COMPUTATIONAL SEMANTICS: DAY 3

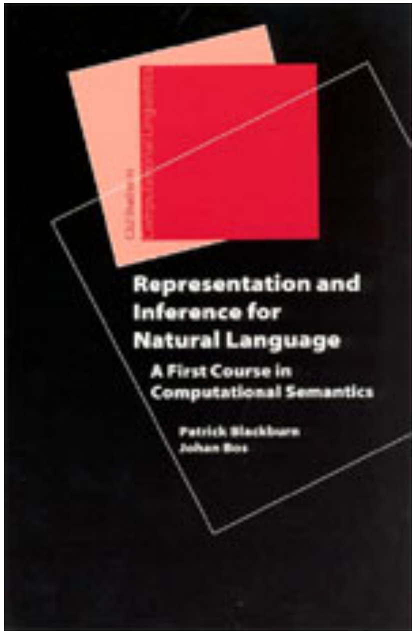
Johan Bos

University of Groningen

www.rug.nl/staff/johan.bos

Computational Semantics

- Day 1: Exploring Models
- Day 2: Meaning Representations
- **Day 3: Computing Meanings**
- Day 4: Drawing Inferences
- Day 5: Meaning Banking



Questions after yesterday’s lecture

- Quantifier scope
- The I function

Questions: Quantifier Scope

- Is there a difference between $\forall x \exists y$ LOVE(x,y) and $\exists y \forall x$ LOVE(x,y) ?

The satisfaction definition

$$
\begin{array}{lll}
M, g \models R(\tau_1, \cdots, \tau_n) & \textit{iff} & (I_F^g(\tau_1), \cdots, I_F^g(\tau_n)) \in F(R), \\
M, g \models \tau_1 = \tau_2 & \textit{iff} & I_F^g(\tau_1) = I_F^g(\tau_2), \\
M, g \models \neg\phi & \textit{iff} & \text{not } M, g \models \phi, \\
M, g \models (\phi \wedge \psi) & \textit{iff} & M, g \models \phi \text{ and } M, g \models \psi, \\
M, g \models (\phi \vee \psi) & \textit{iff} & M, g \models \phi \text{ or } M, g \models \psi, \\
M, g \models (\phi \rightarrow \psi) & \textit{iff} & \text{not } M, g \models \phi \text{ or } M, g \models \psi, \\
M, g \models \exists \mathrm{x}\phi & \textit{iff} & M, g' \models \phi, \text{ for some x-variant } g' \text{ of } g, \\
M, g \models \forall \mathrm{x}\phi & \textit{iff} & M, g' \models \phi, \text{ for all x-variants } g' \text{ of } g.
\end{array}
$$

$I_F^g(\tau)$ is $F(\mathrm{c})$ if the term τ is a constant c, and $g(\mathrm{x})$ if τ is a variable x.

Questions: $I^g{}_F$

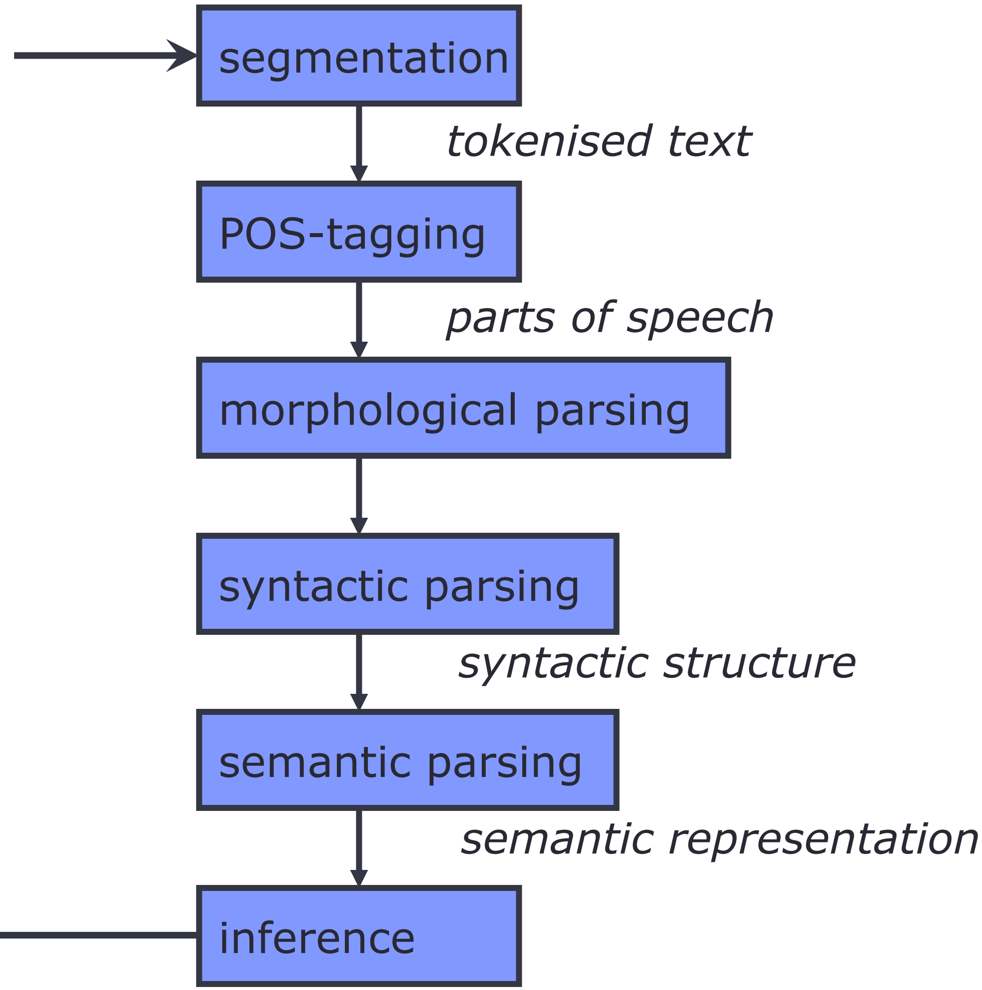
- The horrible $I^g{}_F$ (can’t even typeset it properly in ppt)
- This is a function from terms to entities in the domain
- Recall that terms can be variables or contants
- So basically this function catches two birds with one stone:

  Suppose t is a term.
  If t is a <u>variable</u>, then
  we use the assignment function g: I(t)=g(t)
  If t is a <u>constant</u>, then
  we use the interpretation function F: I(t)=F(t)

Semantic Analysis Pipeline


WHO COULD BUY CHICAGO

FOUR OF THE RICHEST WOMEN IN THE WORLD.



