Module 1B: Semantics

Ann Copestake

Computer Laboratory
University of Cambridge

November 2009

Outline of today's lecture

Lecture 1: Introduction

- Overview of the course
- Notes on lectures and lecture notes
- What is compositional semantics?
- Model-theoretic semantics and denotation
- Natural language vs logical connectives

Syllabus

1. **Introduction**
2. **Introduction to compositional semantics**
3. **Typed lambda calculus**
4. **Constraint-based semantics**
5. **More on scope and quantifiers**
6. **Building underspecified semantics**
7. **Extreme underspecification**
   (Notes will be provided later)

Syllabus

1. **Introduction**
2. **Introduction to compositional semantics**
3. **Typed lambda calculus**
4. **Constraint-based semantics**
5. **More on scope and quantifiers**
6. **Building underspecified semantics**
7. **Extreme underspecification**
   (Notes will be provided later)

Notes and intro exercises

- optional sections
- revision sections
- introductory exercises
- exercises in notes
- past papers
- reading

Natural language interfaces and limited domains

- Natural language interfaces: interpret a query with respect to a very limited domain (microworld).
- CHAT-80 (`http://www.lpa.co.uk/pws_dem5.htm`)

  ```
  What is the population of India?
  ```

  Domain-dependent grammar gives meaning representation:

  ```
  which(X:exists(X:(isa(X,population)
                        and of(X,india))))
  ```

  Inference and match on Prolog database:

  ```
  have(india,(population=900)).
  ```

- But such techniques do not scale up to larger domains.

Semantics in information management?

- Enables abstraction:
  - Paper 1: The synthesis of 2,8-dimethyl-6H,12H-5,11 methanodibenzo[b,f][1,5]diazocine (Troger's base) from p-toluidine and of two Troger's base analogs from other anilines
  - Paper 2: ... Tröger's base (TB) ... The TBs are usually prepared from para-substituted anilines
- Robust inference: e.g., search for papers describing Tröger's base syntheses which don't involve anilines?
- Aiming for domain and application independence.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs ‘attacking Vogons are annoying’ and ‘spelling contests are boring’)
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs ‘the attack of the Vogons’)
- aspirin’s synthesis (noun+pos noun) (vs ‘the Vogons’ attack’)
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs ‘attacking Vogons are annoying’ and ‘spelling contests are boring’)
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs ‘the attack of the Vogons’)
- aspirin’s synthesis (noun+pos noun) (vs ‘the Vogons’ attack’)
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs ‘attacking Vogons are annoying’ and ‘spelling contests are boring’)
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun)
  (vs ‘the attack of the Vogons’)
- aspirin’s synthesis (noun+pos noun)
  (vs ‘the Vogons’ attack’)
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs ‘attacking Vogons are annoying’ and ‘spelling contests are boring’)
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs ‘the attack of the Vogons’)
- aspirin’s synthesis (noun+pos noun) (vs ‘the Vogons’ attack’)
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Syntactic variability.

- Hoffman synthesized/synthesised aspirin (verb+ed NP)
- aspirin was synthesised by Hoffman (NP be verb+ed)
- synthesising aspirin is easy (verb+ing NP) (vs 'attacking Vogons are annoying' and 'spelling contests are boring')
- the synthesised aspirin (verb+ed/adj noun)
- the synthesis of aspirin (noun of noun) (vs 'the attack of the Vogons')
- aspirin's synthesis (noun+pos noun) (vs 'the Vogons' attack')
- aspirin synthesis (noun noun)

Common semantics (ideally) or appropriate entailment patterns.

Semantics in NLP applications

1. Various applications need meaning representations: if possible, we want to use the same sort of meaning representation in as many applications as possible, so we can build modular parsers and generators.
2. Inference, of different sorts, is important in many applications.
3. Formal specification, so meaning representations can be understood and reused.
4. All this argues for some form of logic as a meaning representation.

Compositional semantics

- Compositional semantics: building up the meaning of an utterance in a predicatable way from the meaning of the parts.
  Roughly: semantics from syntax, closed class words and inflectional morphology.
- Lexical semantics.
- Real world knowledge (or micro-world knowledge).

Contradictions

Compositional semantics should account for logical contradiction:

(1) Kim is an aardvark.
Kim is not an aardvark.

(2) If Kim can play chess then Kim can ride a motorbike.
Kim can play chess but Kim cannot ride a motorbike.

(3) Every dog has a tail.
Some dogs do not have tails.

An aardvark

Entailment

Also account for entailment:

(4) Every dog has a tail. Kim is a dog. $\Longrightarrow$ Kim has a tail.

But not:

(5) Kim is a bachelor. $\Longrightarrow$ Kim is not married.

(6) Sandy is a tiger. $\Longrightarrow$ Sandy is an animal.

Entailment

Also account for entailment:

(4) Every dog has a tail. Kim is a dog. $\Longrightarrow$ Kim has a tail.

But not:

(5) Kim is a bachelor. $\Longrightarrow$ Kim is not married.

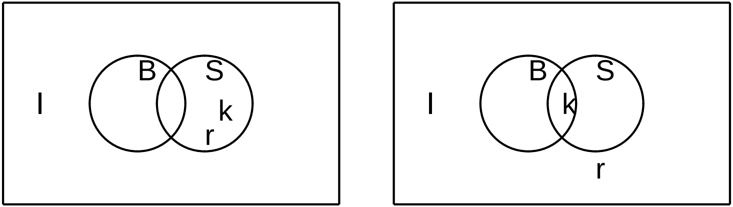
(6) Sandy is a tiger. $\Longrightarrow$ Sandy is an animal.

Model-theoretic semantics

Kitty sleeps is true in a particular model if the individual denoted by Kitty (say k) in that model is a member of the set denoted by *sleep* (S):

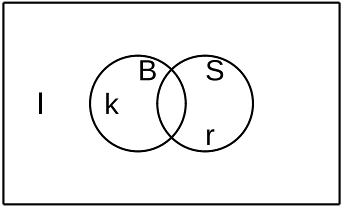
$$k \in S$$

Two models where *Kitty sleeps* is true:

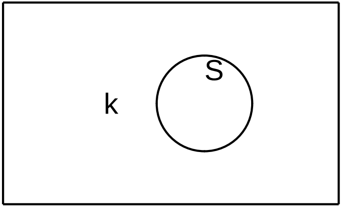


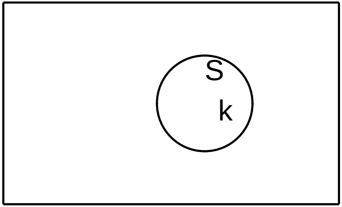
Model-theoretic semantics

A model where *Kitty sleeps* is false:



Only showing relevant entities:





Ordered pairs

- The denotation of *chase* is a set of ordered pairs.
- For instance, if Kitty chases Rover and Lynx chases Rover and no other chasing occurs then *chase* denotes $\{\langle k, r\rangle, \langle l, r\rangle\}$.
- Ordered pairs are not the same as sets.
  - Repeated elements: if *chase* denotes $\{\langle r, r\rangle\}$ then Rover chased himself.
  - Order is significant, $\langle k, r\rangle$ is not the same as $\langle r, k\rangle$ 'Kitty chased Rover' vs 'Rover chased Kitty'

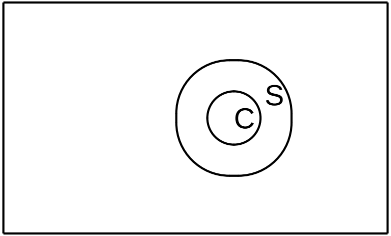
every, some and no

The sentence *every cat sleeps* is true just in case the set of all cats is a subset of the set of all things that sleep.
If the set of cats is $\{k, l\}$ then *every cat sleeps* is equivalent to:

$$\{k, l\} \subseteq S$$

Or, if we name the set of cats C:

$$C \subseteq S$$



every, some and no

The following sentence has two possible interpretations:

(7) every cat does not sleep

It can be interpreted in the same way as either of:

(8) No cat sleeps

(9) It is not the case that every cat sleeps

Logic and model theory

- Model theory: meaning can be expressed set theoretically with respect to a model.
- But set theoretic representation is messy: we want to abstract away from individual models.
- Instead, think in terms of logic: truth-conditions for *and*, *or* etc. e.g., if *Kitty sleeps* is true, then *Kitty sleeps or Rover barks* will necessarily also be true.

Natural language vs logical connectives

The correspondence between English *and*, *or*, *if* . . . *then* and *not* and the logical $\wedge$, $\vee$, $\implies$ and $\neg$ is not straightforward.

(10) a. The Lone Ranger jumped on his horse and rode away.

b. ? The Lone Ranger rode away and jumped on his horse.

(11) The price of the meal includes a glass of wine or a glass of beer.

(12) If Kitty is invisible then everyone will see Kitty.
(If we assume that Kitty is not invisible, then this sentence would be true if we used $\implies$ to translate it.)

Other connectives

English has other connectives whose meaning partially corresponds to the logical connectives, such as *but*:

(14) Lynx growled but Kitty purred.

This is true in the same models as:

(15) Lynx growled and Kitty purred.

However, *but* indicates a contrast, as we can see if we try and use it to conjoin two sentences which intuitively don't contrast:

(16) a. ? Lynx growled but Kitty growled.
b. Lynx growled and Kitty growled.

Next lecture

Building up logical representations (logical forms) compositionally from syntax.

Lecture 2: Introduction to semantic composition

- Semantic composition with propositional logic
- General principles of semantic composition
- Semantic composition with quantifier-free predicate logic
- Quantifiers
- The semantics of some nominal modifiers

Outline of Lecture 2

Semantic composition with propositional logic

- Propositional logic: ignore the internal structure of sentences entirely.
  Kitty sleeps and Rover barks
  $P \wedge Q$
  Interpretation depends on truth values of P and of Q in the model.

Logic and grammar: Grammar Fragment 1

```
S -> S1 and S2
```
$(S1' \wedge S2')$

```
S -> S1 or S2
```
$(S1' \vee S2')$

```
S -> if S1 then S2
```
$(S1' \implies S2')$

```
S -> it-is-not-the-case-that S1
```
$(\neg S1')$

Base sentences:

S -> 'Kitty sleeps' (true)

S -> 'Lynx sleeps' (false)

S -> 'Lynx chases Rover' (false)

Grammar Fragment 1: Interpretation

Interpretation of English sentences with this grammar and model:

(18) 'Lynx sleeps' or 'Lynx chases Rover'
([Lynx sleeps]′ ∨ [Lynx chases Rover]′)
(false ∨ false)
false

(19) it-is-not-the-case-that 'Lynx sleeps'
(¬ [Lynx sleeps]′)
¬ false
true

Grammar Fragment 1: Ambiguous examples

(20) ‘Kitty sleeps’ or ‘Lynx chases Rover’ and ‘Lynx sleeps’
(‘Kitty sleeps’ or ‘Lynx chases Rover’) and ‘Lynx sleeps’
(([Kitty sleeps]′ ∨ [Lynx chases Rover]′) ∧ [Lynx sleeps]′)
((true ∨ false) ∧ false)
false

(21) Kitty sleeps or Lynx chases Rover and Lynx sleeps (other bracketing)
Kitty sleeps or (Lynx chases Rover and Lynx sleeps)
([Kitty sleeps]′ ∨ ([Lynx chases Rover]′ ∧ [Lynx sleeps]′))
(true ∨ (false ∧ false))
true

Semantic composition

- Semantic rules parallel syntax rules.
- Semantics is build up compositionally: meaning of the whole is determined from the meaning of the parts.
- Semantic derivation: constructing the semantics for a sentence.
- Interpretation with respect to a model (true or false).
- The logical expressions constructed (logical form) could (in principle) be dispensed with.
  Maybe . . .

