Semantics

1. What is Semantics

Semantics is the study of how language relates to the world. To do semantics one first needs to be explicit about what sorts of things one assumes the world is made up from, that is, the primitives that will be used in a semantic description. What one assumes to exist is called an **ontology.** As a basic minimum ontology for studying the semantics of the simplest language fragments, we assume two types of things - individuals, i.e. concrete objects of all kinds, and truth values, i.e true and false. We will assume that bits of language refer to bits of the world made up from these ontological primitives. The bit of the world that a linguistic expression refers to is called its **denotation,** or **semantic value,** and is generally indicated by enclosing the linguistic expression in double square brackets. If we represent real world entities in *italics*, then we might have the following correspondences:

[[pete]] = *Peter John Whitelock*
[[the red block in the box]] = *block_37*
[[pete walks]] = *true*

2. Compositionality

What we want to achieve is a system in which primitive linguistic expressions have certain stipulated (i.e. listed) denotations, and compound linguistic expressions have denotations that are composed from the denotations of their parts in some consistent and well-defined manner. A system of this sort is said to obey the principle of **compositionality.** Given that we, as language speakers, understand an infinite number of sentences by understanding their parts and the rules for putting them together, compositionality seems incontrovertible.

To see how this might work, assume the following trivial grammar:

```
s -> np vp
vp -> iv
vp -> tv np
np -> ronnie
np -> nancy
np -> noam
iv -> is_stupid
iv -> is_a_male
tv -> loves
tv -> is_older_than
```

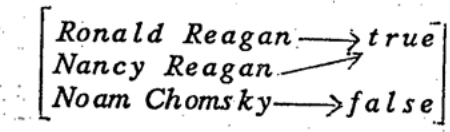
IV = INTRANSITIVE VERB
TV = TRANSITIVE VERB

We now define a **model** for this language. A model is a pair <A,F>, where A is a set of individuals and F is a function which assigns semantic values to the basic expressions. Here is a possible F for this language:

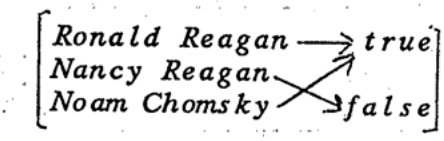
[[ronnie]] = *Ronald Reagan, ex-actor and President of the U.S.*
[[nancy]] = *Nancy Reagan, wife of Ronald*
[[noam]] = *Noam Chomsky, linguist and critic of U.S. foreign policy*

So much for the denotations of noun phrases. We will assume the denotations of intransitive verbs are functions from individuals to truth values, thus:

[[is_stupid]] =

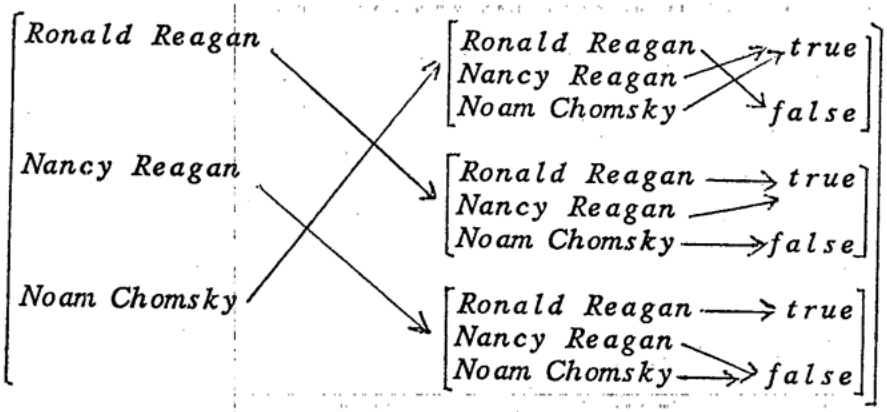


[[is_a_male]] =



Finally, we take the denotations of transitive verbs to be functions from individuals to (functions from individuals to truth values). We could have made the denotations functions from pairs of individuals to truth values, but in the interests of compositionality, we 'curry' this function.

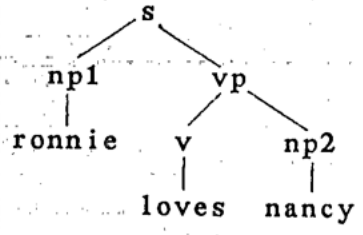
[[loves]] =



We then associate with each syntactic rule a semantic rule which says how the denotation of the whole phrase, i.e. the lhs, is constructed from the denotations of the component parts, i.e. the rhs. In the case of unary rules, the denotations of lhs and rhs are assumed to be the same. The binary rules are given the following semantics:

s -> np vp : [[s]] = [[vp]]([[np]])
vp -> tv np : [[vp]] = [[tv]]([[np]])

To illustrate this system in operation, let us parse a sentence according to the grammar. We will obtain a tree structure as below:



We can associate a semantic value with each node in this tree, as follows:

[[loves]] = as above
[[nancy]] = *Nancy Reagan*
[[v]] = [[loves]]
[[np2]] = [[nancy]]
[[vp]] = [[loves]]([[nancy]]) =

$$\begin{bmatrix} \textit{Ronald Reagan} \longrightarrow \textit{true} \\ \textit{Nancy Reagan} \\ \textit{Noam Chomsky} \longrightarrow \textit{false} \end{bmatrix}$$

[[ronnie]] = *Ronald Reagan*
[[np1]] = [[ronnie]]
[[s]] =

$$\begin{bmatrix} \textit{Ronald Reagan} & \textit{true} \\ \textit{Nancy Reagan} & \\ \textit{Noam Chomsky} & \textit{false} \end{bmatrix} ([[\text{ronnie}]])$$