Natural Language Descriptions

TRUE DESCRIPTIONS

- A white rabbit is eating a carrot.
- A rabbit with a carrot.
- A rabbit is nibbling on a carrot.
- A rabbit holding a carrot in its mouth.
- A carrot is being eaten by a rabbit.

FALSE DESCRIPTIONS

- A rabbit without a carrot.
- A brown rabbit is eating an orange carrot.
- Two rabbits are sharing a carrot.
- A carrot is holding a white rabbit.
- A rabbit with orange flowers.

Natural Language Descriptions

TRUE DESCRIPTIONS

- ....
- ....

FALSE DESCRIPTIONS

- ....
- ....

Natural Language Descriptions

TRUE DESCRIPTIONS

- ....
- ....

FALSE DESCRIPTIONS

- ....
- ....

Description guidelines

- Try to include at least two entities in your description
- Only describe the situation, not what is around it
  i.e., not “a girl is looking <u>into a camera</u>”
- Don’t use relative positional information
  i.e., not “a cat is standing <u>left</u> of a dog”

Goal

- Build first-order meaning representations from natural language descriptions, using the vocabulary of non-logical symbols used in the models
- We assume that we need **syntax** to give structure to the descriptions, providing us means for a compositional way of constructing meaning representation

Goal

- Build first-order meaning representations from natural language descriptions, using the vocabulary of non-logical symbols used in the models
- We assume that we need **syntax** to give structure to the descriptions, providing us means for a compositional way of constructing meaning representation
- Note:
  recent attempts with neural networks skip syntactic analysis entirely!

Goal

- Build first-order meaning representations from natural language descriptions, using the vocabulary of non-logical symbols used in the models
- We assume that we need **syntax** to give structure to the descriptions, providing us means for a compositional way of constructing meaning representation
- We will have a closer look at two grammar formalisms:
  - phrase structure grammar (DCG)
  - combinatory categorial grammar (CCG) TOMORROW

Definite Clause Grammars (Prolog)

```
s --> np, vp.
np --> det, n.
vp --> tv, np.
vp --> iv.
vp --> av, vp.

det --> [a]. det --> [the]. det --> [every].
np --> [someone]. np --> [somebody].
av --> [is]. av --> [are].
n --> [cat]. n --> [dog].
tv --> [eats]. tv --> [eating].
```

Ordinary clauses in Prolog!
Terminals are in square brackets.
Left-recursive rules not allowed.

Adding constraints

- aspectual features (VP):
  - prp (present participle)
  - pap (past participle)
  - inf (infinitival)
  - pss (passive)
- mood features (S):
  - dcl (declarative)
  - int (interrogative)
- agreement features (NP):
  - sg (singular)
  - pl (plural)

Definite Clause Grammars with Features

```
s --> np, vp.
np --> det, n.
vp([F]) --> tv([F]), np.
vp([F]) --> iv([F]).
vp([M]) --> av([M,A]), vp([A]).

det --> [a]. det --> [the]. det --> [every].
np --> [someone]. np --> [somebody].
av([dcl,prp]) --> [is]. av([dcl,prp]) --> [are].
n --> [cat]. n --> [dog].
tv([dcl]) --> [eats]. tv([prp]) --> [eating].
```

Here we use **lists** to be able to add more features. Order is important!

Eliminating left-recursive rules

- DCG can't handle left-recursive grammars
(because of Prolog's top-down search strategy it risks
to go in an infinite loop)
- The simple cases of left recursion (direct left recursion) can be eliminated from a DCG
- These cases are of the form (X is a non-terminal, Y and Z are terminal or non-terminal categories):

```
X --> X, Y.
X --> Z.
```

left-recursive DCG schema

```
X --> Z, X'.
X' --> [].
X' --> Y, X'.
```

left-recursion eliminated by introducing new category and empty production

Example: *italian*

```
np --> det, n.
n --> n, adj.
n --> adj, n.

det --> [una].   det --> [la].
n --> [casa].    n --> [cosa].
adj --> [bella]. adj --> [nuova].
```

Provide DCG analyses

TRUE DESCRIPTIONS

- A white rabbit is eating a carrot.
- A rabbit with a carrot.
- A rabbit is nibbling on a carrot.
- A rabbit holding a carrot in its mouth.
- A carrot is being eaten by a rabbit.

FALSE DESCRIPTIONS

- A rabbit without a carrot.
- A brown rabbit is eating an orange carrot.
- Two rabbits are sharing a carrot.
- A carrot is holding a white rabbit.
- A rabbit with orange flowers.


YOU GET A PARSER FOR FREE WITH PROLOG!
NOT SURE IT IS A PARSER I WANT TO USE

Non-logical symbols

- Concepts (WordNet)
- Relations (spatial relations only)

part of	->	s_part_of
touch	->	s_touch
near	->	s_near
support	->	s_support

- Inferences
  - *support* implies *touch*
  - *near* implies not *touch* and not *part of*
  - *touch* implies not *part of*

The big question

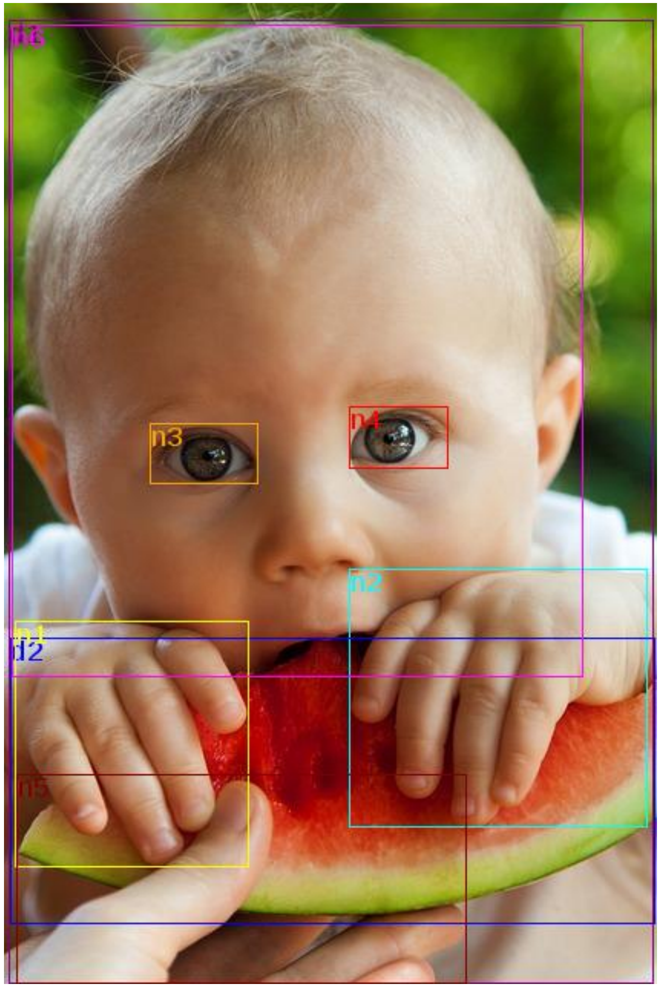
- How can we associate a natural language description like "every cat is drinking milk" with its first-order translation:
  $\forall x[n\_cat\_1(x) \rightarrow \exists y\ [n\_milk\_1(y)\ \&\ s\_near(x,y)]]$?

