Crash course on higher-order logic*

Theodore Sider December 2, 2022

Contents

*Thanks to Daniel Berntson, Eliya Cohen, Cian Dorr, John Keller, Brock Sides, Eric Winsberg, and Alessandro Torza for comments. This was written for a graduate seminar I taught in the spring of 2020. Despite its flaws (such as lack of rigor and comprehensiveness), it may have some value as a first port of call. Other introductions to this material have started to appear, such as the first chapter of Dorr et al. (2021) and Bacon (2022).

1. Introduction

"Higher order metaphysics" is a hot topic, and an excellent one. The work is at a high level, and is intrinsically interesting. It is relatively new, so there is still much to be done. And it connects to adjacent areas, such as logic, philosophy of logic, philosophy of mathematics, and philosophy of language. Getting up to speed on higher-order matters is a good way to broaden your horizons.

But much of the literature on higher-order metaphysics is very hard to understand, if you're coming in from the outside. It is often technical, and familiarity with various issues in logic, philosophy of logic, and philosophy of mathematics is often presupposed.

For any subject, if you lack the background, the experience of reading an advanced paper can be very scary. Nothing makes any sense! And this is especially true if a lot of the paper is written in symbols that you don't understand, or barely understand. It is easy to just assume that the topic is not for you, and move on to something else. But the situation is rarely as bad as it seems; usually, all you need is a little background. Then the papers won't be anywhere near as scary, and you'll understand most of what is going on; and you'll be able to pick up the rest as you continue.

The necessary background for higher-order metaphysics isn't that difficult. But it's spread out in many different places, and (as far as I know) isn't presented anywhere in a concise and introductory way. This document will go through this background slowly and informally, without (I hope!) presupposing knowledge of anything more than introductory logic. This background will include some basic logic, philosophy of logic, philosophy of mathematics, and philosophy of language.

This is intended to be a very informal introduction. I will sacrifice rigor and comprehensiveness to make things accessible.[1]

2. Importance of syntax to logic

In the literature on higher-order metaphysics, syntax—that is, grammar—often matters a great deal. This might seem strange. So I'd like to start by talking about why syntax is important in logic.

Logic is largely about logical implication, about what follows from what.

"Logical implication" means something like: truth preservation in virtue of logical form. For example, 'Jones is a sister' does not logically imply 'Jones is a sibling', because even though the latter is true whenever the former is, this is not because of their logical forms; it is because of the meanings of 'sister' and 'sibling'.

Since logical implication is by virtue of form, our statements of logical rules, such as conjunction elimination:

$$\frac{A \text{ and } B}{A}$$

talk about logical form. The statement of the logical rule says: any sentence that has the logical form above the line (namely, "A and B") logically implies the corresponding sentence below the line. Other rules concern sentences with other logical forms, such as "A or B", "If A then B", and so on.

But we need a clear account of syntax in order to state such rules properly. Why isn't the following an instance of conjunction elimination?

[1] For instance, when sketching formal semantics for quantified languages, I won't deal with variables properly; I'll say things like "$\forall v A$ is true iff A is true of every member of the domain".

Daisy and Luke are siblings

Therefore, Daisy

It's because the variables A and B in the rule of conjunction elimination are supposed to stand for *sentences*, and the word 'and' in the rule is assumed to occur between sentences. Thus conjunction elimination and other logical rules use certain syntactic concepts (such as that of a sentence) and presuppose a certain syntax for the language to which they're applied.[2]

The success of a proposed logic depends crucially on the quality of its assumptions about syntax. To take a famous example, consider how Frege's logic supplanted Aristotle's.[3] Aristotle's syllogistic logic had been the dominant conception of logic for centuries, but in the late nineteenth century, Frege (and others) showed that Aristotle's logic was too weak, and developed a much more powerful logic that we now know as predicate logic. Aristotle's logic was limited in power precisely because of its too-crude syntactic assumptions, namely that the sentences that can occur in syllogisms always have subject-predicate form ('a is a G', 'All Fs are Gs', 'Some Fs are Gs', 'No Fs are Gs', etc.), with a single subject and single predicate. Frege's main innovations were syntactic. Frege allowed predicates to have multiple subjects, as in 'Jones respects Smith', in which the predicate is 'respects' and the subjects are 'Jones' and 'Smith', and replaced Aristotelian general subjects 'All Fs', 'Some Fs', and 'No Fs', with quantifiers $\forall x$ and $\exists x$ (not his notation), which could be attached to sentences containing sentential connectives such as $\wedge$, $\vee$, $\rightarrow$, and $\sim$. Without developing this new syntax, one couldn't even state familiar logical rules of inference such as conjunction elimination, existential generalization, and so on. With the new syntax, Frege was able to explain the correctness of arguments that could not be explained by Aristotle, such as:

Someone respects everyone

Therefore, everyone is respected by someone (or other)

The best representation Aristotle could have given these sentences would have

[2] In typical formal languages, the symbol for 'and' (e.g., '$\wedge$') can only occur between sentences. 'And' in English has a more flexible syntax. For example, in addition to connecting sentences, it can connect names to form complex plural subjects. Stating systematic rules of inference for a natural language like English is much harder than stating rules for formal languages with a simpler, and stipulated, syntax.

[3] The history here is actually more complex than the following caricature suggests.

been:

Some Fs are Gs

Therefore, every F is an H

where F stands for 'is a thing' (a predicate true of everything), G stands for 'respects-everyone', and H stands for 'is-respected-by-someone'. This is obviously not a valid syllogism. The problem is that Aristotle has no way of "breaking up" the predicates G and H. His syntax allows him no way of recognizing the further logical structure they contain. But Frege's syntax does let him recognize this structure; he can represent the sentences thus:

$\exists x \forall y Rxy$

$\forall y \exists x Rxy$

In Frege's logic, the second sentence does indeed follow from the first.

2.1 Syntax in formal languages

Syntax has to do with what combinations of symbols "make sense", or are "well-formed". In English, 'Sally owns a dog' is a grammatical sentence, and 'owns a dog' is a grammatical verb phrase. In contrast, 'Sally dog a owns' isn't a grammatical sentence, and 'dog a owns' isn't a grammatical verb phrase. Those strings of words don't make sense; they aren't English. Concepts like *sentence* and *verb phrase* are syntactic concepts.

In a modern approach to logic, one doesn't deal with natural languages, because their syntax is too complex. Rather, one develops formal languages. The syntax of a formal language is typically similar to that of natural languages in certain ways, but simpler, free of ambiguity, and stipulated.

To give a syntax for a formal language, one gives a rigorous definition of what counts as a grammatical, or "well-formed" formula—or just "formula" for short. The syntactic concept of *formula* is the analog in the formal language of the syntactic concept of being a grammatical sentence of English. Here is a typical definition:

1. "$Rt_1 \ldots t_n$" is a formula, for any n-place predicate R and any n terms (i.e., names or variables), $t_1, \ldots, t_n$
2. If A is a formula, so is "$\sim A$"
3. If A and B are formulas, so are "$(A \wedge B)$", "$(A \vee B)$", "$(A \rightarrow B)$", and "$(A \leftrightarrow B)$"
4. If A is a formula then so are "$\forall v A$" and "$\exists v A$", where v is any variable
5. A string of symbols is a formula only if it can be shown to be a formulas using rules 1–4

To illustrate: $\forall x(Fx \rightarrow Gx)$ is a formula (assuming that x is a variable and F and G are one-place predicates), since: by clause 1, Fx and Gx are formulas; and so by clause 3, $(Fx \rightarrow Gx)$ is a formula; and so, by clause 4, $\forall x(Fx \rightarrow Gx)$ is a formula. But $\forall F \rightarrow)x$ is not a formula, since it can't be shown to be a formula by sequential application of rules 1–4.

Note, by the way, the use of the concepts of *predicate* and *term* in this definition. Like the concept of a formula, these are syntactic concepts. (They are partly, but only partly, analogous to the natural-language syntactic concepts of *verb phrase* and *noun phrase*.)

In logic, then, expressions fall under syntactic categories (predicate, term, formula). Moreover, we make reference to those categories when we state logical principles. For example, the rule of conjunction elimination, for a formula language like the one we just developed, is:

$$\frac{A \wedge B}{A}$$

A fuller statement makes the reliance on the syntax clear:

> For any formulas A and B, the formula $A \wedge B$ logically implies A.

3. First- versus second-order logic

3.1 Syntax

The kind of logic usually taught in introductory philosophy classes is what is known as "first-order logic". A different kind, "second-order" logic (and indeed, third- and higher-order logic) will be important for us.

The difference between first- and second-order logic begins with a difference in syntax. The definition of a formula we considered in the previous section was the one for first-order logic. Given that definition, there are formulas like these:

$$\forall x Gx \qquad \exists x \exists y Bxy$$

but there are no formulas like the following:

$$\exists F Fa \qquad \forall R(Rab \rightarrow Rba)$$

in which 'F' and 'R' are variables. That is, variables are allowed to occur in *subject* position only (the syntactic positions in which you're allowed to fill in names), not predicate position. But in second-order logic, variables *are* allowed to occur in predicate position, and so the second pair of expressions do count as formulas.

To state the syntax for second-order logic, first, we make a distinction amongst variables. Variables now come in two (disjoint) types: *individual variables* $x, y, z \ldots$ (these are the old style) and, for each natural number $n \geq 1$, n-place *predicate variables*: $F, X, R, \ldots$. Second, we change the first clause in the definition of a formula:

> 1′. For any n-place predicate *or predicate variable* R and any n terms (i.e., names or *individual* variables), $t_1, \ldots, t_n$, "$Rt_1 \ldots t_n$" is a formula.

Other than that, everything stays the same.

What do formulas like $\exists F Fa$ and $\forall R(Rab \rightarrow Rba)$ *mean*? Well, there isn't really a fixed answer—different people use such formulas to mean different things. But on one common usage, these formulas quantify over properties.

Thus the first means, to a first approximation, that a has some property; and the second means that b bears every relation to a that a bears to b.

This is only an approximation; and saying something more accurate would be getting ahead of ourselves. But right here at the start we can make an important point. Consider:

> a has some property

Let's make a distinction between a *first-order* and a *second-order* symbolization of this sentence. The second-order symbolization is the one we've just met: $\exists FFa$. The first-order symbolization, on the other hand, is:

$$\exists x(Px \wedge Hax)$$

where 'P' is a one-place predicate symbolizing 'is a property' and 'H' is a two-place predicate symbolizing 'has' (i.e., instantiates). This second symbolization really is a first-order sentence, not a second-order sentence, since its variable (namely, x) only occurs in subject position.

Thus the difference between first-order and second-order logic *isn't* that the latter is needed to talk about properties. One can talk about any entities one wants, including properties, using first-order logic, since there is no limit to the kinds of entities that can be in the range of the first-order quantifiers '$\forall x$' and '$\exists x$'. If there are such things as properties, we can quantify over them; and if it makes sense to speak of objects *having* these properties, then we can introduce a predicate H for having in a first order language.

(Caveat: sometimes the term 'second-order' (or 'higher-order') is used loosely, to refer to all talk about properties. But in this document I will be using the term in the narrowly logical sense, to refer to sentences in which quantified variables occur in predicate position—and later, other non-subject positions.)

