Semantics of First-Order Logic

A notation becomes a "representation" only when we can explain, in a formal way, how the notation can make true or false statements about some domain; this is what model-theoretic semantics does in logic.

We now have a set of syntactic expressions for formalizing knowledge, but we have only talked about the *meanings* of these expressions in an informal way. We now need to provide a **model-theoretic semantics** for this syntax, specifying precisely what the *possible* ways are of using a first-order language to talk *about* some domain of interest.

Why is this important? Because, (1) that is the only way that we can verify that the facts we write down in the representation actually *can* mean what we intuitively *intend* them to mean; representations without formal semantics have often turned out to be incoherent when examined from this perspective; and (2) that is the only way we can judge whether proposed methods of making inferences are reasonable – i.e., that they give conclusions that are guaranteed to be true whenever the premises are, or at least are *probably* true, or at least *possibly* true. Without such guarantees, we will probably end up with an irrational "intelligent agent"!

Models

Our goal, then, is to specify precisely how all the terms, predicates, and formulas (sentences) of a first-order language can be interpreted so that terms refer to individuals, predicates refer to properties and relations, and formulas are either true or false.

So we begin by fixing a domain of interest, called the

domain of discourse: $\mathcal{D}$.

This can be any nonempty set at all – e.g., a set of (actual or imaginary) blocks on a table that you want to encode knowledge about; a set of numbers or other mathematical objects; a set of possible electronic circuits and their components; the set of all physical objects in the universe; the set of all physical objects, events, cultural and social entities in the world; etc.

NB: A common mistake is to think that the individuals in $\mathcal{D}$ are just symbolic objects or abstractions of some sort. They *can* be, but they can perfectly well be real, tangible objects in the world, if that's what we want to talk about in the logic!! The mistake comes from the fact that when we say that such-and-such terms in FOL refer to such and such individuals, we naturally can't place those individuals themselves *on the written page*, but can only use words or other "metasymbols" (e.g., *Abraham Lincoln, a, b, c, ...*, below) for them. Thus we *seem* to be relating symbols of the logic to *other* symbols – but we're not! We're relating symbols of the logic to entities in the domain of discourse.

Next, we need to specify a possible *correspondence* between the basic symbols (constants) of FOL and the domain of discourse. We do this by letting individual constants denote individuals in $\mathcal{D}$, letting function constants denote functions that map individuals (or n-tuples of them) to individuals, and letting predicate constants denote sets (or n-ary relations) formed from individuals in $\mathcal{D}$. More precisely, we define an

interpretation: $\mathcal{I}$

as a *function* that maps

- each individual constant to an individual $\in \mathcal{D}$; e.g., $\mathcal{I}$(ABE) = *Abraham Lincoln*, in other words, ABE *denotes* Abraham Lincoln under interpretation $\mathcal{I}$;
- each n-adic function constant to a function $\in \mathcal{D}^n \rightarrow \mathcal{D}$; (the notation $A \rightarrow B$ in general refers to the class of all function that map the elements of A into elements of B; A^n refers to the set of all n-tuples of elements belonging to set A;) e.g., $\mathcal{I}$(Weight) = *the function that maps each object in $\mathcal{D}$ into the weight in kilograms of that object*; thus Weight *denotes* a function in the class $\mathcal{D} \rightarrow R$ under interpretation $\mathcal{I}$, where R is the set of nonnegative real numbers;
- each n-adic predicate constant to a subset of $\mathcal{D}^n$ (i.e., an n-adic *relation* over $\mathcal{D}$); e.g., $\mathcal{I}$(Loves) = *the set of pairs of people such that the first loves the second*; thus Loves *denotes* a subset of $\mathcal{D}^2$ under interpretation $\mathcal{I}$ (where presumably $\mathcal{D}$ includes people).

