1. Compositional Semantics and Pragmatics

1.1. The Principle of Compositionality

A basic starting point of generative grammar: there are infinitely many sentences in any natural language, and the brain is finite, so linguistic competence must involve some finitely describable means for specifying an infinite class of sentences. That is a central task of syntax.

Semantics: A speaker of a language knows the meanings of those infinitely many sentences, is able to understand a sentence he/she has never heard before or to express a meaning he/she has never expressed before. So for semantics also there must be a finite way to specify the meanings of the infinite set of sentences of any natural language.

1.2. Model-theoretic Semantics

In formal semantics, truth conditions are expressed in terms of truth relative to various parameters — a formula may be true at a given time, in a given possible world, relative to a certain assignment of meanings to its atomic “lexical” expressions and of particular values to its variables. For simple formal languages, all of the relevant variation except for assignment of values to variables is incorporated in the notion of truth relative to a model. Semantics which is based on truth-conditions is called model-theoretic.

Compositionality in the Montague Grammar tradition:
The task of a semantics for language L is to provide truth conditions for every well-formed sentence of L, and to do so in a compositional way. This task requires providing appropriate model-theoretic interpretations for the parts of the sentence, including the lexical items. The task of a syntax for language L is (a) to specify the set of well-formed expressions of L (of every category, not only sentences), and (b) to do so in a way which supports a compositional semantics. The syntactic part-whole structure must provide a basis for semantic rules that specify the meaning of a whole as a function of the meanings of its parts.

Basic structure in classical Montague grammar:

(1) Syntactic categories and semantic “types”: For each syntactic category there must be a uniform semantic type. One possible hypothesis: sentences express propositions, nouns and adjectives express properties of entities, verbs express properties of events.
(2) Basic (lexical) expressions and their interpretation. Some syntactic categories include basic expressions; for each such expression, the semantics must assign an interpretation of the appropriate type. Within the tradition of formal semantics, most lexical meanings are left unanalyzed and treated as if primitive; Montague regarded most aspects of the analysis of lexical meaning as an empirical rather than formal matter; formal semantics is concerned with the types of lexical meanings and with certain aspects of lexical meaning that interact directly with compositional semantics, such as verbal aspect.

(3) Syntactic and semantic rules. Syntactic and semantic rules come in pairs:
<Syntactic Rule n, Semantic Rule n>: in this sense compositional semantics concerns “the semantics of syntax.” (Example: See syntax and semantics of predicate calculus in Section 3.)

Syntactic Rule n: If α is an expression of category A and β is an expression of category B, then F_β(α,β) is an expression of category C. (where F is some syntactic operation on expressions]

Semantic Rule n: If α is interpreted as α’ and β is interpreted as β’, then F_β(α,β) is interpreted as G_β(α’,β’). (where G is some semantic operation on semantic interpretations]

1.3. Semantics and pragmatics

The logico-philosophical tradition divides semiotics (the study of signs, applicable to both natural and constructed languages) into syntax, semantics, and pragmatics (Morris 1938). On this classic view, syntax concerns properties of expressions, such as well-formedness; semantics concerns relations between expressions and what they are “about” (typically “the world” or some model), such as reference and truth conditions; and pragmatics concerns relations between expressions and their uses in context, such as conversational implicature (see Sec. 2.2). Many have challenged the autonomy of semantics from pragmatics implied by the traditional trichotomy, arguing that reference and truth-conditions themselves often depend on context. We will look at these issues in future lectures.

2. Linguistic Examples.

(See also the Larson chapter) These are examples of the kinds of problems that we will be able to solve with the tools of formal semantics and pragmatics. These and other problems will be discussed in future lectures.

2.1. The structure of NPs with restrictive relative clauses.

Consider NPs such as “the boy who loves Mary”, “every student who dances”, “the doctor who treated Mary”, “no computer which uses Windows”. Each of these NPs has 3 parts: a determiner (DET), a common noun (CN), and a relative clause (RC). The question is: Are there semantic reasons for choosing among three different possible syntactic structures for these NPs?

a. Flat structure:

```
NP
| DET CN RC
the boy who loves Mary
```

b. “NP - RC” structure: The relative clause combines with a complete NP to form a new NP.

```
NP
| DET CN RC
who loves Mary
the boy
```

c. “CNP - RC” structure: (CNP: common noun phrase: common noun plus modifiers)

```
NP
| DET CNP RC
who loves Mary
the boy
```

Argument: we can argue that compositionality requires the third structure: that “boy who loves Mary” forms a semantic constituent with which the meaning of the DET combines. We can show that the first structure does not allow for recursivity, and that the second structure cannot be interpreted compositionally. (The second structure is a good structure to provide a basis for a compositional interpretation for non-restrictive relative clauses.)

