Class Intro & Compositional Semantics & Others

1 Class intro

- What does a declarative denote?
  A declarative denotes a proposition. Its extension is a truth value (True or False or undefined); its intension is a set of possible worlds where this proposition is true.

  (1) Anna wears a hat.
      a. \([\text{Anna wears a hat}] = 1 \text{ iff Anna wears a hat, and } [\text{Anna wears a hat}] = 0 \text{ otherwise}\)
      b. \([\text{Anna wears a hat}] = \{w : \text{Anna wears a hat in } w\}\)

- The issue: What does a question denote?

  (2) Who wears a hat?

  - Hard to answer directly. Instead, we usually start with the relation between questions, answers, and statements (viz., indirect questions).
    Question-answer pair: an answer needs to be exhaustive, unless ignorance-marked.

  (3) Who wears a hat?

      Anna
      Billy
      Cindy
      Daisy

      a. Anna and Cindy.
      b. # Anna.

  Indirect question: \textit{wh}-knowledge needs to be exhaustive; beliefs in false answers must be avoided

  (4) John knows who wears a hat.
      (i) For any individual \(x\), if \(x\) wears a hat, John knows that \(x\) wears a hat.
      (ii) For any individual \(x\), if \(x\) doesn’t wear a hat, not [John believes that \(x\) wears a hat].

  - We also consider other “question-containing” constructions:
    Free relative: free relative denotes a short answer of the corresponding question.

  (5) John went to [where he can get help].

  \textit{Wh}-conditional (Mandarin): a \textit{wh}-conditional expresses a relation between the short answers of the two involved \textit{wh}-questions.
Questions themselves also give us some hints:

**Number-marking:** Singular marked questions have a uniqueness requirement:

(7) (Context: the speaker remembers that two of the girls wear a hat, but he cannot remember whom exactly.)
   a. Who wears a hat?
   b. # Which girl wears a hat?
   c. Which girls wear a hat?

**NPI-licensing:** Questions can license the negative polarity item (NPI) *any*:

(8) a. * Anna wears any accessory.
    b. Daisy doesn’t wear any accessory.
    c. Who wears any accessory?

• Views in canonical approaches:

<table>
<thead>
<tr>
<th></th>
<th>A question denotes ...</th>
<th>A wh-item denotes ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamblin Semantics</td>
<td>a set of possible answers</td>
<td>a set of individuals</td>
</tr>
<tr>
<td>Karttunen Semantics</td>
<td>a set of true answers</td>
<td>an $\exists$-quantifier</td>
</tr>
<tr>
<td>Categorial Semantics</td>
<td>an $\lambda$-abstract</td>
<td>an $\lambda$-operator</td>
</tr>
<tr>
<td>Partition Semantics</td>
<td>a partition of worlds</td>
<td>an $\lambda$-operator</td>
</tr>
</tbody>
</table>

Table 1: Canonical approaches of composing questions

– Hamblin/Alternative Semantics

(9) a. [which cat] = \{x : x is a cat\}
    b. [which cat meows] = \{x meows : x is a cat\}

– Karttunen Semantics

(10) a. [which cat] = [some cat]
     b. [which cat meows] = \{x meows : x is a cat $\land$ x meows in w\}

– Categorial approaches

(11) a. [which cat] = $\lambda P \lambda x [\text{cat}(x) \land P(x)]$
     b. [which cat meows] = $\lambda x [\text{cat}(x) \land \text{meows}(x)]$

– Partition Semantics

(12) a. [which cat] = $\lambda P \lambda x [\text{cat}_@ (x) \land P_w(x)]$
     b. [which cat meows] = $\lambda w' \lambda w [\lambda x [\text{cat}_@ (x) \land P_w(x)] = \lambda x [\text{cat}_@ (x) \land P_w'(x)]$

What are the pros and cons of each composition method?
• Schedule and requirements (see syllabus)

• Questionnaire

Send me an email (yxiang@fas.harvard.edu) with the following information:

(13) a. How would you rate the difficulty of the materials in lecture 1? (1 for extremely easy, 7 for extremely difficult)

b. What is your previous exposure to semantics and logic?

c. What is your previous exposure to linguistics outside of semantics?

d. Do you have any experience with conference abstract writing?

e. What do you hope to learn in this class?