Since $\mathcal{I}$ is a function, we have written the interpretation (denotation, semantic value) of atomic symbols like ABE, Weight, etc., as $\mathcal{I}$(ABE), $\mathcal{I}$(Weight), etc.[1] However, we will generally use a slightly more concise "superscript" notation, ABE$^{\mathcal{I}}$, Weight$^{\mathcal{I}}$, etc.

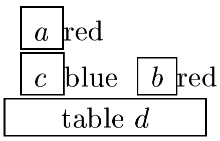
Before giving a more complete illustration, we define a *model* as a domain of discourse together with an interpretation:

[1]**Important note**: $\mathcal{I}$ is a strange function compared to most of the ones you might be used to from mathematics, because it is a function on *symbols*. When we write something like $sin\,\theta$ in calculus, we are not referring to the *sine* of the *symbol* θ (which has a particular oval shape and cross-bar), but rather to the *sine* of whatever angle θ *refers* to. By contrast, when we write $\mathcal{I}$(C), we really are talking about the value of $\mathcal{I}$ for the *symbol* "C" itself. So we might more accurately write $\mathcal{I}$("C"), or in Lisp-like notation, $\mathcal{I}$('C). We could not be referring to the *value* of C, because it doesn't *have* a value – till we supply one, using $\mathcal{I}$! Another way of putting this distinction is that in $sin\,\theta$ we are *using* θ (namely, to refer to an angle), while in $\mathcal{I}$(C) we are *mentioning* C. This is much like the contrast between the use and mention of "Rome" in the sentences, "Rome is an ancient city", and "Rome is a 4-letter word".

model $\mathcal{M} = (\mathcal{D},\mathcal{I})$.

Note that G & N use $|\mathcal{I}|$ for $\mathcal{D}$. This is somewhat misleading, since in general the domain $\mathcal{D}$ is not determined by the interpretation $\mathcal{I}$ – many possible choices of domain may be compatible with an interpretation of a first-order language. In particular, this is so if our first-order language has no function symbols.

As a detailed example of the concepts we've introduced, let's consider a simple world of two red blocks a, b and a blue block c on a table d. Here $a,\ b,\ c,\ d$ are our *metalinguistic* names for the objects we want to talk about. These names are *not* part of the first-order language under consideration, but just "handy" ways of referring to the objects of interest. In other words, they are part of the *metalanguage* – which often includes chunks of English prose – that we use to talk *about* the (first-order) object language under consideration, the domain of discourse, and the way the symbols of the object language relate to the objects in the domain of discourse.



In the object language, we might describe this situation by the formulas:

Block(B1), Block(B2), Red(B1), Red(B2), Blue(B3),
On(B1,B3), On(B3,T), On(B2,T)

where B1 is intended to refer to red block a, B2 is intended to refer to red block b, B3 is intended to refer to the blue block c, and T is intended to refer to the table d.

We can take the domain of discourse to be just

$\mathcal{D} = \{a, b, c, d\}$,

(which is a set of 4 *objects*, not a set of 4 names!) Let's assume that in our first-order language we use only B1, B2, B3, and T as individual constants, and only Block, Red, Blue and On as predicate constants. Then we can express their intended meanings by the following interpretation $\mathcal{I}$:

$\mathrm{B1}^{\mathcal{I}} = a$
$\mathrm{B2}^{\mathcal{I}} = b$
$\mathrm{B3}^{\mathcal{I}} = c$
$\mathrm{T}^{\mathcal{I}} = d$
$\mathrm{Block}^{\mathcal{I}} = \{a, b, c\}$
$\mathrm{Red}^{\mathcal{I}} = \{a, b\}$
$\mathrm{Blue}^{\mathcal{I}} = \{c\}$
$\mathrm{On}^{\mathcal{I}} = \{\langle a, c\rangle\,, \langle c, d\rangle\,, \langle b, d\rangle\,\}$

Let's expand the example a little to also allow for a function. Let Base be a function constant that we interpret as giving us, for any object, the "base" the object rests on:

