Functions, Types, and Lambdas

Document for Linguistics 426 / Philosophy 426

Fall, 2015

Instructor: Richmond Thomason

Version of: October 12, 2015

Note: This version needs proofreading and some expansion at the end.

1. Functions

You understand what a function is by understanding what it does: it associates a single *output* with one or many *inputs*, or (equivalently) a single *value* with one or many *arguments*.

We are most familiar with functions from mathematics. Doubling a number, or multiplying a number by 2, is a function f: $f(3) = 6$, $f(47) = 92$. Adding two numbers is a function g: $g(2,5) = 7$, $g(5,7) = 12$. But wherever there are individuals of any kind, there are functions. The relation h between people and their birthdays is a function; h(George Washington) = February 22, 1732 and h(Abraham Lincoln) = February 12, 1809.

There is no reason, then, why functions can't input and output functions. In fact, let f be the function that inputs a number n and outputs the function that inputs a number m and outputs the result of adding n to m: if $f(n)=g$ then $g(m)=n+m$. For instance, $f(2)$ is the doubling function, so $[f(2)](3)=6$.

This breaks the addition function h, which takes two inputs, into two functions, each of which takes just one input:

$$h(n,m) = [f(n)](m).$$

This trick will make it possible for us to confine our attention to one-place functions only.

2. Functional types

One of the standard solutions to the Russell Paradox insists that every function must have a type, and that the types are layered, or arranged in a hierarchy. At the bottom of the hierarchy, we have a domain of *individuals*; these are things, that for purposes of the theory at hand, are not treated as functions. It is also convenient to have a bottom-level domain of truth-values: this consists of only two objects: $\top$ (true) and $\bot$ (false).

Whenever you have domains D_1 and D_2, you also have a domain consisting of all the functions that input elements from D_1 and output elements in D_2. And this is the mechanism that creates the hierarchy. From the primitive domains of individuals and truth values, we have a domain of functions from individuals to truth values. We then have a domain of functions from functions from individuals to truth values to truth values. In fact, the process yields an infinite family of functional domains, each of which is typed by the type of its inputs and the type of its outputs.

This idea is captured by the following recursive definition of a *type*.[1]

[1]The use of 'e' for the type of individuals was introduced by Richard Montague. See [Montague, 1972, Montague, 1973]. Montague called individuals "entities."

(1) 1. e and t are types.
    2. If σ and τ are types, then $\langle \sigma, \tau \rangle$ is the type of functions from type σ to type τ.

According to this definition, for example:

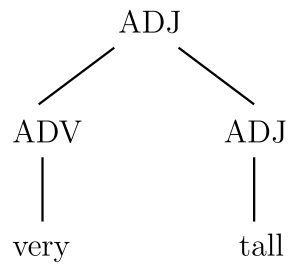
(2) 1 $\langle \mathsf{e}, \mathsf{t} \rangle$ is the type of functions from individuals to truth-values.
    2. $\langle \mathsf{e}, \mathsf{e} \rangle$ is the type of functions from individuals to individuals.
    3. $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \mathsf{t} \rangle$ is the type of functions from functions from individuals to truth values to truth values. (As you can see, very quickly the English descriptions of these types becomes harder to understand than the notation for them.)
    4. $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \langle \mathsf{e}, \mathsf{t} \rangle \rangle$ is the type of functions from functions from individuals to truth values to functions from individuals to truth values.

This type hierarchy consists of functions of various sorts: strictly speaking, there are no sets. But in fact there is no important difference between a set of individuals and a function of type $\langle \mathsf{e}, \mathsf{t} \rangle$. A set of individuals amounts to a separation of the individuals into those that belong to the set and those that do not. A function f from individuals to truth values provides a similar separation: if $f(\mathrm{d}) = \top$, d is "in" and if $f(\mathrm{d}) = \bot$, d is "out."

This makes it easier to understand some of the types listed in Example (2). $\langle \mathsf{e}, \mathsf{t} \rangle$ is the type of sets of individuals. $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \mathsf{t} \rangle$ is the type of sets of sets of individuals $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \langle \mathsf{e}, \mathsf{t} \rangle \rangle$ is the type of functions from sets of individuals to sets of individuals. This makes it easier to see which functional types might be appropriate for interpreting the syntactic types you find in natural language. For instance, if you decide that semantically, adjectives correspond to sets of individuals, then you will want to say that adjectives have the functional type $\langle \mathsf{e}, \mathsf{t} \rangle$.