2.2. “Inclusive” vs. “exclusive” disjunction: semantics or pragmatics?

Intuitively, it often seems that natural language or is often used in an “exclusive” sense: “p or q but not both”. We can write a truth-table for exclusive or, which we will represent with the symbol “$+$”, in contrast with the familiar inclusive or symbolized with “or”.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p + q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The question is, is English or (or German oder, or Russian ili, etc.) really semantically ambiguous between two truth-conditional connectives? Or can one defend an analysis on which or is semantically always inclusive disjunction, and the apparent exceptions can be explained as a result of pragmatics factors such as Gricean implicatures?
3.2. Example. Syntax and semantics of the predicate calculus (PC).

Predicate Calculus is the most well known and in a sense the prototypical example of a formal language. We use it to demonstrate features of formal languages which are most important for us: the notions of model and model-theoretic semantics, and the Principle of Compositionality.

We limit ourselves here to some examples and remarks. More exact definitions are given in Appendix I.

The sentences John loves Mary and Everyone whom Mary loves is happy can be represented as formulas of PC:

John loves Mary: $\text{love}(\text{John}, \text{Mary})$

Everyone whom Mary loves is happy: $\forall x (\text{love}(\text{Mary}, x) \rightarrow \text{happy}(x))$

Formulas and other expressions of PC are built from individual constants (or simply “constants”), (individual) variables, predicate constants (or predicate symbols), logical connectives and quantifiers. Each expression belongs to a certain type. The type structure of PC is very simple: individuals, relations of different arities (unary, binary, etc.), and truth-values.

In our examples we use the following expressions:

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Syntactic categories</th>
<th>Semantic Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>John, Mary</td>
<td>(individual) constant</td>
<td>individuals</td>
</tr>
<tr>
<td>x</td>
<td>variable</td>
<td>individuals</td>
</tr>
<tr>
<td>happy</td>
<td>unary predicate constant</td>
<td>unary relations</td>
</tr>
<tr>
<td>love</td>
<td>binary predicate constant</td>
<td>binary relations</td>
</tr>
<tr>
<td>love(Mary, x)</td>
<td>formulas</td>
<td>truth-values</td>
</tr>
<tr>
<td>happy(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\forall x (\text{love}(\text{Mary}, x) \rightarrow \text{happy}(x))$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expressions are interpreted in models. The structure common to all of the models in which a given language is interpreted (the model structure for the model-theoretic interpretation of the given language) reflects certain basic presuppositions about the “structure of the world” that are implicit in the language. For PC, any given model structure consists of the set of truth-values $\{0,1\}$, a domain $D$ which is some set of objects (or entities), and some n-ary relations on this set.

A model, or interpreted model, consists of a model structure plus a (“lexical”, or “basic”) interpretation function $I$ which assigns semantic values to all constants.

$$M = \langle D, I \rangle$$

An interpretation $I$ is built up recursively on the basis of the basic interpretation function $I$, assigns to every expression $\alpha$ its semantic value $I[\alpha]$ in a given model $M$. (More precisely, $[\alpha]_M$.) These semantic values must correspond to the types of the expressions. Thus, in our examples to the individual constants John and Mary are assigned certain objects, individual variables take their values in the set of objects (entities), to the predicate constant love is assigned a binary relation $[\text{love}]_M$, and to the predicate constant happy, a unary relation (property) $[\text{happy}]_M$. Formulas receive truth values. The formula love (John, Mary) is true in the model $M$ if the pair of objects corresponding to the constants John and Mary belongs to the relation $[\text{love}]_M$. 
The formula $\forall x (\text{love(Mary, } x) \rightarrow \text{happy}(x))$ is true in $M$ iff:
for every object $d$ in the domain, $d \in [\text{happy}]^M$ if $< [\text{Mary} ]^M, d > \in [\text{love}]^M$.

Restating the last statement more carefully and more generally requires talking about semantic values relative to a model and an assignment $g$ of values to variables.

The notation $g[d/x]$ means: The variable assignment which is identical to $g$ except for the (possible) difference that $g[d/x]$ assigns the individual $d$ to the variable $x$.

