Details for a Fregean Symbolism

Phil 387 (4/4/15)

Officially, our symbolism is defined entirely without reference to ordinary language. However, it will be helpful to begin by indicating something of the way in which it is supposed to work. Thus consider some simple subject-predicate sentences as, ‘Bill is happy’ and ‘Hillary is happy’. Such sentences may combine to form ones that are more complex. Thus, ‘Bill is happy and Hillary is happy’. We shall find it convenient to abbreviate this, ‘Bill is happy \( \land \) Hillary is happy’ with operator ‘\( \land \)’. There are five operators of this sort,

\[
\begin{align*}
\neg \text{Bill is happy} & \quad \text{It is not the case that Bill is happy} \\
\text{Bill is happy} \land \text{Hillary is happy} & \quad \text{Bill is happy and Hillary is happy} \\
\text{Bill is happy} \lor \text{Hillary is happy} & \quad \text{Bill is happy or Hillary is happy} \\
\text{Bill is happy} \rightarrow \text{Hillary is happy} & \quad \text{If Bill is happy then Hillary is happy} \\
\text{Bill is happy} \leftrightarrow \text{Hillary is happy} & \quad \text{Bill is happy iff Hillary is happy}
\end{align*}
\]

These may be combined in obvious ways, so that ‘Bill is happy \( \lor \neg \) Hillary is happy’ says Bill is happy or Hillary is not. Sometimes we will find parentheses convenient, so that ‘(Bill is happy \( \land \) Hillary is happy) \( \lor \) Hillary is president’ says Bill and Hillary are both happy \( \lor \) Hillary is president, where ‘Bill is happy \( \land \) (Hillary is happy \( \lor \) Hillary is president)’ says Bill is happy \( \land \) Hillary is happy \( \lor \) president. You should be clear about the difference.

In addition, we will typically take names and predicate terms and represent them by lowercase and uppercase letters respectively; thus we may use ‘b’ for Barack, ‘Hx’ for \( x \) is happy) and ‘Px’ for \( x \) is president). Then we shall read ‘Pb’ to say Barack is president, and ‘\( \forall x Hx \)’ to say for any thing \( x \) it is happy – so as to say that everything is happy. (The upside-down ‘A’ for ‘all’ is the universal quantifier). And we shall read ‘\( \exists x Hx \)’ to say there exists a thing \( x \) such that it is happy – so as to say that something is happy. (The backwards ‘E’ for ‘exists’ is the existential quantifier). As indicated by the official readings, the variable ‘\( x \)’ here works very much like a pronoun in ordinary language. And it is possible to use different variables to the same effect. Thus ‘\( \forall y Hy \)’ says that any thing \( y \) is such that it is happy, and ‘\( \exists y Hy \)’ that there exists a thing \( y \) such that it is happy. The important point is that the variable or pronoun links the use of the quantifier to that of the predicate.

And, of course, our notions may be combined. Thus, ‘\( \exists x Hx \land \exists x Px \)’ says something is happy and something is president. ‘\( \exists (Hx \land Px) \)’ says something is both happy and president. These are not the same! The first might be the case so long as
Hillary is happy and Bill is president. For the second, we would need some one thing, say Bill, that is both happy and president. You should try to see this given our official readings. But if this is not yet clear, do not despair. We will be able to be much more clear about such cases – and ones much more complex – once we have been more precise about how the notation works.

I. Syntax

A person can learn to read aloud in, say, Spanish or Hebrew without any understanding of what is being read. And one can imagine a child able to identify errors of grammar or punctuation before being able to read (maybe she catches missed capitals at the beginning of sentences). Similarly, in this section, we learn to identify and generate grammatical sentences of a Fregean language $L$, without direct thought about their meaning. It is possible to give so simple and concise a statement of the grammar of $L$, that a person may become a perfect grammarian independently of understanding of what is written.

(A) We begin by introducing the vocabulary or symbols of $L$, and then turn to the way they are put together. The vocabulary for our Fregean language $L$ consists of: (1) predicate letters: for any $n \geq 1$, uppercase Roman letters with superscript $n$. Thus $A^1$, $B^1...Z^1$, are one-place predicate letters, $A^2$, $B^2...$ are two-place predicate letters, etc. In addition, to be sure that we never run out of predicate letters of a given type, we allow also integer subscripts, in effect to generate new letters. Thus we would treat ‘$A^1_1$’ as a one-place predicate letter different from $A^1$. (In practice, we will almost never need recourse to so many letters.) (2) Individual constants: lower-case Roman ‘a’ through ‘h’. Again to be sure that we never run out of constants, we allow integer subscripts so that ‘a’, ‘a$_1$’, ‘a$_2$’ etc. would all be different constants – though, again, we shall hardly ever need recourse to the device of subscripts. (3) Individual variables: lower-case Roman ‘i’ through ‘z’. And we allow integer subscripts so that ‘x’, ‘x$_1$’, ‘x$_2$’ etc. are all different variables. (4) Truth functional operators, tilde, caret, wedge, arrow, double arrow: ~, $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$. We have seen these before, and now give them names. (5) Quantifier symbols, universal and existential: $\forall$, $\exists$. And, finally, the vocabulary includes, (6) punctuation symbols: opening and closing parentheses: (, ). And that is all. Insofar as these are all the symbols of the language, any expression which does not consist of some combination of these symbols could not be a grammatical expression of $L$.