Often, you can work out what type is appropriate for a syntactic category. Some adverbs, for instance, modify adjectives. The natural structure for 'very tall', for instance, is this.

(3)



This picture suggests that an adjective-modifying adverb operates on an adjective meaning and produces an adjective meaning. If we have decided that adjectives have type $\langle \mathsf{e}, \mathsf{t} \rangle$ then adverbs like 'very' will have type $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \langle \mathsf{e}, \mathsf{t} \rangle \rangle$.

This arrangment allows us to use functional application to interpret the syntactic pictured in (3). $[\![\text{very}]\!]$ will be a function f of type $\langle \langle \mathsf{e}, \mathsf{t} \rangle, \langle \mathsf{e}, \mathsf{t} \rangle \rangle$. $[\![\text{tall}]\!]$ will be a function g of type $\langle \mathsf{e}, \mathsf{t} \rangle$. From the types of f and g, we can see that f must apply to g and will return an output of type $\langle \mathsf{e}, \mathsf{t} \rangle$.

The interpretation of Rule (3), then, looks like this.

(4)
$$\begin{tikzcd}[column sep=small] & {\llbracket\text{very tall}\rrbracket = \llbracket\text{very}\rrbracket(\llbracket\text{tall}\rrbracket)} \arrow[dl, dash] \arrow[dr, dash] & \\ {\llbracket\text{very}\rrbracket} & & {\llbracket\text{tall}\rrbracket} \end{tikzcd}$$

This simple and satisfying arrangement can work only if it makes sense to say that the set of things that are very tall can be recovered from the set of things that are tall. One issue here is how to deal with the vagueness in ‘very’ and ‘tall’. Even setting this aside, it turns out that this assumption isn’t entirely plausible. This doesn’t mean that we need to give up the idea of functional interpretations, but it does indicate that we can’t interpret adjectives as sets of individuals, and need to look for an interpretation that carries more information.

3. Type assignments and semantic interpretation

We have been assuming that expressions belonging to the same syntactic category are to have semantic interpretations of the same functional type: if α and β are both NPs, for instance, then the types of $\llbracket\alpha\rrbracket$ and $\llbracket\beta\rrbracket$ will be the same. This idea, it seems, can be carried out without requiring any significant changes in familiar syntactic rules. It makes sense, then, to suppose that each syntactic category is associated with one of the semantic types produced by Definition (1).

As the example of ‘very tall’ indicates, we often can figure out how to assign functional types to syntactic categories by looking at the syntactic rules that apply to these categories and seeing what type assignments are needed in order to arrive at simple interpretations of syntactic rules.

4. Lambda expressions and the logic of types

There is a logic that goes along with functional types; this logic was formulated by Alonzo Church in [Church, 1940] and represents many years of simplifying the ideas behind the 1910 edition of *Principia Mathematica*, [Whitehead and Russell, 1910 1913]. Richard Montague added an intensional component to Church’s 1940 type theory and used it in works like , which mark the beginning of formal semantics.

As well as the usual apparatus of boolean logic (negation, disjunction, conjunction, the conditional) and identity, functional type theory has variables of each type. For instance, since $\langle \mathsf{e}, \mathsf{t}\rangle$ is a type, there will be variables like $x_{\langle \mathsf{e},\mathsf{t}\rangle}$. Universal and existential quantifiers can be used with these typed variables. So the logic allows you to formulate things like (5), which says that for each individual there is a set containing nothing but that individual.

$$(5)\ \forall x_{\mathsf{e}} \exists z_{\langle \mathsf{e},\mathsf{t}\rangle} \forall y_{\mathsf{e}} [z(y) = \top \leftrightarrow y = x]$$

But this logic also has a distinctive and new logical construct: *functional abstraction* or *lambda abstraction*, which provides a systematic way to construct expressions that denote functions.

Let’s go back to numerical examples for a moment. When we write a term like x^2 or $x^2 + 3$ we are expressing functional dependence. As the variable ‘x’ takes on various values, ‘x^2’ and ‘$x^2 + 3$’ take on values that systematically depend on these values. When we want to talk about these functions, we use the original terms: we say "x^2 is the squaring function" or (more awkwardly) "$x^2 + 3$" is the operation of squaring a number and adding 3 to the result."