But this naturally raises the question of what the difference in meaning is, between the second-order sentence $\exists FFa$ and the first-order sentence $\exists x(Px \wedge Hax)$. We'll be getting to that soon; but to preview: some people think that there is no difference, whereas others regard second-order quantifiers as being *sui generis*, and thus regard such sentences as not being equivalent.

3.2 Formal logic and logical consequence

Modern mathematical logic is a branch of mathematics. You set up a bunch of definitions—for instance, the notion of a formula in a given formal language, the notion of an interpretation, the notion of a formula being true in an interpretation, etc.—and then you give mathematical proofs of various facts given the definitions—e.g., that the formula $\forall x(Hx \vee \sim Hx)$ is true in all interpretations.

Unless you think there is something wrong with mathematics itself, there can be no disputing the results of mathematical logic. But these results are about stipulatively defined notions. It's a *non*mathematical claim that if a formula (such as $\forall x(Hx \vee \sim Hx)$) is true in all interpretations in some stipulatively defined sense, then meaningful natural language sentences represented by that formula (such as 'Everything is either a human or not a human') are *logical truths*, in a sense that is *not* stipulatively defined. Similarly, it is an indisputable mathematical truth that in every interpretation in which a formula $\sim\sim Ha$ is true, the formula H is true; but it is a nonmathematical claim that, for example, "It's not the case that it's not the case that Ted is human" *logically implies* (again, in a nonstipulated sense) "Ted is human". And indeed, some people deny these claims about logical truth and logical implication, despite admitting the mathematical results. For example, there is a school of thought about logic called intuitionism that rejects both double negation elimination and the law of the excluded middle.

One of the main points of logic is to study the notions of logical truth and logical implication. These notions are not stipulatively defined. They are objects of philosophical reflection, just like knowledge or persistence or goodness or beauty. In mathematical logic we devise various formal gizmos, but it is a philosophical claim that these formal gizmos accurately represent or depict or model logical truth and logical consequence.

(When I say that there are philosophical questions about logical truth and implication, I don't mean to prejudge the question of how deep or substantive those questions are. As I mentioned earlier, it is common to say that 'Jones is a sister' does not *logically* imply 'Jones is a sibling' since this implication does not hold in virtue of "form", but rather in virtue of the meanings of 'sister' and 'sibling'. But a paradigm of logical implication, such as the implication of 'Jones is a sister' by 'Jones is a sister and Jones is a lawyer', also holds by virtue of meaning (in whatever sense the first implication does), namely, by virtue of the

meaning of 'and'. Why do we count 'and', but not 'sister' or 'sibling', as being part of the "forms" of sentences? Put another way, why is 'and', but not 'sister' or 'sibling', a *logical constant*?[4] It's an open question how deep the answers to such questions are, and thus an open question how deep the questions of logical truth and logical consequence are along certain dimensions.[5])

To sum up: it's important to distinguish the notions of truth and logical consequence—whose meanings are not stipulated—from the stipulative notions we construct in mathematical logic (such as truth-in-all-interpretations-of-a-certain-sort), which may or may not yield an accurate model of the former notions.

3.3 Semantics

There are two main kinds of formal/mathematical models of the notions of logical truth and logical consequence: *semantic* models and *proof-theoretic* models.

Let's start with semantic models. Here is the overall idea:

Semantic approach

1. Define *interpretations*—mathematically precise "pictures" of logical possibilities
2. Define the notion of a sentence's being *true-in* a given interpretation
3. Use these notions to define metalogical concepts. E.g., a sentence is *valid* iff it is true in all interpretations; a set of sentences Γ *semantically implies* a sentence S iff S is true in every interpretation in which every member of Γ is true.

[4] See MacFarlane (2005) for an overview of this issue.

[5] But note that a failure of depth about what is, e.g., *logically true* wouldn't imply a failure of depth about what is *true*. The disagreement between intuitionists and classical logicians isn't just about what is logically true, e.g., whether 'the decimal expansion of π either does or does not contain some sequence of 666 consecutive 6s' is a logical truth. It also is about, e.g., whether the decimal expansion of π either does or does not contain some sequence of 666 consecutive 6s; and the substantivity of this latter disagreement isn't undermined by nonsubstantivity of what counts as a logical constant.

For first- and second-order logic, the usual definition of interpretation is exactly the same:

Intepretation
(for both first- and second-order logic)

A nonempty set, D (the "domain"), plus an assignment of an appropriate denotation based on D to every nonlogical expression in the language. Names are assigned members of D; one-place predicates are assigned sets of members of D; two-place predicates are assigned sets of ordered pairs of D; and so on.

But the definition of truth in an interpretation differs a little between first- and second-order logic. In the case of first-order logic, the definition looks roughly like this:[6]

Definition of truth

i) A sentence Fa is true in an interpretation I if and only if the denotation of the name a is a member of the denotation of the predicate F; a sentence Rab is true if and only if the ordered pair of the denotation of the name a and the denotation of the name b is a member of the denotation of the predicate R; etc.

ii) A sentence $\sim A$ is true in I if and only if A is not true in I; a sentence $A \wedge B$ is true in I if and only if A is true in I and B is true in I; etc.

iii) A sentence $\forall vA$ is true in I if and only if A is true of every member of D; a sentence $\exists vA$ is true in I if and only if A is true of some member of D.

In the case of second-order logic, we keep these three clauses in the definition of truth in an interpretation, and add a clause for quantified sentences with second-order quantifiers:

[6] These and other such definitions in this document are not formally rigorous, particularly in the clauses for quantified sentences.

iv) Where R is an n-place predicate variable, a sentence $\forall R A$ is true in I if and only if A is true of every set of n-tuples of members of D; a sentence $\exists R A$ is true in I if and only if A is true of some set of n-tuples of members of D.

Here is the informal summary. In semantics, first- and second-order logic use the same notion of an interpretation: namely, a domain together with denotations for names and predicates. As for the definition of truth-in-an-interpretation, the second-order definition is just like the first-order one, except that we add a provision for second-order sentences: the second-order quantifiers $\forall R$ and $\exists R$ range over the n-tuples of the domain.

3.4 Proof theory

The second main kind of formal/mathematical model of the notions of logical truth and logical implication is the proof-theoretic model. Here the idea is that logical implication is *provability*: a set of formulas Γ logically implies a formula A if and only if there exists a *proof* of A from Γ.

The idea of a proof can be made precise in a number of ways. Here is a particularly simple one (called a Hilbert system of proof). First one chooses a set of formulas called *axioms*, and a set of relations over formulas called *rules*. We say that a formula S *follows via rule* R from other formulas $S_1, \ldots, S_n$ iff the formulas $S_1, \ldots, S_n, S$ stand in R. (For example, the rule *modus ponens* is the relation that holds between any three formulas of the form A, $A \rightarrow B$, and B.) And then we define proofs thus:

A *proof of* A *from* Γ is a finite series of formulas, the last of which is A, in which each formula is either i) a member of Γ, ii) an axiom, or iii) follows from earlier lines in the series by a rule

And finally we say that Γ proves A iff there exists some proof of A from Γ

One more bit of terminology: a *theorem* is a formula, A, such that there exists a proof of A from the empty set $\varnothing$. What is a proof from $\varnothing$? Well, if you think

about the definition of a proof from Γ, a proof from $\varnothing$ is a proof in which no premises (other than axioms) are allowed. Thus a theorem is a formula that can be proven from the axioms alone—this is the proof-theoretic notion of a logical truth.

The idea is to choose axioms that are obviously logical truths—statements such as:

$$\big((A \vee B) \wedge \sim A\big) \rightarrow B$$

and to choose rules that are obviously logical implications, such as modus ponens. Thus a proof of A from Γ is just a chain of good arguments, starting from Γ and culminating in A.

There are other ways of making provability precise.[7] For instance, many other systems of proof allow one to make temporary assumptions, as when one assumes the antecedent of a conditional in a conditional proof, or when one assumes the opposite of what one is trying to prove in a reductio. But let's stick with the Hilbert approach.

The Hilbert approach to provability can be taken with both first- and second-order logic. For example, in first-order logic, one axiom system looks like this:[8]

[7] But all ways of defining what counts as a proof share the feature of being *mechanically checkable* in a certain sense. For example, to check whether a finite series of formulas counts as a legal proof in a given Hilbert system, start with the first formula of the series. Is it a member of Γ or an axiom? If not, we don't have a legal proof. Otherwise move on to the second formula. Is it a member of Γ or an axiom? If not, does it follow from earlier formulas in the series by some rule? If not, then we don't have a legal proof. Otherwise, on to formula three. Eventually we'll reach the final formula, and will have checked whether the series counts as a proof. Note that in any Hilbert system, it is always required that it be mechanically decidable what counts as an axiom or rule of that system.

[8] See Shapiro (1991, section 3.2) for this system and the one for second-order logic. That book is a generally useful although not easy source of information on second-order logic. In universal instantiation, t must be a term that is "free for v in A"; in universal generalization, v must be a variable that doesn't occur freely in A or any member of Γ.

Axioms:

$$A \to (B \to A)$$
$$\big(A \to (B \to C)\big) \to \big((A \to B) \to (A \to C)\big)$$
$$(\sim A \to \sim B) \to (B \to A)$$
$$\forall v A \to A_{v \mapsto t} \qquad \text{(universal instantiation)}$$

Rules:

$$\frac{A \qquad A \to B}{B} \quad \text{(modus ponens)} \qquad\qquad \frac{A \to B}{A \to \forall v B} \quad \text{(universal generalization)}$$

In the axioms, "$A_{v \mapsto t}$" means: the result of starting with A, and then replacing every free occurrence of v with an occurrence of t. For example, if A is $F v$, then $A_{v \mapsto t}$ would be $F t$.

The displayed "axioms" are in fact not axioms. Rather, they are axiom *schemas*. Take the first schema, $A \to (B \to A)$. The letters "A" and "B" are schematic variables, which stand for any formulas. When you replace them with formulas, the result is called an *instance* of the schema. Here are some instances of the schema $A \to (B \to A)$:

$$P \to (Q \to P)$$
$$R \to (S \to R)$$
$$\sim(P \to Q) \to \big((Q \to R) \to \sim(P \to Q)\big)$$
$$\text{etc.}$$

The idea, then, is that *every* instance of $A \to (B \to A)$ counts as an axiom; and similarly for the other three axiom schemas. Thus there are infinitely many axioms, not just four; but each axiom falls under one of four patterns.

To get various proof systems for second-order logic, we can just add new axioms and/or rules to those of the above system for first-order logic. For instance:

New axiom schemas

$$\forall X A \to A_{X \mapsto F} \qquad \text{(second-order universal instantiation)}$$
$$\exists X \forall v_1 \ldots \forall v_n (X v_1 \ldots v_n \leftrightarrow A) \qquad \text{(Comprehension)}$$

(X is any n-place predicate variable, F is any n-place predicate, and A is a schematic variable for any formula, which cannot have X free.)

Second-order universal instantiation is just like the old version of universal instantiation, except that the variable is second-order. Here is an example of an instance:

$$\forall R(Rab \to Rba) \to (Tab \to Tba)$$

where R is a two-place predicate variable and T is a two-place predicate constant. (We might similarly add a second-order version of the rule of universal generalization.)