Semantic composition

- Semantic rules parallel syntax rules.
- Semantics is build up compositionally: meaning of the whole is determined from the meaning of the parts.
- Semantic derivation: constructing the semantics for a sentence.
- Interpretation with respect to a model (true or false).
- The logical expressions constructed (logical form) could (in principle) be dispensed with.
  Maybe . . .

Predicates in grammar

`S -> NP Vintrans` $V'(NP')$	`NP -> Kitty` k
`Vintrans -> barks` bark′	`NP -> Lynx` l
`Vintrans -> sleeps` sleep′	`NP -> Rover` r

Conventions: term in italics refers to the word itself (e.g., *sleep*)
apostrophe symbol indicates the denotation (e.g., sleep′)
k, r and l are constants here.

A first account of transitive verbs

For instance: $\text{chase}'(k, r)$ — *Kitty chases Rover*

```
S -> NP1 Vtrans NP2
```

$V'(NP1', NP2')$

```
Vtrans -> chases
```

chase′

But the syntax is wrong. We want:

```
S -> NP VP
VP -> Vtrans NP
Vtrans -> chases
```

Solution later . . .

Grammar Fragment 2

S -> it-is-not-the-case-that S1
$(\neg S1')$

S -> S1 and S2
$(S1' \wedge S2')$

S -> S1 or S2
$(S1' \vee S2')$

S -> if S1 then S2
$(S1' \implies S2')$

S -> NP Vintrans
$V'(NP')$

S -> NP1 Vtrans NP2
$V'(NP1', NP2')$

Vintrans -> barks
bark′

Vintrans -> sleeps
sleep′

Vtrans -> chases
chase′

NP -> Kitty
k

NP -> Lynx
l

NP -> Rover
r

Example with Grammar Fragment 2

Kitty chases Rover and Rover barks

Quantifiers

(22) $\exists x[\text{sleep}'(x)]$
Something sleeps

(23) $\forall x[\text{sleep}'(x)]$
Everything sleeps

(24) $\exists x[\text{cat}'(x) \wedge \text{sleep}'(x)]$
Some cat sleeps

(25) $\forall x[\text{cat}'(x) \implies \text{sleep}'(x)]$
Every cat sleeps

(26) $\forall x[\text{cat}'(x) \implies \exists y[\text{chase}'(x, y)]]$
Every cat chases something

(27) $\forall x[\text{cat}'(x) \implies \exists y[\text{dog}'(y) \wedge \text{chase}'(x, y)]]$
Every cat chases some dog

Variables

- x, y, z pick out entities in model according to variable assignment function: e.g., sleeps$'(x)$ may be true or false in a particular model, depending on the function.
- Constants and variables:

  (29) $\forall x[\text{cat}'(x) \implies \text{chase}'(x, r)]$
  Every cat chases Rover.
- No explicit representation of variable assignment function: we just care about bound variables for now (i.e., variables in the scope of a quantifier).

Quantifier scope ambiguity

- The truth conditions of formulae with quantifiers depend on the relative scope of the quantifiers
- Natural languages sentences can be ambiguous wrt FOPC without being syntactically ambiguous
- *Everybody in the room speaks two languages* same two languages or not?

The semantics of some nominal modifiers

(33) every big cat sleeps

$$\forall x[(\text{cat}'(x) \wedge \text{big}'(x)) \implies \text{sleep}'(x)]$$

(34) every cat on some mat sleeps
wide scope *every*:

$$\forall x[(\text{cat}'(x) \wedge \exists y[\text{mat}'(y) \wedge \text{on}'(x,y)]) \implies \text{sleep}'(x)]$$

wide scope *some* (i.e., single mat):

$$\exists y[\text{mat}'(y) \wedge \forall x[(\text{cat}'(x) \wedge \text{on}'(x,y)) \implies \text{sleep}'(x)]]$$

$\text{on}'(x,y)$ must be in the scope of both quantifiers.

Adjectives and prepositional phrases (in this use) are syntactically modifiers.
Semantically: intersective modifiers: combine using $\wedge$, modified phrase denotes a subset of what's denoted by the noun.

Going from FOPC to natural language

Well-formed FOPC expressions, don't always correspond to natural NL utterances. For instance:

(35) $\forall x[\text{cat}'(x) \wedge \exists y[\text{bark}'(y)]]$

This best paraphrase of this I can come up with is:

(36) Everything is a cat and there is something which barks.

Next lecture

Typed lambda calculus and composition.

Lecture 3: Composition with typed lambda calculus

- Typing in compositional semantics
- Lambda expressions
- Example grammar with lambda calculus
- Quantifiers again

Outline of Lecture 3

Composition using typed lambda calculus (in a nutshell . . .)

Lecture 3: Composition with typed lambda calculus
- Typing in compositional semantics
- Lambda expressions
- Example grammar with lambda calculus
- Quantifiers again

Overview

- We have developed grammar fragments for quantifier-free predicate calculus but transitive verbs were given a syntactically weird analysis. The rule-to-rule hypothesis is that one semantic rule can be given per syntactic rule — but we must assume plausible syntax.
- We have seen that FOPC can be used to represent sentences, but we have not seen how to compose sentences with quantifiers

In this lecture, we'll introduce:

- typing, which enforces well-formedness (i.e., specifies what expressions can go together)
- lambda calculus is a more powerful notation for semantic composition

Typing

- Semantic typing ensures that semantic expressions are consistent.
  e.g., chase′(dog′(k)) is ill-formed.
- Two basic types:
  - e is the type for entities in the model (such as k)
  - t is the type for truth values (i.e., either ‘true’ or ‘false’)

  All other types are composites of the basic types.
- Complex types are written $\langle$type1, type2$\rangle$, where type1 is the argument type and type2 is the result type and either type1 or type2 can be basic or complex.
  $\langle e, \langle e, t\rangle\rangle$, $\langle t, \langle t, t\rangle\rangle$

Typing

- Semantic typing ensures that semantic expressions are consistent.
  e.g., chase′(dog′(k)) is ill-formed.
- Two basic types:
  - e is the type for entities in the model (such as k)
  - t is the type for truth values (i.e., either ‘true’ or ‘false’)

  All other types are composites of the basic types.
- Complex types are written $\langle$type1, type2$\rangle$, where type1 is the argument type and type2 is the result type and either type1 or type2 can be basic or complex.
  $\langle e, \langle e, t\rangle\rangle$, $\langle t, \langle t, t\rangle\rangle$

Typing

- Semantic typing ensures that semantic expressions are consistent.
  e.g., $\text{chase}'(\text{dog}'(k))$ is ill-formed.
- Two basic types:
  - e is the type for entities in the model (such as k)
  - t is the type for truth values (i.e., either 'true' or 'false')

  All other types are composites of the basic types.
- Complex types are written $\langle \text{type1}, \text{type2} \rangle$, where type1 is the argument type and type2 is the result type and either type1 or type2 can be basic or complex.
  $\langle e, \langle e, t \rangle \rangle$, $\langle t, \langle t, t \rangle \rangle$

Types of lexical entities

First approximation: predicates corresponding to:

- intransitive verbs (e.g. bark′) — $\langle e, t\rangle$
  take an entity and return a truth value
- (simple) nouns (e.g., dog′, cat′) — $\langle e, t\rangle$
- transitive verbs (e.g., chase′) — $\langle e, \langle e, t\rangle\rangle$
  take an entity and return something of the same type as an intransitive verb

Lambda expressions

Lambda calculus is a logical notation to express the way that predicates ‘look’ for arguments. e.g.,

$$\lambda x[\text{bark}'(x)]$$

- Syntactically, λ is like a quantifier in FOPC: the lambda variable (x above) is said to be within the scope of the lambda operator
- lambda expressions correspond to functions: they denote sets (e.g., $\{x : x \text{ barks}\}$)
- the lambda variable indicates a variable that will be bound by function application.

Lambda conversion

Applying a lambda expression to a term will yield a new term, with the lambda variable replaced by the term (lambda-conversion).

For instance:

$$\lambda x[\text{bark}'(x)](k) = \text{bark}'(k)$$

Lambda conversion and typing

Lambda conversion must respect typing, for example:

$$\begin{array}{ccc} \lambda x[\text{bark}'(x)] & k & \text{bark}'(k) \\ \langle e,t\rangle & e & t \end{array}$$

$$\lambda x[\text{bark}'(x)](k) = \text{bark}'(k)$$

We cannot combine expressions of incompatible types.
e.g.,

$$\lambda x[\text{bark}'(x)](\lambda y[\text{snore}'(y)])$$

is not well-formed

Multiple variables

If the lambda variable is repeated, both instances are instantiated:

$\lambda x[\text{bark}'(x) \wedge \text{sleep}'(x)]$	r	$\text{bark}'(r) \wedge \text{sleep}'(r)$
$\langle e, t \rangle$	e	t

$\lambda x[\text{bark}'(x) \wedge \text{sleep}'(x)]$ denotes the set of things that bark and sleep

$$\lambda x[\text{bark}'(x) \wedge \text{sleep}'(x)](r) = \text{bark}'(r) \wedge \text{sleep}'(r)$$

Transitive and intransitive verbs

A partially instantiated transitive verb predicate is of the same type as an intransitive verb:

$\lambda x[\text{chase}'(x, r)]$	k	$\text{chase}'(k, r)$
$\langle e, t \rangle$	e	t

$\lambda x[\text{chase}'(x, r)]$ is the set of things that chase Rover.

$$\lambda x[\text{chase}'(x, r)](k) = \text{chase}'(k, r)$$

Transitive verbs

Lambdas can be nested: transitive verbs can be represented so they apply to only one argument at once.
For instance:

$$\lambda x[\lambda y[\text{chase}'(y,x)]]$$

often written

$$\lambda x \lambda y[\text{chase}'(y,x)]$$

$$\lambda x[\lambda y[\text{chase}'(y,x)]](r) = \lambda y[\text{chase}'(y,r)]$$

Bracketing shows the order of application in the conventional way:

$$\begin{aligned}(\lambda x[\lambda y[\text{chase}'(y,x)]](r))(k) &= \lambda y[\text{chase}'(y,r)](k) \\ &= \text{chase}'(k,r)\end{aligned}$$

