughters.
- Phrase structure rules also (optionally) introduce further RELS and HCONS.
- How do we link the ARGn positions of the relations to the right things?
- How do we link the HARG/LARG of qeqs to the right things?

HOOK

- The CONT.HOOK is the information that a given sign exposes for further composition.
- By hypothesis, this includes only:
    - INDEX (the individual or event denoted by the sign, linked to some ARG0)
    - LBL (the local top handle of the sign)
    - XARG (the external argument of the sign)
- The HOOK of a sign is identified its with the C-CONT.HOOK.
- The C-CONT.HOOK in turn is identified with the semantic head daughter, if there is one.
- Otherwise, the LBL, INDEX, and XARG inside C-CONT.HOOK need to be constrained appropriately.

LKEYS

- The feature LKEYS houses pointers to important relations on the RELS list, most notably LKEYS.KEYREL.
- Only appropriate for lexical items.
- Serves as a uniform place to state linking constraints.
- Linking constraints: equality between HOOK.INDEX or HOOK.LBL of arguments/modifiees and LKEYS.KEYREL.ARGn.

ARGn <> indices

```
intransitive-lex-item := basic-one-arg-no-hcons &
  [ ARG-ST < [ LOCAL.CONT.HOOK.INDEX ref-ind &
                                     #ind ] >,
    SYNSEM.LKEYS.KEYREL.ARG1 #ind ].

intersective-mod-lex := no-hcons-lex-item &
  [ SYNSEM [ LOCAL.CAT.HEAD.MOD
                     < [ ..INDEX #ind ]] >,
             LKEYS.KEYREL.ARG1 #ind ] ].
```

ARGn <> handles (1/2)

```
clausal-second-arg-trans-lex-item := basic-two-arg &
[ ARG-ST < [ LOCAL.CONT.HOOK.INDEX ref-ind & #ind ],
           [ LOCAL.CONT.HOOK.LTOP #larg ] >,
  SYNSEM [ LOCAL.CONT.HCONS <! qeq &
                               [ HARG #harg,
                                 LARG #larg ] !>,
           LKEYS.KEYREL [ ARG1 #ind,
                          ARG2 #harg ] ] ].
```

ARGn <> handles (2/2)

```
basic-determiner-lex := norm-hook-lex-item &
  [ SYNSEM [ LOCAL
     [ CAT [ HEAD det,
             VAL..HOOK [ INDEX #ind,
                         LTOP #larg ]],
       CONT [ HCONS <! qeq &
                     [ HARG #harg,
                       LARG #larg ] !>,
              RELS <! relation !> ] ],
       LKEYS.KEYREL quant-relation &
                    [ ARG0 #ind,
                      RSTR #harg ] ] ].
```

LBL <> LBL

```
isect-mod-phrase :=
  head-mod-phrase-simple &
  head-compositional &
  [ HEAD-DTR.SYNSEM.LOCAL.CONT.HOOK.LTOP #hand ],
    NON-HEAD-DTR.SYNSEM.LOCAL.CONT.HOOK.LTOP #hand
```

- The rule for intersective modifiers identifies the LTOP of the two daughters, and thus the LBL of the main relation introduced by each.

- The HOOK value of the whole thing comes from the syntactic head, thanks to the type head-compositional.

Scopal modifiers (1/2)

```
scopal-mod-phrase :=
   head-mod-phrase-simple &
  [ NON-HEAD-DTR.SYNSEM.LOCAL
     [ CAT.HEAD.MOD < [ LOCAL scopal-mod ] >,
       CONT.HOOK #hook ],
    C-CONT [ HOOK #hook,
             HCONS <! !> ] ].
```

- No identification of LTOPs.
- Non-head (adjunct) daughter is the semantic head.

Scopal modifiers (2/2)

```
scopal-mod-lex := lex-item &
  [ SYNSEM [ LOCAL [
      CAT.HEAD.MOD < [ LOCAL scopal-mod &
                         [ ..LTOP #larg ]] >,
      CONT.HCONS <! qeq &
                    [ HARG #harg,
                      LARG #larg ] !> ],
      LKEYS.KEYREL.ARG1 #harg ]].
```

- Builds qeq between its ARG1 and the MOD's LTOP

Building qeqs

- Determiners
- Scopal adverbs
- Clausal complement verbs (and nouns, adjectives, adpositions...)

Summary

- Phrase structure and lexical rules:
  - ... gather up RELS and HCONS
  - ... potentially add further RELS and HCONS
  - ... unify elements on valence/ mod lists with signs
  - ... pass up and/or modify HOOK information
- Lexical entries:
  - ... orchestrate the linking between valence/mod lists and the ARGn positions in the relations they contribute
  - ... expose certain information in the HOOK

Composition: Overview

- RELS and HCONS on mother nodes
- HOOK, LKEYS
- ARGn <> indices
- ARGn <> handles
- LBL <> LBL
- Building *qeq*s

Overview

- MRS
  - Goals, design principles
  - Flat semantics
  - Underspecified quantifier scope
  - Linguistic questions
  - MRS in feature structures
- Information Structure
  - What it means
  - How we represent it

Information structure

- The way in which a speaker packages the information in an utterance for the addressee

- The distinction between allosentences: Sentences with different form but the same truth conditions

  - Kim gave the book to Sandy.