To get a feel for the Comprehension axiom, let A be the following formula:

$$Hx \wedge \exists y(Cy \wedge Oxy)$$

which might symbolize, e.g.,

x is a hipster who owns at least one chicken

We can then write down the following instance of Comprehension:

$$\exists X \forall x\big(Xx \leftrightarrow Hx \wedge \exists y(Cy \wedge Oxy)\big)$$

where X is a one-place predicate variable. In other words: there exists a property which is had by an object if and only if that object is a hipster who owns at least one chicken.

Thus what Comprehension says, roughly, is that to any formula A there is a corresponding property or relation. (Thus this is a lot like the principle of naïve comprehension from set theory. As we will see later, it doesn't lead to the same trouble.)

3.5 Metalogic

Here's what we have done so far. For each of first- and second-order logic, we've laid out i) a syntax for the language, ii) a semantics, and iii) a proof theory. We are now in a position to understand some of the dramatic differences between first- and second-order logic. The differences have to do with the behavior of the metalogical notions introduced above: e.g., provability, semantic consequence, and so on.

(The notions are called "metalogical" to distinguish them from logical notions such as "and", "not", and "all". Metalogical notions are notions that we employ when we are talking about formal logical languages and investigating their properties.)

3.5.1 Completeness

Let's start with the relationship between theoremhood and validity. As we saw earlier, these are two different ways of formalizing the intuitive notion of logical truth. However, as Kurt Gödel proved in his dissertation in 1929, the two formalizations are equivalent in the case of first-order logic. Gödel proved:

Completeness If a first-order formula is valid then it is a theorem

It's relatively straightforward to prove the converse:

Soundness If a first-order formula is a theorem then it is valid

Thus the two results together tell us that a first order theorem is a theorem *if and only if* it is valid—i.e., that for first-order logic, theoremhood is equivalent to validity.

However, for second-order logic, the situation is very different. The axiom system mentioned above is indeed sound: every theorem is valid. However, that system isn't complete: some valid formulas aren't theorems. Moreover, this isn't just a failing of the particular system mentioned above. It follows from Gödel's first *in*completeness theorem (which he proved two years later, in 1931) that there can be *no* sound and complete axiom system for second-order logic.

There are some details here that matter. First, there is a trivial sense in which we *can* come up with a complete axiom system for second-order logic, if we

allow systems with "rules" like the following:

$$\frac{A_1, A_2, \ldots}{B} \text{ (where } B \text{ is any formula that is semantically implied by } A_1, A_2, \ldots)$$

or if we are allowed to define the "axioms" as being all the valid formulas. In a proper axiomatic system, it is required that there be a mechanical procedure—in a certain precise sense—for deciding what counts as a rule or axiom.

Second, the claim that there can be no sound and complete axiom system for second-order logic is tied to the definition of an interpretation for second-order logic that was given in section 3.3. The interpretations defined there are often called "standard" or "full" interpretations, since in them the second-order quantifiers range over *all* sets of n-tuples drawn from the domain. But one can define other notions of interpretation, in which the second-order quantifiers range over only some such sets; and there is no guarantee that metalogical facts that hold under one notion of interpretation will continue to hold under other notions of interpretation. For example, if "Henkin interpretations" are substituted for full interpretations, then one can have a sound and complete axiom system after all. Similar remarks apply to other metalogical differences between first- and second-order logic to be discussed below.

This is philosophically important. It means that, in order to draw substantive philosophical conclusions from metalogical facts like the incompleteness of second-order logic (a procedure which is philosophically fraught anyway, as we will see) one must be sure that the definition of interpretation involved in the metalogical facts matches (as much as possible!) the intended interpretation of the second-order quantifiers.

3.5.2 Compactness

Call a set of formulas, Γ, *satisfiable* if and only if there is some interpretation in which every member of Γ is true. (This is a semantic notion of consistency.) The following holds for first-order logic (it is a fairly immediate consequence of the completeness theorem):

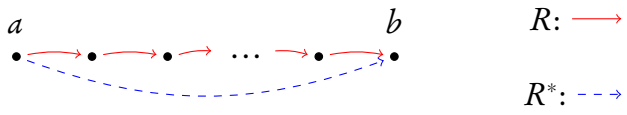
Compactness If every finite subset of Γ is satisfiable, Γ is satisfiable.

To get a sense of why this is important, consider the following. Let R be any two-place predicate. (Not a predicate variable; a predicate constant.) Informally

put, the *ancestral* of R, R^*, is a predicate obeying the following rule:

R^*ab iff: Rab, or
Rax and Rxb, for some x, or
Rax and Rxy and Ryb, for some x and y, or …

That is: a is an R-ancestor of b iff there is some "finite R-chain" from a to b:



('Ancestral' because ancestor-of is the ancestral of parent-of.)

In fact there is no way to define R^* using first-order predicate logic. For suppose otherwise—suppose there is some sentence, R^*ab, of predicate logic that says that a is an R-ancestor of b. Now consider the following infinite set, Γ, of sentences:

$$\Gamma = \{R^*ab, A_1, A_2, A_3, \ldots\}$$

where the sentences $A_1, A_2, A_3, \ldots$ are the following:

A_1 : $\sim Rab$ ("There is no one-link R-chain")
A_2 : $\sim\exists x(Rax \wedge Rxb)$ ("There is no two-link R-chain")
A_3 : $\sim\exists x\exists y(Rax \wedge Rxy \wedge Ryb)$ ("There is no three-link R-chain")
etc.

Notice two things about Γ: 1. Γ is unsatisfiable—that is, there is no interpretation in which every member of Γ is true. For if R^*ab is true in an interpretation then there is some finite R-chain from a to b, in which case one of the A_is must be false in that interpretation. 2. Every finite subset of Γ is satisfiable. For any finite subset of the A_is merely rules out finite chains between a and b up to some particular length (the length being whatever the highest of the A_is is in the finite subset), and R^*ab can still be true if there is a chain longer than that between a and b.

But 1 and 2 together contradict Compactness. Therefore there can exist no such sentence R^*ab. You can't define the ancestral of a predicate using first-order logic.

Thus compactness tells us something very important about first-order logic: that language is expressively weak in a certain way. (There are other ways in which first-order logic is expressively weak. One other related example is that you can't express in first-order logic the idea that there are only finitely many things.)

Second-order logic is very different in this respect. Compactness does *not* hold for second-order logic. And in second-order logic you *can* define the ancestral of a predicate:

$$R^*ab \leftrightarrow \forall F\Big(\Big(\forall x(Rax \rightarrow Fx) \wedge \forall x\forall y((Fx \wedge Rxy) \rightarrow Fy)\Big) \rightarrow Fb\Big)$$

What this says, intuitively, is this:

> a is an R-ancestor of b if and only if for every property, F: IF i) everything that a bears R to is an F, and ii) whenever an F bears R to something, that something is also an F, THEN b is an F

or, to put it differently:

> a is an R-ancestor of b if and only if b has every property that i) is had by every "R-child" of a, and ii) is "closed under" R

Thus there is a sense in which second-order logic is more expressively powerful than first-order logic: you can say more with the language of second-order logic.

3.5.3 Clarifying differences in expressive power

But it's important to be clear about the exact sense in which second-order logic is more expressively powerful. After all, consider the following first-order sentence, in which '$\in$' is a two-place predicate for set-membership:

$$R^*ab \leftrightarrow \forall z\Big(\Big(\forall x(Rax \rightarrow x \in z) \wedge \forall x\forall y\big((x \in z \wedge Rxy) \rightarrow y \in z\big)\Big) \rightarrow b \in z\Big)$$

What this says is that b is a member of every set, z, that contains every R-child of x and is closed under R. This clearly seems like an acceptable definition of R^*; but it's first-order. So what is going on?

Here is the answer. When I said above that no sentence R^*ab "says" that a is an R-ancestor of b, what I meant was that there is no sentence that says this in *every interpretation*. A little more exactly:

> There is no first-order sentence R^*ab, such that in any interpretation I, R^*ab is true in I if and only if, where r is the set of ordered pairs that is denoted by R in I, there is a finite chain of members of the domain of I, pairwise connected by r, leading from the denotation of a to the denotation of b.

The set-theoretic sentence above does succeed in saying that a is an R-ancestor of b *in set-theoretic interpretations*—that is (roughly) in interpretations in which the domain contains a suitable number of sets, and in which '$\in$' denotes set-membership. But there are plenty of non-set-theoretic interpretations in which the set-theoretic sentence doesn't express the ancestral, for instance interpretations in which '$\in$' means something that has nothing to do with sets.

Thus there is a sense in which the usual semantics for second-order logic "stacks the deck" in its favor. It is hard-wired into that semantics that the second-order quantifier $\forall F$ ranges over subsets of the domain, and that second-order predications Fx express set membership. We simply don't consider second-order interpretations in which Fx means anything different from "the denotation of x is a member of the set denoted by F", whereas we do consider second-order interpretations in which '$x \in y$' does not mean 'the denotation of x is a member of the denotation of y'—the reason, in the latter case, is that $\in$ is not treated as a logical constant. It's *not* hard-wired into the semantics for first-order logic that the nonlogical constant $\in$ expresses set-membership, or that the domain contains the sets it would need for the set-theoretic definition to work. We will return to this sort of issue later.

3.6 Metamathematics

There are further important differences between first- and second-order logic, which emerge when we consider the differences between first- and second-order formulations of mathematical theories.

3.6.1 Skolem's paradox

Call a *model* of a set of sentences, an interpretation in which every sentence in the set is true. In first-order logic, the following holds:[9]

Löwenheim-Skolem theorem If a set of sentences, Γ, has a model, it has a model whose domain is at most countably infinite.

Thus, for example, no matter what first-order axioms we write down for set theory, say, or for the theory of real numbers, provided those axioms are satisfiable at all, there is some interpretation whose domain has no more elements than the natural numbers, but in which all the axioms are true.

But how can that be? After all, you can prove in set theory, for example, that there exist sets that are larger than the set of natural numbers.

This is called "Skolem's paradox". Really it isn't a paradox at all. You can prove in set theory a sentence in the language of set theory—containing the predicate '$\in$'—that says, *given its intended interpretation*, that there are sets larger than the set of natural numbers. The interpretations that the Löwenheim-Skolem theorem tells us about, in which the domain is the size of the natural numbers, aren't the intended interpretation. '$\in$' doesn't mean set-membership in such interpretations, and the domain won't contain all the sets.[10]

Skolem's paradox does bring out something important, however, about first-order logic: that sets of first-order sentences can't pin down the intended interpretation of their vocabulary solely by virtue of logic. Take any set of sentences about the real numbers. If it has an intended model, in which the domain is the set of real numbers, it must also have another nonisomorphic model in which the domain has the size of the natural numbers.

Second-order logic is different in this respect. The Löwenheim-Skolem theorem doesn't hold for second-order logic. (But note again that it is just hard-wired into the standard definition of a second-order interpretation that, e.g., "monadic predication means set-membership".)

3.6.2 Nonstandard models of arithmetic

First some terminology.

[9] This is just one of a number of "Löwenheim-Skolem" theorems.

[10] Indeed, in *no* interpretation can the domain contain all the sets.

First-order language of arithmetic: the first-order language with symbols 0, $'$, $+$, and $\cdot$.

Second-order language of arithmetic: the second-order language with those symbols.