Roadmap

• Compositional Semantics
  – The principle of compositionality
  – Semantic types
  – Lambda calculus
  – Composition rules: TN, NN, FA, PM

• Generalized quantifiers
2 Compositional Semantics

2.1 The principle of compositionality

- In any natural language, there are infinitely many sentences, and the brain is finite. So, for syntax, linguistic competence must involve some finitely describable means for specifying an infinite class of sentences.

A speaker of a language knows the meanings of those infinitely many sentences, and is able to understand a sentence he/she hears for the first time. So, for semantics, there must also be finite means for specifying the meanings of the infinite set of sentences of any natural language.

- In generative grammar, a central principle of formal semantics is that the relation between syntax and semantics is compositional.

(14) The principle of compositionality (Fregean Principle):
The meaning of a complex expression is determined by the meanings of its parts and the way they are syntactically combined.

(15) Kitty meows. 
\[ S \]
\( \text{Kitty} \) (Saturated)
\( \text{meows} \) (Unsaturated)

The meaning of (15) is the result of applying the unsaturated part of the sentence (a function) to the saturated part (an argument).

2.2 Semantic Types

- The basic types correspond to the objects that Frege takes to be saturated.
  - \( e \) for individuals, in \( D_e \)
  - \( t \) for truth values, in \( \{1, 0\} \)

From these basic types, we can recursively define complex types:
  - \( \langle e, t \rangle \) for intransitive verbs, predicative adjectives, and common nouns
  - \( \langle e, \langle e, t \rangle \rangle \) for transitive verbs

(16) Types
   a. Basic types: \( e \) (individuals/entities) and \( t \) (truth values).
   b. Functional types: If \( \sigma \) and \( \tau \) are types, then \( \langle \sigma, \tau \rangle \) is a type.
   c. Intensional types: if \( \sigma \) is a type, then \( \langle s, \sigma \rangle \) is a type.

(17) Domains
   a. \( D_t = \{1, 0\} \)
   b. \( D_{\langle \sigma, \tau \rangle} = \{ f \mid f : D_\sigma \to D_\tau \} \) (functions from things of type \( \sigma \) to things of type \( \tau \))
   c. \( D_{\langle s, \tau \rangle} = \{ f \mid f : W \to D_\tau \} \) (functions from possible worlds to things of type \( \tau \))
(18) **Types of functions and properties**

a. A function of type $\langle \sigma, t \rangle$ is a *set* of objects of type $\sigma$.

b. A function of type $\langle \sigma, st \rangle$ is a *property* of objects of type $\sigma$.

For now, let us ignore intensionality. We’ll get back to it later.

- **Syntactic categories and their semantic types**

<table>
<thead>
<tr>
<th>Syntactic category</th>
<th>English expressions</th>
<th>Semantic type (extensionalized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence</td>
<td>$t$</td>
<td></td>
</tr>
<tr>
<td>Proper name</td>
<td>$e$</td>
<td></td>
</tr>
<tr>
<td>e-type/referential NP</td>
<td>the king</td>
<td>$\langle e, t \rangle$</td>
</tr>
<tr>
<td>Common noun</td>
<td>cat</td>
<td>$e$</td>
</tr>
<tr>
<td>IV, VP</td>
<td>run</td>
<td>$\langle e, t \rangle$</td>
</tr>
<tr>
<td>TV</td>
<td>love</td>
<td>$\langle et, t \rangle$</td>
</tr>
<tr>
<td>Predicative ADJ</td>
<td>happy, gray</td>
<td>$\langle e, t \rangle$</td>
</tr>
<tr>
<td>Predicate modifier</td>
<td>skillful, quickly</td>
<td>$\langle et, et \rangle$</td>
</tr>
<tr>
<td>Sentential modifier</td>
<td>perhaps</td>
<td>$\langle t, t \rangle$</td>
</tr>
<tr>
<td>Determiner</td>
<td>some, every, no</td>
<td>$\langle et, \langle et, t \rangle \rangle$</td>
</tr>
<tr>
<td></td>
<td>the, a</td>
<td>$\langle et, e \rangle$</td>
</tr>
</tbody>
</table>

**Exercise 1:** Identify the semantic type of *beautiful* in the following sentences:

(19) a. Anna is a beautiful girl.
    b. Anna is a beautiful dancer.