= *true*

Thus the semantic value of the entire sentence is true with respect to the model given above. If we were to change the model then the semantic value of the sentence might change. Thus the linguistic expression "Ronnie loves Nancy" means what it does - in this case, true or false, by virtue of what the basic expressions mean, what the rules of combination mean, and the way the world (the model) is. Conversely, to know the meaning of a sentence is to know what the world would have to be like in order for it to be true. Such a view of semantics is called **truth-conditional.**

Notice that keeping the same semantic function (i.e assignment of semantic values to basic expressions of the language), but replacing the phrase structure syntax with a categorial lexicon and the rule of function application as described in HO7 makes this whole procedure much simpler. Instead of having to specify how the semantic value of a complex expression is obtained from the values of its parts, we just have the general rule:

if a linguistic expression α is built by applying β to γ, then $[[\alpha]]$ is obtained by applying $[[\beta]]$ to $[[\gamma]]$.

3. Logical Form

One of the obvious problems with the above is the unwieldiness of the semantic objects that are being manipulated. Another is that it would seem that we can say nothing about the meaning of a sentence without specifying everything about the world. A solution to both of these problems is the notion of **logical form (LF).** Logical form is an intermediate level of description between natural language and semantic values. Expressions of LF are associated with semantic values in the same way as above. Expressions of natural language are **translated** into expressions of LF in a compositional manner. Thus, in principle, the expressions of LF are dispensible, since they could be replaced by what they denote. But the expressions are much more compact than their denotata. Also, we can build the LF without having to say anything explicit about what it means (without giving a model for it). The advantages of using LF as the representation of the meaning of a sentence over using the sentence itself as a representation are several. First, it gives us a grasp on various semantic properties, such as ambiguity or paraphrase, that sentences do not. Secondly, it can be used for inference. A system of representation whose expressions were ambiguous or which did not support a proof procedure would have little claim to be called a logic.

Suppose that we decide to use FOPC as our semantic representation language. That is, we

intend the translations of NL sentences to be well-formed formulae of FOPC. If we wish to continue to use function application as the mechanism for building representations of constituents from representations of their parts, we will actually need a higher order language to represent the functions that are the intermediate results of the translation procedure. Thus we will enrich our logic with λ, the abstraction operator. For example:

λX[man'(X)] represents that function which when applied to an individual returns true if that individual is a man, and false otherwise.

λP[P(john')] represents that function which when applied to a predicate returns true if that predicate is true of john, and false otherwise.

λP[man'(john') Λ P] represents that function which when applied to a proposition returns true if that proposition is true and john is a man, and false otherwise.

A problem with our original model-theoretic semantics for an NL fragment was that [[np]] was taken to be an individual. While the only nps in our fragment were proper names, this was satisfactory. But we can hardly countenance noun phrases like "every man" denoting an individual. With the help of our logical representation language, we can now rectify this problem.

Suppose we wish to obtain the following translations of simple sentences containing quantified noun phrases:

(1) every man is_stupid	all(X)[man'(X) -> is_stupid'(X)]
(2) every man is_a_male	all(X)[man'(X) -> is_a_male'(X)]
(3) a woman is_stupid	exists(X)[woman'(X) Λ is_stupid'(X)]
(4) no woman is_a_male	¬ exists(X)[woman'(X) Λ is_a_male'(X)]

Let P denote what the vp in sentences 1 and 2, i.e either is_stupid or is_a_male, denotes. Then these two sentences will be true if and only if:

(5) all(X)[man'(X) -> P(X)]

Therefore we can take the denotation of "every man" to be that function which when applied to a predicate P returns true iff (5), i.e.

λP all(X)[man'(X) -> P(X)]

Similarly, the denotation of "every woman" will be that of

λP all(X)[woman'(X) -> P(X)]

Thus if we let Q denote whatever the noun in these two noun phrase denotes, we will obtain the following value for "every":

λQ λP all(X)[Q(X) -> P(X)]

By the same line of argumentation we obtain the following translations for "a" and "no" respectively.

λQ λP exists(X)[Q(X) Λ P(X)]
λQ λP ¬ exists(X)[Q(X) Λ P(X)]

Now the pairs of syntactic and semantic rules we require for building noun phrases and

sentences and their corresponding semantic values are:

s -> np vp : [[s]] = [[np]]([[vp]])
np -> det n : [[np]] = [[det]]([[n]])

To give a consistent translation to nps, it no longer suffices to treat even proper names as denoting individuals. Instead we treat them as denoting functions from properties to propositions (i.e. functions from (functions from individuals to truth values) to truth values. If we have an individual *Noam Chomsky*, represented by the logical constant 'noam'', then the linguistic expression 'noam' denotes, not 'noam'', but λP[P(noam')] that is, the set of properties that are true of that individual. But the result of applying this value as a function to the value of the vp is identical to applying the value of the vp to the individual, thus:

λX [is_stupid'(X)](noam') = is_stupid'(noam')
λP [P(noam')] (λX [is_stupid'(X)]) =
λX [is_stupid'(X)](noam') = is_stupid'(noam')

Once again, notice that what is going on here follows much more closely what goes in a categorial-style syntax. That is, type-raising a noun phrase from np to s/(s\ np) is exactly analogous to having noun phrases denote not individuals but functions from properties (as denoted by s\ np) to propositions (as denoted by s). Thus it seems that type-raising, which was originally introduced to capture some syntactic facts about coordination etc., is actually required for sound semantic reasons as well.