To proceed, we need some brief conventions for talking about expressions of $L$. Observe that the constants, variables and predicate letters of $L$ are lower- and upper-case Roman letters. We will let “curly” ‘$\mathcal{P}$’, ‘$\mathcal{Q}$’...’$\mathcal{Q}$’ represent arbitrary expressions of $L$, lower-case italics, ‘a’, ‘b’, ‘c’ represent arbitrary constants and ‘x’, ‘y’, ‘z’ represent arbitrary variables. Truth functional operators, quantifier symbols, and punctuation symbols will stand for themselves. Insofar as these are symbols for symbols, they are “meta-symbols.” Concatenated or joined meta-symbols stand for the concatenation of that which they represent. Thus, e.g., any constant c or variable x is a term; any expression of the form $\forall x$ is a universal x-quantifier, and any expression of the form $\exists x$ is
an existential \( x \)-quantifier. Here \( 'c' \), \( 'x' \), \( \forall x' \) and \( \exists x' \) are not expressions of \( \mathcal{L} \). (Why?) Rather, we've said of expressions in \( \mathcal{L} \) that \( 'a' \), \( 'b' \), \( 'x' \), \( 'y' \) are terms, \( \forall x' \) is a universal \( x \)-quantifier, \( \forall y' \) is a universal \( y \)-quantifier, \( \exists x' \) is an existential \( x \)-quantifier, \( \exists y' \) is an existential \( y \)-quantifier, etc. In the metalinguistic expressions, \( \forall' \) and \( \exists' \) stand for themselves, \( 'c' \) for the arbitrary constant, and \( 'x' \) for the arbitrary variable. Thus, when we say something about expressions of the form \( \forall x \) we say something about a range of expressions, all at once. In the following, we use this method to say which expressions are formulas and sentences.

(B) We are now ready to introduce the core notion of a formula. I'll give a complete statement, then turn to discussion and examples.

1. If \( \mathcal{P} \) is an \( n \)-place predicate letter followed by \( n \) terms, then \( \mathcal{P} \) is a formula.

2. If \( \mathcal{P} \) is a formula, then \( \neg \mathcal{P} \) is a formula.

3. If \( \mathcal{P} \) and \( \mathcal{Q} \) are formulas, then \( (\mathcal{P} \land \mathcal{Q}) \), \( (\mathcal{P} \lor \mathcal{Q}) \), \( (\mathcal{P} \rightarrow \mathcal{Q}) \) and \( (\mathcal{P} \leftrightarrow \mathcal{Q}) \) are formulas.

4. If \( x \) is a variable and \( \mathcal{P} \) is a formula then \( \forall x \mathcal{P} \) and \( \exists x \mathcal{P} \) are formulas.

5. Nothing is a formula unless it can be formed by repeated applications of (1) - (4).

(1) tells us that \( 'A^1x' \), \( 'B^2cw' \) and \( 'B^2zz' \) are formulas. It does not allow that \( 'A^1cz' \) is a formula, because of the mismatch between the number of places and the number of terms; and it does not allow that \( 'A^1' \) is a formula, for the same reason. It is common to assume that predicate letters are followed by the correct number of terms, and to abbreviate by omission of superscripts. Let us do so as well. Given this, \( 'Ax' \) and \( 'Axy' \) involve different predicate letters, for \( 'Ax' \) abbreviates \( 'A^1x' \) and \( 'Axy' \) abbreviates \( 'A^2xy' \). We have seen this convention at work already in our initial intuitive presentation.

Clauses (2) through (4) tell us that one expression is a formula given that some other or others are formulas. So, e.g., by (1), \( 'Ax' \) is a formula; given this, by (2), \( \neg 'Ax' \) is a formula. (2) tells us that if any expression \( \mathcal{P} \) is a formula, then a tilde, followed by that expression, is a formula. And the process goes on! We have just seen that \( \neg 'Ax' \) is a formula; given this, by (2), \( \neg \neg 'Ax' \) is a formula. Etc. It is natural to represent this, and other cases, by a diagram which indicates how one rule application feeds into the next.
And the process could continue indefinitely.

Clause (3) works very much like (2), except that a given application of the rule requires two inputs. So, e.g., by (1), ‘Ax’ and ‘Bx’ are formulas. Given this, by (3), ‘(Ax \& Bx)’ is a formula. Again, the process goes on, and charts are helpful. For a case which combines rules (1) - (3),

\[
\begin{array}{c}
1 \\
Ax \\
\downarrow
2 \\
\sim Ax
\end{array}
\quad
\begin{array}{c}
1 \\
Bx \\
\downarrow
3 \\
(\sim Ax \rightarrow Ba)
\end{array}

By (1), 'Ax' and 'Bx' are formulas; given this, by (2), '\sim Ax' is a formula; given these, by (3), '(\sim Ax \rightarrow Ba)' is a formula.

Notice that different applications of (1) might produce the same formula; so, by a chart like the one above, but with ‘Ax’ uniformly replacing ‘Ba’, ‘(\sim Ax \rightarrow Ax)’, is a formula. For a more complex case, let’s use a chart to see that ‘(\sim Fxz \lor \sim (Rw \leftrightarrow Az))’ is a formula:

\[
\begin{array}{c}
1 \\
Fxz \\
\downarrow
2 \\
\sim Fxz
\end{array}
\quad
\begin{array}{c}
1 \\
Rw \\
\downarrow
3 \\
(Rw \leftrightarrow Az)
\end{array}
\quad
\begin{array}{c}
1 \\
Az \\
\downarrow
2 \\
(\sim Rw \leftrightarrow Az)
\end{array}
\quad
\begin{array}{c}
1 \\
(\sim Fxz \lor \sim (Rw \leftrightarrow Az))
\end{array}

By (1), these are formulas; given these, by (2) and (3), these are formulas; given this, by (2), this is a formula; given these, by (3), this is a formula.

Clause (4) is like (2) in that it requires only a single input. We thus say that \(\sim\) and the quantifiers are unary operators and \(\&\), \(\lor\), \(\rightarrow\), \(\leftrightarrow\) are binary. It works with a quantifier symbol and any variable. Again, charts are useful.
These charts demonstrate that a given expression is a formula. Clause (5) tells us that the only expressions that are formulas, are expressions that can be shown to be formulas this way. Thus, e.g., ‘(Ax)’ is not a formula. Here’s one way to see it: clause (3) is the only clause that introduces parentheses; but (3) always introduces parentheses along with some binary operator; since ‘(Ax)’ has parentheses but no binary operator, it could not be formed by (3), and so could not be formed by any rule; so, by (5), it is not a formula. Similarly, ‘(Ax~Bx)’ is not a formula: ‘Ax’ is a formula, and ‘~Bx’ is a formula, but there is no way to put them together, by the rules, without a binary operator in between; so ‘(Ax~Bx)’ is not a formula. If an expression is a formula, then there is some way to construct it by the rules on a chart; in general, a chart for an expression that is not a formula must break one of the rules. Notice that we are in a position to deal with arbitrarily complex cases by our methods!