This way of talking produces an ambiguity: 'x^2' can either refer to a number, the result of squaring x, or it can refer to a *function*, the operation of squaring x. This ambiguity does no great harm in informal mathematics, but logical formalisms need to dispense with it. In type logic, 'x^2' would unambiguously refer to the number we get by squaring x, and $\lambda x\ x^2$ would unambiguously refer to the function that inputs any number x and outputs x^2.

If α is a logical expression of type τ and x is a variable of type σ, $\lambda x\ \alpha$ is an expression of type $\langle\sigma,\tau\rangle$ and refers to the corresponding function from objects of type $\langle\sigma,\tau\rangle$.

For instance, if f and g have type $\langle \mathsf{e},\mathsf{t}\rangle$—the type of sets of individuals—then $\lambda x_{\mathsf{e}}[f(x) \wedge g(x)]$ denotes the intersection of f and g, and $\lambda x_{\mathsf{e}}[f(x) \vee g(x)]$ denotes their union. And $\lambda f_{\langle \mathsf{t},\mathsf{t}\rangle}\lambda x_{\mathsf{t}}\neg(x)$ denotes the function that inputs a one-place truth function and outputs its negation.

Returning again to mathematical examples, recall that $\lambda x\ x^2$ is the squaring function. So if we apply this function to a number, say 5, we get 5^2 or 25:

$$(6)\ \ [\lambda x\ x^2](5) = 5^2.$$

(6) is an instance of the *principle of lambda conversion.* On the left side of the equation is a lambda expression, denoting a function, is applied to a constant, denoting an input to the function. So $[\lambda x\ x^2](5)$ denotes the output this function returns for the input 5. The right side of the equation is obtained by *substituting* '5' for 'x' in 'x^2'

The principle of lambda conversion generalizes this.

(7) Principle of Lambda Conversion:

Let α be an expression of type τ, so that $\lambda x_\sigma \alpha$ has type $\langle\sigma,\tau\rangle$. Let β be an expression of type σ, so that $[\lambda x_\sigma \alpha](\beta)$ has type τ. Finally, let $\alpha[\beta/x]$ be the result of substituting β_σ for x_σ in α. Then the following equation will be true:

$$\lambda x_\sigma \alpha = \alpha[\beta/x_\sigma].$$

5. NPs as generalized quantifiers

We are going to see how functional types can resolve an apparent mismatch between the syntax and semantics of noun phrases.

Two sorts of expressions that from a semantical point of view look very different can appear in noun phrase positions: (1) on the one hand, proper names, pronouns, demonstratives, and definite descriptions, and (2) on the other hand, quantified NPs, such as 'every registered voter', 'a woman from Boston', and 'many elderly patients'.

There are very good reasons for thinking that, from a semantical standpoint, these two kinds of expressions are very different. For one thing, the traditional logical formalizations of the two are strikingly different. Compare, for instance, the way that 'Jill voted' and 'Every registered voter voted' would be formalized.

(8.1) Voted(Jill)
(8.2) $\forall x[[\text{Voter}(x) \wedge \text{Registered}(x)] \rightarrow \text{Voted}(x)]$

The difference between (8.1) and (8.2) is accompanied by dissimilarities in the intuitive meanings. (8.1) is a proposition about a single voting event, involving a single individual. If there are 10,232 registered voters, (8.2) has to do with 10,232 voting events at various times and maybe in various places, each with a different voter.

Finally, there are important inferential differences. For instance, (9.1) is a logical truth, while we would expect (9.2) to be false in most elections.

(9.1) Jill voted or Jill didn't vote.
(9.2) Every registered voter voted or every registered voter didn't vote.

These differences seem to suggest that the "referential" NPs and the "quantificational" NPs must have different functional types. However, it turns out that if we think things through beginning with the apparently more complex case—the quantificational NPs—we will then have an account that will also work for the simpler case—the referential NPs. Logical differences like the one between (9.1) and (9.2) will follow from special properties of referential NPs that do not, however, have to do with differences between the functional types of these NPs and the types of quantificational NPs.

This solution was first proposed in [Montague, 1973] and accounts for the title of that paper: "The Proper Treatment of Quantification in Ordinary English."

To begin with, we need to decide what the functional type must be of an NP like 'every registered voter'. In subject position, such an NP combines with a VP to form a sentence. Suppose sentences have the functional type t and VPs have the functional type $\langle \mathsf{e}, \mathsf{t} \rangle$. This is natural enough: sentences are true or false, and VPs like 'voted' are true or false of single individuals.