Standard interpretation: the interpretation whose domain is the set of natural numbers, and in which '0' denotes the number 0, '$'$' denotes the successor (or add-one) function, '$+$' denotes the addition function, and '$\cdot$' denotes the multiplication function.

So: you can write things like these this in the first-order language of arithmetic: '$0 = 0$', '$0' + 0' = 0''$', '$\exists x\, x \cdot 0'' = 0''''$', and so on. And in the standard interpretation, they mean, respectively, that the number 0 is self-identical, that $1 + 1 = 2$, and that there is some number that when multiplied by 2 yields the number 4; and they are all true in that interpretation.

The second-order language of arithmetic is the same, but allows predicate variables in addition to individual variables. Thus '$\exists F\, F0$' ("the number zero has at least one property") is a formula in the second-order language of arithmetic, though not the first-order.

OK. Here is a question. Is there any set of sentences in the first-order language of arithmetic whose *only* model is the standard interpretation?

The answer to this is trivially no. Suppose the standard interpretation is a model of some set of arithmetic sentences, Γ. We can construct a new interpretation in which all the members of Γ are also true, by "swapping" the numbers 2 and 0: let '0' denote the number 2 in the new interpretation, let '$'$' denote the function that maps 2 to 1, 1 to 0, 0 to 3, and so on; and similarly for '$+$' and '$\cdot$'. For that matter, we could construct an interpretation just like the standard interpretation but in which Julius Caesar is swapped in for 0, both in the domain and in the interpretations of 0, $'$, $+$, and $\cdot$; and the members of Γ will all be true in this interpretation as well.

In general, exactly the same sentences are true in "isomorphic" models: models that have the same pattern, but in which the identities of the objects have been altered.

OK, so the standard interpretation can't be a unique model of any set of sentences. But let's ask a further question: is there any set of sentences whose models are all *isomorphic* to the standard interpretation?

More surprisingly, the answer to this question is also no.[11] Let Γ be any set of sentences that are all true in the standard interpretation. In fact, let Γ be the set of *all* such sentences. In addition to having the standard interpretation as a model, Γ has what are called "nonstandard models", which are models that are structured as follows. They consist in part of a copy of the standard model: there is the model's "zero"—that is, the member of the domain that is denoted by '0'. Then there is the model's "one"—that is, the member of the domain that is the successor of the model's zero (that is: the member of the domain that results by applying the model's denotation of '′' to the model's denotation of '0'). And so on. But in addition to these "standard numbers", the model also contains infinitely many (and densely ordered) other series of numbers, each of which is structured like the negative and positive integers: discrete, but without beginning or end.

$$\underbrace{0, 1, 2, \ldots}_{\text{standard numbers}} \quad \ldots \quad \underbrace{\ldots a_{-1}, a_0, a_1, \ldots}_{\text{some nonstandard numbers}} \quad \ldots \quad \underbrace{\ldots b_{-1}, b_0, b_1, \ldots}_{\text{more nonstandard numbers}} \quad \ldots$$

Thus we again have an example of the expressive weakness of the language of first-order logic. Using first-order sentences in the language of arithmetic, there is no way to "force" models to look like the standard interpretation.

The situation is very different with second-order logic. There is a second-order sentence in the language of arithmetic, all of whose models are isomorphic to the standard interpretation. (That's not to say that this sentence gives us an axiomatic basis for arithmetic: since the completeness theorem fails for second-order logic, there is no axiomatic method we can use to draw out the consequences of this sentence.)

3.6.3 Schematic and nonschematic axiomatizations

There is a further difference between first- and second-order logic in the formulation of mathematical theories that will be very important going forward.

[11] There is actually a further reason for the answer being no: because of the "upward" Löwenheim-Skolem theorem, if the standard interpretation is a model of Γ, Γ must also have models in which the domain has cardinality greater than that of the natural numbers. The nonstandard models discussed in the text have domains of the same size as the natural numbers.

Let's start with arithmetic. Consider the project of giving axioms for arithmetic: axioms whose consequences (perhaps proof-theoretic consequences, perhaps semantic consequences—we'll confront this question later) are some, perhaps all, of the truths of arithmetic. How will we do this?

Here are some of the axioms we will want:

$$\forall x \forall y (x' = y' \to x = y)$$
$$\forall x\ 0 \neq x'$$
$$\forall x (x \neq 0 \to \exists y\ x = y')$$
$$\forall x\ x + 0 = x$$
$$\forall x \forall y\ x + y' = (x + y)'$$
$$\forall x\ x \cdot 0 = 0$$
$$\forall x \forall y\ x \cdot y' = (x \cdot y) + x$$

Never mind the details (though these should look like intuitively correct and very basic statements about arithmetic). What's important is that these axioms are not enough. A further key principle about arithmetic is the principle of *induction*. This says, roughly, that if i) the number 0 has a certain property, and ii) whenever a number has that property, so does that number's successor, then: every number has the property.

But how to state the principle of induction? You do it differently depending on whether your language is first- or second-order:

$$\forall F \Big(\big(F0 \wedge \forall x (Fx \to Fx') \big) \to \forall x Fx \Big)$$

(second order induction *principle*)

$$\Big(A(0) \wedge \forall x \big(A(x) \to A(x') \big) \Big) \to \forall x A(x)$$ (first order induction *schema*)

Thus in second-order logic, you just add one further axiom: the second-order induction principle is a single formula, that actually fits the gloss above—it

begins with a quantifier $\forall F$—"for every property…". But in first-order logic, you can't say "for every property". Rather, what you do is state an induction axiom *schema* and say that every instance of this schema is an axiom. In this schema, A is some formula with some variable v free; $A(\mathsf{0})$ is the result of changing free vs in A to $\mathsf{0}$s, $A(x)$ is the result of changing free vs in A to xs, etc.[12]

So in first-order logic, if we want to state principles of induction, we can't do it with one sentence; we need infinitely many. However, even those infinitely many sentences don't really capture the full principle of induction. What they in effect say, taken together, is that induction holds for every *statable* property—that is, every property of natural numbers corresponding to some formula A in the first-order language of arithmetic. But there are many more properties of natural numbers than that (there are only as many formulas as there are natural numbers, but there are as many properties of natural numbers as there are real numbers). So there is a sense in which we can't really state the principle of induction in a first-order language.

(This fact is closely connected to the existence of nonstandard models. It is because we can include nothing stronger than the instances of the induction schema that we get nonstandard models of first-order arithmetic. Including the second-order induction principle rules them out.)

As with many of these differences between first- and second-order logic, there are philosophical questions about the significance of this distinction involving schemas. It isn't as if one avoids the need for schemas by adopting second-order logic. Even though second-order axiomatizations of arithmetic and set theory don't require schemas, one still needs schemas in the axioms of second-order logic itself, such as the comprehension schema. These latter logical axioms are essential to the use of the second-order theories of arithmetic and set theory. For instance, one conclusion we should be able to draw from any induction principle is that "if $\mathsf{0}$ is either even or odd, and if whenever n is even or odd, $n+1$ is also either even or odd, then every number is either even or odd". Now, the second-order principle of induction says that for any property, if $\mathsf{0}$ has it and if $n+1$ has it whenever n has it, then every number has that property. But in order

[12] In the terminology I was using earlier, the schema can be written this way:

$$\left(A_{v\mapsto \mathsf{0}} \wedge \forall x(A_{v\mapsto x} \rightarrow A_{v\mapsto x'})\right) \rightarrow \forall x A_{v\mapsto x}$$

to apply this principle to get the desired result, we need to know that there is a property of being either even or odd; that's what the comprehension schema tells us. So one must tread carefully with arguments that second-order logic's ability to avoid schemas in certain axiomatizations constitutes an advantageous sort of expressive power.

4. Paradoxes

4.1 Abstract mathematics and set-theoretic foundations

In the nineteenth century, mathematics underwent a dramatic transformation, moving toward an "abstract" approach. Traditionally, mathematics had been associated with relatively "concrete" matters: numbers were for counting or measuring; geometry was about physical points and lines; etc. But in the nineteenth century, mathematicians moved toward a conception of mathematics in which one studies the properties of arbitrary mathematical *structures*, regardless of their association with anything concrete.

In very abstract mathematics, one must be wary of relying on intuitions. In order to investigate what is true in an arbitrary structure, one must avoid smuggling in assumptions that seem intuitively correct but are not made explicit in the assumptions about the structure. This requires sophistication about *logic*, since what is needed is the ability to draw only the logical consequences from the assumptions laid down about the structure. Not coincidentally, it was in the late nineteenth and early twentieth century when modern logic was developed.

Modern logic was also key to a second development around the same time: a surge of interest in the foundations of mathematics. (This surge naturally accompanied the increase in abstraction, for it is natural to wonder what we ultimately doing, when we are studying arbitrary "structures".)

A second key tool in the foundations of mathematics was set theory (which was also discovered around the same time). An arbitrary "structure" was understood to be a certain sort of *set*. Thus on the set-theoretic point of view, what we are ultimately doing, when investigating mathematical structures, is investigating the properties of various kinds of sets.

What is a set? Well, it's something that contains other things; those other things are its *members*. We name a set by enclosing a list of its members within set braces, { and }. Thus the set containing Ted Sider and Barack Obama would

be named this way:

$$\{\text{Ted Sider, Barack Obama}\}$$

But sets can also have infinitely many members, and we may not be able to list the members. In such a case, in order to name the set we might give the conditions under which something is in that set. If E is the set of all and only the even natural numbers, we might write this:

$$x \in E \text{ if and only if } x \text{ is an even natural number}$$

Alternatively, we might write:

$$E = \{x | x \text{ is an even natural number}\}$$

(In general, "$\{x|\phi(x)\}$" means: "the set whose members are exactly those things, x, such that $\phi(x)$".)

In addition to containing things like people, sets can also contain other sets. For instance, the set $\{\{\text{Ted Sider, Barack Obama}\}\}$ has just one member. (Note the two set braces on each side.) That one member, {Ted Sider, Barack Obama}, is itself a set, the set whose members are Ted Sider and Barack Obama.

There is also such a thing as the *null set*, the set with no members at all: $\varnothing$.

Sets have members, but not in any particular order. Thus

$$\{\text{Ted Sider, Barack Obama}\} = \{\text{Barack Obama, Ted Sider}\}$$

But sometimes it is helpful to talk about set-like things that do have an order. These are called *ordered sets*. We write them using angle-braces ⟨ and ⟩ instead of the set-braces { and } for unordered sets. Since the order of the members of an ordered set is significant, reordering the members results in a different ordered set:

$$\langle\text{Ted Sider, Barack Obama}\rangle \neq \langle\text{Barack Obama, Ted Sider}\rangle$$

How could all of mathematics be regarded as just being about sets? The reason is that we can *construct* any mathematical objects as sets. For example, the natural numbers. We can regard the number 0 as just being the null set, the number 1 as just being the set containing the null set, the number 2 as just being the set containing those previous two sets, and so on:

$$0 \Rightarrow \varnothing \qquad 1 \Rightarrow \{\varnothing\} \qquad 2 \Rightarrow \{\varnothing, \{\varnothing\}\} \qquad \ldots$$

We can regard a rational number as just being the ordered pair of its numerator and denominator:[13]

$$\frac{1}{2} \Rightarrow \langle 1, 2 \rangle$$

Ordered pairs can in turn be defined as sets:

$$\langle x, y \rangle \Rightarrow \big\{\{x\}, \{x, y\}\big\}$$

We can regard real numbers as just being certain sets of rational numbers (e.g., "Dedekind cuts"). We can regard complex numbers as being ordered pairs of real numbers:

$$a + b\,i \Rightarrow \langle a, b \rangle$$

And so on.[14]

Here is one more reason for the importance of sets. The development of calculus in the 17^{th} century was a major step forward for mathematics. But as developed by Newton and Leibniz, the foundations of calculus were profoundly unclear (cf. Berkeley's famous critique). Gradually the foundations of calculus became clearer (as concepts like limits and epsilon-delta definitions were developed). As this happened, it became clearer that *functions* were an important part of the story. (What a derivative is, is a derivative of a function.) At first, functions were just regarded as being formulas in languages, but later it became clear that they must rather be regarded as "arbitrary mappings". But what are those? With set theory we have an answer: a function is just a set of ordered pairs, in which no two ordered pairs in the set share the same first member.