- Moreover: how can we do this in a <u>**systematic**</u> way? We want to make our method scalable to other kinds of natural language expressions, including those that we have never seen before!

Another example

Someone is holding a melon.

∃x [n_person_1(x) &
∃y [n_melon_2(y) &
∃z [n_hand_1(z) &
s_part_of(z,x) &
s_supports(z,y)]]]

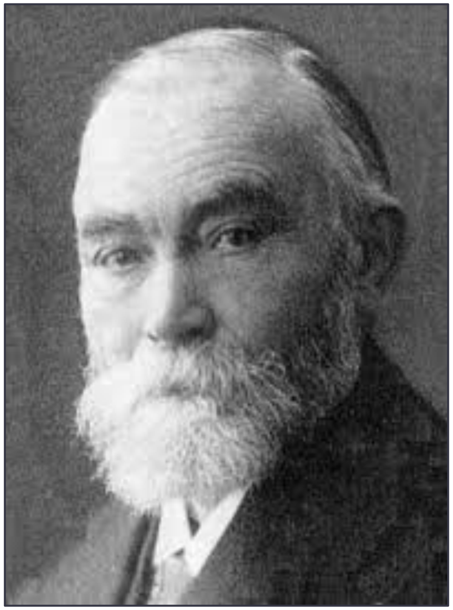


Next

- We will have a look at DCG the again
- But now we will specify the lexical semantics
- And we show how composition works
- But first, more about compositionality

Compositionality

- We assume that the meaning representation of a sentence is composed out of the (partial) meaning representations of its parts (i.e., the words)
- This principle is known as *compositionality*, often misattributed to Frege [Janssen 2012]

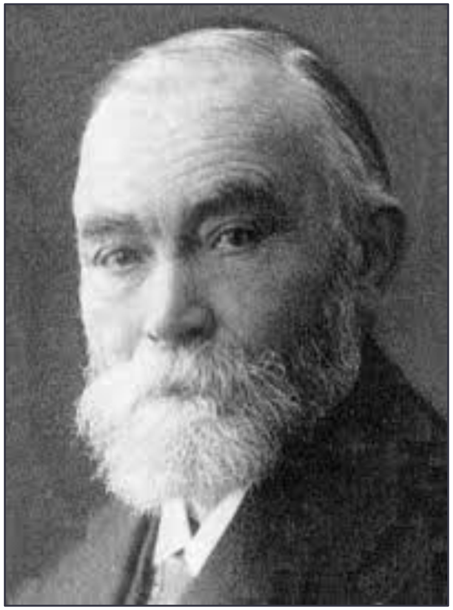


Frege

Compositionality

- We assume that the meaning representation of a sentence is composed out of the (partial) meaning representations of its parts (i.e., the words)
- This principle is known as *compositionality*, often misattributed to Frege [Janssen 2012]

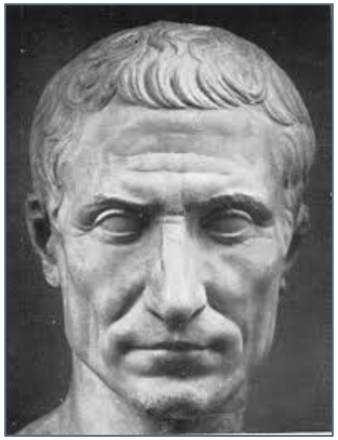
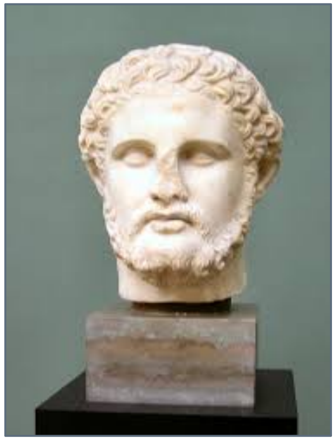
Frege



Carnap

Compositionality

- Generally speaking, the motivation for compositionality is not for principled, but for practical reasons
- This follows an old wisdom, often attributed to Julius Caesar, but probably from Philippus of Macedonia (father of Alexander the Great): compositionality implements the rule <u>**divide et impera**</u> [Janssen 2012]

Caesar   *Philippus*

Decomposing

"every cat is drinking milk" ≈
$\forall x[\mathrm{CAT}(x) \rightarrow \exists y\, [\mathrm{MILK}(y)\ \&\ \mathrm{NEAR}(x,y)]]$

"every" ≈ ...

"cat" ≈ ...

"is" ≈ ...

"drinking" ≈ ...

"milk" ≈ ...

Decomposing

"every cat is drinking milk" ≈
$\forall x[\mathrm{CAT}(x) \rightarrow \exists y\,[\mathrm{MILK}(y)\ \&\ \mathrm{NEAR}(x,y)]]$

"every" ≈ ...

"cat" ≈ CAT(x)

"is" ≈ ...

"drinking" ≈ NEAR(x,y)

"milk" ≈ MILK(y)

Decomposing

"every cat is drinking milk" ≈
$\forall x[CAT(x) \rightarrow \exists y\ [MILK(y)\ \&\ NEAR(x,y)]]$

"every" ≈ $\forall x\ [\ ...(x) \rightarrow\ ...(x)\ ]$

"cat" ≈ $CAT(x)$

"is" ≈ ...

