Lecture 13: Noun Phrases and Quantification

1. Quantifiers: Introduction

We’ve studied quantifiers in first-order quantificational logic (= predicate logic), and you’ve had some experience translating sentences from English into predicate logic. You’ve also had some experience with the lambda-calculus and type theory. Now let’s look at English sentences containing quantifiers and see whether logic helps us analyze them semantically. The answer will be yes, but we’ll see more clearly why first-order quantificational logic isn’t enough if we want to do justice to the structure of natural language, and how the treatment of NPs as ‘generalized quantifiers’ of type (e→t)→t helps solve a lot of problems and open up new perspectives.

For some historical perspective behind what we’ll talk about today and next time, see (Partee 1996, Partee 2004) and the longer version of the latter on my website: http://people.umass.edu/partee/docs/BHP_Essay_Feb05.pdf. A good introduction to much of what we’ll do in these two classes can be found in (Larson 1995), which you’ve read. And a somewhat more advanced introduction, including an introduction to the lambda calculus, which we won’t go into here, can be found in Part D, “English as a Formal Language”, of the Partee, ter Meulen and Wall textbook: Chapter 13 “Basic Concepts” and Chapter 14 “Generalized Quantifiers”.

Consider the following sentence containing a universal quantifier-word every and an indefinite article a. The sentence is semantically ambiguous: we can think of the indefinite article as introducing an existential quantifier and every as introducing a universal quantifier, and the two quantifiers can be interpreted in either ‘scope order’.

(1) a. Every student read a book. (Quantifier scope ambiguity)

Just one (surface) syntactic structure:

b. S
   / \        /
  NP VP
 /   |
DET CNP V NP
/   |   |
every student read DET CNP
|   |   |
a book

Predicate logic representations of the two readings:

(2) (i) ∀x ( Student (x) → ∃y ( Book (y) & Read (x,y))
   (ii) ∃y ( Book (y) & ∀x ( Student (x) → Read (x,y))
Compositional interpretation of the English sentence: How do we derive the meaning of the whole from the meaning of the parts? -- First question: what are the “parts”?

The difficulty for compositionality if we try to use predicate calculus to represent “logical form”: What is the interpretation of “every student”? There is no appropriate syntactic category or semantic type in predicate logic. Inadequacy of 1st-order predicate logic for representing the semantic structure of natural language. We can solve this problem when we have (the lambda-calculus and)\(^1\) a richer type theory.

**Categories of PC:**

<table>
<thead>
<tr>
<th>PC</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Sentence</td>
</tr>
<tr>
<td>Predicate</td>
<td>Verb, Common Noun, Adjective</td>
</tr>
<tr>
<td>Term</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Proper Noun</td>
</tr>
<tr>
<td>Variable</td>
<td>Pronoun (he, she, it)</td>
</tr>
<tr>
<td>(no more)</td>
<td>Verb Phrase, Noun Phrase, Common Noun Phrase, Adjective Phrase, Determiner, Preposition, Prepositional Phrase, Adverb</td>
</tr>
</tbody>
</table>

Where in the formula below is the meaning of every student?

(3) a. \( \forall x (\text{Student}(x) \rightarrow \text{Walk}(x)) \)

Answer: it’s all the underlined parts, that is, everything except the predicate “Walk”!

b. \( \forall x (\text{Student}(x) \rightarrow \text{Walk}(x)) \)

It’s not a *constituent* in the logical formula, although it is a constituent in the English sentence. Predicate logic helps us express the truth-conditional *content* of these English sentences, but it does not capture the *structure* of such English sentences.

Consider the following sentences and their structures: English on the left, PC on the right:

(4) a. \[ S \]
    \[
    \begin{array}{ccc}
    \text{NP} & \text{VP} & \text{Pred} & \text{Term} \\
    \text{John} & \text{walks} & \text{Walk} & \text{John} \\
    \end{array}
    \]

\( John \text{ walks.} \)

b. \[ S (\text{Formula}) \]

\( Walk (\text{John}) \)

\(^1\) Using the lambda-calculus to help express higher-type interpretations is often very helpful, but it’s not strictly essential.
Every student walks.

Some student walks.

No student walks.

This last sentence, *No student walks*, has two equivalent translations into predicate logic: the one in the tree above and \((\forall x)(\text{student}(x) \rightarrow \neg \text{walk}(x))\). That would have a different tree.