  - Kim gave the BOOK to Sandy.

  - It was the book that Kim gave to Sandy.

  - The book, Kim gave to Sandy.

(Lambrecht 1996)

Information structural components

- Focus: That which is new and/or important in an utterance
- Topic: That which an utterance is about
- Contrast: Subtype of focus or topic which entails an alternative set
- NB: Information structure interacts with but is distinct from information status

(Song 2014)

Information structure cross-linguistically

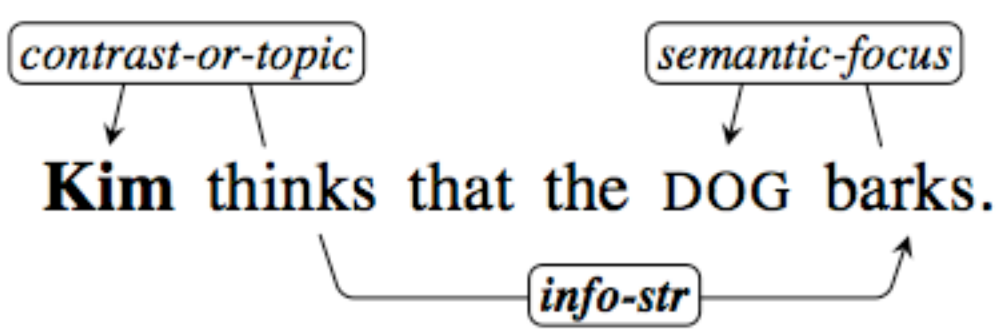
- Different languages make available different strategies for marking information structure, including:
  - Special positions within the clause
  - Special morphology
  - Lexical markers (“particles”)
  - Prosody
  - Special syntactic constructions

(Song 2014)

Representing Information Structure in MRS

- Add a new feature to MRS: ICONS (Individual CONstraintS)
- ICONS elements of the *info-str* type have two features: CLAUSE and TARGET.
- The subtype of the ICONS element provides (possibly partial) information about the information structural role the index in TARGET plays with respect to the clause picked out by the event variable in CLAUSE.

(2) a.



(Song and Bender 2012;
Song 2014:230)

info-str hierarchy

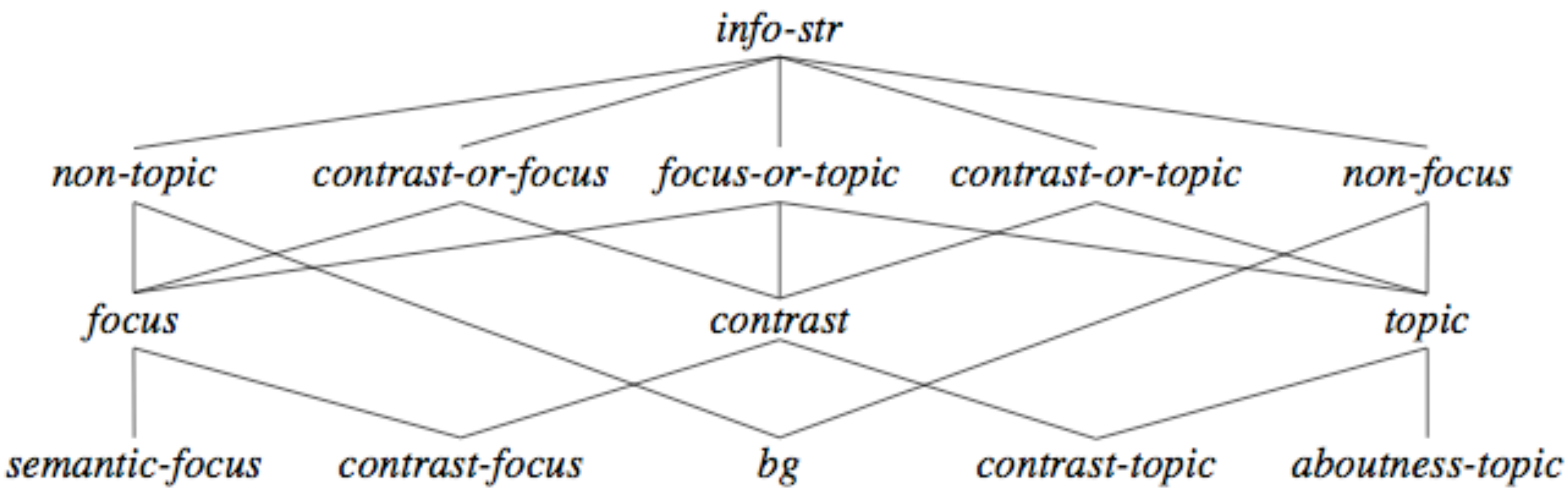


Figure 9.1: Type hierarchy of *Info-str*

(Song 2014:174)

Information structure in Lab 4

- http://courses.washington.edu/ling567/testsuites.html#info
- http://depts.washington.edu/uwcl/sanghoun/matrix.cgi

Overview

- MRS
  - Goals, design principles
  - Flat semantics
  - Underspecified quantifier scope
  - Linguistic questions
  - MRS in feature structures
- Information Structure
  - What it means
  - How we represent it