4.2 Russell's paradox

So sets increasingly became important in the foundations of mathematics. At first, mathematicians weren't very clear about what they were. Some didn't clearly distinguish sets from formulas picking them out. Many thought of sets as being completely unproblematic, or (relatedly) as just being part of logic.

But gradually it became clear that set theory itself is a substantial part of mathematics.

[13] Really it's better to regard rational numbers as equivalence classes of such ordered pairs.

[14] See a textbook on set theory, e.g., Enderton (1977), for a discussion of these constructions.

There were many reasons for this, including the discovery by Cantor of different sizes of infinity (which we'll discuss in a bit), and the discovery of the need for the axiom of choice. But most worrisome was the discovery of certain paradoxes. The kinds of assumptions mathematicians had been making about sets, it turned out, were contradictory!

The simplest of these paradoxes (though not the first to be discovered) was Bertrand Russell's. The following assumption had been made (sometimes implicitly):

Naïve comprehension for any "condition", there exists a corresponding set—a set of all and only those things that satisfy the condition

We use this principle implicitly when we speak of, for example, "the set of all even natural numbers": we form the condition "even natural number" and infer from Naïve comprehension that there exists a set of all and only those things that satisfy that condition.

But Russell then said: well, what about the condition "is a set that is not a member of itself"? Naïve comprehension implies that there exists a set R (the "Russell set") containing all and only the sets that aren't members of themselves. Thus:

i) Any set that is *not* a member of itself *is* a member of R

ii) Any set that *is* a member of itself is *not* a member of R

But now, Russell asked, is R a member of itself?

The claim that R *isn't* a member of itself leads to a contradiction: if R isn't a member of itself, then by i), R would be a member of R, and so it would be a member of itself after all—contradiction.

But the claim that R *is* a member of itself also leads to a contradiction: if R is a member of itself, then by ii) it wouldn't be a member of R, and so wouldn't be a member of itself—contradiction.

So each possibility is ruled out: that R is a member of itself and that R is not a member of itself. But one of those possibilities has to be correct—either R is a member of itself or it isn't. Thus the principle of Naïve comprehension has let to a contradiction.

Russell's argument can be put more simply. Naïve comprehension implies that

there exists a set, R, such that:

$$\text{for all } z, z \in R \text{ iff } z \notin z$$

But this implies, instantiating the variable z to R:

$$R \in R \text{ iff } R \notin R$$

which is a contradiction.

So the principle of Naïve comprehension isn't true. But that principle seems to be at the core of the very idea of a set. The very idea of a set, the main idea in the foundations of mathematics, seems to be contradictory!

4.3 Axiomatic set theory and ZF

In the decades after the paradoxes were discovered, mathematicians gradually found ways to develop a consistent approach to set theory, which avoids Russell's and other paradoxes. The approach that is now standard in mathematics is called "Zermelo-Frankel", or "ZF" set theory.

The ZF method for avoiding Russell's paradox is to reject the principle of Naïve comprehension. However, we can't just stop talking about sets of things satisfying various conditions we specify—the ability to do that is what makes sets so useful. So in place of Naïve comprehension, ZF includes new principles telling us that various sets exist. These new principles will be unlike Naïve comprehension in being consistent, but like Naïve comprehension in implying the existence of all the sorts of sets we need in the foundations of mathematics.

The main principles of ZF are these:

Extensionality Sets with the same members are identical

Null set There exists a set $\varnothing$ containing no members

Pairing For any sets a and b, there exists a set $\{a, b\}$ containing just a and b

Unions For any sets a and b, there exists a set $a \cup b$ containing all and only those things that are members of a or members of b (or both)

Infinity There exists a set, A, that i) contains the null set, and i) is such that for any $a \in A$, $a \cup \{a\}$ is also a member of A. (Any such set A must be infinite, since it contains all of these sets: $\varnothing, \{\varnothing\}, \{\{\varnothing\}, \varnothing\}, \ldots$.)

Power set For any set, A, there exists a set containing all and only the subsets of A (this is called A's "power set")

Separation Suppose some set x exists, and let $\mathscr{C}$ be any condition. Then there exists a set y consisting of all and only the members of x that satisfy $\mathscr{C}$.

Here is the rough idea of how they avoid Russell's paradox. Instead of a naïve comprehension principle telling us that certain sets exist, we have a two-step process telling us that certain sets exist. First, we have "expansion" principles which assert the existence of certain kinds of sets: Null set, Pairing, Unions, Infinity, and Power set.[15] Second, we have the principle of Separation, which lets us "contract" the sets that we obtain from the expansion principles.

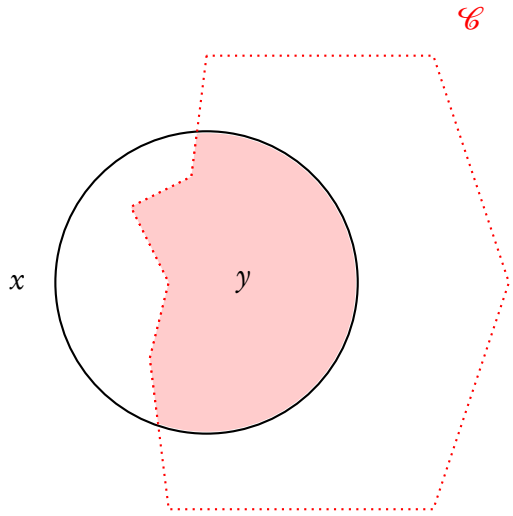
Separation is like the principle of Naïve comprehension, in that it assures us that there exist sets corresponding to various conditions; but it does this in a way that doesn't lead to contradiction. It says, roughly:

Separation Suppose some set x exists, and let $\mathscr{C}$ be any condition. Then there exists a set y consisting of all and only the members of x that satisfy $\mathscr{C}$.

So the principle of separation doesn't quite say that there exists a set corre-

[15] Actually there is also a stronger schema, called "Replacement".

sponding to each condition, since to apply it, we need to already know that a certain set x exists (say, by proving its existence using the expansion axioms). We can then use any chosen condition to pick out a subset of the given set x:



Here is why Separation doesn't lead to Russell's paradox. Suppose you start with a set, x. You can then use the principle of separation, choose the condition "is not a member of itself", and conclude that there exists a *subset* of x, call it y, that contains all and only the non-self-membered members of x:

$$\text{For all } z : z \in y \text{ iff } z \in x \text{ and } z \notin z$$

But this doesn't lead to a contradiction. It does imply this:

$$y \in y \text{ iff } y \in x \text{ and } y \notin y$$

But now we can consistently suppose that $y \notin y$. There is no need to suppose[16] that $y \in x$, so we can't use the right-to-left-hand direction of this biconditional to infer $y \in y$.

Suppose there existed a *universal set*—a set, U, that contains *every* set. We could then use the principle of separation to pick out the subset, R, of U containing all and only the nonself-membered members of U:

$$\text{For all } z : z \in R \text{ iff } z \in U \text{ and } z \notin z$$

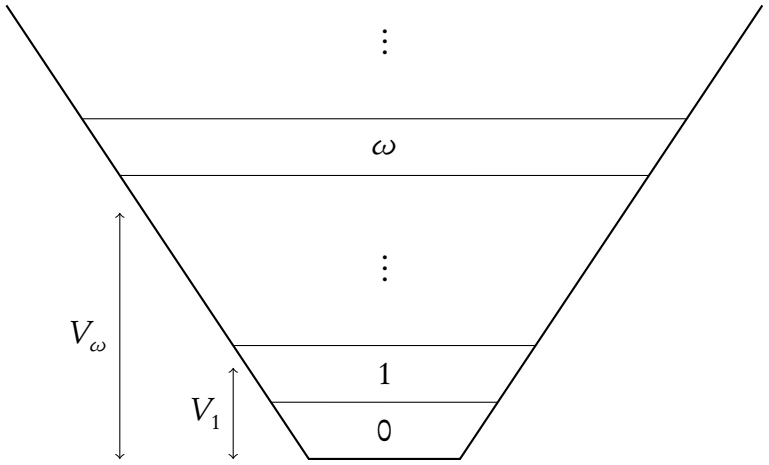
[16] In fact, other principles of ZF imply that $y \notin x$. The axiom of regularity implies that no set is a member of itself. Thus this principle implies that $y = x$ (given y's definition); and then, applying this principle again, we have that $y \notin x$.

But since *every* z is a member of U, this implies:

$$\text{For all } z : z \in R \text{ iff } z \notin z$$

which is Russell's contradiction.

So what we've learned is that in ZF set theory, there does not exist a universal set. What, then, is the picture of sets according to ZF? It is that of an "iterative hierarchy":



V_0 is the set of urelements

You first start with a set of "urelements"—a set of things that don't have members. (In "pure" ZF, you don't have urelements, and so the bottom of the diagram is "pointy"; in applied mathematics, nonmathematical objects will be the urelements.) Then you form a new level of sets, V_1, which adds to V_0 all the sets of urelements. Then you form a new level, V_2, which adds all sets you can form from V_1. And (in a certain sense) you keep going infinitely.[17]

One final thing about ZF. My statement above of the principle of separation wasn't quite right. An axiom of set theory must only talk about things like sets; it can't talk about things like "conditions". In fact, you state Separation differently in second- and first-order logic:

[17] See Boolos (1971).

$$\forall x \exists y \forall z \left(z \in y \leftrightarrow (z \in x \wedge A)\right) \quad \text{(first-order separation } \textit{schema}\text{)}$$
$$\forall x \forall X \exists y \forall z \left(z \in y \leftrightarrow (z \in x \wedge Xz)\right)$$
(second-order separation *principle*)

Thus in *first-order* ZF—that is, ZF set theory stated using first-order logic—there is no single axiom of separation. Rather, there is an axiom schema. ("A" is a schematic variable; whenever you replace 'A' with a formula in the language of set theory, the result is an axiom.) But in *second-order* set theory, there is a single axiom of separation—instead of the schematic variable A, there is a universally quantified second-order variable F.

Thus the situation is parallel to the situation with the principle of induction. In first-order set theory, we need infinitely many separation axioms; but even then, as a group they say, in effect, merely that a set has subsets corresponding to properties we can formulate in our language. This is much weaker than the second-order separation axiom, which says that the set has subsets corresponding to every property, not just those properties we can formulate. (As a consequence of this weakness, first-order set theory has all sorts of unwanted models.)