$\text{Base}^{\mathcal{I}} = \{\langle a, c\rangle\ ,\ \langle b, d\rangle\ ,\ \langle c, d\rangle\ ,\ \langle d, d\rangle\ \}$

Note that this function is almost the same, set-theoretically, as $\text{On}^{\mathcal{I}}$. But recall that function constants in FOL denote *total* functions. Thus, the above interpretation of Base also supplies a value for argument d (the table), namely itself – it is its own "base". This is just an arbitrary choice, assuming that we don't really intend Base to have any particular value for the table. (And if we were to say it rests on the floor, we would then have to specify what the value of the Base function is when applied to the floor, etc.!)

> **N.B.:** Don't assume from the above specification of $\mathcal{I}$ that to use logic we actually have to spell out such interpretations! In an application, we will usually *have in mind* that certain constants should denote certain individuals or sets, but the only time we actually spell such things out is when we're illustrating or analyzing properties of the logic.

Given that we have a function symbol like Base, how do we now interpret a function term like

Base(B1) ?

The answer is just that we take the function denoted by Base, i.e., $\text{Base}^{\mathcal{I}}$, and apply it to the individual denoted by B1, i.e., $\text{B1}^{\mathcal{I}}$. The resulting individual, $\text{Base}^{\mathcal{I}}(\text{B1}^{\mathcal{I}}) = \text{Base}^{\mathcal{I}}(a) = c$ (the blue block), is the denotation (or semantic value) of Base(B1).

Variable and Term Assignments

Before we can state the general rule for interpreting function terms, we need to consider the possibility that a term may be a *variable*, or a function term involving a variable, such as Base(x), or Base(Base(x)). To be able to interpret such terms, we introduce a

variable assignment $\mathcal{U} \in$ VARIABLES $\rightarrow \mathcal{D}$,

i.e., $\mathcal{U}$ maps each variable to an individual.

Let's extend our example accordingly by also adding VARIABLES = {x, y, z} to our first-order language, and assigning them values

$\text{x}^{\mathcal{U}} = \text{y}^{\mathcal{U}} = \text{z}^{\mathcal{U}} = a.$

This assignment is completely arbitrary, and in fact, in general the initially given assignment function serves only to get us "off the ground" in the recursive semantic interpretation of our first-order language. As we shall see, the values assigned to variables will be systematically altered when we take account of the quantifiers binding them.

We can now state the general rule for interpreting a function term as follows, where we use $\text{T}_{\mathcal{IU}}$ for the general *term assignment*, which assigns a semantic value to every term, given an interpretation $\mathcal{I}$ and a variable assignment $\mathcal{U}$. (Thus $\mathrm{T}_{\mathcal{IU}}$ *extends* both $\mathcal{I}$ and $\mathcal{U}$ so as to interpret not only simple terms but also function terms.)

term assignment $\mathrm{T}_{\mathcal{IU}}$:

$$\begin{aligned}
\mathrm{T}_{\mathcal{IU}}(\tau) &= \tau^{\mathcal{I}} \quad \text{if } \tau \text{ is an individual constant} \\
&= \tau^{\mathcal{U}} \quad \text{if } \tau \text{ is a variable} \\
\mathrm{T}_{\mathcal{IU}}(\alpha(\tau_1, ..., \tau_n)) &= \alpha^{\mathcal{I}}(\mathrm{T}_{\mathcal{IU}}(\tau_1),\ ...,\ \mathrm{T}_{\mathcal{IU}}(\tau_n)) \quad \text{if } \alpha \text{ is an } n\text{-ary function constant} \\
&\quad \text{and } \tau_1, ..., \tau_n \text{ are terms}
\end{aligned}$$