Grammar 2, revised

`S -> NP VP`
$VP'(NP')$

`VP -> Vtrans NP`
$Vtrans'(NP')$

`VP -> Vintrans`
$Vintrans'$

`Vtrans -> chases`
$\lambda x \lambda y[\text{chase}'(y, x)]$

`Vintrans -> barks`
$\lambda z[\text{bark}'(z)]$

`Vintrans -> sleeps`
$\lambda w[\text{sleep}'(w)]$

`NP -> Kitty`
k

`NP -> Lynx`
l

`NP -> Rover`
r

Example 1: lambda calculus with transitive verbs

1. `Vtrans -> chases`
   $\lambda x \lambda y[\text{chase}'(y, x)]$ (type: $\langle e, \langle e, t \rangle \rangle$)
2. `NP -> Rover`
   r (type: e)
3. `VP -> Vtrans NP`
   $\text{Vtrans}'(\text{NP}')$
   $\lambda x \lambda y[\text{chase}'(y, x)](r)$
   $= \lambda y[\text{chase}'(y, r)]$ (type: $\langle e, t \rangle$)
4. `NP -> Lynx`
   l (type: e)
5. `S -> NP VP`
   $\text{VP}'(\text{NP}')$
   $\lambda y[\text{chase}'(y, r)](l)$
   $= \text{chase}'(l, r)$ (type: t)

Ditransitive verbs

The semantics of *give* can be represented as $\lambda x \lambda y \lambda z[\text{give}'(z, y, x)]$. The ditransitive rule is:

```
VP -> Vditrans NP1 NP2
```

$(Vditrans'(NP1'))(NP2')$

Two lambda applications in one rule:

$$
\begin{aligned}
&(\lambda x[\lambda y[\lambda z[\text{give}'(z, y, x)]]](l))(r) \\
&= \lambda y[\lambda z[\text{give}'(z, y, l)]](r) \\
&= \lambda z[\text{give}'(z, r, l)]
\end{aligned}
$$

Here, indirect object is picked up first (arbitrary decision in rule/semantics for *give*)

Ditransitive verbs with PP

PP form of the ditransitive uses the same lexical entry for *give*, but combines the arguments in a different order:

```
VP -> Vditrans NP1 PP
```

$(Vditrans'(PP'))(NP1')$

Example 2

Rover gives Lynx Kitty

1. `Vditrans -> gives`
   $\lambda x[\lambda y[\lambda z[\text{give}'(z, y, x)]]]$ type: $\langle e, \langle e, \langle e, t\rangle\rangle\rangle$
2. `NP -> Lynx`
   l type: e
3. `NP -> Kitty`
   k type: e
4. `VP -> Vditrans NP1 NP2`
   (Vditrans′(NP1′))(NP2′)
   $(\lambda x[\lambda y[\lambda z[\text{give}'(z, y, x)]]](l))(k)$
   $= \lambda y[\lambda z[\text{give}'(z, y, l)]](k)$ type: $\langle e, \langle e, t\rangle\rangle$
   $= \lambda z[\text{give}'(z, k, l)]$ type: $\langle e, t\rangle$

Example 2, continued

5 `NP -> Rover`

r type: e

6 `S -> NP VP`

$VP'(NP')$

$= \lambda z[\text{give}'(z, k, l)](r)$

$= \text{give}'(r, k, l)$ type: t

PP ditransitive: Exercise

Rover gives Kitty to Lynx

Assumptions:

- has the same semantics as
  Rover gives Lynx Kitty
- No semantics associated with *to*
- Same lexical entry for *give* as for the NP case
- So difference has to be in the VP rule

Coordination

Before we introduced *and* etc syncategorematically (i.e., we wrote 'and' in the grammar rule).
Alternative using lambdas:

```
S[conj=yes] -> CONJ S1[conj=no]
```

$CONJ'(S1')$

```
S[conj=no] -> S1[conj=no] S2[conj=yes]
```

$S2'(S1')$

```
CONJ -> and
```

$\lambda P[\lambda Q[P \wedge Q]]$ *type:* $\langle t, \langle t, t \rangle \rangle$

```
CONJ -> or
```

$\lambda P[\lambda Q[P \vee Q]]$ *type:* $\langle t, \langle t, t \rangle \rangle$

Why aren't we using Kleene +?

Coordination

Before we introduced *and* etc syncategorematically (i.e., we wrote ‘and’ in the grammar rule).
Alternative using lambdas:

```
S[conj=yes] -> CONJ S1[conj=no]
```
$CONJ'(S1')$

```
S[conj=no] -> S1[conj=no] S2[conj=yes]
```
$S2'(S1')$

```
CONJ -> and
```
$\lambda P[\lambda Q[P \wedge Q]]$ *type:* $\langle t, \langle t, t \rangle \rangle$

```
CONJ -> or
```
$\lambda P[\lambda Q[P \vee Q]]$ *type:* $\langle t, \langle t, t \rangle \rangle$

Why aren’t we using Kleene +?

Example 3: lambda calculus and coordination

Lynx chases Rover or Kitty sleeps

1. `CONJ -> or`
   $\lambda P[\lambda Q[P \vee Q]]$
2. `S[conj=yes] -> CONJ S1[conj=no]`
   CONJ$'$(S1$'$)
   $\lambda P[\lambda Q[P \vee Q]](\text{sleep}'(k)) = \lambda Q[\text{sleep}'(k) \vee Q]$
3. `S[conj=no] -> S1[conj=no] S2[conj=yes]`
   S2$'$(S1$'$)
   $\lambda Q[\text{sleep}'(k) \vee Q](\text{chase}'(l, r)) = \text{sleep}'(k) \vee \text{chase}'(l, r)$

VP coordination

- sentential conjunctions are of the type $\langle t, \langle t, t\rangle\rangle$
- conjunctions can also combine VPs, so $\langle\langle e, t\rangle, \langle\langle e, t\rangle, \langle e, t\rangle\rangle\rangle$: conjunctions are of polymorphic type
- general schema for conjunctions is $\langle type, \langle type, type\rangle\rangle$.

VP conjunction rule uses the same lexical entries for *and* and *or* as sentential conjunction:

```
VP[conj=yes] -> CONJ VP1[conj=no]
```

$\lambda R[\lambda x[(\text{CONJ}'(R(x)))(\text{VP1}'(x))]]$

```
VP[conj=no] -> VP1[conj=no] VP2[conj=yes]
```

$\text{VP2}'(\text{VP1}')$

This looks complicated, but doesn't use any new formal devices.

Example 4

1. chases Rover
   $\lambda y[\text{chase}'(y, r)]$
2. `CONJ -> and`
   $\lambda P \lambda Q[P \wedge Q]$
3. and chases Rover
   `VP[conj=yes] -> CONJ VP1[conj=no]`
   $\lambda R \lambda x[(\text{CONJ}'(R(x)))(\text{VP1}'(x))]$ (grammar rule)
   $\lambda R \lambda x[(\lambda P[\lambda Q[P \wedge Q]](R(x)))(\lambda y[\text{chase}'(y, r)](x))]$
   (substituted CONJ and VP1)
   $= \lambda R \lambda x[(\lambda P[\lambda Q[P \wedge Q]](R(x)))(\text{chase}'(x, r))]$ (lambda y)
   $= \lambda R \lambda x[\lambda Q[R(x) \wedge Q](\text{chase}'(x, r))]$ (applied lambda P)
   $= \lambda R \lambda x[R(x) \wedge \text{chase}'(x, r)\ ]$ (applied lambda Q)

Example 4, continued

4 `Vintrans -> barks`

$\lambda z[\text{bark}'(z)]$

5 barks and chases Rover

```
VP[conj=no] ->
   VP1[conj=no] VP2[conj=yes]
```

$\text{VP2}'(\text{VP1}')$ (grammar rule)

$\lambda R \lambda x[R(x) \wedge \text{chase}'(x, r)](\lambda z[\text{bark}'(z)])$ (sub. VP1, VP2)

$= \lambda x[\lambda z[\text{bark}'(z)](x) \wedge \text{chase}'(x, r)]$ (applied lambda R)

$= \lambda x[\text{bark}'(x) \wedge \text{chase}'(x, r)]$ (applied lambda z)

6 Kitty barks and chases Rover

```
S -> NP VP
```

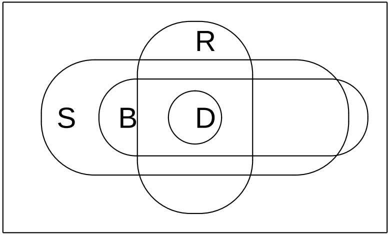
$\text{VP}'(\text{NP}')$

$\lambda x[\text{bark}'(x) \wedge \text{chase}'(x, r)](k)$

$= \text{bark}'(k) \wedge \text{chase}'(k, r)$

Denotation and type of quantifiers

every dog denotes the set of all sets of which dog′ is a subset. i.e., a function which takes a function from entities to truth values and returns a truth value.
For instance, *every dog* might denote the set $\{\text{bark}', \text{run}', \text{snore}'\}$:

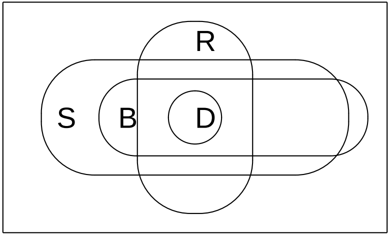


D= dog′, B= bark′, S= snore′, R= run′

What does *some dog* denote?

Denotation and type of quantifiers

every dog denotes the set of all sets of which dog′ is a subset. i.e., a function which takes a function from entities to truth values and returns a truth value.
For instance, *every dog* might denote the set $\{\text{bark}', \text{run}', \text{snore}'\}$:



D= dog′, B= bark′, S= snore′, R= run′

What does *some dog* denote?

Denotation and type of quantifiers, continued

The type of *every dog* is $\langle\langle e, t\rangle, t\rangle$ (its argument has to be of the same type as an intransitive verb).
every dog:

$$\lambda P[\forall x[\mathrm{dog}'(x) \implies P(x)]]$$

Semantically, *every dog* acts as a functor, with the intransitive verb as the argument:

$$\lambda P[\forall x[dog'(x) \implies P(x)]](\lambda y[sleep(y)])$$
$$= \forall x[dog'(x) \implies \lambda y[sleep(y)](x)]$$
$$= \forall x[dog'(x) \implies sleep(x)]$$

This is higher-order: we need higher-order logic to express the FOPC composition rules. Problem: *every dog* acts as a functor, *Kitty* doesn't, so different semantics for S -> NP VP, depending on whether NP is a proper name or quantified NP.

Denotation and type of quantifiers, continued

The type of *every dog* is $\langle\langle e, t\rangle, t\rangle$ (its argument has to be of the same type as an intransitive verb).
every dog:

$$\lambda P[\forall x[\text{dog}'(x) \implies P(x)]]$$

Semantically, *every dog* acts as a functor, with the intransitive verb as the argument:

$$\lambda P[\forall x[dog'(x) \implies P(x)]](\lambda y[sleep(y)])$$
$$= \forall x[dog'(x) \implies \lambda y[sleep(y)](x)]$$
$$= \forall x[dog'(x) \implies sleep(x)]$$

This is higher-order: we need higher-order logic to express the FOPC composition rules. Problem: *every dog* acts as a functor, *Kitty* doesn't, so different semantics for `S -> NP VP`, depending on whether NP is a proper name or quantified NP.