"drinking" ≈ $NEAR(x,y)$

"milk" ≈ $MILK(y)$

Decomposing

"every cat is drinking milk" ≈
$\forall x[CAT(x) \rightarrow \exists y\ [MILK(y)\ \&\ NEAR(x,y)]]$

"every" ≈ $\forall x\ [\ ...(x) \rightarrow\ \ ...(x)\ ]$

"cat" ≈ $CAT(x)$

"is" ≈ *nothing?*

"drinking" ≈ $NEAR(x,y)$

"milk" ≈ $MILK(y)$

Decomposing

"every cat is drinking milk" ≈

$\forall x[\mathrm{CAT}(x) \rightarrow \exists y\ [\mathrm{MILK}(y)\ \&\ \mathrm{NEAR}(x,y)]]$

"every" ≈ $\forall x\ [\ \ldots(x) \rightarrow\ \ldots(x)\ ]$

"cat" ≈ $\mathrm{CAT}(x)$

"is" ≈ *nothing?*

"drinking" ≈ $\mathrm{NEAR}(x,y)$

"milk" ≈ $\mathrm{MILK}(y)$

Decomposing

“every cat is drinking milk” ≈
$\forall x[CAT(x) \rightarrow \exists y\ [MILK(y)\ \&\ NEAR(x,y)]]$

“every” ≈ $\forall x\ [\ ...(x) \rightarrow\ \ ...(x)\ ]$

“cat” ≈ $CAT(x)$

“is” ≈ *nothing?*

“drinking” ≈ $NEAR(x,y)$

“milk” ≈ $\exists y\ [MILK(y)\ \&\ ...(y)\ ]$

What do we observe?

- Open spaces for formulas (the ...), sometimes more than one!
- Variables need to be correctly bound, sometimes more than one!
- Some lexical items seem to have no "semantic contribution"

Partial formulas

We will add a couple of new operators to describe partial formulas:

λ @

- The lambda operator **λ** signals **missing information**
  The lambda binds variables (like the quantifiers) and is placed in front of a formula (like the quantifiers)

- The application operator **@** indicates that two pieces of information need to be **combined**

Adding lambdas and applications

"every cat is drinking milk" ≈
$\forall x[CAT(x) \rightarrow \exists y\ [MILK(y)\ \&\ NEAR(x,y)]]$

"every" ≈ $\lambda p \lambda q \forall x\ [\ (p@x) \rightarrow (q@x)\ ]$

"cat" ≈ $\lambda x\ CAT(x)$

"is" ≈ $\lambda f\ f$

"drinking" ≈ $\lambda y\ \lambda x\ NEAR(x,y)$

"milk" ≈ $\lambda p \exists y\ [MILK(y)\ \&\ (p@y)\ ]$

Higher order logic

- lambda-bound variables can also range over non-entities (i.e. properties and formulas)
- this means that we have left the (relatively safe) domain of first-order logic
- we will use the lambdas purely as a device to construct formulas from smaller parts
- it will provide us a way to control free and bound variables

With a little help of syntactic structure

- Syntax (DCG, CCG, or something else) helps us to find out what combines with what
- Consider the following (simplified) DCG

s → np vp	det → [every]
np → det n	n → [cat]
np → n	n → [milk]
vp → tv np	av → [is]
vp → av vp	tv → [drinking]

- Next step: add semantics

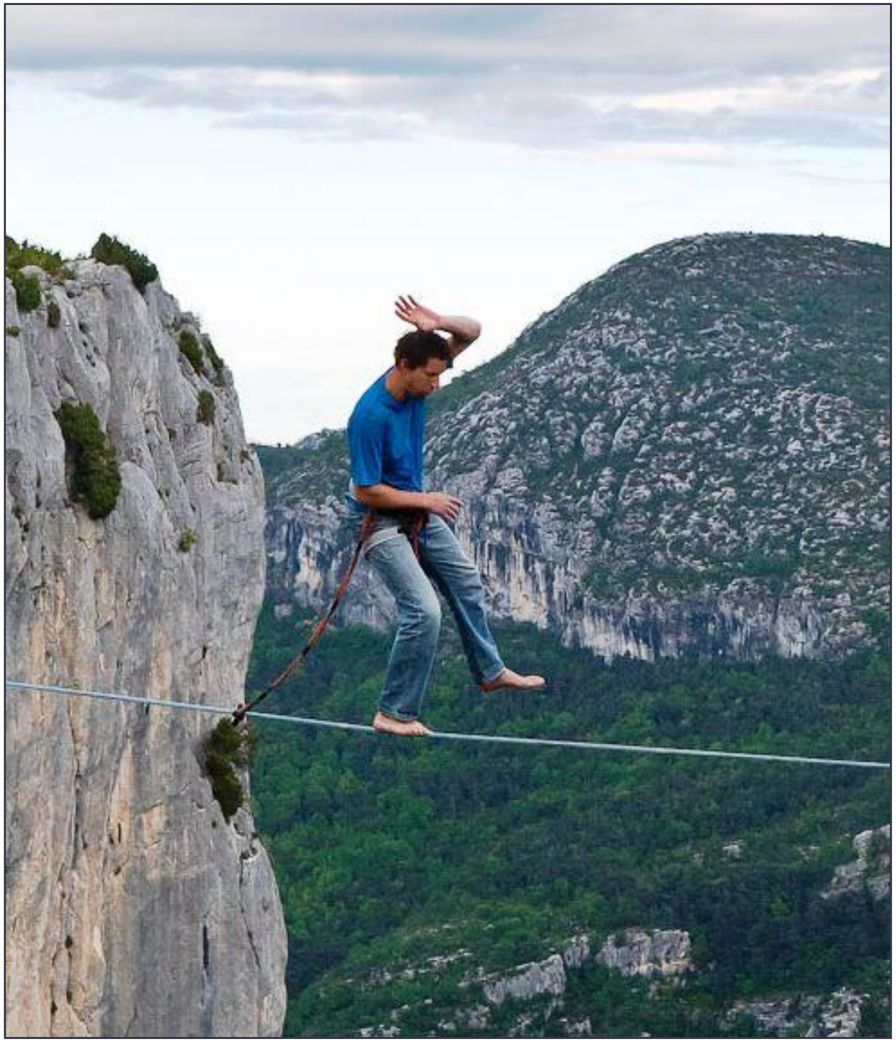
The semantics in the lexicon

det [sem: λpλq∀x[(p@x) → (q@x)]] → [every]
n [sem: λx CAT(x)]→ [cat]
n [sem: λx MILK(x)]→ [milk]
av [sem: λf f]→ [is]
tv [sem: λxλy NEAR(x,y)]→ [drinking]