Going back to our example, we can see that, e.g.,

$$\begin{aligned}
&\mathrm{T}_{\mathcal{IU}}(\mathrm{Base(B1)}) = \mathrm{Base}^{\mathcal{I}}(\mathrm{T}_{\mathcal{IU}}(\mathrm{B1})) = \mathrm{Base}^{\mathcal{I}}(\mathrm{B1}^{\mathcal{I}}) = \mathrm{Base}^{\mathcal{I}}(a) = c; \\
&\mathrm{T}_{\mathcal{IU}}(\mathrm{Base(x)}) = \mathrm{Base}^{\mathcal{I}}(\mathrm{T}_{\mathcal{IU}}(\mathrm{x})) = \mathrm{Base}^{\mathcal{I}}(\mathrm{x}^{\mathcal{U}}) = \mathrm{Base}^{\mathcal{I}}(a) = c; \\
&\mathrm{T}_{\mathcal{IU}}(\mathrm{Base(Base(B1))}) = \mathrm{Base}^{\mathcal{I}}(\mathrm{T}_{\mathcal{IU}}(\mathrm{Base(B1)}) = \mathrm{Base}^{\mathcal{I}}(c) = d.
\end{aligned}$$

Satisfaction

Since we now know how to interpret terms, we are ready to interpret a predicate applied to a sequence of terms, i.e., an atomic formula. First we recursively (inductively) define *satisfaction* of a formula ϕ by a variable assignment $\mathcal{U}$ relative to a model $\mathcal{M}$, written as

$$\models_{\mathcal{M}} \phi\,[\mathcal{U}].$$

This notion is close to that of truth, but is dependent on a particular variable assignment $\mathcal{U}$. Later we will eliminate the dependence on the variable assignment, and define *truth* of ϕ in model $\mathcal{M}$, written

$$\models_{\mathcal{M}} \phi.$$

Starting with atomic formulas, we consider an n-ary predicate π applied to arbitrary terms $\tau_1, ..., \tau_n$:

$$\models_{\mathcal{M}} \pi(\tau_1, ..., \tau_n)\,[\mathcal{U}] \quad \text{iff} \quad \langle \mathrm{T}_{\mathcal{IU}}(\tau_1), ..., \mathrm{T}_{\mathcal{IU}}(\tau_n)\rangle \in \pi^{\mathcal{I}},$$

in other words, the variable assignment $\mathcal{U}$ satisfies the formula relative to $\mathcal{M}$ $(= (\mathcal{D},\mathcal{I}))$ iff the n-tuple which is the interpretation of the argument terms $\tau_1, ..., \tau_n$ belongs to the relation $\pi^{\mathcal{I}}$ that is the interpretation of the predicate π. In the case of our earlier blocks example, we can write, e.g.,

$$\models_{\mathcal{M}} \mathrm{Block(B1)}\,[\mathcal{U}],\ \ \models_{\mathcal{M}} \mathrm{Red(B1)}\,[\mathcal{U}],\ \ \models_{\mathcal{M}} \mathrm{On(B1,B3)}\,[\mathcal{U}], \text{ etc.}$$

Recall that equality receives a fixed interpretation in FOL, rather than being freely interpretable. This fixed interpretation is given by the following satisfaction conditions for equations (which are also considered atomic formulas):

$$\models_{\mathcal{M}} \sigma = \tau\,[\mathcal{U}] \quad \text{iff} \quad \mathrm{T}_{\mathcal{IU}}(\sigma) = \mathrm{T}_{\mathcal{IU}}(\tau).$$

For the earlier blocks example, we have, e.g.,

$$\models_{\mathcal{M}} \mathrm{B1} = \mathrm{x}\,[\mathcal{U}],\ \ \models_{\mathcal{M}} \mathrm{Base(B1)} = \mathrm{B3}\,[\mathcal{U}],\ \ \models_{\mathcal{M}} \mathrm{Base(Base(B1))} = \mathrm{Base(B3)}\,[\mathcal{U}], \text{ etc.}$$