Type raising

Change the type of the proper name NP: instead of the simple expression of type e, we make it a function of type $\langle\langle e,t\rangle,t\rangle$
So instead of k we have $\lambda P[P(k)]$ for the semantics of *Kitty*.
But, what about transitive verbs? We've raised the type of NPs, so now transitive verbs won't work.
Type raise them too ...
chases:
$\lambda R[\lambda y[R(\lambda x[chase(y,x)])]]$

Executive Summary:

- this gets complicated,
- and *every cat chased some dog* only produces one scope!

Type raising

Change the type of the proper name NP: instead of the simple expression of type e, we make it a function of type $\langle\langle e, t\rangle, t\rangle$
So instead of k we have $\lambda P[P(k)]$ for the semantics of *Kitty*.
But, what about transitive verbs? We've raised the type of NPs, so now transitive verbs won't work.
Type raise them too . . .
chases:
$\lambda R[\lambda y[R(\lambda x[chase(y, x)])]]$

Executive Summary:

- this gets complicated,
- and *every cat chased some dog* only produces one scope!

Semantics in computational grammars

- Underspecified representations preferred
- Complexity of type raising should be avoided
- Integrated approach to syntax and semantics
- Composition using typed feature structures
- Preliminary step: event-based semantics

Next lecture

Lecture 4: Introduction to constraint-based semantics

- Events
- Semantics in typed feature structures
- Semantics in the lexicon
- Composition

Outline of Lecture 4

Why events?

(42) A dog barked loudly
$\exists x[\text{dog}'(x) \wedge \text{loud}'(\text{bark}'(x))]$

(43) A dog barked in a park
$\exists x \exists y[\text{dog}'(x) \wedge \text{park}'(y) \wedge \text{in}'(\text{bark}'(x), y)]$

Problematic because:

- Indefinite number of higher-order predicates
- *a loud bark* gets very different semantics from *bark loudly*
- Unwarranted ambiguity in scopes

Event semantics

(44) A dog barks
$\exists x \exists e[\text{dog}'(x) \wedge \text{bark}'(e, x))]$
i.e., There is a dog and there is an event of that dog barking

(45) A dog barks loudly
$\exists x \exists e[\text{dog}'(x) \wedge \text{loud}'(e) \wedge \text{bark}'(e, x))]$
i.e., There is a dog and there is an event of that dog barking and that event is loud.

(46) A dog barks in a park
$\exists x \exists y \exists e[\text{dog}'(x) \wedge \text{park}'(y) \wedge \text{bark}'(e, x) \wedge \text{in}'(e, y)]$
i.e., There is a dog and there is an event of that dog barking and that event is in a park.

Event semantics in general

reify events (i.e., make them into things)

- most nouns don’t denote physical objects anyway
- events are spatio-temporally located, so have some physical attributes

Scopal modifiers versus events

probably expresses something about the truth-conditions of a sentence and its semantics interacts with quantifiers.

$$\forall x[\text{dog}'(x) \implies \text{probably}'(\text{bark}'(e, x))]$$

means something different from:

$$\text{probably}'([\forall x[\text{dog}'(x) \implies \text{bark}'(e, x)]])$$

Example: Suppose probably′ means ‘with a probability of more than 0.5’, r and s are the only dogs in our model, $P(\text{bark}'(r)) = 0.6$, $P(\text{bark}'(s)) = 0.6$ and probabilities are independent.
$\forall x[\text{dog}'(x) \implies \text{probably}'(\text{bark}'(e, x))]$ is true
$\text{probably}'([\forall x[\text{dog}'(x) \implies \text{bark}'(e, x)]])$ is false

More examples

Notation — use of e is a convention to show the sort. The following are essentially equivalent:

(47) A dog barks

$\exists x \exists y[\text{dog}'(x) \wedge \text{event}'(y) \wedge \text{bark}'(y, x))]$

$\exists x \exists y_{ev}[\text{dog}'(x) \wedge \text{bark}'(y_{ev}, x))]$

$\exists x \exists e[\text{dog}'(x) \wedge \text{bark}'(e, x))]$

(48) A dog knows a cat

$\exists x \exists y \exists e[\text{dog}'(x) \wedge \text{cat}'(y) \wedge \text{know}'(e, x, y))]$

Semantics in typed feature structures: a simple grammar

- Practical session 4
- Event semantics
- Quantifier-free predicate calculus
  this dog will correspond to [this(c) $\wedge$ dog(c)]
  (underspecified quantifier scope in next lecture)
- Only connective is conjunction: represented implicitly
- Variant of semantics in Sag and Wasow (1999), modified slightly so underspecification works

Flat semantics

$$\begin{bmatrix} \textbf{syn-struc} \\ \text{ORTH } \textbf{*dlist*} \\ \text{HEAD } \textbf{pos} \\ \text{SPR } \textbf{*list*} \\ \text{COMPS } \textbf{*list*} \\ \text{SEM } \textbf{semantics} \\ \text{ARGS } \textbf{*list*} \end{bmatrix}$$

- **semantics** has two appropriate features, HOOK and RELS
- HOOK contains INDEX (more later)
- INDEX takes a **sement**
- INDEX is for composition (very very roughly like lambda variable) — it can be ignored in semantics of full sentences
- a **sement** has subtypes **object** and **event**
- RELS takes a difference list of elementary predications

Flat semantics, example

$$
\begin{bmatrix}
\textbf{semantics} \\
\text{HOOK.INDEX } \boxed{2}\ \textbf{event} \\
\text{RELS } <! \begin{bmatrix} \textbf{relation} \\ \text{PRED } \textbf{this\_rel} \\ \text{ARG0 } \boxed{4} \begin{bmatrix} \textbf{object} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \textbf{relation} \\ \text{PRED } \textbf{dog\_rel} \\ \text{ARG0 } \boxed{4} \end{bmatrix}, \begin{bmatrix} \textbf{arg1-relation} \\ \text{PRED } \textbf{bark\_rel} \\ \text{ARG0 } \boxed{2} \\ \text{ARG1 } \boxed{4} \end{bmatrix} !>
\end{bmatrix}
$$

$$[\text{this}(c) \wedge \text{dog}(c) \wedge \text{bark}(e, c)]$$

Elementary predications

- **relation** has the appropriate feature PRED: string value corresponding to the predicate symbol
- ARG0: event for verbs (e.g., *e* in bark(e, c)), argument for ordinary nouns (e.g., the *c* in dog(c))
- ARG1, ARG2 and ARG3, as required.
- equivalence of arguments is implemented by coindexation

$$\left[\begin{array}{l}\textbf{semantics}\\ \text{HOOK.INDEX } \boxed{e}\ \textbf{event}\\ \text{RELS } <! \left[\begin{array}{l}\textbf{relation}\\ \text{PRED } \textbf{this\_rel}\\ \text{ARG0 } \boxed{c}\left[\textbf{object}\right]\end{array}\right], \left[\begin{array}{l}\textbf{relation}\\ \text{PRED } \textbf{dog\_rel}\\ \text{ARG0 } \boxed{c}\end{array}\right], \left[\begin{array}{l}\textbf{arg1-relation}\\ \text{PRED } \textbf{bark\_rel}\\ \text{ARG0 } \boxed{e}\\ \text{ARG1 } \boxed{c}\end{array}\right] !>\end{array}\right]$$

$$[\text{this}(c) \wedge \text{dog}(c) \wedge \text{bark}(e, c)]$$

Semantics in the lexicon

```
dog := noun-lxm &
[ ORTH.LIST.FIRST "dog",
  SEM.RELS.LIST.FIRST.PRED "dog_rel" ].
```

- lexical entries are a triple consisting of orthography, semantic predicate symbol and lexical type (e.g., `"dog"`, `"dog_rel"` and **noun-lxm**)
- the lexical type (e.g., **noun-lxm**) encodes both syntax and a skeleton semantic structure
- for the practical, predicate values are string-valued, so they don't have to be explicitly declared as types
- one predicate per lexeme (simplifying assumption): *dog* and *dogs* will both be `"dog_rel"`
  full-scale grammars relate sg and pl forms of regular nouns by rule

Linking in lexical entries

```
[ trans-verb
  ORTH <! chase ! >
  HEAD [ verb
         MOD <> ]
  SPR < [ phrase
          HEAD noun
          SPR <>
          COMPS <>
          SEM [ semantics
                HOOK.INDEX [1] sement ] ] >
  COMPS < [ phrase
            HEAD noun
            SPR <>
            COMPS <>
            SEM [ semantics
                  HOOK.INDEX [2] sement ] ] >
  SEM [ semantics
        HOOK.INDEX [3] event
        RELS <! [ arg1-2-relation
                  PRED chase_rel
                  ARG0 [3]
                  ARG1 [1]
                  ARG2 [2] ] ! > ]
  ARGS *list* ]
```

Linking in lexical entries

- semantic argument positions are coindexed with the appropriate part of the syntax

In this approach:

- access to the semantics of a phrase is always via its HOOK slot
- RELS list is never accessed directly (only function is to accumulate list of EPs)

Composition 1

Schematically, three types of information in the semantics:

Accumulators — RELS. Implemented as difference lists, only for accumulating values, only operation during parsing is difference list append

Hooks — INDEX. Hooks give arguments for predicates, only way of accessing **parts** of the semantics of a sign. Lexically set up, pointers into RELS.

Slots e.g., SPR.SEM.HOOK.INDEX. Syntactic features which also specify how the semantics is combined. A syntax ‘slot’ will be coindexed with a hook in another sign.