The semantics in the rules

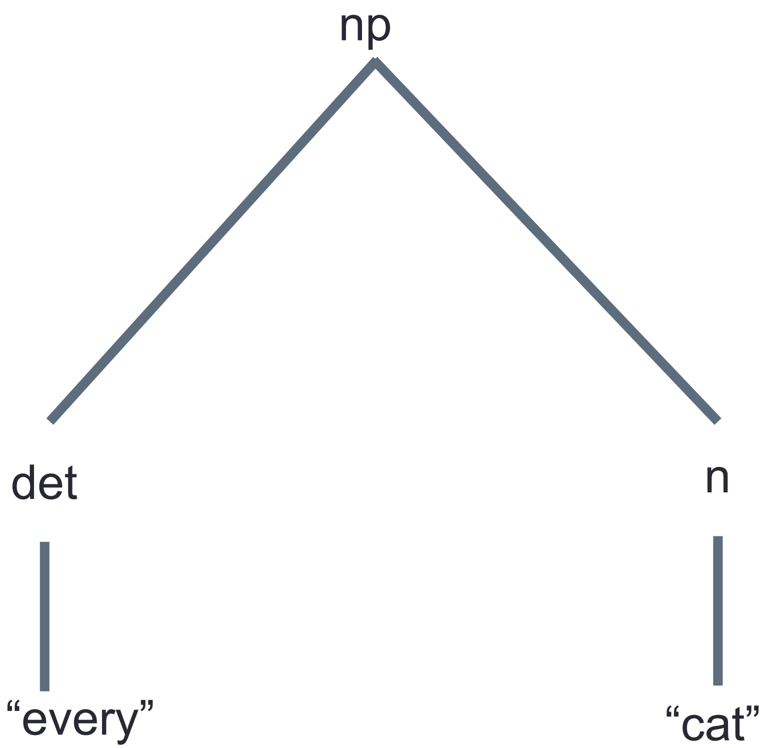
s[sem: (X@Y)] → np[sem:X] vp[sem:Y]
np[sem: (X@Y)]→ det[sem:X] n[sem:Y]
np [sem: ∃x(Y@x)]→ n[sem:Y]
vp [sem: λx(Y@(X@x))]→ tv[sem:X] np[sem:Y]
vp [sem: (X@Y)]→ av[sem:X] vp[sem:Y]

One picture says more than a thousand ~~words~~ variables

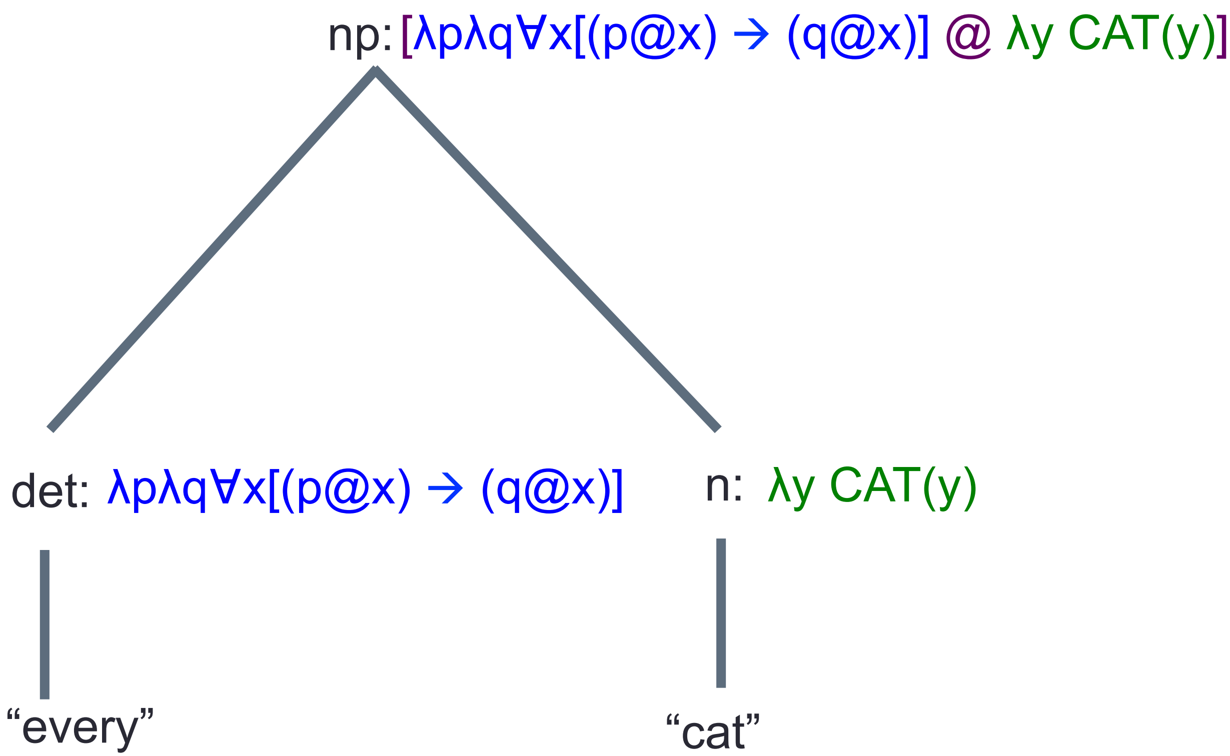


Butch on his chopper

np → det: n



$np{:}[\varphi @ \psi] \rightarrow det{:}\varphi \; n{:}\psi$



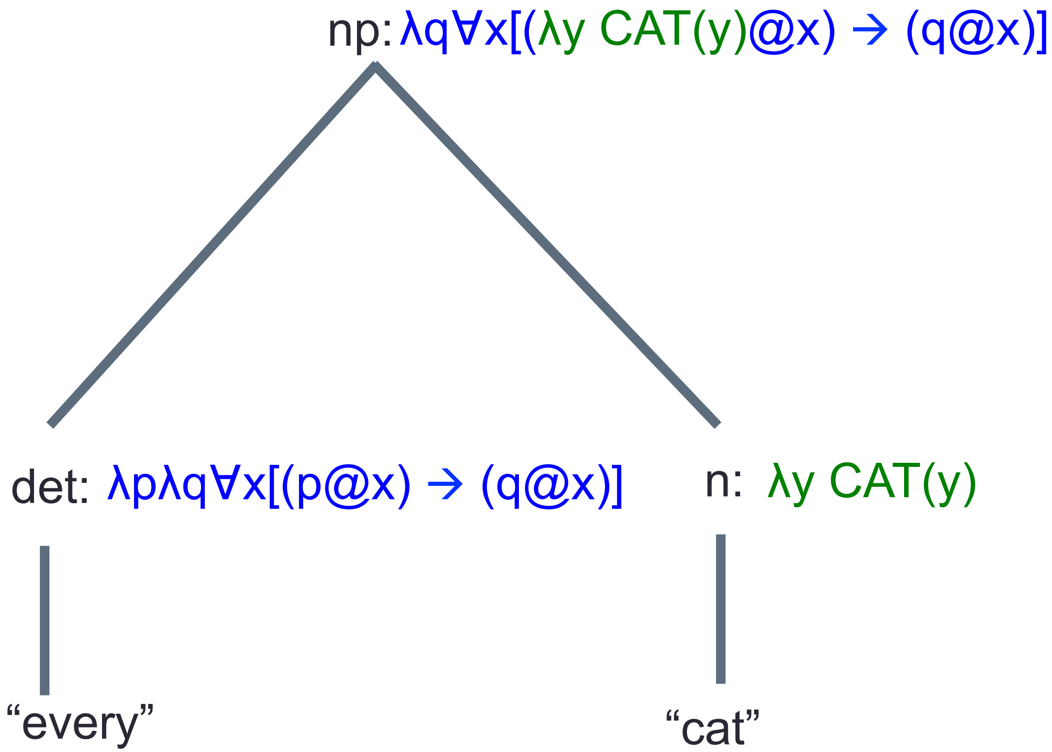
β-conversion

- Consider the application: (λxφ@ψ)
  - Here the functor is: λxφ
  - And the argument is: ψ
- The process of replacing every free occurrence of x in φ by ψ is called

