Type-driven interpretation

Jeff Speaks
PHIL 43916

September 10, 2014

1 Simplifying our theory

Recall that in our discussion of our semantic theory last time, we noted that the theory incorporates, for every syntactic rule, a corresponding semantic rule. This suggests that someone learning the language will have to, in addition to mastering the rules by which expressions can be grammatically combined, associate with each of these syntactic rules a separate semantic rule.

Once you put things this way, you might think that it would be vastly better if we could simplify things, and not have to come up with a separate semantic rule for every rule of grammatical combination. There's an analogy here with theory construction in, for example, physics – it is vastly better to have a theory which explains the motion of all physical things than a theory which needs to posit separate laws to explain the movement of different types of physical objects. (One for planets, one for bricks, one for rabbits . . .)

Today our focus will be on the attempt to simplify our semantic theory in something like this way.

2 Modifying our assignment of semantic values

Remember the semantic rule corresponding to 'it is not the case that':

(c) $[\![\ [_{\mathrm{S}}\ \text{neg S}]\ ]\!] = [\![\text{neg}]\!]\ ([\![\text{S}]\!])$

The idea was that $[\![\text{neg}]\!]$ is a function from truth-values to truth-values, and that the semantic value of $[_S$ neg S$]$ is the value of that function given $[\![\text{S}]\!]$ as argument.

The most ambitious attempt to simplify semantic theory would hold that this rule – functional application – is the only, or almost the only, semantic rule.

To make this work, we're obviously going to have to change our assignments of semantic values to our basic vocabulary. Consider, for example, our view about $[\![\text{V}_\text{i}]\!]$. We took, for example, $[\![\text{is boring}]\!]$ to be $\{x\colon x \text{ is boring}\}$. But this won't do if we want our only semantic rule to be functional application, since

Pavarotti is boring.

is a grammatical expression of our language, and neither $[\![\text{Pavarotti}]\!]$ nor $[\![\text{is boring}]\!]$ is a function. And if neither of x, y are a function, then neither will be an argument for the other, and functional application can't get started.

The basic idea is this: our two basic categories are sentences and names. As before, the semantic value of a sentence is a truth-value, and the semantic value of a name is the object for which it stands. But the semantic value of everything else in our language will be a function. Let's go through some examples to see how this might work.

2.1 Intransitive verbs

Let's start with V_i's like 'is boring.' What function might make a good semantic value for expressions of this sort?

Well, we know that in simple sentences like the above VPs inherit the semantic value of the relevant V_i. And we know that combining a N with a VP makes an S. So, if functional application is going to be our only semantic rule, we know that $[\![\text{V}_\text{i}]\!]$ must be something which can combine by functional application with an object to yield a truth-value. Hence it is natural to think that, on the present sort of approach, $[\![\text{V}_\text{i}]\!]$ is a function from objects to truth-values.

Let's use 'e' to stand for entity – the sort of thing which can be the semantic value of a name – and 't' to stand for truth-value. Then the present view is that the semantic value of an intransitive verb is a function of type $\langle$e,t$\rangle$ – a function from entities to truth-values.

Which function of this sort is it? Intuitively, $[\![\text{is boring}]\!]$ will be the function which gives value true for any argument which is boring, and the value false otherwise. I.e.:

$[\![\text{is boring}]\!]^v$ = the function f such that f(x)=1 if $x \in \{x\colon x \text{ is boring in v}\}$ and f(x)=0 otherwise

This is the *characteristic function* of the set of boring things: it delivers one value – 1 – for all of the things in the set, and another – 0 – for all of the things not in the set. The proposal, then, is that rather than assign sets as the semantic values of intransitive verbs, we use the characteristic functions of those sets.

Why do this? Again, the point is that by doing this we avoid having to appeal to different semantic rules at each turn: one for combining ⟦N⟧ with ⟦VP⟧, another for combining ⟦V$_i$⟧ with ⟦N⟧, and so on.

2.2 Transitive verbs

As before, things get a bit hairier when we get to transitive verbs. The only semantic values in our system are objects, truth-values, and functions; so we know that the semantic value of a V$_t$ must be a function. But a function from what to what?

To answer this question we think about what transitive verbs do, grammatically, in our language. They combine with N's to form VP's. So – given that ⟦N⟧ is an entity – their semantic value must be a function from entities to whatever the semantic value of a VP is. Since the semantic value of a VP is itself a function from entities to truth values, the semantic value of a V$_t$ must be a function from entities to a function from entities to truth-values – that is, a function of type ⟨e,⟨e,t⟩⟩.

Exactly which function of this type will ⟦likes⟧ be? Intuitively, you might think of this as a function from two objects to a truth-value – one that returns the value 'true' iff the first likes the second. But this is no good if we want our only semantic rule to be functional application. So we need to reduce this two-place function – i.e., a function which takes a pair of arguments – to two one-place functions. This method is known as *currying* a function.