Composition constraints

1. The RELS of the mother of the phrase is the difference list append of the RELS of the daughters.
2. One phrase has one or more syntactic slots (MOD, SPR or COMPS) filled by the other daughters.
   The phrase with the slot is the semantic head (not always same as syntactic head — e.g. modifier is semantic head in `head-modifier-phrase`)
   The semantic head coindexes its argument positions with the HOOKs of the other daughters.
3. The HOOK on the phrase as a whole is coindexed with the HOOK of the semantic head daughter.
4. Unsaturated slots are passed up to the mother.

chase the cat

```
[ binary-head-initial
  ORTH <! chase the cat ! >
  HEAD [1] verb
  SPR [2] < [ SEM.HOOK.INDEX [3] ] >
  COMPS < >
  SEM  [ HOOK.INDEX [4] event
         RELS <! [ PRED chase_rel
                   ARG0 [4]
                   ARG1 [3]
                   ARG2 [5] ] , [ PRED the_rel
                                  ARG0 [5] ] , [ PRED cat_rel
                                                 ARG0 [5] ] ! > ]
  ARGS < [ plur-verb
           HEAD [1]
           SPR [2]
           COMPS < [6] >
           SEM.HOOK.INDEX [4] ] , [6] [ binary-head-second
                                        SEM.HOOK.INDEX [5] ] > ]
```

$$\text{chase}'(e_4, x_3, y_5) \wedge \text{the}'(y_5) \wedge \text{cat}'(y_5)$$

Next lecture

Lecture 5: More on scope and quantifiers

FOPC ‘issues’.
An introduction to generalized quantifiers.
Scope ambiguity expressed with generalized quantifiers
LFs as trees
Underspecification as partial description of trees
Constraints on underspecified forms
Intersective modification and implicit conjunction

Outline of Lecture 5

Lecture 5: More on scope and quantifiers

- FOPC 'issues'.
- An introduction to generalized quantifiers.
- Scope ambiguity expressed with generalized quantifiers
- LFs as trees
- Underspecification as partial description of trees
- Constraints on underspecified forms
- Intersective modification and implicit conjunction

FOPC deficiencies and solutions

1. adverbial modification: events (last lecture)
2. scopal modification: higher order predicates
3. determiners other than *every* and *some*: generalised quantifiers
4. multiple representations for different quantifier scopes (and no syntactic ambiguity): underspecified representations

At end of Lecture 5, new representation: underspecified predicate calculus with generalised quantifiers and (limited) higher-order scopal modifiers.
Lecture 6: compositional semantics with this representation using typed feature structure grammars.

Scopal modification

FOPC works for some types of modification:

every big dog barks

$\forall x[\text{big}'(x) \wedge \text{dog}'(x) \implies \text{bark}'(e,x)]$

every dog barks loudly

$\forall x[\text{dog}'(x) \implies \text{bark}'(e,x) \wedge \text{loud}'(e)]$

But: non-first-order predicates:

kitty probably sleeps

$$\text{probably}'(\text{sleep}'(k))$$

L believes it is not the case that K sleeps

L believes K doesn’t sleep

$$\text{believe}'(l, \text{not}'(\text{sleep}'(k)))$$

Determiners other than *every* and *some*

some A is a B

$$\exists x[A(x) \wedge B(x)]$$

every A is a B

$$\forall x[A(x) \implies B(x)]$$

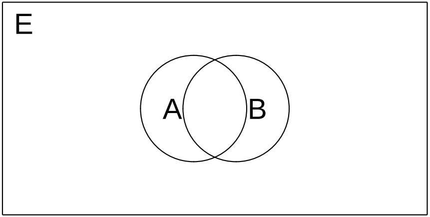
two As are Bs

$$\exists x[\exists y[x \neq y \wedge A(x) \wedge A(y) \wedge B(x) \wedge B(y)]]$$

most As are Bs
?

An introduction to generalized quantifiers.

Generalized quantifiers involve the relationship between two sets of individuals, A and B, within a domain of discourse E.



$D_E AB$.

True quantifiers depend only on the cardinality of the sets A and $A \cap B$ (i.e., $|A|$ and $|A \cap B|$).

every: $|A| = |A \cap B|$

some: $|A \cap B| \geq 1$

at least two: $|A \cap B| \geq 2$

most (interpreted as *more than half*): $|A \cap B| > |A|/2$

Terminology and notation

A: *restriction* of the quantifier
B: *body* (or *scope*)
every white cat likes Kim
white cat: restriction of the quantifier
likes Kim: body
Notation: quantifier(bound-variable,restriction,body)
every white cat likes Kim
$\text{every}'(x, \text{white}'(x) \wedge \text{cat}'(x), \text{like}'(x, k))$

Ambiguity

(52) most white cats like some squeaky toys

$$most'(x, white'(x) \wedge cat'(x), some'(y, toy'(y) \wedge squeaky'(y), like'(x, y))$$

(preferred reading)

$$some'(y, toy'(y) \wedge squeaky'(y), most'(x, white'(x) \wedge cat'(x), like'(x, y))$$

(dispreferred reading)

Ambiguity

(53) most mothers of two white cats like Kim

$$\text{most}'(x, \text{two}'(y, \text{white}'(y) \wedge \text{cat}'(y), \text{mother}'(x, y)), \text{like}'(x, \text{Kim}'))$$

(preferred reading)

$$\text{two}'(y, \text{white}'(y) \wedge \text{cat}'(y), \text{most}'(x, \text{mother}'(x, y), \text{like}'(x, \text{Kim}')))$$

Scope ambiguity

(58) Every person in the room speaks two languages.

every person is bilingual OR two languages every person shares

every dog did not sleep

every dog was awake OR some dog was awake

All the people who were polled by our researchers thought that every politician lies to some journalists in at least some interviews.

Number of readings is (roughly) 120 (5!)
Underspecification of quantifier scope allows us to avoid an unmotivated ambiguity in tree structures, while preserving the possibility of representing scope distinctions.

Underspecification and Sudoku solving

			7					8
		9					2	
	5			3			9	
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Solving.

			7					8
		9					2	
	5			3			9	
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Possibility 1.

			7					8
		9					2	7
	5			3			9	
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Possibility 2.

			7					8
		9					2	
	5			3			9	7
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Underspecification.

			7					8
		9					2	7
	5			3			9	7
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Inference on underspecified form.

			7					8
		9					2	7
	5			3			9	7
8					2			
		6				7		
			4					1
	3			9			6	
	2					4		
7					1			

Inference on underspecified form.

			7					8
		9					2	7
	5			3			9	7
8					2			
		6				7		
			4					1
	3			9			6	
	2					4	7	
7					1			

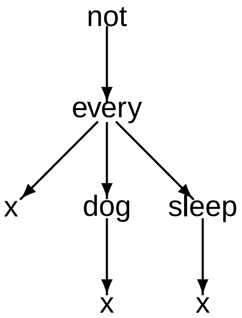
LFs as trees

- Every conventional logical formula can be represented as a tree (syntactically).

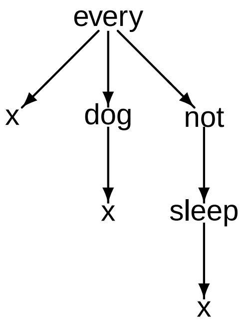
  (63) every dog did not sleep
  (64) not(every(x,dog(x),sleep(x)))
  (65) every(x,dog(x),not(sleep(x)))

- nodes correspond to predicates and variables
- branches correspond to predicate argument relationships
- trees for different scopes normally share some part of their structure.

LFs as trees



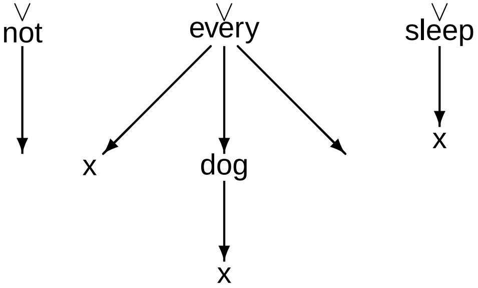
not(every(x,dog(x),sleep(x)))



every(x,dog(x),not(sleep(x)))

Underspecification as partial description of trees

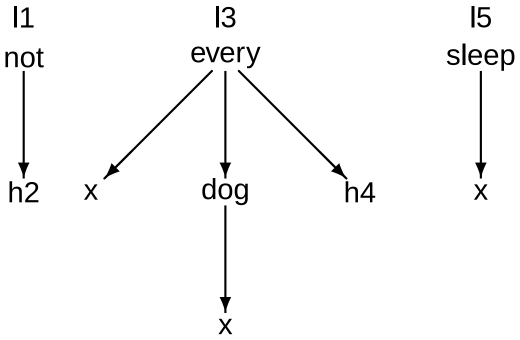
One structure captures the commonalities between scopes and can be specialized to produce exactly the required scopes.



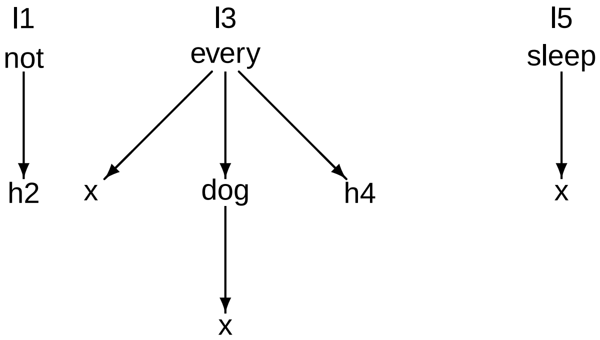
- arrows pointing to nothing — arguments are missing
- ∨ on upper node — structure can fill argument position
- Exactly two ways the pieces can be completely recombined to give trees as before.

Holes and labels

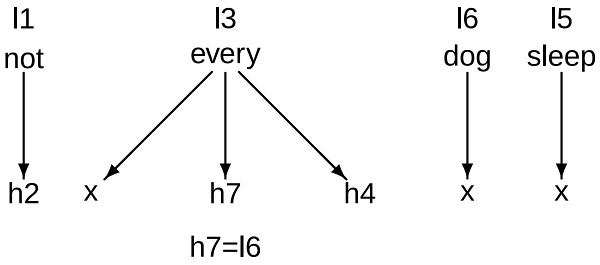
- Distinguish the different arguments and fragments (to make it easier to manipulate)
- argument position identifier is a *hole*
- fragment identifier is a *label*



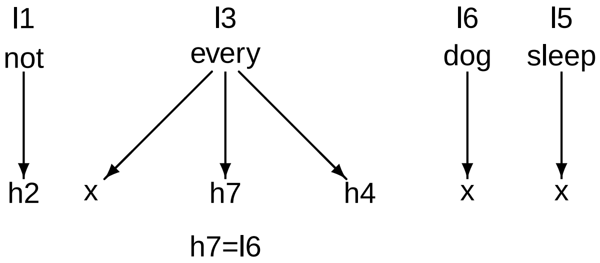
Elementary predications



Elementary predications

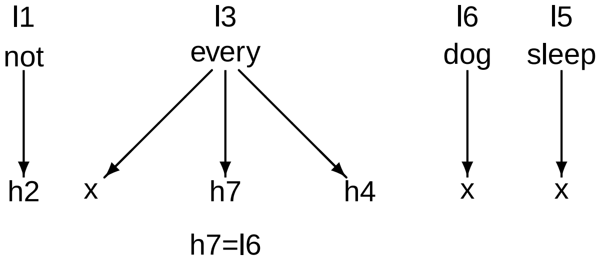


Elementary predications



- Underspecified representation is broken up into elementary predications (EPs): i.e., combinations of a predicate with its arguments.
- Each EP has one label, one predicate and one or more arguments.