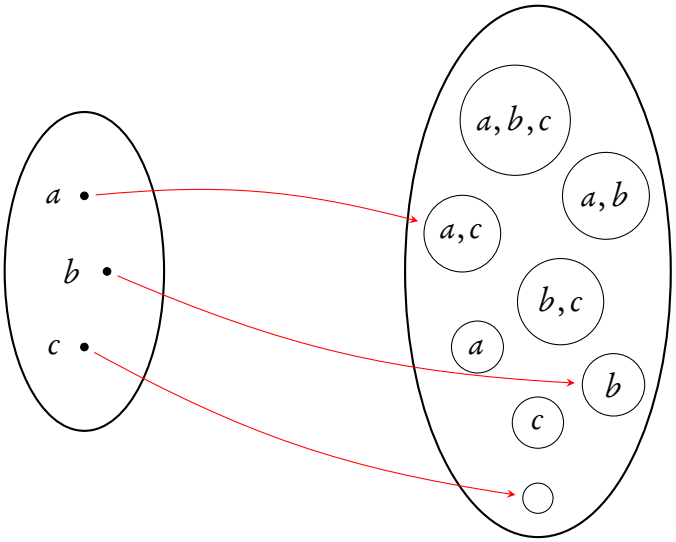
4.4 Other paradoxes, other solutions

Philosophers are generally aware that sets are entities that are "susceptible to paradoxes", that one needs to be cautious when reasoning about them, that natural-seeming principles about sets (such as the principle of naïve comprehension) need to be restricted in some way, and so on. But many other kinds of entities are just as susceptible to paradoxes, and this isn't always appreciated.

For example, propositions and properties face paradoxes similar to those confronting sets. Consider, for example, the property of *being a non-self-exemplifying property*: it seems that it exemplifies itself if and only if it does not exemplify itself—a contradiction. The problem even arises for predicates: the predicate 'heterological', meaning *is a predicate that does not apply to itself*, seems to apply to itself if and only if it does not apply to itself—again a contradiction.

Here is another paradox confronting properties and propositions, which demon-

strates the need for caution in reasoning about such entities. First a bit of background. The mathematician Georg Cantor famously showed that the power set of a set is always larger than that set, in the sense that if you take each member of the set and pair it with a unique member of the powerset, some members of the powerset will be left over, not paired with any member of the set. (See the diagram. In the diagram, A is a finite set; but Cantor's argument applies to all sets, even infinite ones.)



A Power set of A

Here is the argument. Let f be any function that maps every member of a set, A, to some subset of A. We'll show that some subset of A isn't in the range of this function—the function doesn't map any member of A to that subset. (In the diagram, this means showing that there is a circle in the right-hand-side of the diagram to which no red arrow is pointing.)

Let's begin by noting that some members of A have the following feature: they are members of the sets to which they are mapped by f. For example, in the diagram, a and b have the feature, since a is mapped to $\{a, c\}$ and b is mapped to $\{b\}$. But c does not have the feature, since it is mapped to the empty set.

Let's form the set, D, of members of A that do *not* have the feature. That is, let:

$$\text{for any } x,\ x \in D \text{ iff: } x \in A \text{ and } x \notin f(x) \qquad (*)$$

So in the diagram, $D = \{c\}$.

Now, suppose for reductio that every member of the powerset of A is in the range of f. That is, suppose that for every subset, X, of A, some member of A is mapped to X by f. It follows that some member of A is mapped to D by f. Let's call that member of A "d". Thus $f(d) = D$.

But there can be no such d. (This is true in the diagram: no red arrow is pointing to $\{c\}$; but we will give a general argument that doesn't rely on the diagram.) By (*),

$$d \in D \text{ iff: } d \in A \text{ and } d \notin f(d)$$

But $f(d) = D$, so:

$$d \in D \text{ iff: } d \in A \text{ and } d \notin D$$

But $d \in A$, so:

$$d \in D \text{ iff } d \notin D$$

which is a contradiction. (Note the similarity to Russell's argument. Russell thought of his argument by reflecting on Cantor's.)

Back, now, to propositions. Consider any set, S of propositions. It would be natural to assume that there exists a unique proposition that *some god is currently entertaining all and only the propositions that are members of S*. But that can't be: it would mean that there are at least as many propositions as there are sets of propositions. But how could there fail to be such a proposition? Or how could it fail to be unique? The argument seems to show that our ordinary ways of thinking about propositions lead to contradictions.[18]

Back to set theory and Russell's paradox. We discussed the ZF solution, but it isn't the only one. One might, for instance, give up classical logic, and say that $R \in R \leftrightarrow R \notin R$ isn't a contradiction. Others have developed other set theories, in which there is a universal set (e.g., Quine's "New foundations" system). There is also the "indefinite extensibility" tradition. On this view, there is something wrong with the idea that one can quantify over absolutely everything. One can accept the principle of naïve comprehension, and so accept a set of all and only the non-self-members. That is, given the current range of our quantifiers, we can then introduce a set that includes all and only the members in that range which are not self-members. However, in doing so we

[18] See Kaplan (1994). We'll also discuss the Russell-Myhill paradox in more detail later, which is in this vicinity.

have expanded the domain of quantification, since that set we just introduced wasn't in the original domain.[19]

And finally, and more important for our purposes, there are attempts to solve paradoxes by broadly "syntactic" means: by adopting languages in which attempts to raise the paradox are not grammatical. As an example, take second-order logic. In second-order logic, we can think of the predicate variables as standing for properties, but we can't even grammatically raise the question of whether one of those properties applies to itself. The reason is that the way you attribute properties in second-order logic is by predicating: e.g., you say that object a has some property by saying $\exists FFa$. So the way to say that some property has itself would seem to be this: $\exists FFF$. But that's not grammatical: the syntax of second-order logic says that if F is a one-place predicate, it needs to combine with a *term* to form a sentence; it can't combine with another predicate. So you can't even grammatically formulate anything like Russell's paradox here.

We'll look into this more when we develop a stronger higher-order logic.

5. Higher-order logic and λ-abstraction

In the following sections we'll develop a very expressively powerful language—the language in which higher-order metaphysics takes place.

5.1 Third-order logic and beyond

Let's consider languages that are even more expressively powerful than that of second-order logic.

In second-order logic, we have expressions which attach to terms in order to form sentences; these are predicates (whether predicate variables or predicate constants). However, we don't have any predicates of predicates: expressions that attach to predicates in order to form sentences.

We could add them—we could modify the definition of a formula as follows:

> In addition to the ordinary kind of predicates (both constant and variable), there are some "predicates of predicates": $\boldsymbol{F}, \boldsymbol{G}, \ldots$. For any predicate of predicates, $\boldsymbol{F}$, and any one-place ordinary predicate G, "$\boldsymbol{F}G$" is a formula.

[19] See Fine (2007).

And we could modify the semantics in the obvious way:

> In any interpretation, the denotation of a predicate of predicates is a set of sets of members of the domain.
>
> The formula "$\boldsymbol{F}G$" is true in an interpretation if and only if the denotation of the ordinary predicate G is a member of the denotation of the predicate of predicates $\boldsymbol{F}$.

This might be useful, for example, in symbolizing a sentence like:

> Sally and John have exactly the same virtues
>
> $\forall X(\boldsymbol{V}X \rightarrow (Xs \leftrightarrow Xj))$

(Although notice that the syntax of the English sentence 'Sally and John have exactly the same virtues' seems more like a first-order sentence quantifying over properties and employing a predicate 'has' for instantiation.)

Once we have predicates of predicates, it is natural to introduce quantifiers binding variables in that syntactic position, so that in addition to saying "Sally and John have exactly the same virtues" as above, we can also express a sort of existential generalization of that sentence:

> There is some type of property such that Sally and John have exactly the same properties of that type
>
> $\exists \boldsymbol{Y} \forall X(\boldsymbol{Y}X \rightarrow (Xs \leftrightarrow Xj))$

This is third-order logic.

And we could keep iterating, introducing predicates that can apply to predicates of predicates, and variables for such predicates, resulting in fourth-order logic, even higher-order predicates, and so on. Also we could allow these higher-order predicates to be multi-place. But there is a further sort of syntactic generalization that we will explore.

5.2 Higher-order logic and types

So far we have discussed languages in which one can quantify into predicate position, the position of predicates of predicates, and so on. I want next to discuss an even more expressive sort of language, in which one can quantify

into many more grammatical positions. In a sense, quantification can be into *any* grammatical position, in a broad sense of grammatical position.

5.2.1 More about syntax

To introduce this language, let's think a little more about grammar/syntax in general.

Syntax is about well-formedness—about what kinds of expressions "make sense". Syntactic rules tell you what kinds of expressions combine with what other kinds of expressions to form still other kinds of expressions. We have seen a few such rules, such as:

> If you take a one-place predicate, F, and attach it to a term, t, the result Ft is a formula
>
> If you take the expression $\sim$, and attach it to a formula, A, the result $\sim A$ is a formula

There is a common pattern here:

> If you take an expression of category X, and attach it to an expression of category Y, the result is an expression of category Z

For the first rule, X was *one-place predicate*, Y was *term*, and Z was *formula*; for the second rule, X was *the negation sign*, Y was *formula*, and Z was *formula*.

This makes it natural to ask whether it would make sense to introduce new syntactic categories other than those we have been examining so far. For example, in each of the two rules above, the "output" of the rule (the expression of category Z) was a formula. But could we have a rule letting us form complex expressions, in which the output (Z) was, say, the category *one-place predicate*?

We can. We can, for instance, introduce a syntactic category of *predicate functors*, which combine with one-place predicates to form new one-place predicates. The syntactic rule governing predicate functors would be this:

> If you take a predicate functor $\boldsymbol{q}$, and attach it to a one-place predicate, F, the result $\boldsymbol{q}F$ is a one-place predicate

Here X is *predicate functor*, Y is *one-place predicate*, and Z is *one-place predicate*. Natural language has something like predicate functors, namely adverbs:

'quickly' can combine with the predicate 'runs' to form the predicate 'runs quickly'.

Once we have seen syntactic categories and rules in this light, a possibility for generalizing them suggests itself. For *any* syntactic categories Y and Z, we ought to be able to introduce a new category, X, with this feature: expressions of category X combine with expressions of category Y to form expressions of category Z. Since this process can be iterated, there will be infinitely many syntactic categories. Moreover, in all categories, we will allow quantified variables of that category. The result is a certain sort of "higher-order logic", based on the "theory of types".[20]

5.2.2 Types

If we are going to develop a language with infinitely many syntactic categories, we need a way of talking about syntactic categories as entities, so that we can make generalizations about all syntactic categories ("for any syntactic categories, a and b, …".) Syntactic categories, thought of as entities, are called *types*.

Let's first get a handle on what these types are supposed to do for us. Each one is an entity representing a syntactic category. Thus we will speak of expressions in our language of higher-order logic as being *of* types: an expression is *of* type a if and only if that expression falls under the grammatical category that the type a represents. For instance, as we'll see in the next paragraph, the type that represents the grammatical category of *formulas* is: t. Thus expressions that are formulas, such as $Fa \wedge Gb$ or $\exists x Fx$, are expressions *of* type t. (What *is* the entity t? It doesn't matter; think of it as being the letter 't' if you like. What matters about the types is what they represent, not what they are.)