It will be convenient to have in hand a few more abbreviations. First, it is sometimes convenient to use a pair of square brackets ‘[ ]’ in place of a pair of parentheses ‘( )’. This is purely for visual convenience. E.g., ‘(((0(0)))’ may be easier to absorb as, ‘((0(0)))’. On a chart, it’s natural simply to introduce the square brackets, in an application of (3), in place of the parentheses. Second, if the very last step of some chart is justified by (3), then it is OK to take the last step with the outermost pair of parentheses (or brackets) dropped. Thus, e.g., ‘∀xFxy → ¬Gz’ abbreviates (∀xFxy → ¬Gz); notice that it does not abbreviate ∀x(Fxy → ¬Gz) – for we only drop parentheses associated with a last step which is justified by (3); the last step for ‘∀x(Fxy → ¬Gz)” is justified by (4), not (3), so we would not be allowed to drop these parentheses. Again, dropping parentheses is purely for visual convenience. Also, it will be convenient to have a fixed two-place predicate letter (say, ‘E’’) for the relation of equality. Supposing that we have fixed on a predicate letter ‘P’ for the relation of equality, we’ll capitulate to ordinary usage, and abbreviate ‘Px’ as (x = y). In a chart, this abbreviation always appears in the top line, in application of rule (1). Putting all this together, the following (unannotated) chart shows that ‘∀z(~Fz→∃x[(x = z)∧Lz])→∀xCx’ is an abbreviation of a formula.
We drop superscripts on predicate letters, introduce the abbreviation for equality in the top row, use square brackets, and introduce the last arrow without parentheses. Where ‘$E_7$’ is the predicate letter for equality, ‘$(\neg Fz \rightarrow \exists x((x = z) \land Lz)) \rightarrow \forall x Cx$’ abbreviates the official expression, ‘$(\forall z(\neg F^1z \rightarrow \exists x(E^1z x \land L^1z)) \rightarrow \forall x C^1 x)$’. In general, we won’t distinguish between a formula and its abbreviations.

(C) Our discussion of syntax wraps up with a few final definitions – including (finally) the definition of a sentence. The definitions all involve formulas, and are presented in relation to a formula’s chart.

First, a formula’s main operator is the last operator added in its chart. From the chart just above, the second ‘$\rightarrow$’ is the main operator of ‘$\forall z(\neg Fz \rightarrow \exists x((x = z) \land Lz)) \rightarrow \forall x Cx$’. Second, every formula in a chart for $P$ (including $P$ itself) is a subformula of $P$. A subformula is atomic iff it appears in the top row, and so is justified by (1). From the above chart, the atomic subformulas of ‘$\forall z(\neg Fz \rightarrow \exists x((x = z) \land Lz)) \rightarrow \forall x Cx$’ are ‘Fz’, ‘(x = z)’, ‘Lz’ and ‘Cx’. An immediate subformula of $\exists$ is a subformula to which $\exists$ is directly connected by lines. From the chart above, ‘$\forall z(\neg Fz \rightarrow \exists x((x = z) \land Lz))$’ and ‘$\forall x Cx$’ are the immediate subformulas of ‘$\forall z(\neg Fz \rightarrow \exists x((x = z) \land Lz)) \rightarrow \forall x Cx$’. Of course, a formula with a unary main operator has just one immediate subformula. We sometimes speak of a formula by means of its main operator: a formula of the form $\neg P$ is a negation; $\forall x P$ is a universal generalization; $\exists x P$ is an existential generalization; a formula of the form ($P \lor \exists$) is a disjunction with $P$ and $\exists$ as disjuncts; a formula of the form ($P \land \exists$) is a conjunction with $P$ and $\exists$ as conjuncts; a formula of the form ($P \rightarrow \exists$) is a (material) conditional where $P$ is the antecedent of the conditional and $\exists$ is the consequent; and a formula of the form ($P \leftrightarrow \exists$) is a (material) biconditional.

Moving now toward the definition of a sentence: If a formula includes a quantifier, that quantifier’s scope includes just the subformula in which the quantifier first appears. Using underlines to indicate scope,
A variable $x$ is **bound** iff it appears in the scope of an $x$-quantifier, and a variable is **free** iff it is not bound. In the above chart, each variable is bound. The $x$-quantifier binds both instances of ‘$x$’; the $y$-quantifier binds both instances of ‘$y$’; and the $z$-quantifier binds both instances of ‘$z$’. In ‘$xFxy$’, however, both instances of ‘$x$’ are bound, but the ‘$y$’ is free. Finally, an expression of $\mathcal{L}$ is a **sentence** iff it is a formula and it has no free variables. To determine whether an expression is a sentence, use a chart to see if it is a formula. If it is a formula, use underlines to check whether any variable $x$ falls outside the scope of an $x$-quantifier. If there is a chart, and no such variable, then the expression is a sentence. Thus, from the chart above, ‘$\exists z(Lz \rightarrow \exists y \forall x Lxy)$’ is a sentence. From this chart,

‘$\forall y(\neg Rx \leftrightarrow \exists x Hxy)$’ is not. It has a chart so it is a formula. The $x$-quantifier binds the last two instances of ‘$x$’, and the $y$-quantifier binds both instances of ‘$y$’. But the first instance of ‘$x$’ is free. Since it has a free variable, ‘$\forall y(\neg Rx \leftrightarrow \exists x Hxy)$’ is not a sentence. By contrast ‘$\forall y(\neg Ra \leftrightarrow \exists x Hxy)$’ is both a formula and a sentence; all the variables are bound and ‘$a$’ is a **constant** not a variable -- so there are no free variables.