Linear notation



l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

Underspecification specialisation

To reconstruct the scoped structures, equate holes and labels (like putting the trees back together).
Two valid possible sets of equations:

(66) l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6, h4=l1, h2=l5
every(x,dog(x),not(sleep(x)))
top label is l3

(67) l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6, h4=l5, h2=l3
not(every(x,dog(x),sleep(x)))
top label is l1

Full scoping

In a fully scoped structure:

- every hole is filled by a label.
- every label apart from one is equated with a hole
- the unique label which isn't the value of a hole is the top of the tree: i.e., the outermost thing in the scoped structure

Order of the elementary predications and the name of the variables are not significant:
l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6
l6:dog(x), l5:sleep(x), l1:not(h2), l3:every(x,h7,h4), h7=l6
l0:dog(x), l11:not(h2), l3:every(x,h7,h4), h7=l0, l5:sleep(x)

Full scoping

In a fully scoped structure:

- every hole is filled by a label.
- every label apart from one is equated with a hole
- the unique label which isn't the value of a hole is the top of the tree: i.e., the outermost thing in the scoped structure

Order of the elementary predications and the name of the variables are not significant:

l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

l6:dog(x), l5:sleep(x), l1:not(h2), l3:every(x,h7,h4), h7=l6

l0:dog(x), l11:not(h2), l3:every(x,h7,h4), h7=l0, l5:sleep(x)

Full scoping

In a fully scoped structure:

- every hole is filled by a label.
- every label apart from one is equated with a hole
- the unique label which isn't the value of a hole is the top of the tree: i.e., the outermost thing in the scoped structure

Order of the elementary predications and the name of the variables are not significant:

l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

l6:dog(x), l5:sleep(x), l1:not(h2), l3:every(x,h7,h4), h7=l6

l0:dog(x), l11:not(h2), l3:every(x,h7,h4), h7=l0, l5:sleep(x)

Full scoping

In a fully scoped structure:

- every hole is filled by a label.
- every label apart from one is equated with a hole
- the unique label which isn't the value of a hole is the top of the tree: i.e., the outermost thing in the scoped structure

Order of the elementary predications and the name of the variables are not significant:
l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6
l6:dog(x), l5:sleep(x), l1:not(h2), l3:every(x,h7,h4), h7=l6
l0:dog(x), l11:not(h2), l3:every(x,h7,h4), h7=l0, l5:sleep(x)

Constraints on underspecified forms

Implicit constraints on how the EPs can be combined:

1. all scoped structures must be singly rooted trees (therefore, no cycles etc)
2. variables must be bound by a quantifier

Constraints on underspecified forms

Explicit constraints are needed for more complicated examples.

(68) every nephew of a dragon snores

(69) every(x, a(y, dragon(y), nephew(x,y)) snore(x))
i.e., the arbitrary dragon reading

(70) a(y, dragon(y), every(x, nephew(x,y), snore(x)))
i.e., the specific dragon reading

Underspecification?:
l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h6=l8

Need for more constraints

l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h6=l8
has invalid solutions besides the valid ones in 69 and 70:

(72) l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h2=l9, h3=l5, h6=l8, h7=l4
every(x,snore(x),a(y,dragon(y),nephew(x,y)))
(which means roughly — every snorer is the nephew of a dragon)

(73) l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h2=l9, h3=l5, h6=l8, h7=l4
a(y,dragon(y),every(x,snore(x),nephew(x,y)))

Problem is that the verb has been able to instantiate the restriction of *every*, which should be restricted to the corresponding Nbar.

qeq constraints

$=_q$ (*qeq*) constraints (equality modulo quantifiers).
If a hole h is $=_q$ a label l, then one of the following must be true:

- $h = l$
- there is an intervening quantifier, quant, such that quant has a label l' where $l' = h$ and the body of quant is h' (i.e., quant(var,h_r,h')) and $h' = l$
- there is a chain of such intervening quantifiers, all linked via their bodies.

qeq constraints

Revised example:

(74) every nephew of a dragon snores

(75) l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h6=l8, h2 $=_q$ l4

In general, every quantifier corresponding to a determiner will have a restrictor hole which is qeq the top label of its Nbar.

(77) l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h6 $=_q$ l8, h2 $=_q$ l4

Intersective modification and implicit conjunction

- Assume *young black cat* can be represented in conventional logic as young(x) ∧ black(x) ∧ cat(x).
- maybe: h1:and(h2,h3), h2:young(x), h3:and(h4,h5), h4:black(x), h5:cat(x)
- or: h1:and(h2,h3), h2:and(h4,h5), h4:young(x), h4:black(x), h5:cat(x)
- Equivalent logical forms but syntactically very different.
- so maybe: h1:and(h2,h3,h4), h2:young(x), h3:black(x), h4:cat(x)
- But no real need for the explicit ‘and’, so:
  h1:young(x), h1:black(x), h1:cat(x)
  Equal labels indicate implicit conjunction.
  Use the predicate ‘and’ for the actual lexeme *and*
- This corresponds to the typed feature structure representation we want to use . . .

Intersective modification and implicit conjunction

- Assume *young black cat* can be represented in conventional logic as young(x) $\wedge$ black(x) $\wedge$ cat(x).
- maybe: h1:and(h2,h3), h2:young(x), h3:and(h4,h5), h4:black(x), h5:cat(x)
- or: h1:and(h2,h3), h2:and(h4,h5), h4:young(x), h4:black(x), h5:cat(x)
- Equivalent logical forms but syntactically very different.
- so maybe: h1:and(h2,h3,h4), h2:young(x), h3:black(x), h4:cat(x)
- But no real need for the explicit 'and', so: h1:young(x), h1:black(x), h1:cat(x)
  Equal labels indicate implicit conjunction.
  Use the predicate 'and' for the actual lexeme *and*
- This corresponds to the typed feature structure representation we want to use . . .

Intersective modification and implicit conjunction

- Assume *young black cat* can be represented in conventional logic as young(x) $\wedge$ black(x) $\wedge$ cat(x).
- maybe: h1:and(h2,h3), h2:young(x), h3:and(h4,h5), h4:black(x), h5:cat(x)
- or: h1:and(h2,h3), h2:and(h4,h5), h4:young(x), h4:black(x), h5:cat(x)
- Equivalent logical forms but syntactically very different.
- so maybe: h1:and(h2,h3,h4), h2:young(x), h3:black(x), h4:cat(x)
- But no real need for the explicit 'and', so:
  h1:young(x), h1:black(x), h1:cat(x)
  Equal labels indicate implicit conjunction.
  Use the predicate 'and' for the actual lexeme *and*
- This corresponds to the typed feature structure representation we want to use . . .

Next lecture

Lecture 6: Building underspecified representations

- MRS in TFSs
- Semantic composition in constraint-based grammars
- Composition rules for phrases

Outline of Lecture 6

Objectives

1. Develop composition principles for underspecified representations
2. Extend TFS grammars to allow scope (with underspecified quantifiers)

Expressing MRS in TFS:

- labels and holes in the structures have to be unifiable (i.e., of the same type): *handles*
- distinguish between RELS, for the elementary predications and HCONS (handle constraints: qeqs)
- Every EP has a LBL (MRS label)
- Quantifiers have features ARG0, RSTR and BODY (for bound variable, restriction and body)

An example MRS in TFSs

$$
\begin{bmatrix}
\textbf{semantics} \\
\text{RELS} < \begin{bmatrix} \text{PRED } \textbf{every\_rel} \\ \text{LBL } \boxed{2}\ \textbf{handle} \\ \text{ARG0 } \boxed{3}\ \textbf{ref-ind} \\ \text{RSTR } \boxed{4}\ \textbf{handle} \\ \text{BODY } \textbf{handle} \end{bmatrix}, \begin{bmatrix} \text{PRED } \textbf{dog\_rel} \\ \text{LBL } \boxed{6}\ \textbf{handle} \\ \text{ARG0 } \boxed{3} \end{bmatrix}, \begin{bmatrix} \text{PRED } \textbf{not\_rel} \\ \text{LBL } \boxed{7}\ \textbf{handle} \\ \text{ARG0 } \boxed{8} \end{bmatrix}, \begin{bmatrix} \text{PRED } \textbf{sleep\_rel} \\ \text{LBL } \boxed{9} \\ \text{ARG0 } \textbf{event} \\ \text{ARG1 } \boxed{3} \end{bmatrix} > \\
\text{HCONS} < \begin{bmatrix} \textbf{qeq} \\ \text{SC-ARG } \boxed{4} \\ \text{OUTSCPD } \boxed{6} \end{bmatrix}, \begin{bmatrix} \textbf{qeq} \\ \text{SC-ARG } \boxed{8} \\ \text{OUTSCPD } \boxed{9} \end{bmatrix} >
\end{bmatrix}
$$

$$\{h2: \text{every}(x, h4, h5), h6: \text{dog}(x), h7: \text{not}(h8), h9: \text{sleep}(e, x)\}$$
$$\{h4 =_q h6, h8 =_q h9\}$$

l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

Semantic composition

Accumulators — RELS (as before) and HCONS (qeqs). Both implemented with difference-list append.

Hooks — INDEX (as before) and LTOP. LTOP is the handle of the EP with highest scope in the phrase. Scopal EPs (e.g., *probably*, *believe* and *not*) have an argument hole which is qeq the LTOP of the phrase they combine with. Also RSTR of quantifiers.

Slots as before

Scopal relationships

- All EPs have LBL features which correspond to their label.
- LTOP is the label of the EP in an MRS which has highest scope, except that the labels of quantifiers are not equated with the LTOP, so they can float
- All scopal relationships are stated via qeqs: qeqs are accumulated in HCONS

Notation: e.g., *probably*

$[h_p, e]$	$[h_v, e]_{mod}$	$[h_p : \text{probably}(h_u)]$	$\{h_u =_q h_v\}$
hook	syntax slot	EP	qeqs

```
[ scopal-adverb
  HEAD [ MOD < [ phrase
                 HEAD verb
                 SPR <syn-struc>
                 COMPS < >
                 SEM [ semantics
                       HOOK [ LTOP  [hv]
                              INDEX [e] ] ] ] > ]
  SEM [ semantics
        HOOK [ LTOP  [hp]
               INDEX [e] event ]
        RELS <! [ PRED probably_rel
                  LBL  [hp]
                  ARG0 [hu] ] !>
        HCONS <! [ qeq
                   SC-ARG   [hu]
                   OUTSCPD  [hv] ] !> ] ]
```

Rules for linking EPs to hooks in the lexicon

Each lexical item has a single *key* EP

1. If the key is non-scopal or fixed scopal EP (i.e., not a quantifier), LTOP of the MRS is equal to LBL of key.
   e.g., h_d is the LTOP of *dog* and h_p is the LTOP of *probably*:

   dog:
   $[h_d, x] \quad [h_d : \text{dog}(x)] \quad \{\}$
   probably:
   $[h_p, e] \quad [h_l, e]_{mod} \quad [h_p : \text{probably}(h_u)] \quad \{\}$

2. If the key is a quantifier EP, LTOP is not related to a handle.

   every:
   $[h_f, x_v] \quad [h_n, x_v]_{spec} \quad [h_v : \text{every}(x_v, h_r, h_b)] \quad \{\}$

   This allows the quantifier to float.

Semantic head and SPEC feature

- The determiner has to be the semantic head to get the correct semantic effect.
- SPEC is a syntactic feature that allows the determiner to select for the noun.
- The noun still syntactically selects for the determiner via SPR, but for the semantic rules we ignore SPR in this case.
- There are other ways one could do this, but use this in the practical.