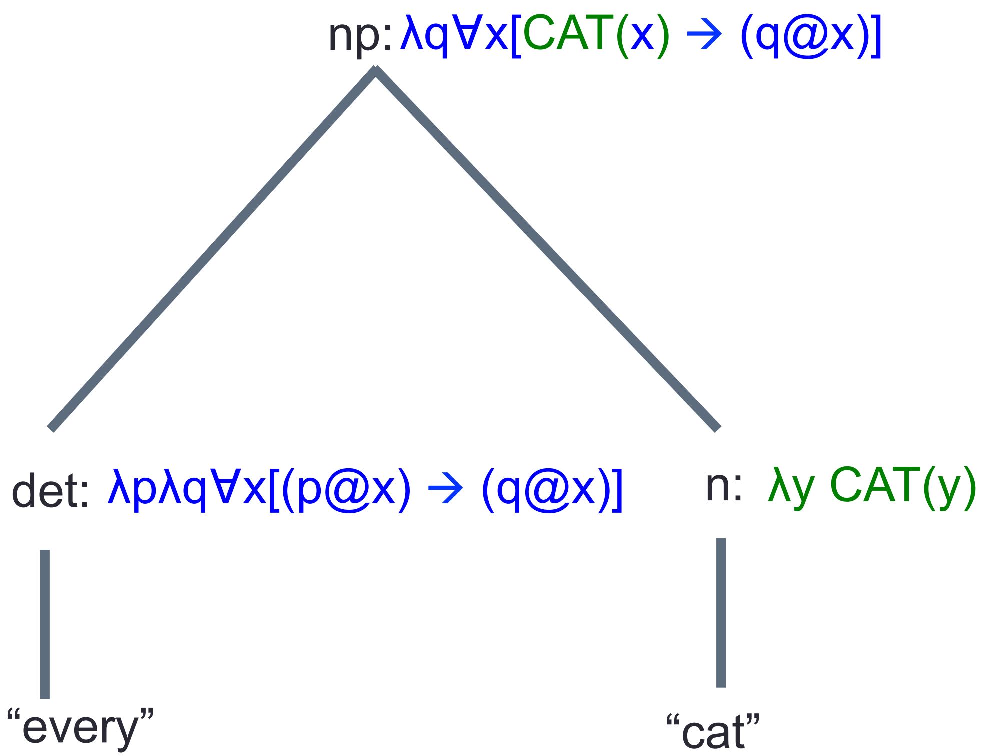
  β-conversion

  (or β-reduction, or λ-conversion)

np:[φ@ψ] → det:φ n:ψ



np:[φ@ψ] → det:φ n:ψ



Demo

- ~/doc/tea/ComputationalSemantics % cat essligrammar.pl
- ~/doc/tea/ComputationalSemantics % cat lexicon.pl
- ~/doc/tea/ComputationalSemantics % cat semdcg.pl

- [semdcg], s(Sem,[a,man,rides,a,bicycle],[]).

Exercise 1

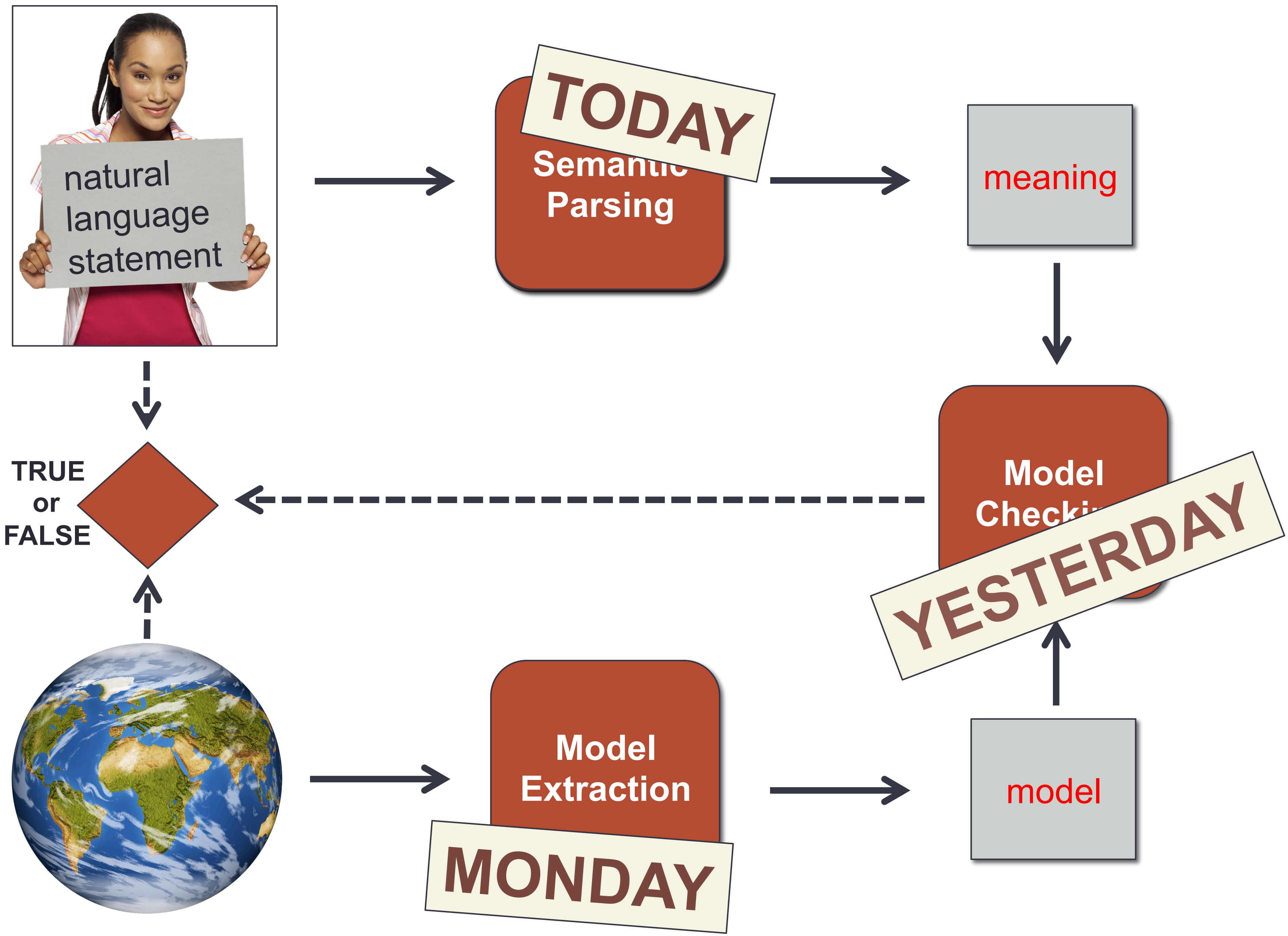
- Not only sentences, also noun phrases.