**Exercise 2:** Classify the following words based on their (extensional) semantic types:

*not, if...then, student, John, Boston, a man, buy, fast, carefully, necessarily*

- With type assignments to expressions of natural language, we can determine the semantic types of new expressions/morphemes.

(20) Ann is pretty.
    (21) Kitty meowed.
    (22) Cambridge is in MA.

- But, be careful!
  
  (i) an expression can be type-ambiguous;
  
  (ii) there can be covert elements in the LF;
  
  (iii) there can be type-shifting operations, ....
2.3 Lambda calculus

• Functions

A function \( f \) from \( A \) to \( B \) (written as \( 'f : A \rightarrow B' \)) is a relation such that

(i) \( f \) maps every element in \( A \) to some element in \( B \). Formally: \( \text{Dom}(f) = A \)
(ii) Each element in \( A \) is paired with just one element in \( B \)

– Examples:

(23) \([\text{the mother of}] = f : D_e \rightarrow D_e \) s.t. for all \( x \in D_e, f(x) = \text{the mother of } x \).
(24) \([\text{meow}] = f : D_e \rightarrow \{1, 0\} \) s.t. for all \( x \in D_e, f(x) = 1 \) if \( x \) meows, \( f(x) = 0 \) otherwise.
(25) \([\text{hit}] = f : D_e \rightarrow D_{(e,t)} \) s.t. for all \( x \in D_e, \)
\( f(x) = g : D_e \rightarrow \{1, 0\} \) s.t. for all \( x \) in \( D_e, g(x) = 1 \) iff \( y \) hits \( x \).

• It is more handy and common to write functions in lambda (\( \lambda \))-notations.

(26) **Schema of lambda terms:**
\( \lambda v[\beta.\alpha] \) read as “the function which maps every \( v \) such that \( \beta \) to \( \alpha \)”
   a. \( v \) is the argument variable
   b. \( \beta \) is the domain condition (the domain over which the function is defined)
   c. \( \alpha \) is the value description (a specification of the value/output of the function)

(27) **Semantic types of lambda terms**
If \( v \) is of type \( \sigma \) and \( \alpha \) is of type \( \tau \), then \( \lambda v.\alpha \) is of type \( \langle \sigma, \tau \rangle \).

(28) **Lambda reduction/conversion**
\( (\lambda v.\alpha)(a) = \alpha' \) where \( \alpha' \) is like \( \alpha \) but with every free occurrence of \( v \) replaced by \( a \).
(Note: Occurrences of \( v \) that are free in \( \alpha \) are bound \( \lambda v \) in \( \lambda v.\alpha \))

– Examples in math:

(29) \( \lambda x[ x \in N. \ x + 1 ] \) read as “the function that maps every \( x \) such that \( x \) is in \( N \) to \( x + 1 \).”
   a. \( (\lambda x[ x \in N. \ x + 1 ])(2) = 2 + 1 = 3 \)
   b. \( (\lambda x[ x \in N. \ x + 1 ])(a) \) is undefined

– Examples in natural languages:

Verb:

(30) a. \([\text{meow}] = \lambda x_e. \text{meow}'(x) \)
    b. \([\text{meow}](k) = (\lambda x_e.\text{meow}'(x))(k) = \text{meows}'(k) \)
(31) a. \([\text{hit}] = \lambda y_e \lambda x_e. \text{hit}'(x, y) \)
    b. \([\text{hit}](j) = \lambda x_e. \text{hit}'(x, j) \)
    c. \([\text{hit}](j)(m) = \text{hit}'(m, j) \)

Non-verbal predicates:
(32)  a. \([\text{cat}] =\)
    b. \([\text{gray}] =\)
    c. \([\text{fast}] =\)

Vacuous words:

(33)  a. \([\text{is}] =\)
    b. \([\text{that}] =\)

Exercise 3: Specify its semantic types of the following \(\lambda\)-abstracts.