That is all. If you’ve understood these rules and definitions, you are an expert in the grammar of $\mathcal{L}$!
II. Semantics

Having said which expressions are sentences of \(L\), we need to say something about their semantics – about the way they connect with the world and, in particular, the conditions under which they are true and false. Roughly, we are going to say that the “status” of a complex formula is determined by its main operator with the status of its immediate subformulas, and the status of an atomic subformula is determined directly. This makes it possible to calculate the status of a complex formula, moving subformula by subformula, from the atomics to the whole. Following Frege, it is thus natural to see the fundamental semantic notion as that of a function. After a brief word about functions generally, we’ll take up atomic formulas, formulas with truth functional main operators, formulas with quantifier main operators, and work some examples.

(A) One might think of a function as a “black box” which takes some object(s) as input and, in response, extrudes an object as output. The output may or may not be the same as an input, but the output is always the same for any given inputs. The functions with which we are most familiar are those from mathematics. Set aside metaphysical questions about the nature of numbers and functions. Then we may think of, e.g., \(2 \cdot x\) as a function which takes an input object in the \(x\)-place and supplies another object as output. When the input is 1, the output is 2; when the input is 2, the output is 4; etc. Similarly, \(x + y\) is a function which takes one object in the \(x\)-place, and another in the \(y\)-place. With 1 in the \(x\)-place and 2 in the \(y\)-place ([1/\(x\), 2/\(y\)]), the output is 3; with [2/\(x\), 415/\(y\)] the output is 417, etc.

It is important that functions may combine, so that the output of one is the input to another. Such combination may be illustrated by a “tree” diagram. So, e.g., we may understand \(y + (2 \cdot x)\) by,

\[
\begin{array}{c}
\text{y + (2} \cdot \text{x)} \\
\text{y} \\
\text{2} \cdot \text{x} \\
\text{x}
\end{array}
\]

\(y + (2 \cdot x)\) results from operation of the addition function on \(y\) and \(2 \cdot x\), and \(2 \cdot x\) results from operation of the multiplication function on 2 and \(x\). We do the calculation from the “tips” to the whole. So, e.g., with [2/\(x\), 1/\(y\)], \(2 \cdot x\) is 4, and \(y + (2 \cdot x)\) is 5. This way of seeing things may seem to make complicated what is intuitively obvious. But it may also make explicit what has been going on all along. It will be helpful to have this explicit understanding as we turn to the functions of our canonical notation.

Individual constants work something like names. They pick out a single object. Thus we have imagined that ‘b’ might pick out Barack. After that, it is possible to understand predicate letters, along with each of the truth functional connectives and quantifiers, as designating functions. In each case, the functions have, as output, the true
or the false \((T \text{ or } F)\). Again, let’s set aside metaphysical worries about functions, and about \(T\) and \(F\). As it turns out, related entities are commonplace in mathematics. So, for now at least, we’ll proceed in faith that somehow there is an acceptable account of them, and exploit the function picture for our understanding of the Fregean symbolism.

\(B\) Predicate letters designate functions which apply to objects or not. If a predicate function applies to an object then the output is \(T\), if it does not apply then the output is \(F\). In a given context, we specify the relevant functions. Thus, as before, we might say \(P_x\) designates the same function as \((x \text{ is president})\). Then an atomic formula with some assignment of objects to its variables designates a truth value, \(T\) or \(F\) depending on whether the predicate applies to the thing. So both \(P_b\) and \(P_x\) with \([\text{Barack}/x]\) are \(T\), and \(P_x\) with \([\text{Hillary}/x]\) is \(F\). Similarly, if \(L_{xy}\) is \((x \text{ loves } y)\), then \(L_{xy}\) \([\text{Søren}/x, \text{Regina}/y]\) is \(T\) and, \(L_{xy}\) \([\text{Regina}/x, \text{Søren}/y]\) is \(F\).\(^1\) Notice that we designate functions in the metalanguage. Thus, e.g., if \(P_x\) designates \((x \text{ is president})\), \(P_x\) is \((x \text{ is president})\) and \(P_y\) is \((y \text{ is president})\), and if \(L_{xy}\) is \((x \text{ loves } y)\), \(L_{zw}\) is \((z \text{ loves } w)\). In practice, it will be convenient sometimes to produce “word salads” mixing ordinary language with \(\mathcal{L}\). So, e.g., ‘\(\exists x \forall y (x \text{ loves } y)\)’ is like ‘\(\exists x \forall y L_{xy}\)’ where \(L_{xy}\) is \((x \text{ loves } y)\).

A few comments: First, it is an idealization to suppose that ordinary predicates represent functions. Functions are never vague. So, e.g., if \(B_x\) is \((x \text{ is bald})\), \(B_x\) may seem definitely to return \(T\) for some objects, and \(F\) for others. But, where hair is merely thin, we may be tempted to say that \(B_x\) returns neither \(T\) nor \(F\) – or maybe a bit of both. We’ll simply assume that the application of predicate functions is or can be specified so that there is always some definite output. Thus we might take \(B_x\) to be \((x \text{ has } \leq 64 \text{ hairs on his head})\). Even so, \(B_x\) may seem undefined on certain objects: it is now clear enough what it is for \(B_x\) to apply to you or to me, but is it so clear what it is for it to apply to a computer? Again, we’ll simply assume that predicate functions are or can be specified so that they have some definite output for any input object. So, e.g., we might take \(B_x\) to be the same as \((x \text{ has a head with } \leq 64 \text{ hairs on it})\), or we might resort to specifying its application by list, say, \(B_x\): \(\{\text{Telly}, \text{Yule}…\}\).

Second, observe that predicate functions, once they are made precise, do correspond to lists. Thus, where \(S_x\) is \((x \text{ is over 6 feet tall})\) there is a corresponding collection \(\{\text{Shaq}, \text{Wilt}…\}\); and asking whether \(S_x [\text{Shaq}/x]\) is true is the same as asking whether \(\text{Shaq}\) is on the list (he is). Similarly, where \(L_{xy}\) is \((x \text{ loves } y)\) there is a collection, \(\{\langle \text{Søren}, \text{Regina} \rangle, \langle \text{Romeo}, \text{Juliet} \rangle, \langle \text{Juliet}, \text{Romeo} \rangle…\}\) of ordered pairs such that the first member loves the second. In this case, asking whether \(L_{xy} [\text{Søren}/x, \text{Regina}/y]\) is true is the same as asking whether \(\text{Søren}, \text{Regina}\) is in the list (it is); and asking whether \(L_{xy} [\text{Regina}/x, \text{Søren}/y]\) is true is the same as asking whether \(\text{Regina}, \text{Søren}\) is on the list (it is not). Sometimes we will find it easier to think of the functions in these terms.