For formulas that are not satified, we use the notation

$$\not\models_{\mathcal{M}} \phi\,[\mathcal{U}];$$

e.g.,

$$\not\models_{\mathcal{M}} \text{Block(T)}\,[\mathcal{U}],\ \ \not\models_{\mathcal{M}} \text{On(B1,T)}\,[\mathcal{U}],$$
$$\not\models_{\mathcal{M}} \text{x} = \text{B3}\,[\mathcal{U}],\ \ \not\models_{\mathcal{M}} \text{Base(B1)} = \text{Base(B2)}\,[\mathcal{U}], \text{ etc.}$$

Next we consider logically compound formulas, built up by using the logical connectives:

$$\models_{\mathcal{M}} \neg\phi\,[\mathcal{U}] \ \text{ iff } \ \not\models_{\mathcal{M}} \phi\,[\mathcal{U}]$$
$$\models_{\mathcal{M}} (\phi \wedge \psi)\,[\mathcal{U}] \ \text{ iff } \models_{\mathcal{M}} \phi\,[\mathcal{U}] \ \text{ and } \ \models_{\mathcal{M}} \psi\,[\mathcal{U}]$$
$$\models_{\mathcal{M}} (\phi \vee \psi)\,[\mathcal{U}] \ \text{ iff } \models_{\mathcal{M}} \phi\,[\mathcal{U}] \ \text{ or } \ \models_{\mathcal{M}} \psi\,[\mathcal{U}]$$
$$\models_{\mathcal{M}} (\phi \Rightarrow \psi)\,[\mathcal{U}] \ \text{ iff } \not\models_{\mathcal{M}} \phi\,[\mathcal{U}] \ \text{ or } \models_{\mathcal{M}} \psi\,[\mathcal{U}]$$
$$\models_{\mathcal{M}} (\psi \Leftarrow \phi)\,[\mathcal{U}] \ \text{ iff } \not\models_{\mathcal{M}} \phi\,[\mathcal{U}] \ \text{ or } \models_{\mathcal{M}} \psi\,[\mathcal{U}]$$
$$\models_{\mathcal{M}} (\phi \Leftrightarrow \psi)\,[\mathcal{U}] \ \text{ iff } \models_{\mathcal{M}} \phi\,[\mathcal{U}] \text{ and } \models_{\mathcal{M}} \psi\,[\mathcal{U}] \quad \text{or} \quad \not\models_{\mathcal{M}} \phi\,[\mathcal{U}] \text{ and } \not\models_{\mathcal{M}} \psi\,[\mathcal{U}]$$

Arguably, these interpretations of the logical connectives are quite close to those of the corresponding English words *not, and, or, if…then…, …if…, if and only if.* In the case of English *or*, it has often been suggested that this is more like *exclusive or* (i.e., one or the other but not both). For instance, in

Every guest drank tea or coffee

we feel that the alternatives are exclusive. However, this is only a "conversational implicature" – something we infer by default, but which can be denied without contradiction.[2] For instance, we can say without contradiction,

Every guest had tea or coffee. In fact, some had both.

In the case of $\Rightarrow$, we should mention that this corresponds to *if…then…* only when we use *if…then… factually*, as in

If John is awake, (then) he heard what you just said

rather than *counterfactually*, as in

If John were awake, (then) he would have heard what you just said

where it is implied (or at least implicated!) that John is not awake. Such a sentence is not true merely because the antecedent is false. For instance,

If John were asleep, he would have heard what you just said

[2]This concept is due to H.P. Grice ("Logic and conversation", in *The Logic of Grammar*, ed. by D. Davidson & G. Harman, Dickenson, 1975), who suggested that conversational implicatures arise from "conversational maxims", such as "Be relevant", "Be brief", "Be truthful", and (roughly) "Be as informative as is required". Concerning the last maxim, suppose you know that John had tea *and* coffee. Then you would be withholding information if you said "John had tea *or* coffee," even though this is true. The hearer assumes you are conforming with the conversational maxims, so if you say "John had tea *or* coffee," the inference is that John did *not* have both, else you would have said *and*!

seems to be false even when the antecedent is false (John is awake) and the consequent is true (he did hear what you just said).