The complication of needing to talk about $g[d/x]$ comes from formulas with more than one variable, like:
$\forall x \forall y (\text{love}(x, y) \rightarrow \text{happy}(x))$ and
$\exists x \forall y (\text{love}(x, y) \rightarrow \text{happy}(x))$.

So let us restate more carefully, according to the semantics given in Appendix 1, the truth conditions for the formula: $\forall x (\text{love(Mary, } x) \rightarrow \text{happy}(x))$:

$\forall x (\text{love(Mary, } x) \rightarrow \text{happy}(x))$ $\models_{M,d}$ if:

- For each $d$ in $D$, $d \models [\text{happy}]^M$ if $< [\text{Mary}]^M, d > \in [\text{love}]^M$.

For each constant $\alpha$, $\alpha \models [\text{happy}]^M = I(\alpha)$.

And for any variable $x$, $x \models [\text{d}^M] = g[d]/x(x) = d$. So the condition above is equivalent to:

- If for each $d$ in $D$, $d \models I(\text{love})$, then $d \models I(\text{happy})$.

Example
Let us consider a very simple PC language which has (as in the formulas above) only two constants John and Mary and two predicate symbols love (binary) and happy (unary).

Let us consider two models, $M_1$ and $M_2$:

$M_1 = \langle D, I_1, >, D = \{j, m\}$,
$I_1(\text{John}) = j, I_1(\text{Mary}) = m$, $I_1(\text{love}) = \langle[j,j], j, m, m, \rangle, I_1(\text{happy}) = \langle j, m \rangle$,

$M_2 = \langle D, I_2, >, D = \{j, m\}$,
$I_2(\text{John}) = j, I_2(\text{Mary}) = m$, $I_2(\text{love}) = \langle[j,j], m, j, \rangle, I_2(\text{happy}) = \{m\}$.

It is easy to see that both formulas love (John, Mary) and love (Mary, John) are true in $M_1$ but only the second one is true in $M_2$.

The formula $\forall x (\text{love(Mary, } x) \rightarrow \text{happy}(x))$ is true in $M_1$. But it is false in $M_2$, since for the evaluation $g$ such that $g(x) = j$ we have $\text{love(Mary, } x) \models_{M_2,d} = 1$ and $\text{happy}(x) \models_{M_2,d} = 0$.

The semantics of PC illustrates the Principle of Compositionality.

As we know the infinite set of formulas of PC are built from terms (individual variables and constants) and predicate symbols by recursive syntactic rules (rules R1 – R8 in Appendix 1).

The semantics of these formulas – their interpretation in every given model – is defined by semantic rules $S1 – S8$, which correspond in a direct way to the syntactic rules. The semantics of the whole is based on the semantics of parts by means of this pairing of semantic interpretation rules with syntactic formation rules. See trees 1 and 2 in the
(\(\phi(D)\) is the power set (the set of all subsets) of \(D\)).

Pred-\(n\): \(n\)-place relations; sets of \(n\)-tuples of entities. Values: members of \(\phi(D\times\cdots\times D)\).

Form: Truth values. Values: members of \([0,1]\).

**Semantic interpretation relative to \(M, g\):**

We use the notation \(\llbracket q \rrbracket_M^g\) for the semantic value of an expression \(q\) relative to \(M, g\).

**Basic Expressions ("lexical semantics"):**

A. If \(\alpha\) is a variable, then \(\llbracket \alpha \rrbracket_M^g = g(\alpha)\).

B. If \(\alpha\) is a constant, then \(\llbracket \alpha \rrbracket_M^g = 1(\alpha)\).

**Semantic Rules ("semantics of syntax"):**

**S1:** If \(P \in \text{Pred-}1\) and \(T \in \text{Term}\), then \(\llbracket P(T) \rrbracket_M^g = 1\) iff \(\llbracket P \rrbracket_M^g = 1\), \(\llbracket T \rrbracket_M^g = 1\).

**S2:** More general rule: If \(R \in \text{Pred-}n\) and \(T_1, \ldots, T_n \in \text{Term}\), then \(\llbracket R(T_1, \ldots, T_n) \rrbracket_M^g = 1\) iff

\[\llbracket T_1 \rrbracket_M^g = 1, \ldots, \llbracket T_n \rrbracket_M^g = 1\]

**S3:** If \(\varphi \in \text{Form}\), then \(\llbracket \neg \varphi \rrbracket_M^g = 1\) iff \(\llbracket \varphi \rrbracket_M^g = 0\).