---

\(^1\) Kierkegaard was much influenced by his unrequited love for Regina Olsen. That relationship is said to be behind his discussion of the Knights of Faith and of Infinite Resignation in his *Fear and Trembling*. 
Finally, I have used names to say how predicate functions work. Px is T with Barack in the x place. But don’t suppose that predicate functions therefore assume or require names for their operation. It’s Barack, not ‘Barack’ that is in the x place. A predicate function applies or doesn’t apply to the thing – apart from any names of it. When we get around to saying whether a predicate function applies or not (has output T or F), it is natural to use a name. But the function doesn’t itself depend on names. This is important. For McCulloch will argue that names are required in some sense. To understand his argument we need to be clear about the interest of his claim: The claim is interesting precisely because functions themselves don’t seem to require names for their operation!

(C) The “truth-functional” operators ~, \&, ∨, →, and ↔ designate functions which take truth values as inputs and have a truth value as output. Say \( \mathcal{P} \) and \( \mathcal{Q} \) are formulas, and \( \left[ \alpha \right] \) includes assignments to all the free variables in \( \mathcal{P} \) and \( \mathcal{Q} \). Then \( \mathcal{P} \left[ \alpha \right] \) and \( \mathcal{Q} \left[ \alpha \right] \) have truth value outputs. Complex functions generated by the operators may be represented on a sort of “tree” diagram. Trees have a “forward” and a “backward” direction. In the forward direction, a formula is broken into its parts according to its main operator. In the backward direction, truth or falsity of the whole is calculated from truth or falsity of the parts. For formulas with truth functional main operators:

<table>
<thead>
<tr>
<th>Forward:</th>
<th>Backward:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sim \mathcal{P} \left[ \alpha \right] )</td>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \sim )</td>
<td>The trunk is T iff the branch is F; the trunk is F iff the branch is T.</td>
</tr>
<tr>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \sim \rightarrow \mathcal{P} \left[ \alpha \right] )</td>
<td>( \sim \rightarrow \mathcal{Q} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \sim \rightarrow )</td>
<td>The trunk is T iff both branches are T; the trunk is F iff at least one branch is F.</td>
</tr>
<tr>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \sim \leftrightarrow \mathcal{P} \left[ \alpha \right] )</td>
<td>( \sim \leftrightarrow \mathcal{Q} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \sim \leftrightarrow )</td>
<td>The trunk is T iff at least one branch is T; the trunk is F iff both branches are F.</td>
</tr>
<tr>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
<td>( \mathcal{P} \left[ \alpha \right] )</td>
</tr>
<tr>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
<td>( \mathcal{Q} \left[ \alpha \right] )</td>
</tr>
</tbody>
</table>
An atomic subformula $\varphi$ does not branch; it’s value is output of the function given objects assigned to the terms as described above. 2

To see how this works, let us consider an example. Suppose we are given that $(Rx \rightarrow Sx) [a/x]$ is T and $Tx [a/x]$ is F. The main operator of ‘$(Rx \rightarrow Sx) \land Tx$’ is $\land$, so $((Rx \rightarrow Sx) \land Tx) [a/x]$ branches as follows:

$$((Rx \rightarrow Sx) [a/x]) \land Tx [a/x]$$

If we are given that the upper branch is T, and the lower F, at least one branch is F; so, by the “backwards” condition for $\land$, $((Rx \rightarrow Sx) \land Tx) [a/x]$ is F.

2 It is traditional to represent this information in table form as follows.

<table>
<thead>
<tr>
<th>$\varphi [a]$</th>
<th>$\varphi \land \varphi [a]$</th>
<th>$\varphi \lor \varphi [a]$</th>
<th>$\rightarrow \varphi [a]$</th>
<th>$\leftrightarrow \varphi [a]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Thus when $\varphi [a]$ has output T, $\sim \varphi [a]$ is F; when $\varphi [a]$ is F, $\sim \varphi [a]$ is T; when $\varphi [a]$ is T and $\varphi \land [a]$ is F, $\varphi \rightarrow \varphi [a]$ is F; etc. Again, $\sim \varphi$ is any formula with main operator $\sim$ and immediate subformula $\varphi$; $\varphi \land \varphi$, is any formula with main operator ‘$\land$’ and immediate subformulas $\varphi$ and $\varphi$; etc. You should be able to see that this comes to the same thing as the trees.
We’ll get to an extended example in a moment. For now, observe that these conditions for trees mesh with the intuitive presentation with which we began, and at the same time make precise just what is involved. Thus the standard reading of \( \neg \varphi \): “it is not the case that \( \varphi \)”: \( \neg \varphi \) is \emph{T} just when \( \varphi \) isn’t \emph{T}. The standard reading of \( \varphi \land \psi \): “\( \varphi \) and \( \psi \)”: \( \varphi \land \psi \) is \emph{T} just when both \( \varphi \) and \( \psi \) are \emph{T}. The standard reading of \( \varphi \lor \psi \): “\( \varphi \) or \( \psi \)”: \( \varphi \lor \psi \) is \emph{T} just when \( \varphi \) is \emph{T} or \( \psi \) is \emph{T} (or both). The standard reading of \( \varphi \rightarrow \psi \): “if \( \varphi \) then \( \psi \)”: \( \varphi \rightarrow \psi \) is \emph{T} just when one can move from the truth of \( \varphi \) to the truth of \( \psi \). And the standard reading of \( \varphi \leftrightarrow \psi \): “\( \varphi \) if and only if \( \psi \)”: \( \varphi \leftrightarrow \psi \) is \emph{T} just when one can move from the truth of \( \varphi \) to the truth of \( \psi \), and from the truth of \( \psi \) to the truth of \( \varphi \). We’ll say more about associations with ordinary language, after taking up quantifier functions.