OK, let's define the infinite class of types:

[20] The theory of types we'll be discussing derives from Alonzo Church (1940), who was improving the theory of types in Russell and Whitehead's *Principia Mathematica*. There are a number of different kinds of systems that go by the name of "type theory"; this is just one. For a quick peek at some of the alternatives, see section 5.4.

There are two undefined types, e and t

(e is the type of singular terms; t is the type of formulas)

For any types, $a_1, \ldots, a_n$ and b, there is another type $\langle a_1, \ldots, a_n, b \rangle$

($\langle a_1, \ldots, a_n, b \rangle$ is the type of expressions that combine with expressions of types $a_1, \ldots, a_n$ to form an expression of type b)

The undefined types e and t are so-named because they are the types of expressions that name *e*ntities and can be *t*rue or false, respectively. Thus individual variables ($x, y, \ldots$) and names ($a, b, \ldots$) are of type e; and formulas ($Fx, \forall x(Fx \rightarrow Gx), \ldots$) are of type t. As for complex types $\langle a_1, \ldots, a_n, b \rangle$, an expression E that is of this type is the sort of expression that can combine with n expressions, of types $a_1, \ldots, a_n$, respectively, to form an expression of type b. Thus such an expression E is an "n-place" expression: in order to make a syntactically well-formed result, E needs to combine with n other expressions, of the appropriate types. The "appropriate types" are $a_1, \ldots, a_n$—the types of the expressions with which E can combine. And b is the type of the resulting expression—the type of the expression that results when E is combined with those n expressions of appropriate types. One might think of E as a kind of function, mapping expressions to expressions: $a_1, \ldots, a_n$ are the types of its "inputs", and b is the type of its "output".

In the simplest case where $n = 1$, the type $\langle a, b \rangle$ is that of an expression which can combine with an expression of type a to make an expression of type b. In the terminology of the previous section, X is $\langle a, b \rangle$, Y is a, and Z is b.

Let's look at some examples.

Example 1: the type $\langle e, t \rangle$. Expressions of this type combine with something of type e to make an expression of type t. That is: they combine with terms to make formulas. Thus they are one-place predicates! (That is, they are one-place predicates of individuals.)

Example 2: the type $\langle e, e, t \rangle$. Expressions of this type combine with *two* expressions of type e to make an expression of type t. That is, they combine with pairs of terms to make formulas. Thus they are two-place predicates (of individuals). Similarly, a three place-predicate is of type $\langle e, e, e, t \rangle$, and so on.

Example 3: the type $\langle t, t\rangle$. Expressions of this type combine with a formula to make a formula. An example is the symbol for negation, $\sim$. If you combine it with a formula, A, the result $\sim A$ is another formula. (The operators $\Box$ and $\Diamond$ from modal logic are further examples.) Expressions of type $\langle t, t\rangle$ are thus one-place sentence operators.

Example 4: the type $\langle t, t, t\rangle$. Expressions of this type combine with two formulas to make a formula. That is, they are two-place sentence operators. Examples include the symbol for conjunction, $\wedge$, the symbol for disjunction, $\vee$, and the conditional symbol $\rightarrow$. (Another example is the counterfactual conditional symbol $\Box\!\!\rightarrow$ in counterfactual logic.)

Example 5: the type $\langle\langle e, t\rangle, t\rangle$. Expressions of this type combine with an expression of type $\langle e, t\rangle$ to make an expression of type t. That is, they combine with predicates to make formulas. Thus they are what we called higher-order predicates in section 5.1.[21] For example, we might represent 'is patient' as an ordinary predicate P, and 'is a virtue' as a higher-order predicate V, and then represent (with a little violation to English syntax) 'patience is a virtue' as "VP".

[21]There is a confusing terminological issue I'd like to mention. Recall the "higher-order" predicates of section 5.1: predicates of predicates, predicates of predicates of predicates, and so on. What numerical orders should be assigned to such predicates? Different authors use different terminology here, and there is no happy choice. On one hand, it's natural to call predicates of predicates *second-order* predicates, since they're "one jump up" from ordinary predicates. On the other hand, if we called them second-order predicates, we would expect them to have the same syntactic type as the variables that we call "second-order variables"; but what are usually called second-order variables are variables of type $\langle e, t\rangle$—the "predicate variables" of section 3.1. (After all, the latter are the variables that are distinctive of second-order logic.) Relatedly, if predicate constants of type $\langle\langle e, t\rangle, t\rangle$ are "second-order", then presumably predicate constants of individuals (such as P for 'is patient' or 'F' for 'is a fox') should be called "first-order". But that means we're calling an expression of type $\langle e, t\rangle$ "first order" if it's a constant and "second-order" if it's a variable, which is unfortunate.

The problem is that there are two distinct sorts of "jumps" that we feel inclined to call an increase in "order". On one hand, there is the move from a language in which you can't quantify into (ordinary) predicate position to a language in which you can. The first sort of language is standardly called "first-order logic" and the second is standardly called "second-order logic". On the other hand, there is the jump from a language in which you only have predicates of individuals—that is, expresions of syntactic type $\langle e, t\rangle$ (whether constant or variable)—to a language in which you have predicates of predicates—that is, expressions of syntactic type $\langle\langle e, t\rangle, t\rangle$ (whether constant or variable). The latter also feels like a jump in order, which explains the impulse to call predicates of predicates "second-order". But it's a different sort of jump than the first jump; and the two aren't "in sync".

Example 6: the type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$. Expressions of this type combine with expressions of type $\langle e,t\rangle$ and make expressions of type $\langle e,t\rangle$. That is, they combine with one-place predicates to form one-place predicates. Adverbs! (Or: predicate functors.) The adverb 'quickly' combines with the one-place predicate 'runs' to form the one-place predicate 'runs quickly'.

All this, then, inspires the following syntax for the language of our higher-order logic.

1. For each type, there are infinitely many variables which are expressions of that type, and there may also be zero or more constants which are expressions of that type. (Any chosen set of constants is called a "signature", and determines a language.)
2. $\sim$ is a constant expression of type $\langle t,t\rangle$; $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ are constant expressions of type $\langle t,t,t\rangle$
3. If $E_1,\ldots,E_n$ are expressions of types $a_1,\ldots,a_n$, and E is an expression of type $\langle a_1,\ldots,a_n,b\rangle$, then $E(E_1,\ldots,E_n)$ is an expression of type b
4. If v is a variable of type a and E is an expression of type t then $\forall vE$ and $\exists vE$ are expressions of type t

Notice, by the way, a difference between this syntax and, e.g., the syntax for first-order logic in section 2.1. In that earlier syntax, we didn't assign $\sim$ any syntactic category (since the only syntactic categories mentioned there were *term* and *formula*, and $\sim$ isn't either of those). Rather, we simply gave a rule that specifies the syntactic role of $\sim$—a rule saying how $\sim$ combines with an expression of an appropriate syntactic category to yield another expression of a certain syntactic category. That rule was the following: "if A is a formula then so is $\sim A$". But here, we don't have any such rule for $\sim$, or for any of the other propositional connectives. Why not? Because we have instead explicitly assigned types to them. $\sim$, for example, is said in clause 2 to be of type $\langle t,t\rangle$: it combines with one expression of type t to form an expression of type t. As a result, clause 3 covers the syntax of $\sim$ and the other propositional connectives (along with covering the syntax of variables and nonlogical constants). For instance, since $\sim$ is of type $\langle t,t\rangle$, it follows from clause 3 that for any expression

E of type t (i.e., any formula), $\sim(E)$ is an expression of type t.[22]

5.2.3 Meaning: Frege and functions

What do all these expressions of different types *mean*?

For the types that have corresponding expressions in natural language, this is comparatively easy. For example, since we are already familiar with predicates/verb phrases (like 'runs') and adverbs/predicate functors (like 'quickly'), we already have a handle on what sorts of meanings are possessed by expressions of types $\langle e, t\rangle$ and $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$. But what about expressions of much more complex types, for which there are no natural-language counterparts?

In such cases, it's natural to think of the meaning of an expression in terms of what it *does*—in terms of how the meaning of that expression interacts with the meanings of other expressions with which it is combined, to generate the meanings of more complex expressions. And there is a systematic way of thinking about meaning in this way: Frege's (1997*b*).

Let's begin by thinking about meaning informally. (Later on we'll consider a Frege-inspired formal semantics for higher-order logic.) For Frege, in one sense of meaning, the meaning of a proper name (like 'Ted') is an individual (me), and the meaning of a sentence (like 'Ted is a philosopher') is a truth value, of which there are two: T (or 1, or "The True"); and F (or 0, or "The False").

Now, Frege (1952/1892) famously pointed out that expressions that mean the same, in this sense of meaning, can mean different things in another sense of meaning. 'Ted is a philosopher' and 'Barack Obama was born in Hawaii' are both true sentences, and so each denote the same thing, namely T (The True); nevertheless there is clearly some sense of meaning in which they have different meanings. Similarly, '$2+3$' and '$1+4$' both denote the number 5, but again, they have different meanings in some sense. Frege dealt with this by distinguishing one kind of meaning, called *Bedeutung*, or reference, or as I will say, denotation, from another sort of meaning called *Sinn*, or sense (which is roughly the rule by which the denotation is determined). Here I will consider only denotation, and ignore sense. So I can rephrase the claims about meaning from the previous paragraph: the *denotation* of a name is an individual, and the denotation of a sentence is a truth value.

[22] Note that we are still treating quantifiers in the old way. But they too can be assigned types once we have introduced λ-abstraction; see section 5.4.3.

(It's admittedly somewhat awkward to think of sentences as denoting their truth values, particularly if, when we get to quantification into sentence position, we think of $\forall P$ (variable of type t) as meaning "for all propositions…". But let's stick with Frege's original approach for now, for simplicity.)

So: names denote individuals and sentences denote truth values. What about predicates and connectives? For Frege they denote *functions*.

The notion of a function is one that we've mentioned a few times before. It's a familiar notion from mathematics. A function is a rule that yields an output if you give it appropriate inputs. For example, $f(x) = x^2 + 4$ is a function that yields the output 4 if you give it the input 0, and yields the output 8 if you give it the input 2. Another way of putting this: the function f "maps 0 to 4" and "maps 2 to 8". Now, f is a "one-place" function, since it requires one input in order to yield an output. Other functions have more than one place. For example, the addition function, which we might represent as $g(x, y) = x + y$, maps two numbers to a single output: it maps, e.g., $1, 2$ to 3, and maps $4, 7$ to 11. The examples so far have been functions of numbers, but functions can have objects of any sort as inputs and outputs. The biological mother of function, for example, is a function from persons to persons, which maps each person to his or her biological mother.[23]

According to Frege, predicates denote functions from *individuals to truth values*. For example, the one-place predicate 'runs', for Frege, denotes the function that maps any object to T if it runs and to F if it does not run. That is, 'runs' denotes this function:

$$r(x) = \begin{cases} T \text{ if } x \text{ runs} \\ F \text{ if } x \text{ does not run} \end{cases}$$

[23] In section 4.1 we mentioned the set-theoretic definition of a function, according to which a function is a set of ordered pairs. But in some contexts one might prefer some other conception of what functions are. i) According to the set-theoretic definition, any "two" functions that assign the same values to all arguments are in fact identical. For instance, the "add one" set-theoretic function on natural numbers is the very same as the "add two and then subtract one" set-theoretic function. One might prefer a more "fine-grained" conception of function in certain circumstances. ii) According to the set-theoretic definition, functions are sets. Thus, given ZF, there cannot exist functions defined on absolutely all entities (including all sets). Consider, for example, the function that Frege would call the denotation of the predicate '$\in$' of set-membership: the two-place function that maps any a and any set b to T if and only if $a \in b$. Given ZF there is no such set-theoretic function.