(34)  a. \(\lambda f_{e,t}\lambda x_e[f(x) \land \text{gray}'(x)]\)
    b. \(\lambda f_{e,t}\lambda y_{e,t} \exists x[f(x) \land g(x)]\)

Exercise 4: Simplify the following formulas:

(35)  a. \([\lambda x\lambda y(\lambda z.\text{introduce-to}(x,y,z)(a))](b)\)
    b. \((\lambda f_{e,t}\lambda x_e[f(x) \land \text{gray}'(x)])(\lambda y_e.\text{cat}(y))\)
    c. \((\lambda P_{e,t}.P(k))(\lambda y.\text{cat}(y))\)

2.4 Composition

- Basic composition rules

(36)  Terminal Nodes (TN)
If \(\alpha\) is a terminal node, \([\alpha]\) is specified in the lexicon.

Non-Branching Nodes (NN)
If \(\alpha\) is non-branching node, and \(\beta\) is its daughter node, then \([\alpha] = [\beta]\).

Functional Application (FA)
If \(\{\beta, \gamma\}\) is the set of \(\alpha\)'s daughters, \([\beta] \in D_{(\sigma,\tau)}\) and \([\gamma] \in D_{\sigma}\), then \([\alpha] = [\beta][\gamma]\)

(37)  Kitty meows.

\[
\begin{array}{c}
S_t \\
\downarrow \\
\text{NP}_e & \downarrow \\
\downarrow & \downarrow \\
N_e & \text{VP}_{e,t} \\
\downarrow & \downarrow \\
\text{Kitty}_e & \text{meows}_{e,t} \\
\end{array}
\]

. a. \([\text{NP}] = [\text{N}] = [\text{Kitty}] = k\) By TN, NN
b. \([\text{VP}] = [\text{V}] = [\text{meows}] = \lambda x_e.\text{meows}'(x)\) By TN, NN
c. \([\text{S}] = [\text{VP}][\text{NP}]\) By FA

d = [\lambda x_e.\text{meows}'(x)](k)
    = \text{meows}'(k)

- What if two sister nodes do not hold a function-argument relation?

(38)  Cambridge is \([\text{a city} [\text{in MA}]]\).
(39) **Predicate Modification (PM)**
If $\alpha$ is a branching node, $\{\beta, \gamma\}$ is the set of $\alpha$’s daughters, and $[\beta]$ and $[\gamma]$ are both in $D_{\langle e,t \rangle}$, then $\lambda x_\sigma[[\beta](x) = [\gamma](x) = 1]$

- **Exercise 5:** Categorial approaches treat $wh$-words as lambda-operators. Using the lexical entries in (a-b), can we compose the semantics of the LF in (c)? What composition rules can we use for Node 1 and 2?

(40) Who bought what?
   a. $[who] = \lambda P_{\langle e,t \rangle} \lambda x_\sigma[\text{man}(x) \land P(x)]$
   b. $[what] = \lambda P_{\langle e,t \rangle} \lambda x_\sigma[\text{thing}(x) \land P(x)]$
   c. $\lambda x_\sigma x_1 \lambda y_{\text{IP}} x$ bought $y$

- **Take the meaning as a whole!**
  For any two meanings $\alpha$ and $\beta$, the combination of $\alpha$ and $\beta$ can only depend on what $\alpha$ and $\beta$ are, each “taken as a whole”; it cannot depend on the meanings that $\alpha$ and $\beta$ were formed from by earlier semantic operations. In other words, the meaning of an expression has no internal structure.
  (This constraint will be crucial in the formation of short answers and the derivation of quantificational variability effects.)

3 **Generalized quantifiers**

3.1 **Generalized quantifiers**
- Quantificational DPs (e.g. *everything, something, every cat, some cat*) are not individuals (cf. proper names like *John*), nor individual sets (cf. common nouns like *cat*).
  E.g. Only an $e$-type NP can normally license a singular discourse pronoun.