(D) An \( x \)-quantifier designates a function which takes as inputs truth values from the immediate subformula for \emph{every} assignment to \( x \). Again, it will be helpful to specify quantifier functions in terms of trees. For any variable assignment \([a]\), let \([\alpha, \beta/x]\) include all the assignments in \([a]\) and \(\beta/x\) as well. Suppose all the things in the universe are \(a, b, \ldots\). Then quantifier conditions are:

**Forward:**

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]

\[
(\alpha \rightarrow \beta) \upharpoonright a/x
\]
there are no things, and so no branches, we understand the existential to be false (no
branch is T), and the universal to be T (no branch is F). Completing a tree is impractical
if there are more than three or four objects, and impossible if there are infinitely many.
Still, we can use trees to see how the different functions work. As we have seen, the
standard readings of $\forall x \varphi$ are “for any $x$, $\varphi$” and “any $x$ is such that $\varphi$’.” The standard
readings of $\exists x \varphi$ are “for some $x$, $\varphi$” and “there is an $x$ such that $\varphi$” – where “some” is
understood to mean, “at least one.”

A full tree typically begins with a sentence $\varphi$, and an empty assignment $[\[ \]]$; the
sentence branches according to its immediate subformula(s), which branch according to
their immediate subformula(s), etc. to atomics at the tips. Since a sentence has no free
variables, for any $\varnothing$ $[\[a\]]$, in the tree, $[\[a\]]$ is sure to include assignments to all the free
variables in $\varnothing$; so, if $\varnothing$ $[\[a\]]$ is at a tip, $[\[a\]]$ is sure to include assignments to all the variables
in $\varnothing$; so the backward calculation is sure to be possible. If $\varphi$ returns T with the empty
assignment, $\varphi$ is simply true, and if $\varphi$ returns F with the empty assignment, $\varphi$ is simply
false.

Again, an example should help. For now, let’s pretend there are only three things
in the universe, $a$, $b$, and $c$. Assume ‘a’ names $a$; and $a$ loves $a$, $a$ loves $b$, $a$ loves $c$ and $b$
loves $a$, where these are the only loving relations; also assume that $b$ is happy and $c$ is
happy, where these are the only happy individuals. $Lxy$ is ($x$ loves $y$), and $Hx$ is ($x$ is
happy). In compact form, we consider an interpretation $I$ of the language as follows,

\[
\begin{align*}
U & = \{a, b, c\} \\
I(a) & = a \\
I(H) & = \{b, c\} \\
I(L) & = \{(a,a), (a,b), (a,c), (b,a)\}
\end{align*}
\]

The universe $U$ of the interpretation consists of $\{a, b, c\}$. The individual constant ‘a’
designates object $a$. We understand ‘H’ as applying just to $b$, $c$. And ‘L’ applies just to
the ordered pairs, $(a,a)$, $(a,b)$, $(a,c)$, $(b,a)$. We’ll consider the function $\neg \forall x(\exists yLyx \rightarrow Hx)$. This is complicated! But we can attack it, one operator at a time. First, in the forward
direction, we begin with the sentence, and empty assignment, working in the forward
direction:
At stage (1) we begin with the sentence $\neg \forall x(\exists y \text{Lyx} \rightarrow \text{Ha})$ and the empty assignment $[\ ]$. The main operator is $\neg$, so there is only one branch at stage (2). Notice that, as is always the case for truth functional operators, the variable assignment simply carries forward. The main operator of the formula at stage (2) is $\forall x$. In this case, there are as many branches at the next stage (3) as there are things, though assignments on the branches have the different objects in the $x$ place. Now $\rightarrow$ is the main operator on each branch; so, for each stage (3) branch there are two new branches at stage (4). Again, variable assignments carry forward from the branch before. There is nothing to be done with the lower limbs of each pair at stage (4). But the upper limbs get as many new branches as there are things. Notice that existing variable assignments are augmented with appropriate assignments in the $y$ place. This completes the forward tree.
Now we are ready to calculate truth values from the tips back toward the trunk. Recall that \( H \) applies just to \( b, c \) and \( L \) just to \( \langle a, a \rangle, \langle a, b \rangle, \langle a, c \rangle, \langle b, a \rangle \).

The easiest to see are the lower branches of the pairs at (4). For the evaluation of \( Ha \) the assignment to \( x \) does not matter -- the constant ‘\( a \)’ returns the object \( a \) and the function returns \( F \) since it is given that \( b \) and \( c \) are happy but \( a \) is not. For the top tip at (5), we have that \( a \) loves \( a \), so \( Lyx \) has output \( T \) with \( \langle a/x, a/y \rangle \); for the next, \( b \) loves \( a \), so \( Lyx \) has output \( T \) with \( \langle a/x, b/y \rangle \); but \( c \) does not love \( a \), so \( Lyx \) is \( F \) with \( \langle a/x, c/y \rangle \). You should be able to check other tips at (5) by similar reasoning. Now, moving back from the tips, an existential is \( T \) if and only if there is at least one branch that is \( T \). For each existential quantifier, there is at least one branch that is \( T \), so the upper branches of the pairs at (4) are each \( T \). The \( \rightarrow \) is \( F \) if the top branch is \( T \) and the lower branch is \( F \). So the branches at (3) are all \( F \). Since the universal at (2) has a branch that is \( F \), the result is \( F \). Finally, the branch for the negation is \( F \); so the trunk at (1) is \( T \). So, on our assumptions, \( \sim \forall x(\exists yLyx \rightarrow Hx) \) is \( T \).
III. Reading the Symbols

So far, we’ve been able to calculate the truth or falsity of a sentence from an arrangement of truth values at the tips of its tree. Now, in order to see “what a sentence says” about the world, we need to think more generally about the range of conditions under which it is true. I’ll continue to work with trees, though you should *use* them to reach the point where you can “read” sentences on your own. Let \( r \) be Rover; \( Lx, (x \text{ is lucky}) \); \( Dx, (x \text{ is a dog}) \); and \( Bxy, (x \text{ barks at } y) \). Say all the things in the universe are \( a,b,... \). The simplest case is of the sort ‘Dr’; this is true just in case Rover is a dog. It is easy to see that the function returns true if Rover is among the dogs, and otherwise not. After that:

