Lambda abstraction and its uses

Jeff Speaks
PHIL 43916

December 11, 2014

1 The lambda operator

Our goal today is to introduce a new operator into our theory – the lambda operator, symbolized by the Greek letter 'λ' – and show some of the natural language constructions it can be used to analyze.

We've discussed a few different sorts of operators so far. Negation combines with a sentence to form a sentence; conjunction and disjunction combine with a pair of sentences to form a sentence; and the complementizer 'that' combines with a sentence to form a complementizer phrase. 'λ' combines with sentences to form predicates (verb phrases).

Intuitively, and very roughly, the semantic value of

$\lambda x(\psi)$

is the property of being such that ψ is true. So, for example,

λx(Grass is green)

is the property of being such that grass is green – in our semantics, the set of individuals which are such that grass is green. Why do we write 'λx' rather than just 'λ'? Because the principal purpose of this operator is to combine with formulae which contain an unbound (free) variable, like 'x is green or x is blue.' The formula

λx(x is green or x is blue) stands for, intuitively, the property of being green or being blue. So its intension will be a function from worlds and times to sets, and its semantic value will be the set of things which are either green or blue. So, like intransitive verbs, expressions of this sort can combine with names to form sentences.

How, exactly, should we understand the semantics of this operator? One way to state the semantic value of expressions containing it is the following:

$$\lambda x[\psi]\text{t} \iff \psi[\text{t/x}]$$

where 'ψ[t/x]' stands for the sentence obtained by replacing every occurrence of 'x' with the term 't'. So, applying this to our example above would give us

$$\lambda x(x \text{ is green or } x \text{ is blue})\text{t} \iff (\text{t is green or t is blue})$$

We can divide this biconditional into its two directions:

Lambda reduction

$\lambda x[\psi]\text{t} \rightarrow \psi[\text{t/x}]$

Lambda abstraction

$\psi\rightarrow \lambda x[\psi[\text{x/t}]]\text{t}$

These are schemata; when we say that these are rules governing 'λ', what we are saying is that no matter what formula you plug in for ψ and no matter what term you plug in for t, you get a true sentence.

We can also write out its semantics in more familiar terms as

$$[\![\ \lambda x\text{S}]\!]^{M,w,i,g} = \{\text{u}\in\text{U: } [\![\text{S}]\!]^{M,w,i,g[u/x]}\text{=1}\}$$

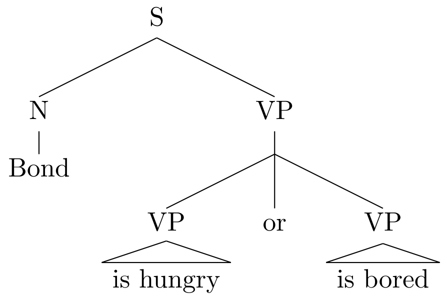
Now that we have this operator on the table, let's see what it might be good for.

2 VP conjunction and disjunction

Consider the sentence

Bond is hungry or is bored.

From the point of view of the semantic theory we have been developing, this sentence is a little puzzling. One wants to provide a tree something like



But the problem is that the meaning of 'or' is a function from a pair of truth-values to a truth value; this simply gives us no way of computing the semantic value of the highest VP node on the basis of $\llbracket$is hungry$\rrbracket$ and $\llbracket$is bored$\rrbracket$, which are sets of individuals.

A natural thought is that we can reduce predicate disjunction and conjunction to ordinary sentence-level disjunction and conjunction, by taking the logical form of the above sentence to be

> Bond is hungry or Bond is bored.

which has an unproblematic structure. The problem is that this sort of move seems not to be always available. Consider the sentences

> Everyone is hungry or is bored.
>
> Someone is bored and is hungry.

Pursuing the strategy sketched above would give us the wrong truth conditions for each.

This suggests that we need some way of understanding how conjunction and disjunction can be used to form complex VPs; ideally, we should be able to do this while avoiding the surprising conclusion that the meaning of 'and' as it occurs in predicates is wholly unrelated to the meaning of 'and' when used as a sentence connective.

Our lambda operator provides us with one way to do this. We might provide something like the following rules for predicate conjunction and disjunction:

$$\llbracket_{\text{VP}} \text{ VP}_1 \text{ or VP}_2\rrbracket = \llbracket\lambda x[\text{VP}_1(x) \text{ or VP}_2(x)]\rrbracket$$
$$\llbracket_{\text{VP}} \text{ VP}_1 \text{ or VP}_2\rrbracket = \llbracket\lambda x[\text{VP}_1(x) \text{ and VP}_2(x)]\rrbracket$$

where 'or' and 'and' on the right hand side of these identities are interpreted in the way to which we are accustomed, as sentence connectives. This does not avoid the conclusion that 'and' has a different semantic value when used as a predicate connective than when used as a sentence connective — but it does show how the two uses are systematically connected.

Here's a complication with this story. Consider a case in which we have a complex predicate one part of which contains a quantifier, as in

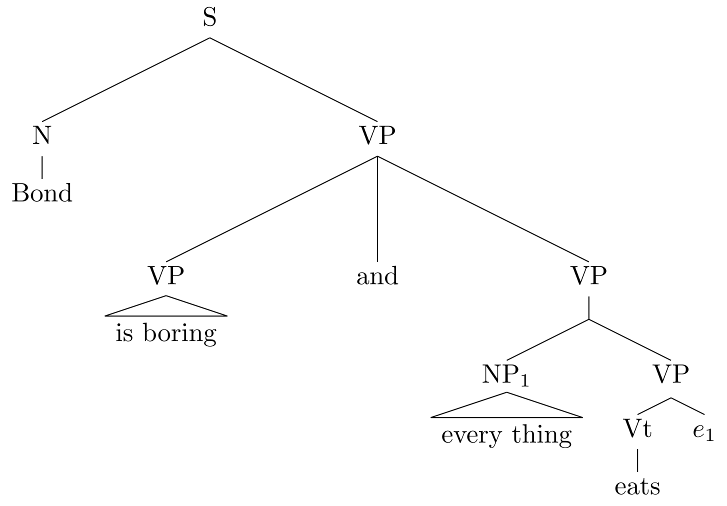
Bond is boring and eats everything.