General rules for phrases

1. The RELS of the mother is constructed by appending the RELS of the daughters.
2. The HCONS of daughters are all preserved (may be added to, see below).
3. One slot of the semantic head is equated with the hook in the other daughter (where the semantic head and the particular slot involved are determined by the grammar rule).
4. The hook features of the mother are the hook features of the semantic head.
5. Unsaturated slots are passed up to the mother.

Rules for combining hooks and slots, 1

1. Intersective combination. The LTOP of the daughters are equated with each other and with the LTOP of the phrase.

white:

$[h_w, x_w]$ $[h_w, x_w]_{mod}$ $[h_w : \text{white}(x_w)]$ $\{\}$

cat:

$[h_c, x_c]$ $[h_c : \text{cat}(x_c)]$ $\{\}$

white cat:

$[h_w, x_w]$ $[h_w : \text{white}(x_w), h_w : \text{cat}(x_w)]$ $\{\}$

Rules for combining hooks and slots, 2

2. Scopal combination (i.e., one daughter, always the semantic head, contains a scopal EP which scopes over the other daughter). The handle-taking argument of the scopal EP is qeq the LTOP of the scoped-over phrase.

sleep:

$[h_s, e_s]$ $[h_z, x_z]_{spr}$ $[h_s : \text{sleep}(e_s, x_z)]$ $\{\}$

probably:

$[h_p, e]$ $[h_l, e]_{mod}$ $[h_p : \text{probably}(h_u)]$ $\{\}$

probably sleeps:

$[h_p, e_s]$ $[h_z, x_z]_{spr}$ $[h_p : \text{prob}(h_u),\ h_s : \text{sleep}(e_s, x_z)]$ $\{h_u =_q h_s\}$

Rules for combining hooks and slots, 3

3. Quantifiers. The restriction of the quantifier is scopal, as above, and the body is left unconstrained.

every:

$[h_f, x_v] \quad [h_n, x_v]_{spec} \quad [h_v : \text{every}(x_v, h_r, h_b)] \quad \{\}$

dog:

$[h_d, x_d] \quad [h_d : \text{dog}(x_d)] \quad \{\}$

every dog:

$[h_f, x_v] \quad [h_v : \text{every}(x_v, h_r, h_b),$
$h_d : \text{dog}(x_v)] \quad \{h_r =_q h_d\}$

Composition shown with feature structures

```
[ mrs
  LTOP [7]
  INDEX [10]
  RELS < [ PRED probably_rel   , [ PRED sleep_rel  >
           HNDL [7]                HNDL [9]
           ARG0 [8] ]              EVENT [10]
                                   ARG1 [3] ]
  HCONS < [ qeq
            SC-ARG [8]
            OUTSCPD [9] ] > ]

[ mrs                              [ mrs
  LTOP [7]                           LTOP [9]
  INDEX [10]                         INDEX [10]
  RELS < [ PRED probably_rel  >      RELS < [ PRED sleep_rel  >
           HNDL [7]                           HNDL [9]
           ARG0 [8] ]                         EVENT [10]
  HCONS <> ]                                  ARG1 [3] ]
                                     HCONS <> ]
```

Composition shown with feature structures

```
[ mrs
  LTOP [7]
  INDEX [3]
  RELS < [ PRED every_rel  ,  [ PRED dog_rel  >
           HNDL [2]              HNDL [6]
           BV [3]                ARG0 [3] ]
           RSTR [4]
           BODY handle ]
  HCONS < [ qeq
            SC-ARG [4]
            OUTSCPD [6] ] > ]

[ mrs                                  [ mrs
  LTOP [7]                               LTOP [6]
  INDEX [3]                              INDEX [3]
  RELS < [ PRED every_rel  >             RELS < [ PRED dog_rel  >
           HNDL [2]                               HNDL [6]
           BV [3]                                 ARG0 [3] ]
           RSTR [4]                      HCONS <> ]
           BODY handle ]
  HCONS <> ]
```

Composition shown with feature structures

```
[ mrs
  LTOP [7]
  INDEX [10]
  RELS < [ PR evy_r      , [ PR dog_r  , [ PR pbly_r , [ PR slp_r     >
           HNDL [2]          HNDL [6]      HNDL [7]      HNDL [9]
           BV [3]            ARG0 [3] ]    ARG0 [8] ]    EVENT [10]
           RSTR [4]                                      ARG1 [3] ]
           BY handle ]
  HCONS < [ qeq            , [ qeq            >
            SC-ARG [4]         SC-ARG [8]
            OUTSCPD [6] ]      OUTSCPD [9] ]
]

[ mrs                                   [ mrs
  LTOP [7]                                LTOP [7]
  INDEX [3]                               INDEX [10]
  RS < [ PR evy_r  , [ PR dog_r  >        RS < [ PR pbly_r , [ PR slp_r    >
         HNDL [2]      HNDL [6]                  HNDL [7]      HNDL [9]
         BV [3]        ARG0 [3] ]                ARG0 [8] ]    EVENT [10]
         RSTR [4]                                              ARG1 [3] ]
         BY hnd ]
  HCONS < [ qeq          >                HCONS < [ qeq          >
            SC-ARG [4]                              SC-ARG [8]
            OUTSCPD [6] ]                           OUTSCPD [9] ]
]                                       ]
```

Lambda calculus vs CBG semantics

Final representations may be equivalent, what differs is composition

Base entries:

cat	$\lambda x[\text{cat}'(x)]$
big	$\lambda P \lambda y[\text{big}'(y) \wedge P(y)]$

The predicates big′ and cat′ are the same but the adjective *big* has to act as a semantic functor, hence the $P(y)$.

In MRS:

cat

$$\left[\text{SEM} \left[\begin{array}{l} \text{HOOK.INDEX } \boxed{3} \\ \text{RELS} <! \left[\begin{array}{l} \text{PRED } \textbf{cat\_rel} \\ \text{ARG0 } \boxed{3} \end{array} \right] !> \end{array} \right] \right]$$

big

$$\left[\begin{array}{l} \text{MOD} < \left[\text{SEM} \left[\text{HOOK.INDEX } \boxed{1} \right] \right] > \\ \text{SEM} \left[\begin{array}{l} \text{HOOK.INDEX } \boxed{1} \\ \text{RELS} <! \left[\begin{array}{l} \text{PRED } \textbf{big\_rel} \\ \text{ARG0 } \boxed{1} \end{array} \right] !> \end{array} \right] \end{array} \right]$$

SEM part is equivalent, except that semantic functor has a connection to the syntactic slot.

Lambda calculus vs CBG semantics

Final representations may be equivalent, what differs is composition

Base entries:

cat $\lambda x[\text{cat}'(x)]$

big $\lambda P \lambda y[\text{big}'(y) \wedge P(y)]$

The predicates big′ and cat′ are the same but the adjective *big* has to act as a semantic functor, hence the $P(y)$.

In MRS:

cat $\left[\text{SEM} \left[\begin{array}{l} \text{HOOK.INDEX } \boxed{3} \\ \text{RELS} <! \left[\begin{array}{l} \text{PRED } \textbf{cat\_rel} \\ \text{ARG0 } \boxed{3} \end{array} \right] !> \end{array} \right] \right]$

big $\left[\begin{array}{l} \text{MOD} < \left[\text{SEM} \left[\text{HOOK.INDEX } \boxed{1} \right] \right] > \\ \text{SEM} \left[\begin{array}{l} \text{HOOK.INDEX } \boxed{1} \\ \text{RELS} <! \left[\begin{array}{l} \text{PRED } \textbf{big\_rel} \\ \text{ARG0 } \boxed{1} \end{array} \right] !> \end{array} \right] \end{array} \right]$

SEM part is equivalent, except that semantic functor has a connection to the syntactic slot.

Lambda calculus vs CBG semantics: rules

```
N1 -> Adj N2
```

$Adj'(N2')$

λP in *big* needed for this to work, $\wedge P(y)$ is required to get the semantics for the noun in the result.

head-modifier-rule specifies that the MOD of the modifier is the value of the (syntactic) head.

```
head-modifier-rule := binary-phrase &
[ SEM.HOOK #hook,
  ARGS < #mod,
         [ HEAD.MOD < #mod >,
           SEM.HOOK #hook ] > ].
```

Function application is not directly reflected in SEM.
General principle of concatenation of RELS instead of $\wedge P(y)$

Lambda calculus vs CBG semantics: rules

```
N1 -> Adj N2
```
$Adj'(N2')$

λP in *big* needed for this to work, $\wedge P(y)$ is required to get the semantics for the noun in the result.
head-modifier-rule specifies that the MOD of the modifier is the value of the (syntactic) head.

```
head-modifier-rule := binary-phrase &
[ SEM.HOOK #hook,
  ARGS < #mod,
         [ HEAD.MOD < #mod >,
           SEM.HOOK #hook ] > ].
```

Function application is not directly reflected in SEM.
General principle of concatenation of RELS instead of $\wedge P(y)$

Outline of Lecture 7

- Deep processing: big hand-built grammars.
  Good things:
  - Can produce detailed semantics
  - Bidirectional: generate and parse

  Bad things:
  - Relatively slow (around 30 words per second)
  - Lexical requirements, robustness
  - Parse selection
- Shallow and intermediate processing: e.g., POS taggers, noun phrase chunkers, RASP.
  Good things:
  - Faster (POS taggers: 10,000 w/sec; RASP: 100 w/sec)
  - More robust, less resource needed
  - Integrated parse ranking

  Bad things:
  - No conventional semantics
  - Not precise, no generation

Semantic representation: MRS

The mixture was allowed to warm to room temperature.

⟨ l3:_the_q(x5,h6,h4), l7:_mixture_n(x5),
l9:_allow_v_1(e2,u11,x5,h10), l13:_warm_v_1(e14,x5),
l13:_to_p(e15,e14,x16), l17:udef_q(x16,h18,h19),
l20:compound(e22,x16,x21), l23:udef_q(x21,h24,h25),
l26:_room_n(x21), l20:_temperature_n(x16) ⟩
⟨ qeq(h6,l7), qeq(h18,l20), qeq(h24,l26), qeq(h10,l13) ⟩

DELPH-IN MRS: main features

- Flat: list of EPs (each with label), list of qeqs.
- Underspecified quantifier scope: labels and holes, linked with qeqs.
- Conjunction from modification etc indicated by shared labels: l13:_warm_v_1(e14,x5), l13:_to_p(e15,e14,x16)
- Lexical predicates (leading underscore): lexeme, coarse sense (POS), fine sense.
- Construction predicates (e.g., compound).
- Sorted variables: tense, etc (and simple information structure).

Semantic representation: RMRS

The mixture was allowed to warm to room temperature.

