Supplement to Logic Unit: Logical Structure in Natural Language

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. Now let's look at English sentences containing quantifiers and see whether logic helps us analyze them semantically. The answer: yes, but first-order quantificational logic isn't enough if we want to do justice to the structure of natural language.

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). 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.

(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) → Walk (x))
    (ii) ∀y ( Book (y) & ∀x ( Student (x) → Read (x,y))

1 Using the lambda-calculus to help express higher-type interpretations is often helpful, but it’s not strictly essential, and we will manage to do without it here.
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:

\[\text{Every student walks.}\]
\[(\forall x)(\text{student}(x) \rightarrow \text{walk}(x))\]

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

Functional types: \(a \rightarrow b\): 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 \(e \rightarrow t\). 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.

CNP: type \(e \rightarrow t\)

VP: type \(e \rightarrow t\)

DET: 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: \((e \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t)\)

NP: \((e \rightarrow t) \rightarrow t\)

\(\neg\): type \(e \rightarrow t\)

\((\exists x)(\text{student}(x) \& \text{walk}(x))\)

\((\forall x)(\text{student}(x) \rightarrow \neg \text{walk}(x))\)

\((\exists x)(\text{student}(x) \& \text{walk}(x))\)

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.

\[\text{Every: as a relation between sets A and B ("Every A B")}: A \subseteq B\]
\[\text{Some, a}: A \cap B \neq \emptyset\]
\[\text{No}: A \cap B = \emptyset\]
\[\text{Most (not first-order expressible)}: |A \cap B| > |A - B|\]

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

This last sentence, No student walks, has two equivalent translations into predicate logic: the one in the tree above and \((\exists x)(\text{student}(x) \& \text{walk}(x))\). 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?
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:

\[(a) \quad \text{Every}(A) = \exists B (A \subseteq B)\]
\[(b) \quad \text{[Some]}(A) = \exists B (A \cap B \neq \emptyset)\]
\[(c) \quad \text{[No]}(A) = \exists B (A \cap B = \emptyset)\]

Every(A) denotes a predicate of predicates: a set of predicates. We will call such a creature a 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 \[\{B| A \subseteq B\}\]: the set of all sets that contain A as a subset. Equivalently: \[\text{Every}(A) = \{B| \forall x (x \in A \rightarrow x \in B)\}\]

Some: a: takes as argument a set A and gives as result \[\{B| A \cap B \neq \emptyset\}\].

How would you express the meaning of Not every this way? Most?

Applying a function to its argument (blackboard): Every (Student)

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


Examples:
No solution is perfect = No solution is a perfect solution.

Some a: only males are male astronauts (true).

Every boy is singing = every boy is a boy who is singing.

The semantic generalization discovered by (Ladusaw 1979) is that NPI's occur inside the argument of “monotone decreasing functions”. This notion is much more general than the notion of negation, and covers all of the above examples and many others; and it is an intrinsically model-theoretic concept — a real semantic property of the interpretation of the expressions, not a “formal” property of “representations” in some sort of “logical form”.

Definition (general):
A function f is monotone increasing if whenever a ≤ b, f(a) ≤ f(b).
A function f is monotone decreasing if whenever a ≤ b, f(b) ≤ f(a).

Application to determiner meanings: (Note: on the domains in our model, the basic ordering relation begins from the ordering on type t: 0 < 1; and for all types whose interpretations are sets, the corresponding notion of “less than” then becomes “subset of”. See Appendix 1 below for more details.)

Definitions:
A determinant D is right monotone increasing (sometimes called right upward entailing or monotone↑) iff whenever B ⊆ C, D(A)(B) entails D(A)(C).
A determinant D is left monotone increasing (left upward entailment or monotone↑) iff whenever C ⊆ A, D(A)(C) entails D(A)(B).
A determinant D is left monotone decreasing (left downward entailment or monotone↓) iff whenever A ⊆ C, D(C)(B) entails D(A)(B).
Illustrations: In a structure "Det CNP VP", the left position is the CNP argument, and the right position is the VP argument.

1. To show, for instance, that no is right monotone decreasing, we use a test like the following:
   (i) \( B \subseteq C \): knows Turkish and Chinese \( \sqsubseteq \) knows Turkish
   (ii) test entailment: No student knows Turkish \( \rightarrow \) no student knows Turkish and Chinese. Valid. So no is right monotone decreasing.

2. To show that no is left monotone decreasing, we use a test like the following:
   (i) \( B \subseteq C \): Italian student \( \subseteq \) student
   (ii) No student knows Urdu \( \rightarrow \) no Italian student knows Urdu. Valid.

3. Similarly we can show that some is right monotone increasing.
   (i) \( B \subseteq C \): knows Turkish and Chinese \( \subseteq \) knows Turkish
   (ii) Some student knows Turkish and Chinese \( \rightarrow \) some student knows Turkish.

4. Some is also left monotone increasing.
   (i) \( B \subseteq C \): student
   (ii) Some Italian student knows Urdu \( \rightarrow \) some student knows Turkish. Valid.