**S4:** If \(\varphi, \psi \in \text{Form}\), then \(\llbracket \varphi \& \psi \rrbracket_M^g = 1\) iff \(\llbracket \varphi \rrbracket_M^g = 1\) and \(\llbracket \psi \rrbracket_M^g = 1\).

**S5:** If \(\forall x \varphi \in \text{Form}\), then \(\llbracket (\forall x \varphi) \rrbracket_M^g = 1\) iff \(\forall x \in D\), \(\llbracket \varphi \rrbracket_M^g = 1\).

**S6:** If \(\varphi, \psi \in \text{Form}\), then \(\llbracket \varphi \rightarrow \psi \rrbracket_M^g = 1\) iff \(\llbracket \varphi \rrbracket_M^g = 1\) or \(\llbracket \psi \rrbracket_M^g = 1\).

**S7:** If \(\varphi\) is a variable and \(\varphi \in \text{Form}\), then \(\llbracket \exists \varphi \rrbracket_M^g = 1\) iff there is a \(d \in D\) such that \(\llbracket \varphi \rrbracket_M^{d(g)} = 1\).

**Truth:** Some formulas are true independent of the choice of assignment; those can be called true relative to just \(M\), i.e. simply true on the given interpretation.

If \(\varphi \in \text{Form}\), then:

- \(\llbracket \varphi \rrbracket_M^g = 1\) iff for all assignments \(g, \llbracket \varphi \rrbracket_M^g = 1\).
- \(\llbracket \varphi \rrbracket_M^g = 0\) iff for all assignments \(g, \llbracket \varphi \rrbracket_M^g = 0\).
- Otherwise \(\llbracket \varphi \rrbracket_M^g\) is undefined.

**APPENDIX 2: For Seminar Feb 20: A Practice Homework**

*(to do together in class, not to turn in)*

**Background:**

1. We will first work on the formula \(\forall x \text{ happy}(x)\), and work out its interpretation with respect to the model M2, working compositionally. We’ll do it basically the same way as #2 below, but just on the blackboard.

2. Below you will find a syntactic “derivation” tree for the formula \(\forall x (\text{love}(M, x) \rightarrow \text{happy}(x))\), which expresses the same proposition as the English sentence *Everyone who Mary loves is happy*. That is followed by a derivation of the truth-conditions of the formula according to the compositional semantic rules of the predicate calculus. Each line is annotated to identify what semantic rule was applied in the derivation of that line, and what node of the syntactic derivation tree it corresponds to. (The problem you are asked to solve is stated after all of that.)
3. Exercise: (to do in seminar together) This one gives more practice with using g.

The predicate logic formula \( \forall x (\exists y \text{ love}(x, y) \rightarrow \text{happy}(x)) \) is equivalent to the English sentence “Everyone who loves someone is happy.”

- **Draw a syntactic tree** analogous to Tree 1 above which shows how the formula is built up from its parts according to the syntactic rules of the predicate calculus (in the Appendix above).
- **Give each node a label** that identifies both the syntactic category of the expression it dominates and the number of the syntactic rule by which its immediate constituents were combined (or “Basic”, if that node dominates a basic expression.)
- **Work out the truth-conditions** of the formula according to the semantic rules of the predicate calculus, analogous to the step-by-step derivation of truth conditions for the example above (see NOTE below). **Annotate each line** by identifying the semantic rule that was applied anywhere within that line (show where), and the node of the tree to which it corresponds. (According to the principle of compositionality, there should be a perfect match between syntactic rule and semantic rule applied at each node.)
- In addition, **further annotate the syntactic tree** by adding to the label of each non-terminal node the number of the semantic rule which was used to combine the meanings of the daughter-node expressions to get the meaning of the whole expression dominated by that node. For nodes dominating basic expressions, indicate whether the semantic rule used is Rule A or Rule B. (If you’ve done it right, there should be a perfect correspondence between syntactic rules and semantic rules applied at a given node, as in Tree 2 above.)

**NOTE:** What happens when you are working with \( g[d/x] \) and you need to make a further substitution, e.g. for the variable \( y \)? Answer: you need to consider another arbitrary element \( d' \) of \( D \), and modify the assignment again, resulting in \( g[d/x][d'/y] \): the assignment just like \( g \) except it assigns \( d \) to \( x \) and \( d' \) to \( y \).

**REFERENCES.**

(As many of these are not referred to in this lecture but will be referred to in later lectures.)