⟨ l3:a1:_the_q(x5), l7:a2:_mixture_n(x5), l9:a3:_allow_v_1(e2), l13:a5:_warm_v_1(e14), l13:a6:_to_p(e15), l17:a7:udef_q(x16), l20:a8:compound(e22), l23:a9:udef_q(x21), l26:a10:_room_n(x21), l20:a11:_temperature_n(x16)⟩

⟨ a1:RSTR(h6), a1:BODY(h4), a3:ARG2(x5), a3:ARG3(h10), a5:ARG1(x5), a6:ARG1(e14), a6:ARG2(x16), a7:RSTR(h18), a7:BODY(h19), a8:ARG1(x16), a8:ARG2(x21), a9:RSTR(h24), a9:BODY(h25) ⟩

⟨ qeq(h6,l7), qeq(h18,l20), qeq(h24,l26), qeq(h10,l13) ⟩

MRS vs RMRS

- l9:_allow_v_1(e2,u11,x5,h10) in MRS
  l9:a3:_allow_v_1(e2), a3:ARG2(x5), a3:ARG3(h10) in RMRS.
- Further factorization: separation of arguments.
- All EPs have an anchor which relates args to EPs.
- RMRS can omit or underspecify ARGs: robust to missing lexical information.

Character positions

The mixture was allowed to warm to room temperature.

⟨ l3:a1:_the_q(x5)$_{\langle 0,3\rangle}$, l7:a2:_mixture_n(x5)$_{\langle 4,11\rangle}$,
l9:a3:_allow_v_1(e2)$_{\langle 16,23\rangle}$, l13:a5:_warm_v_1(e14)$_{\langle 27,31\rangle}$,
l13:a6:_to_p(e15)$_{\langle 32,34\rangle}$, l17:a7:udef_q(x16)$_{\langle 35,52\rangle}$,
l20:a8:compound(e22)$_{\langle 35,52\rangle}$, l23:a9:udef_q(x21)$_{\langle 35,52\rangle}$,
l26:a10:_room_n(x21)$_{\langle 35,39\rangle}$, l20:a11:_temperature_n(x16)$_{\langle 40,52\rangle}$⟩
⟨ a1:RSTR(h6), a1:BODY(h4), a3:ARG2(x5), a3:ARG3(h10),
a5:ARG1(x5), a6:ARG1(e14), a6:ARG2(x16), a7:RSTR(h18),
a7:BODY(h19), a8:ARG1(x16), a8:ARG2(x21), a9:RSTR(h24),
a9:BODY(h25) ⟩
⟨ qeq(h6,l7), qeq(h18,l20), qeq(h24,l26), qeq(h10,l13) ⟩

RMRS from POS tagger

The mixture was allowed to warm to room temperature.

⟨ l1:a2:_the_q(x3), l4:a5:_mixture_n(x6), l7:a8:_allow_v(e9), l10:a11:_warm_v(e12), l13:a14:_to_p(e15), l16:a17:_room_n(x18), l19:a20:_temperature_n(x21)⟩
⟨⟩
⟨⟩

All variables distinct, no ARGs, no qeqs.

Chunker: equate nominal indices, etc.

RMRS from POS tagger

The mixture was allowed to warm to room temperature.

⟨ l1:a2:_the_q(x3), l4:a5:_mixture_n(x6), l7:a8:_allow_v(e9), l10:a11:_warm_v(e12), l13:a14:_to_p(e15), l16:a17:_room_n(x18), l19:a20:_temperature_n(x21)⟩
⟨⟩
⟨⟩

All variables distinct, no ARGs, no qeqs.
Chunker: equate nominal indices, etc.

RMRS as semantic annotation of lexeme sequence.

- Annotate most lexemes with unique label, anchor, arg0. Note: null semantics for some words, e.g., infinitival *to*.
- Partially disambiguate lexeme with n, v, q, p etc.
- Add sortal information to arg0.
- Implicit conjunction: add equalities between labels.
- Ordinary arguments: add ARGs (possibly underspecified) between anchors and arg0.
- Scopal arguments: add ARG plus qeq between anchors and labels.

Standoff annotation on original text via character positions.

RMRS Elementary Predication

An RMRS EP contains:

1. the label of the EP: this is shared by other EPS to indicate implicit conjunction.
2. an anchor, not shared by any other EPS.
3. a relation
4. up to one argument of the relation (the arg0)

This is written as label:anchor:relation(arg0).
l13:a5:_warm_v_1(e14)
l13:a6:_to_p(e15)

RMRS ARGs

An RMRS ARG relation contains:

1. an anchor, which must also be the anchor of an EP.
2. an ARG relation, taken from a fixed set (here: ARG1, ARG2, ARG3, RSTR, BODY, plus the underspecified relations: ARG1-2, ARG1-3, ARG1-2, ARG2-3, ARGN).
3. exactly one argument. This must be 'grounded' by an EP: i.e., if it is a normal variable it must be the ARG0 of an EP, or if it is a hole, it must be related to the label of an EP by a qeq constraint.

a5:ARG1(x5), l13:a5:_warm_v_1(e14), l7:a2:_mixture_n(x5)

RMRS Matching

lb1:every_q(x),	lb1:every_q(x),
lb1:RSTR(h9),	lb1:RSTR(h9),
lb1:BODY(h6),	lb1:BODY(h6),
lb2:cat_n(x),	lb2:cat_n(x),
lb4:some_q(y),	lb4:some_q(y),
lb1:RSTR(h8),	lb1:RSTR(h8),
lb1:BODY(h7),	lb1:BODY(h7),
lb5:dog_n_1(y),	lb5:dog_n_1(y),
lb3:chase_v(e),	lb3:chase_v(e),
lb3:ARG1(x),	lb3:ARG1(x),
lb3:ARG2(y)	lb3:ARG2(y)

RMRS Matching

lb1:every_q(x),
lb1:RSTR(h9),
lb1:BODY(h6),
lb2:cat_n(x),
lb4:some_q(y),
lb1:RSTR(h8),
lb1:BODY(h7),
lb5:dog_n_1(y),
lb3:chase_v(e),
lb3:ARG1(x),
lb3:ARG2(y)

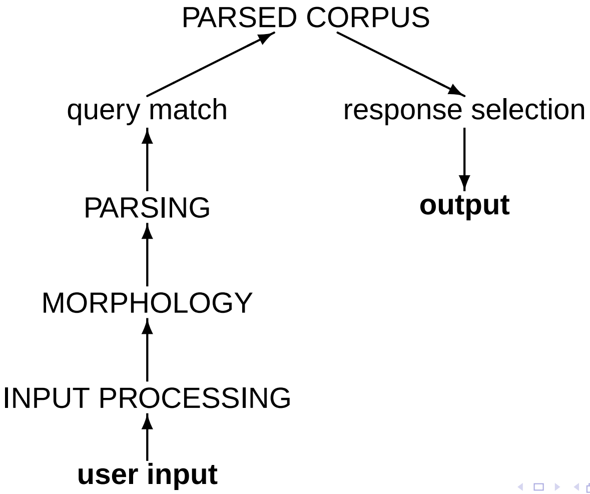
lb1:every_q(x),
lb1:RSTR(h9),
lb1:BODY(h6),
lb2:cat_n(x),
lb4:some_q(y),
lb1:RSTR(h8),
lb1:BODY(h7),
lb5:dog_n(y),
lb3:chase_v(e),
lb3:ARG1-2(x),

RMRS Matching

lb1:every_q(x),
lb1:RSTR(h9),
lb1:BODY(h6),
lb2:cat_n(x),
lb4:some_q(y),
lb1:RSTR(h8),
lb1:BODY(h7),
lb5:dog_n_1(y),
lb3:chase_v(e),
lb3:ARG1(x),
lb3:ARG2(y)

lb1:every_q(x),
lb2:cat_n(x),
lb4:some_q(y),
lb5:dog_n(y),
lb3:chase_v(e)

QA with parsed corpus



Questions and answers: QA, NLID etc

A valid answer should entail the query (with suitable interpretation of *wh*-terms etc).
Is a dog barking?
$\exists x[\text{dog}'(x) \wedge \text{bark}'(x)]$

A dog is barking entails *A dog is barking*

Rover is barking and *Rover is a dog* entails *A dog is barking.*
$\text{bark}'(\mathit{Rover}) \wedge \text{dog}'(\mathit{Rover})$ entails $\exists x[\text{dog}'(x) \wedge \text{bark}'(x)]$

which dog is barking?
$\text{bark}'(\mathit{Rover}) \wedge \text{dog}'(\mathit{Rover})$ entails $\exists x[\text{dog}'(x) \wedge \text{bark}'(x)]$
Bind query term to answer.

QA example 1

Example

What eats jellyfish?

Simplified semantics:
[a:eat(e), ARG1(a,x), ARG2(a,y), jellyfish(y)]
So won't match on *jellyfish eat fish*.

What eats jellyfish?

Example

Turtles eat jellyfish and they have special hooks in their throats to help them swallow these slimy animals.

What eats jellyfish?

Example

Turtles eat jellyfish and they have special hooks in their throats to help them swallow these slimy animals.

Match on [a:eat(e), ARG1(a,x), ARG2(a,y), jellyfish(y)]

A logically valid answer which entails the query since the conjunct can be ignored.

What eats jellyfish?

Example

Sea turtles, ocean sunfish (Mola mola) and blue rockfish all are able to eat large jellyfish, seemingly without being affected by the nematocysts.

What eats jellyfish?

Example

Sea turtles, ocean sunfish (Mola mola) and blue rockfish all are able to eat large jellyfish, seemingly without being affected by the nematocysts.

Pattern matching on semantics:
[a:eat(e), ARG1(a,x), ARG2(a,y), large(y), jellyfish(y)]

eat large jellyfish entails *eat jellyfish* (because *large* is intersective)

What eats jellyfish?

Example

Also, open ocean-dwelling snails called Janthina and even some seabirds have been known to eat jellyfish.

What eats jellyfish?

Example

Also, open ocean-dwelling snails called Janthina and even some seabirds have been known to eat jellyfish.

[a1:know(e), ARG2(a1,h1), qeq(h1,lb), lb:a:eat(e), ARG1(a,x), ARG2(a,y), jellyfish(y)]

Logically valid if *know* is taken as truth preserving.

$$\forall P \forall y [know(y, P) \implies P]$$

Axioms like this required for logically valid entailment: missing axiom would cause failure to match.

What eats jellyfish?

Example

Also, open ocean-dwelling snails called Janthina and even some seabirds have been known to eat jellyfish.

[a1:know(e), ARG2(a1,h1), qeq(h1,lb), lb:a:eat(e), ARG1(a,x), ARG2(a,y), jellyfish(y)]

Logically valid if *know* is taken as truth preserving.

$$\forall P \forall y [know(y, P) \implies P]$$

Axioms like this required for logically valid entailment: missing axiom would cause failure to match.

Take a question:
What debts did Qintex group leave?
Find a short piece of text (a sentence for the practical) from a large collection of documents which answers the question:
Qintex's failure left corporate debts of around ADollars 1.5bn (Pounds 680m) and additional personal debts.
Deep parse the question, RASP parse the answer texts, produce RMRS in both cases, find the best matches.
Evaluate on large set of questions.