(A) ‘\( \exists xDx \)’ is read, “there is an \( x \) such that \( Dx \)”; it is true iff there is at least one dog.

\[
\begin{array}{c}
1 \\
\exists xDx \\downarrow \\
\exists x
\end{array}
\]

\( \exists xDx \) \( \downarrow \) is T at stage (1) just in case at least one branch at (2) is T. So \( \exists xDx \) \( \downarrow \) is T iff there is at least one object \( o \) such that \( Dx \) \( [o/x] \) is T – that is, iff there is at least one dog. The only way for \( \exists xDx \) \( \downarrow \) to be F is if there are no dogs. This, of course, is just what we thought it said from our original intuitive presentation. ‘\( \exists xBrx \)’ is true iff Rover barks at something. This works on a tree like the one above with ‘Brx’ for ‘Dx’.

(B) ‘\( \forall xDx \)’ is read, “for any \( x \), \( Dx \)”; it is true iff everything is a dog.

\[
\begin{array}{c}
1 \\
\forall xDx \\downarrow \\
\forall x
\end{array}
\]

\( \forall xDx \) \( \downarrow \) is T at (1) just in case each of the branches at (2) is T. So \( \forall xDx \) \( \downarrow \) is T iff for any object \( o \), \( Dx \) \( [o/x] \) is T; so it’s T iff everything is a dog. \( \forall xDx \) \( \downarrow \) is F, if there is even one thing that isn’t a dog. ‘\( \forall xBrx \)’ is true iff Rover barks at everything (including himself).

(C) ‘\( \sim \exists xDx \)’ is read, “it is not the case that there is an \( x \) such that \( Dx \)”; it is true iff there are no dogs.
~∃x Dx [ ] is T at (1), just in case ∃x Dx [ ] is F at (2); but the only way for ∃x Dx [ ] to be F is for each of the branches at (3) to be F. If even one branch at (3) is T, then ∃x Dx [ ] is T at (2), and ~∃x Dx [ ] is F.

(D) ‘∃x~Dx’ is read, “there is an x such that it is not the case that Dx”; it is true iff at least one thing isn’t a dog.

∃x~Dx [ ] is T at (1) just in case at least one branch at (2) is T, and the only way for a branch at (2) to be T, is for the corresponding branch at (3) to be F; if a branch at (3) is F, the corresponding branch at (2) is T, and the existential at (1) is T. The only way for ∃x~Dx [ ] to be F, is for everything to be a dog; if everything is a dog, then all the branches at (3) are T; so all the branches at (2) are F; so ∃x~Dx [ ] at (1) is F.
(E) ‘∃x Dx ∧ ∃x Lx’ is read, “there is an x such that Dx, and there is an x such that Lx”; it is true iff something is a dog, and something is lucky.

∃x Dx ∧ ∃x Lx[] is T at (1), just in case both branches are T at (2); for the top branch to be T, at least one thing has to be a dog; for the bottom branch to be T, at least one thing has to be lucky. It doesn’t matter which thing is a dog, and which is lucky, so long as there is at least one of each (notice that nothing prevents the lucky thing from being the dog). ∃x Dx ∧ ∃x Lx[] is F at (1) if one or both of the branches at (2) is F; to make the top branch F, there would have to be no dogs; to make the bottom branch F, there would have to be no lucky things. If there are no dogs, or there are no lucky things, ∃x Dx ∧ ∃x Lx[] is F.

(F) ‘∃x (Dx ∧ Lx)’ is read, “there is an x such that Dx and Lx”; it is true iff there is at least one lucky dog.

∃x (Dx ∧ Lx)[] is T at (1) just in case at least one of the branches at (2) is T; for one of those branches to be T, both branches of the corresponding pair at (3) must be T; so if ∃x (Dx ∧ Lx)[] is T, some one object o must be both a dog and lucky. ∃x (Dx ∧ Lx)[] is
F iff all the branches at (2) are F; this is the case if no dogs are lucky. Notice: if \( \exists x(Dx \land Lx) \) is T, \((\exists x dx \land \exists x Lx) \) is T as well, but not the other way around; for \( \exists x(Dx \land Lx) \) is T only if some one thing is both lucky and a dog. This is just the contrast with which we left off in our original intuitive presentation.

\((G) \) ‘\( \forall x(Dx \rightarrow Lx) \)’ is read, “for any x, if Dx then Lx”; it is true iff all dogs are lucky.

\( \forall x(Dx \rightarrow Lx) \) is T at (1) just in case all of the branches at (2) are T; for this to be the case, there can be no pair at (3) for which the top is T and the bottom is F; so \( \forall x(Dx \rightarrow Lx) \) is T just in case anything that is a dog is also lucky. Notice: If \( o \) isn’t a dog, then the corresponding branch at (2) is T whether \( o \) is lucky or not. It’s only when \( o \) is a dog, that the corresponding branch at (2) has any chance of being F. But if some \( o \) is a dog and isn’t lucky, then the corresponding branch at (2) is F, and \( \forall x(Dx \rightarrow Lx) \) is F as well.

\((H) \) ‘\( \forall x(Dx \rightarrow \exists y Bxy) \)’ is read, “for any x, if Dx then there is a y such that Bxy”; it is true iff every dog barks at something (which may or may not be itself).
As in the previous example, \(\forall x (Dx \rightarrow \exists y Bxy) \land \neg \exists y Bxy\) is T at (1) just in case all of the branches at (2) are T; for this, there can be no pair at (3) for which the top is T and the bottom is F. As before, if \(o\) isn’t a dog, the corresponding branch at (2) is automatically T. But the cases differ with respect to the condition dogs must satisfy. If \(o\) is a dog, the bottom branch at (3) is T just in case one of the corresponding branches at (4) is T—that is, just in case there is some \(p\) such that \(Bxy \land \neg Bxy\) is T. If some \(o\) is a dog and there is no \(p\) at which it barks, then at (3), a top branch is T and bottom F; so the corresponding branch at (2) is F, and the universal at (1) is F.