Exercise 2

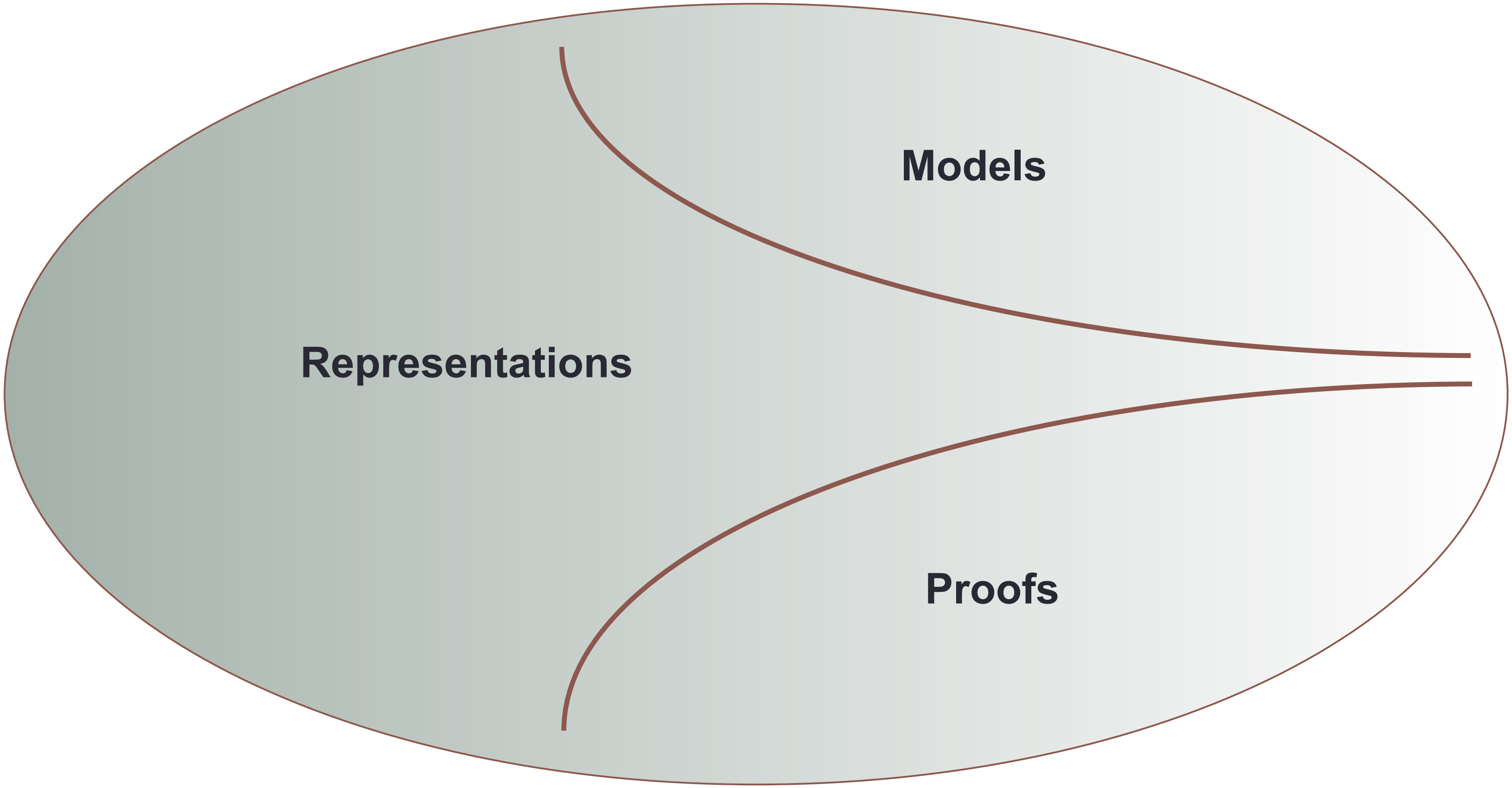
- Look at the natural language statements associated with the images in GRIM
- Pick a frequently occurring verb that is not in the lexicon already
- Specify the lexical semantics of this verb in

  a) no events (pre-Davidsonian)
  b) Davidsonian
  c) neo-Davidsonian
  d) the spatial relations only