5. Interesting fact about every. While most determiners are like some and no in being either left and right increasing or left and right decreasing (so that it makes sense to call some "positive" and no "negative"), there are some determiners, of which the universal quantifier every is the most basic example, which have different properties for their left and right arguments.

   5a. Every is left monotone decreasing:
   (i) \( B \subseteq C \): student
   (ii) Every student knows Urdu \( \rightarrow \) Every Italian student knows Turkish. Valid.

   5b. Every is right monotone increasing:
   (i) \( B \subseteq C \): knows Turkish and Chinese \( \subseteq \) knows Turkish
   (ii) Every student knows Turkish and Chinese \( \rightarrow \) Every student knows Turkish.

The distribution of polarity items in the CNP part and the VP part of the sentences (4-9) above, and others like them, is accounted for by the monotonicity properties of the determiners in them. This account reinforces the analysis of determiners as functions which take a CNP as first argument, and the resulting NP interpretation (a generalized quantifier) as a function which takes the VP as its argument.

Appendix 1. Recursive definition of "≤" on semantic types that "end with a t".

Relevant background – (Partee and Rooth 1983), (Ladusaw 1980). (The first of these doesn't mention monotonicity, but it defines conjunction recursively on the same family of semantic types. There are considerable formal parallels in the two recursive definitions.) Both can be found in the useful collection (Portner and Partee 2002).

Semantic types: from the extensional part of Montague's intensional logic

Basic types: e, t (entities, truth values)
Recursively defined functional types: If \( a, b \) are types, then \( <a,b> \) is a type. \( <a,b> \) is the type of expressions which denote functions from a-type things to b-type things.

Negation as a sentence operator is of type \( <t,t> \). One-place predicates (nouns, some adjectives, maybe intransitive verbs) are of type \( <e,t> \). Generalized quantifiers are of type \( <<e,t>,t> \).

Semantic domains.
Start with a set D of entities and the set of two truth values \( \{0,1\} \). Then recursively define the domain of possible denotations \( D_a \) for expressions of any type \( a \).

\( D_a \) the domain of possible denotations for expressions of type t, is \( \{0,1\} \).
\( D_{a,b} \), the domain of possible denotations for expressions of type \( <a,b> \), is the set of all functions from \( D_a \) to \( D_b \).

Now we're ready to start the definition of "less than" that will be used in the definition of "monotone" across semantic types. First we need to define the set of "types that end with a t".

Recursive definition of "types that end with a t":
1. Type t "ends with a t."
2. If \( a \) is any type at all, and \( y \) is a type that "ends with a t", then \( <a,y> \) is a type that "ends with a t."

(The result is in fact the set of all types that have t as their last 'letter' symbol, with any number of other symbols preceding the 't'.)

Recursive definition of ≤:
Now we can define ≤ on the set of all types that end in a t.
1. 0 ≤ 0, 0≤1, 1=1. [In fact, we can divide ≤ into < and = in the natural way here.]
2. For any functional type \( <a,b> \), and any \( f, g \) in \( D_{a,b} \), \( f \leq g \) iff:
   For all \( x \) in \( D_a \), \( f(x) \leq g(x) \).

That's it. Let's see how it applies to types t, \( <e,t> \) and \( <<e,t>,t> \).

First, type t. This is the type of truth values, the extensions of sentences. Using \( [a] \) to represent the semantic value of \( a \), we can observe that for the case of sentences, \( [a] \leq [b] \) is equivalent to "a implies b" (material implication, defined by the usual truth table), because it's true when a and b are both true (1≤1), when they are both false (0≤0), or when a is false and b is true (0≤1). The only case where it's false is where a is true and b is false (NOT: 1≤0).

(Note: ≤ is not defined on type e, nor on any type that "ends with an e".)

Then type \( <e,t> \), the type of characteristic functions of sets of entities, the semantic values of one-place predicate expressions (nouns, simple adjectives, maybe simple intransitive verbs, some prepositional phrases, etc.) Let \( A, B \) be two expressions of type \( <e,t> \), e.g. two common noun
phrases. \(|A| \leq |B|\) is true iff for all d in D, \(|A|(d) \leq |B|(d)|\). But since \(\leq\) on sentences means "implies", this is another way of saying that A is a subset of B.

– So we've derived the fact that less-than on predicates means "subset of" from the fact that less-than on propositions means "implies".

Now type \(<e,t>,t>\): Let me go straight to model-theoretic terms, bypassing the expressions. For all P, P' in D\(<e,t>,t>\), P \(\leq\) P' iff for all Q in D\(<e,t>\), P(Q) \(\leq\) P'(Q).

Since the possible denotations of type \(<e,t>,t>\) are sets of sets, this again turns out to be the subset relation, though this time it's the subset relation among sets of sets rather than among sets of entities.

Monotonicity:

Having defined \(\leq\) across all the types that end with a t, we can define monotonicity for all functional types that end with a t which have arguments that also end with a t:

For any type \(<a,b>\) such that both a and \(<a,b>\) "end with a t", and any function f of type \(<a,b>\), f is monotone increasing iff for all x,y in Da,

if x \(\leq\) y, then f(x) \(\leq\) f(y).

References.