(I) ‘\(\forall x \forall y ((Dx \land Dy) \rightarrow (x = y))\)’ is read, “for any \(x\) and any \(y\), if \(Dx\) and \(Dy\), then \((x = y)\)”; it is T iff there is at most one dog. Here is the beginning of its tree.
∀x∀y((Dx ∧ Dy) → (x = y)) \[\text{is T at (1)}\] just in case all the branches at (2) are T; and these are T just in case all the branches at (3) are T. But now consider what it is for one of the branches at (3) to be T.

Each branch at (3) is T, if no pair at (4) has the top T and the bottom F. For a top at (4) to be T, both o and p have to be dogs; if one or both isn’t a dog, then the top at (4) is F, and the branch at (3) is T. But suppose o and p are both dogs, then the bottom is false unless \(o = p\). If o and p are distinct dogs, then (4) will have the top T and bottom F; so the branch at (3) will be F, and the universals at (2) and (1) will be F as well. So ∀x∀y((Dx ∧ Dy) → (x = y)) \[\text{is T if there are no dogs (all the tops at (4) will be F), or there is one dog (distinct dogs are required to make the bottom at (4) F), and F if there is more than one. In effect, if } o \text{ is a dog, we’re saying that anything that is a dog, is it.}

IV. A Final (important) Case

Bertrand Russell proposes a three part analysis of definite descriptions -- phrases of the sort, “the so-and-so is...”; he was especially interested in, ‘the present king of France is bald’. On his account these expressions fail when there is more than one so-and-so. Similarly, phrases of the sort, “the so-and-so...” fail when there aren’t any so-and-sos. Thus, e.g., neither ‘the desk at CSUSB has graffiti on it’ nor ‘the present king of France is bald’ seem to be true. The first because the description fails to pick out just one object, and the second because the description doesn’t pick out any object at all. Of course, if a description does pick out just one object, then the predicate must apply. So, e.g., ‘The president of the USA is a woman’ isn’t true. There is exactly one object which is the president of the USA, but it isn’t a woman. And ‘The president of the USA is a man’ is true. In this case, exactly one object is picked out by the description, and the predicate does apply. Thus Russell proposes that a statement of the sort, “the \(P\) is \(Q\)” must meet three conditions:
1. At least one thing is $P$.
2. At most one thing is $P$.
3. Whatever is $P$ is $Q$.

If all three conditions are met, says Russell, the statement is true; if one or more isn’t met, then the statement is false.

Russell uses Frege’s notation for his account of truth conditions. Thus we’ll express his conditions for “the $P$ is $Q$” in $L$. Speaking in the metalanguage, where $Px$ and $Qx$ are (maybe atomic) formulas with just $x$ free, $\exists x([Px \land \forall y(Py \rightarrow (x = y))] \land Qx)$ does the job. Now the trees for this get messy. We will see how this goes. But I think we can see up front what is happening. First, for visual convenience, let $Rx$ be the subformula, $\forall y(Py \rightarrow (x = y))$, then $\exists x([Px \land \forall y(Py \rightarrow (x = y))] \land Qx)$ appears as $\exists x([Px \land Rx] \land Qx)$. Then what we need is an object that satisfies each of the three conditions, $Px$, $Qx$, and $Rx$. Here’s that much of the tree:

$\exists x([Px \land Rx] \land Qx)$ is T at (1) just in case at least one of the branches at (2) is T. But a branch at (2) is T just in case both of the corresponding branches at (3) are T; and a top branch at (3) is T just in case both of the corresponding branches at (4) are T. Thus a branch at (2) is T just in case each of the three corresponding branch tips is T. If an object isn’t $P$ or isn’t $Q$, then the corresponding branch at (2) is F. So far, so good.

Now suppose we are on a branch where some $o$ is such that $Px [o/x]$ and $Qx [o/x]$ are T. What does the condition $Rx$, $\forall y(Py \rightarrow (x = y))$, require? Given that $Rx [o/x]$ is T, it requires that anything that is $P$ is identical to object $o$. Thus, as in the last example we considered above, it requires that at most one thing is $P$. This is just right. Thus, the tree continues:
∀y(∀y → (x = y) [[o/x]]) at (4) is T just in case all the branches at (5) are T; for this to be the case, there can be no pair at (6) for which the top is T and the bottom is F. Suppose some object p is ∃; then the top at (6) is T, and the bottom is false unless o is p. If o and p are both ∃ but distinct, then the corresponding branch at (5) is F, the universal at (4) is F, and the conjunctions at (2) and (3) are F as well. If o = p then the top at (6) is T, but the bottom is T as well, so the branch at (5) is T. Having determined that o is ∃, R requires that anything that is ∃ is it. So, again, it’s like the condition from the last example above, and ∃ requires that there is at most one ∃. So ∃x([∃x ∧ ∀y(∀y → (x = y))] ∧ ∃x) is T just in case some o is ∃, is the only ∃, and is ∃. So each of Russell’s conditions is imposed.

Let’s return to ‘The present King of France is bald’. Relaxing constraints on predicate vocabulary, with PKFx for (x is present king of France) and Bx for (x is bald), we get,

∃x([PKFx ∧ ∀y(PKFy → (x = y))] ∨ Bx)

If there is no present king of France, this is false because it fails the top tips at (4). If there is more than one present king of France, some top tips at (4) may be T, but the corresponding bottom tips are (F). If there is just one present king of France who isn’t bald, then a pair at (4) may be T, but the corresponding tip at (3) is F. If there is just one present king of France who is bald, then a pair of tips at (4) is T, as well as the corresponding tip at (3); so the conjunctions at (3) and (2) are T, and the existential at (1) is T as well. Notice: on this account we do not treat ‘the present king of France’ as a name. Rather, whatever things there are, we consider each, and ask whether it meets certain conditions. If none of the things meets the conditions, then ‘The present king of France is bald’ is false; if some thing meets them, then ‘The present king of France is bald’ is true.