The basic idea is this: we begin with the two-place function mentioned above, which can be defined as follows:

> f(x,y)=1 iff x likes y, and 0 otherwise.

Intuitively, what we are looking for is a pair of functions to do the job of f. The way to find them is to let the first function g be a function from an object x to the function h which is such that, for any object y, h(y)=1 iff f(x,y)=1. That is:

> g(x)=the function h which is such that h(y)=1 iff y likes x

or, equivalently,

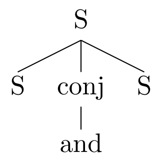
> g(x)=the function h which is such that h(y)=1 iff $y \in \{z\colon z$ likes x$\}$

2.3 Operators

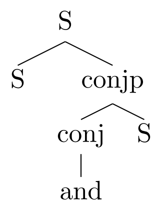
Extending our new semantics to 'it is not the case that' is pretty easy; we can just leave its semantic value as a function of type ⟨t,t⟩. But our sentence connectives are a little tricky;

we must reduce their semantic values to one-place functions if we want to maximize the simplicity of our semantic rules.

To get the simplification of our semantics that we're looking for, we have to slightly complicate our syntax, by introducing a new semantic category, which in the text is called 'conjP.' Rather than treating conjunctive sentences as having the structure



we treat them as having the structure



Given that we are viewing the sentences this way, what must $[\![\text{conjp}]\!]$ and $[\![\text{and}]\!]$ be?

3 Simplifying the interpretation function

In the text, this is called 'type-driven' as opposed to 'rule-to-rule' semantics. The reason for this is that all of the heavy lifting in the semantics is done by an assignment of the right sorts of semantic values to the different syntactic types. Once this is done, then functional application does the rest; there's no need to specify different rules for different syntactic constructions.

In fact, we can summarize the interpretation function for our new semantics – i.e., the list of rules for computing the semantic values of complex expressions from the semantic values of simpler ones – using just these two rules:

(51) a. Pass up

If Δ is a non-branching node whose only child is a, then $[\![\Delta]\!]^v = [\![\text{a}]\!]^v$

(51) b. Functional application

If Δ is a branching node with children a and b and $[\![\text{a}]\!]^v$ is a function which takes as argument entities of type $[\![\text{b}]\!]^v$, then $[\![\Delta]\!]^v = [\![\text{a}]\!]^v([\![\text{b}]\!]^v)$

You should be able to see how much simpler this is than the semantic theory with which we began. We have one rule for non-branching nodes, another for branching nodes – and that's it.

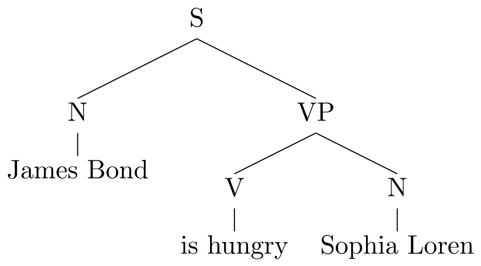
As mentioned in the text, this approach to semantics may also allow us to simplify our syntax by, for example, eliminating the need for a distinction between transitive and intransitive verbs in favor of a single syntactic category V. The relevant syntactic rules could then just be

$$\text{VP} \rightarrow \text{V(N)}$$
$$\text{VP} \rightarrow \text{V}$$

i.e., a rule which tells us that you can form a VP by taking any V, and either concatenating it with an N, or not. This will mean that no syntactic rules will prohibit the formation of ungrammatical strings like

James Bond is hungry Sophia Loren.

But the ungrammaticality of these strings can now be explained not by positing extra semantic rules, but via our claim that the interpretation function – i.e., the set of our rules for semantically combining different expressions – is exhausted by Pass up and Functional Application. For when we look at the structure of the above we get



But when we try to get the semantic value of the VP, we end up combining an e with a function of type ⟨e,t⟩, which gives us a truth-value for the VP. And then we're stuck with the S node with daughters of type e and type t – and neither of these is a function, and so we can't combine them to get any semantic value at all for the sentence. The fact that the sentence is, in this sense, uninterpretable might be used to explain its ungrammaticality.

A puzzle for type-driven interpretation: what does our interpretation function say about cases in which we introduce a name for a function of type ⟨e,t⟩, and then concatenate this with another (normal) name? Or a case in which we concatenate it with a predicate whose semantic value is also of type ⟨e,t⟩? These cases might suggest, respectively, that we can't derive all the facts about well-formedness from the system we've been developing, and that we can't always derive the right truth conditions from Functional Application without adding in – as a rule-to-rule approach would do – facts about whether a given semantic value is the value of the name or the predicate in a sentence.

...

Type driven approaches to semantics have important theoretical advantages, but are often harder to work with; it's more difficult to think about ⟦likes⟧ and ⟦and⟧ on the new approach, for example. So going forward we will not always try to fit our theory into a semantics which uses only functional application. In many cases, it will be clear how our theory could be modified to fit with the present approach.