Finally, the satisfaction conditions for quantification can be stated as follows (this is the trickiest part of truth-conditional semantics, and why it took decades, from Frege's first formulation of FOL till Tarski's work on semantics, to properly formalize it):

$$\begin{array}{ll} \models_{\mathcal{M}} (\forall \nu\, \phi)\, [\mathcal{U}] & \text{iff for all } d \in \mathcal{D},\ \models_{\mathcal{M}} \phi\, [\mathcal{U}'], \text{ where } \mathcal{U}' \text{ is the same as } \mathcal{U} \\ & \qquad \text{except that } \nu^{\mathcal{U}'} = d \\ \models_{\mathcal{M}} (\exists \nu\, \phi)\, [\mathcal{U}] & \text{iff for some } d \in \mathcal{D},\ \models_{\mathcal{M}} \phi\, [\mathcal{U}'], \text{ where } \mathcal{U}' \text{ is the same as } \mathcal{U} \\ & \qquad \text{except that } \nu^{\mathcal{U}'} = d \end{array}$$

Intuitively, the $\forall$-condition just says that ϕ (the sentence that comprises the scope of the quantifier) must be satisfied for all ways of choosing an individual d as the value of variable ν. Similarly for the $\exists$-condition, except that there need only be *some* such d for which ϕ is satisfied.

Truth

It is now a simple step to define **truth in a model** in terms of satisfaction. In most presentations of FOL semantics, truth is defined only for *closed* formulas – ones with no free variables. For instance,

Loves(John,Mary), (∀x (Person(x) ⇒ (∃y (Person(y) ∧ Loves(x,y)))))

are closed formulas, while

Loves(John,x), (Ghost(x) ⇒ (∃x Supernatural-being(x)))

are *open*, i.e., contain at least one occurrence of a free variable. (Note that in the second formula one occurrence of x is free, the other is bound; the two occurrences have nothing to do with each other – the formula is equivalent to (Ghost(x) ⇒ (∃y Supernatural-being(y))).)

Closed formulas are the ones that are similar to English sentences, while open formulas are incomplete, in the sense that they are "missing" at least some quantifiers. From that perspective, it is sufficient to define truth for closed formulas.

For a closed formula ϕ we can define truth in a model $\mathcal{M}$, written $\models_{\mathcal{M}} \phi$, as

$$\models_{\mathcal{M}} \phi \ \text{ iff for all variable assignments } \mathcal{U},\ \models_{\mathcal{M}} \phi\, [\mathcal{U}].$$

In other words, $\mathcal{U}$ satisfies ϕ relative to $\mathcal{M}$, regardless of $\mathcal{U}$. But it is easy to see that we could just as easily have said

$$\models_{\mathcal{M}} \phi \ \text{ iff for } \textit{some} \text{ variable assignment } \mathcal{U},\ \models_{\mathcal{M}} \phi\, [\mathcal{U}],$$

since for closed wffs, one variable assignment is as good as any other; the quantifiers binding the variables "try out" different values for them, regardless of the "initial" values.

The text also defines truth for open formulas, using the first definition above (quantifying over *all* assingments). In other words, G & N wish to be able to treat open formulas as if they made complete, well-defined statements – namely, as if all free variables were universally quantified. Perhaps they do this in anticipation of *clause form*, used later in the text in connection with inference based on the *resolution principle*; in clause form, all quantifiers are implicit. However, this leads to some technical problems which are not quite properly dealt with in the text, so let's limit ourselves to talking about the truth of *closed* formulas.

The important point is that we now have a complete notion of how a given first-order language can be interpreted so as to make true (or false) statements about a given domain of discourse. This puts us in a position to define *logical consequence*, and to relate logical consequence to *deductive inference*.