(41) a. John /the man/ a man walked in. He looked tired.
   b. Every man /no man/ more than one man walked in. *He looked tired.
We treat quantificational DPs as second-order functions of type \( \langle et, t \rangle \), called “generalized quantifiers (GQs)”. In (42), *meows* is an argument of *every cat*.

(42) \[
\begin{array}{c}
\text{DP}_{\langle et, t \rangle} \\
\text{VP}_{\langle et, t \rangle} \\
\text{every} \\
\text{cat}_{\langle e, t \rangle} \\
\end{array}
\]

(43) a. \([\text{every cat}] = \lambda P_{\langle e, t \rangle}. \forall x[\text{cat}'(x) \to P(x)]\]
    b. \([\text{some cat}] = \lambda P_{\langle e, t \rangle}. \exists x[\text{cat}'(x) \land P(x)]\]

- The determiner *every* combines with a common noun of type \( \langle e, t \rangle \) to return a generalized quantifier of type \( \langle et, t \rangle \). Therefore, its type is quite complex: \( \langle et, \langle e, t \rangle \rangle \).

(44) a. \([\text{every}] = \lambda Q_{\langle e, t \rangle}. \lambda P_{\langle e, t \rangle}. \forall x[Q(x) \to P(x)]\]
    b. \([\text{some}] = \lambda Q_{\langle e, t \rangle}. \lambda P_{\langle e, t \rangle}. \exists x[Q(x) \land P(x)]\]

- In comparison, a *the*-phrase denotes a referential individual of type \( e \). The determiner *the* is of type \( \langle et, t \rangle \): it combines with a common noun of type \( \langle e, t \rangle \) to return a referential individual of type \( e \).

(45) a. \([\text{the cat}] = \lambda x[\text{cat}'(x)]\)
    (The unique cat)
    b. \([\text{the}] = \lambda P.tx[P(x)]\) \hspace{1cm} (To be revised)

- Individuals (of type \( e \)) can also be shifted into generalized quantifiers via *type-lifting*.

(46) a. \([\text{John}] = j\)
    b. \(\text{LIFT}([\text{John}]) = \lambda P_{\langle e, t \rangle}. P(j)\)
    c. \((\text{LIFT}([\text{John}]))([\text{came}]) = (\lambda P_{\langle e, t \rangle}. P(j))(\lambda x.\text{came}'(x)) = \lambda x.\text{came}'(j) = \text{came}'(j)\)

3.2 Be-shifting

- We can extract the quantification domain of an \( \exists \)-quantifier via the BE-shifter (Partee 1986):

(47) \(\text{BE} = \lambda P. \lambda z[P(\lambda y. y = z)]\)
(48) \(\text{BE}([\text{some cat}]) = \lambda z[(\lambda f_{\langle e, t \rangle}. \exists x \in \text{cat}'[f(x)])(\lambda y. y = z)] = \lambda z[\exists x \in \text{cat}'[x = z]] = \{z : z \in \text{cat}'\} = \text{cat}'\)

- **Discussion**: What do we get by applying BE to \([\text{every cat}]\) and \(\text{LIFT}([\text{John}])\)?
3.3 Quantifier raising

- Problem in (49a): a type-mismatch arises when a verb takes a quantificational DP as its object.

Solution in (49b): At logical form (LF), the generalized quantifier can raise to adjoin to any propositional node, leaving a trace of type $e$ and introducing a lambda abstraction over the traces variable. This operation is called “Quantifier Raising (QR)”.

\[(49) \text{Anna loves every cat.}\]

- Scope ambiguity can be characterized based on QR.

\[(50) \text{Every kid loves some cat.}\]

- In generative grammar, lambda-abstraction is to represent phrasal movement. The sister node of an $\lambda$-abstract is the moved phrase. The variable bound by the $\lambda$-operator is the trace.

- Discussion: Hamblin-Karttunen semantics defines a question as a set of propositions. Under this view, is the following LF well-formed? Why or why not?

\[(51) \text{Which cat does every kid love?}\]