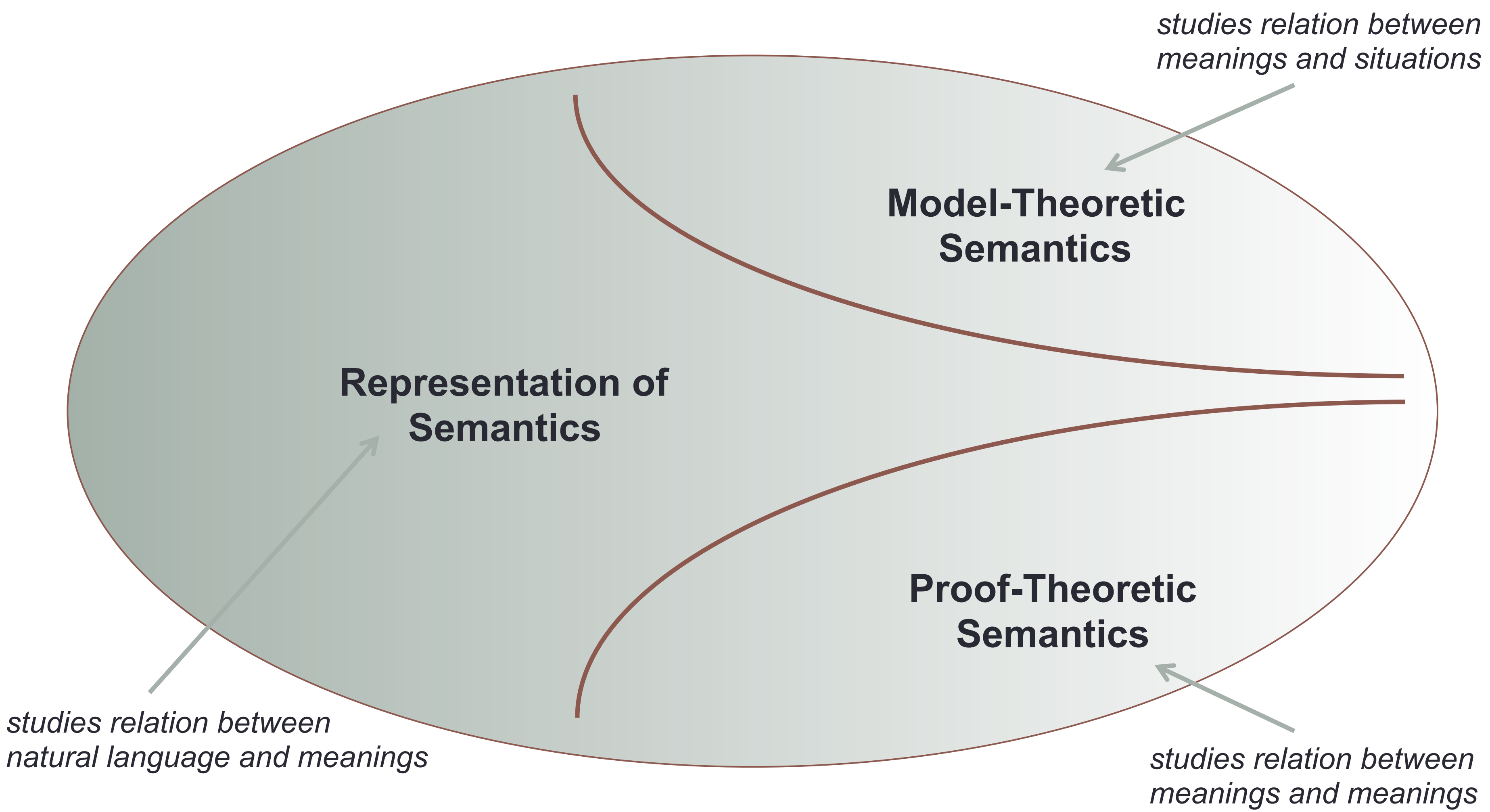
The Big Picture



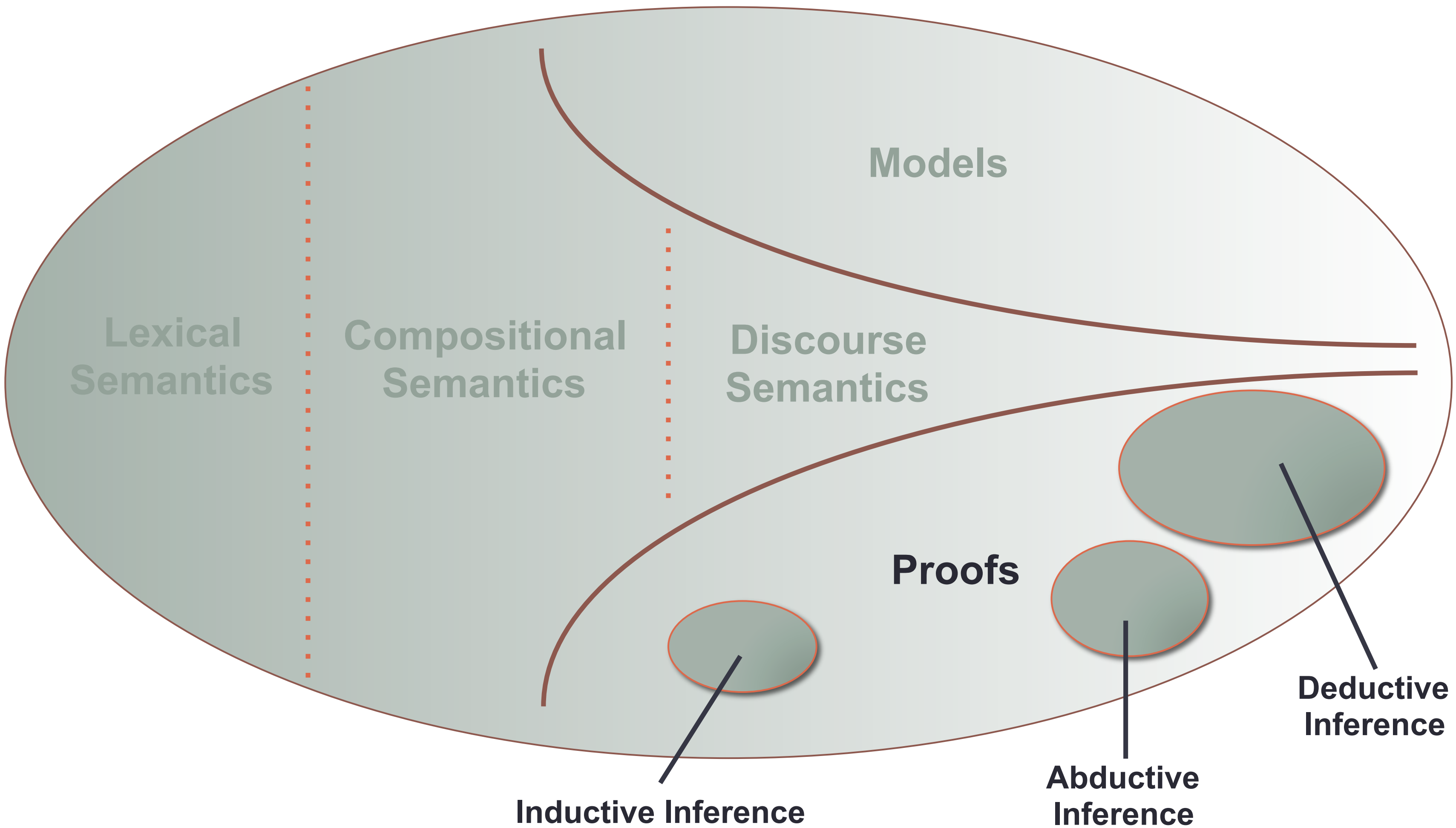
Planet Semantics



Planet Semantics



Proof-Theoretical Semantics



Computational Semantics

- Day 1: Exploring Models
- Day 2: Meaning Representations
- Day 3: Computing Meanings with DCG
- **Day 4: Computing Meanings with CCG**
- Day 5: Drawing Inferences and Meaning Banking