What similarities do you see in the four “English” trees? And what differences do you see between the English syntactic structures and the structures of the formulas that are their logical translations?
2. A solution (Montague 1973): NPs as Generalized Quantifiers

Determiner meanings: Relations between sets, or functions which apply to one set (the interpretation of the CNP) to give a function from sets to truth values, or equivalently, a set of sets (the interpretation of the NP).

Typical case:

```
            S
           /   \
          NP   VP
 /       \
DET     CNP
```

Semantic types: Basic types: \( e \), the type of entities, and \( t \), the type of truth values.

Functional types: \( a \to b \): the type of functions from \( a \)-type things to \( b \)-type things

Example: 1-place predicates denote sets of entities; the type of the characteristic function for a set of entities is \( e \to t \). So this is the type for simple nouns like student, intransitive verbs like walks, and simple adjectives like red. We’ll also assume it’s the type for all VPs. The type for \( S \) will be \( t \). Proper names in English, like terms in logic, can be assumed to be of type \( e \).

\[
\begin{align*}
\text{CNP:} & \quad \text{type } e \to t \\
\text{VP:} & \quad \text{type } e \to t \\
\text{DET:} & \quad \text{interpreted as a function which applies to CNP meaning to give a generalized quantifier, which is a function which applies to VP meaning to give Sentence meaning (extension: truth value). type: } (e \to t) \to ((e \to t) \to t) \\
\text{NP:} & \quad \text{type } (e \to t) \to t \\
\end{align*}
\]

(Let’s work through these functional types on the blackboard)

Sometimes it is simpler to think about DET meanings in relational terms, as a relation between a CNP-type meaning and a VP-type meaning, using the equivalence between a function that takes a pair of arguments and a function that takes two arguments one at a time.

\[
\begin{align*}
\text{Every:} & \quad \text{as a relation between sets } A \text{ and } B \quad (\text{“Every } A \text{ B”}): \quad A \subseteq B \\
\text{Some, } a: & \quad A \cap B \neq \emptyset . \\
\text{No:} & \quad A \cap B = \emptyset . \\
\text{Most (not first-order expressible):} & \quad | A \cap B | > |A - B|. \\
\end{align*}
\]

\[
\begin{align*}
[[\text{Every}]](A,B) = 1 & \iff A \subseteq B. \\
[[\text{Some}]](A,B) = 1 & \iff A \cap B \neq \emptyset. \\
[[\text{No}]](A,B) = 1 & \iff A \cap B = \emptyset \\
[[\text{Most}]](A,B) = 1 & \iff | A \cap B | > |A - B|. \\
\end{align*}
\]

Determiners as one-place functions whose value is also a function:

But to mimic the structure of the English NP, we want every to combine with one predicate,
not with two predicates at the same time. Here’s the trick: we can define *every* as an expression that combines with a predicate to yield a predicate that combines with another predicate:

\[(5)\]

\(\llbracket \text{Every} \rrbracket (A)(B) = 1 \iff A \subseteq B\)

\(\llbracket \text{Some} \rrbracket (A)(B) = 1 \iff A \cap B \neq \emptyset\)

\(\llbracket \text{No} \rrbracket (A)(B) = 1 \iff A \cap B = \emptyset\)

*Every*\(\llbracket\) denotes a predicate of predicates: a set of predicates. We will call such a creature an *generalized quantifier*. A predicate *B* should be in the extension of *EVERY*\((A)\) iff *A* is a subset of *B*. Similarly for *Every*:

- Takes as argument a set *A* and gives as result \(\{B\mid A \subseteq B\}\): the set of all sets that contain *A* as a subset. Equivalently: \(\llbracket \text{Every} \rrbracket (A) = \{B\mid \forall x (x \in A \rightarrow x \in B)\}\)
  - With lambdas: \(\text{TR}(\text{every}_GQ) = \lambda Q \lambda P[\forall x (Q(x) \rightarrow P(x))]\)

*Some*, \(\alpha\): takes as argument a set *A* and gives as result \(\{B\mid A \cap B \neq \emptyset\}\).
  - With lambdas: \(\text{TR}(\alpha_GQ) = \lambda Q \lambda P[\exists x (Q(x) \& P(x))]\)

-- How would you express the meaning of *Not every* as a function that applies to a set *A* and gives some set of sets \(\{B\mid \ldots\}\)? *Most?*

**Note:** All this is really just a different way of looking at what we already said about determiners like *every* before, with lambdas. Montague treated all determiners that way, and in the first years of formal semantics, formal semanticists followed Montague’s idea of treating all NPs uniformly as “generalized quantifiers” of type \((e \rightarrow t) \rightarrow t\). We’ll talk next time about where the idea of treating NPs as having multiple types came from.