So, whatever the functional type of a quantified NP is, this type must combine with $\langle \mathsf{e}, \mathsf{t} \rangle$ to produce a truth value: an item of type t. We could achieve this simply by taking the type of these NPs to be e, since functions of type $\langle \mathsf{e}, rt \rangle$ applied to individuals (type e) output items of type t. But to assign quantified NPs the type e would be to treat them as denoting individuals, and this would be totally wrong, because of the semantic differences we have just listed.

But there is another alternative. We can assign quantified NPs the type $\langle\langle \mathsf{e}, \mathsf{t} \rangle, \mathsf{t} \rangle$, the type of sets of sets of individuals. We can then interpret subject-predicate combinations functionally, but the function is now expressed by the NP and applied to the value of the VP.

(10)
$$\begin{tikzcd}[column sep=small]
 & \llbracket \text{every registered voter voted} \rrbracket = \llbracket \text{every registered voter} \rrbracket(\llbracket \text{voted} \rrbracket)\ \mathsf{t} \arrow[dl, no head] \arrow[dr, no head] & \\
\llbracket \text{every registered voter} \rrbracket\ \langle\langle \mathsf{e}, \mathsf{t} \rangle, \mathsf{t} \rangle & & \llbracket \text{voted} \rrbracket\ \langle \mathsf{e}, \mathsf{t} \rangle
\end{tikzcd}$$

We have now arrived at the idea that quantified NPs should have the functional type $\langle\langle \mathsf{e}, \mathsf{t} \rangle, \mathsf{t} \rangle$. We can confirm this idea by seeing exactly what sets of sets these NPs should denote. Take 'every registered voter', for instance. If this denotes the set of sets containing every voter, then, according to the rule pictured in (10), we should get the sentence 'Every registered voter voted' by testing whether the set of voters is a member of the set of sets containing every registered voter. But this amounts to saying that the set of voters contains every registered voter, i.e. that every registered voter voted.

All this can be elegantly formulated using lambda conversion.

(11.1) $[\![\text{every } \alpha]\!] = \lambda f_{\langle\langle \mathsf{e},\mathsf{t}\rangle,\mathsf{t}\rangle} \forall x_{\mathsf{e}}[\alpha(x) \rightarrow f(x)]$.

(11.2) $[\![\text{a } \alpha]\!] = \lambda f_{\langle\langle \mathsf{e},\mathsf{t}\rangle,\mathsf{t}\rangle} \exists x_{\mathsf{e}}[\alpha(x) \wedge f(x)]$.

(11.3) $[\![\text{the } \alpha]\!] = \lambda f_{\langle\langle \mathsf{e},\mathsf{t}\rangle,\mathsf{t}\rangle} \exists x_{\mathsf{e}}[f(x) \wedge \forall u_{\mathsf{e}}[\alpha(u) \leftrightarrow u{=}x]]$.

Finally, we can capture the special characteristics of referring NPs by the postulates like the following one for 'Jill'. (This, and (11.1–11.3) are called *meaning postulates* by semanticists.)

(12.1) $\exists x_{\mathsf{e}} \forall f_{\langle e,t\rangle}[[\![\text{Jill}]\!](f) \leftrightarrow f(x)]$

6. Intensional types

Throughout this discussion, we have ignored intensionality. Montague extended Church's functional type theory to include intensional types and showed how this extended theory could deal with many intensional constructions. Perhaps we'll return to this topic at a later time.

7. Bibliography

[Church, 1940] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

[Montague, 1972] Richard Montague. Pragmatics and intensional logic. In Donald Davidson and Gilbert H. Harman, editors, *Semantics of Natural Language*, pages 142–168. D. Reidel Publishing Co., Dordrecht, 1972.

[Montague, 1973] Richard Montague. The proper treatment of quantification in ordinary English. In Jaakko Hintikka, Julius M.E. Moravcsik, and Patrick Suppes, editors, *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pages 221–242. D. Reidel Publishing Co., Dordrecht, Holland, 1973. Reprinted in *Formal Philosophy*, by Richard Montague, Yale University Press, New Haven, CT, 1974, pp. 247–270.

[Whitehead and Russell, 1910 1913] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, England, first edition, 1910–1913. Three volumes.