So far we've understood quantifier phrases like 'everything' as combining with sentences to form sentences. But the only way to fit the above sentence into that form would be for the quantifier to move out of the complex predicate to take wide scope over the whole sentence; and this (given the above treatment of predicate conjunction) would seem to violate our constraints about quantifiers moving out of conjuncts, which we used to explain the ungrammaticality of sentences like

Everyone went to the store and he was disappointed.

This means that we have to treat 'everything' as combining with the verb 'eats'. What rule might you give for $[\![$[every β] VP$]\!]$?

This is not completely spelled out in the text, but the rough idea is this. Our tree might look something like like this:



The question is how we get the semantic value of 'eats everything' by combining a quantified noun phrase and a VP. The rule suggested in the text is this: we begin by adding a trace to 'eats e_1' to give us the formula 'x eats e_1.' We then have a formula which enables us to use our usual rule for combining NP's with sentences. This tells us that the semantic value of the combining the quantified noun phrase with our formula=1 iff for every $u \in U$, $[\![\text{x eats } e_1]\!]^{M,w,i,g[u/e_1]}$=1. But of course we want this VP node to have a set of things, not a truth value, as its semantic value. So our semantics requires us to use lambda abstraction to give us as the semantic value of the VP the set of things which are the semantic value of

λx (for all $u \in U$, $x \in \{y{:}< y, u > \in [\![\text{eats}]\!]\}$)

which will be the set of things that eat everything – which is what we want.

This might seem a bit ad hoc. But some evidence that in sentences like this the quantifier does attach to the VP is given by the fact that whereas

> A napkin was next to every plate.

is ambiguous (and favors the reading on which 'every plate' has wide scope), the sentence

> A napkin was next to every plate but had already been used.

forces the reading on which 'a napkin' has wide scope. The idea would be that this is explained by the constraint on movement of 'every plate' out of conjuncts.

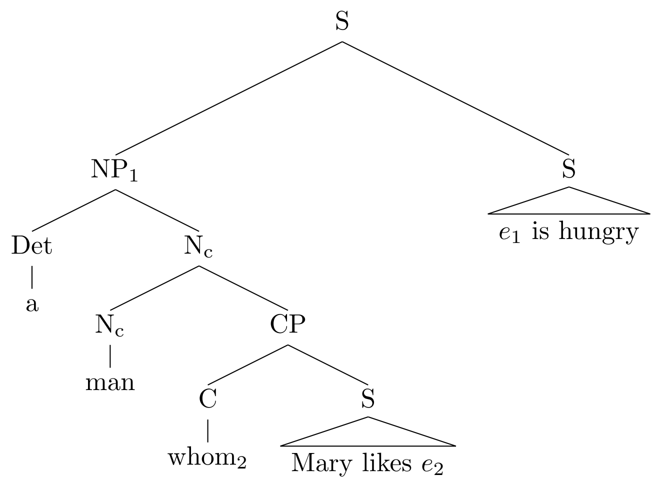
3 Relative clauses

The following are some example sentences involving relative clauses:

> A man *whom Mary likes* is hungry.
>
> A man *that likes Mary* is bored.

It looks like these clauses are functioning as predicates; in particular, they look like predicates derived from sentences. This makes it very natural to use the lambda operator in the analysis of these clauses.

Consider the first sentence above. We might (simplifying a bit the view defended in the text) take its tree to be something like



There are two steps the computation of the semantic value of this tree that need comment.

The first is the way in which we derive the semantic value of the CP from 'whom' and the S. The lambda operator, intuitively, gives us a way to represent a set of things which corresponds in a certain way to a sentence. It is the set of things which are such that, when those things are made the value of a certain node in the relevant sentence, the sentence comes out true. The role of 'whom' in the above sentence is to combine with a sentence to yield a CP whose semantic value is the set of things which, when supplied as the semantic value of the trace with which 'whom' is co-indexed, makes that sentence true. So, for example, if the semantic value of the S node is

$$[\![\text{Mary likes } e_2]\!]$$

then the semantic value of the CP will be

$$[\![\lambda y(\text{Mary likes } y)]\!]$$

i.e.

$$\{y\text{: } \langle \text{ Mary}, y \ \rangle \in [\![\text{likes}]\!]\}$$

Here it looks like lambda abstraction is playing an essential role, since this set is not the semantic value of any node in the tree for 'Mary likes e_2.'

This gives us a set of individuals – the set of individuals that Mary likes – as the semantic value of the CP. How do we combine this with $[\![\text{man}]\!]$ to get the semantic value of the parent NP?

We again use lambda abstraction, in the way just illustrated by our discussion of predicate conjunction and disjunction. The idea is, very roughly, that

$$[\![\mathrm{N_c}]\!] = [\![\lambda x(\text{man}(x) \ \& \text{ whom Mary likes}(x))]\!]$$
$$= \{x\text{:x} \in [\![\text{man}]\!] \text{ and } x \in \{y\text{: } \langle \text{ Mary}, y \ \rangle \in [\![\text{likes}]\!]\}\}$$