**Linguistic universal: Natural language determiners are conservative functions.** (Barwise and Cooper 1981)

**Definition:** A determiner meaning *D* is *conservative* iff for all *A,B*, *D*(A)(B) = *D*(A)(A \(\cap\) B).

Examples:

- No solution is perfect = No solution is a perfect solution.
- Exactly three circles are blue = Exactly three circles are blue circles.
- Every boy is singing = every boy is a boy who is singing.

“Non-example”: *Only* is not conservative; but it can be argued that *only* is not a determiner.

- Only males are astronauts (false) \(\neq\) only males are male astronauts (true).

**Theorem:** (Keenan and Stavi 1986, van Benthem 1986) Starting from *every* and \(\alpha\) as basic determiners, and building other determiner meanings by the Boolean operations of negation, conjunction, and disjunction, the resulting set of determiners consists of exactly the conservative determiners.

Suggested consequence: The conservativity universal is probably linked to the Boolean structure that is found throughout natural language semantics. It may be conjectured (Chierchia and McConnell-Ginet 1990) that we are mentally endowed with cross-categorial Boolean functions as the basic combinatory tool of our capacity for concept formation.
3. Quantifier scope ambiguity.

Let’s return to the problem described in Section 1, namely the ambiguity of the sentence (1a) *Every student read a book*. It seems to have just one (surface) syntactic structure, as shown in the tree (1b), but two semantic interpretations, with the predicate logic representations given in (2 i-ii).

How can we derive the two meanings of the whole compositionally from the meaning of the parts?

Here is (an informal statement of) Montague’s Quantifying In rule (Montague 1973); it is similar to the Quantifier-Lowering rule of Generative Semantics and Quantifier Raising (QR) of (May 1977, 1985); various alternative treatments of quantifier scope ambiguity exist, including Cooper-storage (Cooper 1975) and Herman Hendriks’s flexible typing approach (Hendriks 1988, 1993).

**Quantifying In Rule, Syntax:** (informally stated): An NP combines with a sentence with respect to a choice of variable (“*he*” in MG). Substitute the NP for the first occurrence of the variable; change any further occurrences of the variable into pronouns of the appropriate number and gender.

**Semantic rule:** NP’(λx[S’]) (“The set of properties denoted by the NP includes the property denoted by the λ-expression derived from the sentence.”)

We illustrate with two derivations for the ambiguous sentence *Every student read a book*.

**Syntactic derivation (i)** (rough sketch; read from bottom to top. Bold is used here to show which variables are substituted for at each step.)

S: every student read a book

NP: every student       S: he3 read a book

NP: a book       S: he3 read him2

Compositional Translation: (every student)’(λx3[(a book)’(λx2[read (x3, x2)])])

Rough paraphrase: Every student has the property that there is a book that he read.

If you write out the interpretations of the NPs and apply Lambda-Conversion as many times as possible, the result will be (some alphabetic variant of) the first-order PC formula ∀x(student(x) → ∃y(book(y) & read(x,y))).

**Syntactic derivation (ii)**

S: every student read a book

NP: a book       S: every student read him2

NP: every student       S: he3 read him2
Compositional Translation: (a book’)((\lambda x_2[(\lambda x_3 [\text{read (x_3, x_2)])])))

Paraphrase: Some book has the property that every student read it.

After applying Lambda-Conversion as many times as possible, the result will be (some alphabetic variant of) the first-order PC formula $\exists y (\text{book}(y) \land \forall x (\text{student}(x) \rightarrow \text{read}(x,y))).$

**Observation:** Compositional semantics requires that every ambiguous sentence be explainable on the basis of ambiguous lexical items and/or multiple syntactic derivations. Semantic structure mirrors syntactic part-whole structure, which in Montague Grammar is represented by syntactic derivational structure, not surface structure. There are different theories of the semantically relevant syntactic structure: “Derivation trees” or “analysis trees” (MG), LF (Chomskian GB or Minimalist theory), Tectogrammatic Dependency Trees (Prague), Deep Syntactic Structure (Mel’čuk) Underlying Structure (Generative Semantics), ... GPSG, HPSG, and various contemporary versions of Categorial Grammar are attempts to represent all the necessary syntactic information directly in a single “level” of syntax.

**References.**


May, Robert. 1977. The grammar of quantification, MIT: Ph.D.