Similarly, the two-place predicate 'bites' denotes this function:

$$b(x,y) = \begin{cases} T \text{ if } x \text{ bites } y \\ F \text{ if } x \text{ does not bite } y \end{cases}$$

In general, n-place predicates for Frege denote n-place functions from entities to truth values.

As for connectives, they denote functions from *truth values to truth values*. For instance, the one-place connective 'not' denotes this function, n:

$$n(T) = F$$
$$n(F) = T$$

This function "reverses" truth values, mapping truth to falsity and falsity to truth; it is the *negation* function. Similarly, according to Frege, the two-place connectives 'and' and 'or' denote the *conjunction* and *disjunction* functions c and d:

$$c(T,T) = T \qquad d(T,T) = T$$
$$c(T,F) = F \qquad d(T,F) = T$$
$$c(F,T) = F \qquad d(F,T) = T$$
$$c(F,F) = F \qquad d(F,F) = F$$

These should be familiar from introductory logic; Frege's idea is that the functions that truth tables depict are the meanings of the connectives.

5.2.4 Formal semantics for higher-order logic

In this section I'll give a partial sketch of a formal semantics for our language of higher-order logic inspired by Frege's functional approach to denotation.[24]

As in first-order logic, an interpretation will consist in part of a domain—a nonempty set. And as before, an interpretation assigns denotations to nonlogical expressions. In the case of names—constant expressions of type e, these denotations will be exactly what they were earlier: members of the domain. But

[24]The definition sketched here is not the only possible one. There are alternatives, for instance alternatives in the spirit of Henkin interpretations for second-order logic (section 3.5.1).

for expressions of other types, denotations will be different sorts of entities—albeit always entities that are "based" on the domain.

First let's give a general statement of the kind of entity denoted by expressions of any type in a given interpretation. For each type, a, we'll specify what kind of object counts as an "a denotation" (or: "denotation of type a")—the kind of thing that can be denoted by an expression of type a.[25] The definition first specifies the kinds of denotations for the undefined types, and then it gives a rule covering complex types:

Kinds of denotations, for a given domain D

e denotations are members of D

t denotations are truth values (i.e., T, F)

An $\langle a_1, \ldots, a_n, b \rangle$ denotation is an n-place function that maps an a_1 denotation, an a_2 denotation, …, and an a_n denotation, to a b denotation

Consider, for example, the type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$—the type of predicate functors. Given the above definition, an $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ denotation for a domain D is a function that maps $\langle e,t\rangle$ denotations to $\langle e,t\rangle$ denotations. And $\langle e,t\rangle$ denotations are themselves functions—functions from D to truth values. Thus an $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ denotation is a function that maps any function from D to truth values to another function that maps D to truth values.

Recall from the end of section 5.2.2 that in our syntax for higher order logic, the logical constants $\wedge$, $\vee$, etc. were treated as being expressions of certain types.

Now we can give a proper definition of an interpretation:

[25] In some contexts, types are thought of as being, in the first instance, types of semantic values, rather than as types of expressions, as I have been thinking of them.

Definition of interpretation for higher-order logic

An interpretation consists of a domain, D, and a specification, for each nonlogical expression of type a, of an a denotation for D.

Additionally, any interpretation assigns to the propositional connectives as denotations their Fregean truth-functions. (For example, the denotation of $\sim$ in any interpretation is the one-place truth function mapping T to F and F to T.)

So, for example, if G is some nonlogical predicate (type $\langle e, t \rangle$), then any interpretation must assign to G some particular function from its domain to truth values.

Next we give a general rule for computing the denotations of complex expressions based on the denotations of their parts:

Denotations of complex expressions (in a given interpretation)

If $E_1, \ldots, E_n$ are expressions of types $a_1, \ldots, a_n$, with denotations $d_1, \ldots, d_n$, and expression E has type $\langle a_1, \ldots, a_n, b \rangle$ and denotes a function f, then the denotation of the complex expression $E(E_1, \ldots, E_n)$ (which has type b) is: $f(d_1, \ldots, d_n)$

Thus denotations of complexes are derived by applying functions to arguments: the function that is the "outer" expressions's denotation is applied to the arguments that are the "inner" expressions' denotations. Indeed, the attraction of the Fregean functional approach to denotation is that it leads to such a simple rule.

To see all this in action, let's work through an example. Consider an interpretation in which the following formula symbolizes "Ted doesn't run quickly":

$$\sim \boldsymbol{q}(R)(c)$$

Thus c is a name symbolizing "Ted", R is a one-place predicate symbolizing "runs", and $\boldsymbol{q}$ is a predicate functor symbolizing "quickly". Thus the denotations of these expressions will be of the following sorts ('fx' abbreviates 'function'):

	type	kind of denotation	particular denotation
c	e	member of the domain	Ted
R	$\langle e,t\rangle$	fx from entities to truth values	the fx r that maps o to T iff o runs
$\boldsymbol{q}$	$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$	fx from $\langle e,t\rangle$ fxs to $\langle e,t\rangle$ fxs	the fx q that maps g to the "g-ing quickly" fx
$\sim$	$\langle t,t\rangle$	fx from TVs to TVs	the fx n that maps T to F and F to T

Now let's figure out what the denotation—i.e., truth value—of the entire sentence $\sim\boldsymbol{q}(R)(c)$ is. To make this more readable, let's write "$|E|$" for "the denotation of expression E". Thus, what we're trying to figure out is what $|\sim\boldsymbol{q}(R)(c)|$ is. We'll start by writing down the denotations of simple expressions, and then work up to the denotations of more complex expressions.

$$|R| = r \qquad \text{(see table)}$$
$$|\boldsymbol{q}| = q \qquad \text{(see table)}$$
$$|\boldsymbol{q}(R)| = q(r) \qquad \text{(rule for denotations of complexes)}$$
$$|c| = \text{Ted} \qquad \text{(see table)}$$
$$|\boldsymbol{q}(R)(c)| = q(r)(\text{Ted}) \qquad \text{(rule for denotations of complexes)}$$
$$|\sim| = n \qquad \text{(see table)}$$
$$|\sim\boldsymbol{q}(R)(c)| = n(q(r)(\text{Ted})) \qquad \text{(rule for denotations of complexes)}$$

Thus $\sim\boldsymbol{q}(R)(c)$ denotes $n(q(r)(\text{Ted}))$. But what is $n(q(r)(\text{Ted}))$? Well, $q(r)$ is the function that maps any member of the domain to T iff it runs quickly. (This is so because q is the function that maps a function from the domain to truth values to the corresponding "quickly function" from members of the domain to truth values.) In fact, I do run quickly. (Very quickly.) So $q(r)(\text{Ted}) = T$. And so, $n(q(r)(\text{Ted})) = F$. Our sentence denotes The False.

In a full presentation of the semantics, we would need to give a definition of the denotation for all expressions of all types, including expressions containing variables (see next section) and connectives. I won't do this here, since my goal has been to say just enough to give the intuitive feel for meaning in higher-order logic. However, it's important to note that there are philosophical questions about how formal, set-theoretic semantics of the type sketched here relates to intended meaning. Such questions are especially pressing for higher-order logic. For instance, under one common interpretation of the language of higher-order logic, it is common to assume that a comprehension principle holds, so that (for example) the sentence $\exists F\forall x(Fx \leftrightarrow x = x)$ is true. But then, if the quantifier $\forall x$ includes all sets in its range, it would seem that the

quantifier $\exists F$ cannot range over sets, since a set of all self-identical sets would be a set of all sets whatsoever, which does not exist given ZF set theory. So: take the semantics sketched in this section as being a sort of model (in the informal, philosophy-of-science sense) of one possible intended meaning of the language of higher-order logic—a model which may not be fully accurate but may nevertheless be useful for certain purposes.

5.2.5 Variables

Back to syntax. Consider the symbolization of "Ted is sitting and Ted is eating":

$$Sc \wedge Ec$$

In first-order logic we can generalize into subject position:

$$\exists x(Sx \wedge Ex)$$

And in second-order logic we can generalize into predicate position:

$$\exists X(Xc \wedge Ec)$$

So why not also allow generalization into the 'and' position:

$$\exists \bigcirc(Sc \bigcirc Ec)$$

to mean, roughly, that there is some relation between propositions that holds between the proposition that Ted is sitting and the proposition that Ted is eating? '$\bigcirc$' here would be a variable with the same type as '$\wedge$', namely: $\langle t, t, t\rangle$.

In higher-order logic, we do allow this. More generally, since we are allowing constants of any of the infinitely many types, we will also allow *variables* of each of those types, together with quantifiers binding them.

For example, we will have a variable $\triangle$ of the same type as $\sim$, namely $\langle t, t\rangle$, so that we can write:

$$\exists \triangle \, \triangle Sc$$

meaning, roughly, that there is some property of propositions had by the proposition that Ted is sitting. ('$\triangle$' and '$\bigcirc$' and other symbols I'm using here aren't standard; I'll talk later about how we write variables of different types.)

Similarly, suppose $\boldsymbol{q}$ is a constant predicate functor (type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$), representing the adverb 'quickly', so that we can symbolize 'Ted runs quickly' as:

$$\boldsymbol{q}(R)(c)$$

Then we can also write:

$$\exists \boldsymbol{x}\, \boldsymbol{x}(R)(c)$$

where $\boldsymbol{x}$ is a variable of type $\langle\langle e,t\rangle\langle e,t\rangle\rangle$—a "predicate functor variable".

We can even have variables of type t, i.e., the type of formulas. If P is such a variable, then "$\exists P\, P$" means roughly that there is some true proposition.

5.2.6 Typing symbols

We need some way of writing symbols, whether variables or constants, that indicates their types, and our ad hoc method for doing this so far (x, X, $\triangle$, $\boldsymbol{x}$, P) won't work in general. There are infinitely many types (for any types $a_1,\ldots,a_n,b$, there is a new type $\langle a_1,\ldots,a_n,b\rangle$), so we can't go on picking a different font or symbol set for each type—we'll run out.

Instead, in official contexts anyway, let's write all variables as "x", and all constants as "c", and indicate the type of the variable or constant by including that type as a superscript on the symbol. So for any type, a, x^a will be a variable of type a, and c^a will be a constant of type a. We can get as many variables and constants of any type as we like by using subscripts, e.g.: $x^a_1, x^a_2, \ldots$.

Unofficially, however, we'll continue to use the old ad-hoc methods, to improve readability. Thus we'll write:

official	unofficial	
x^e	$x, y, \ldots$	individual variables
c^e	$c, d, \ldots$	names
x^t	$P, Q \ldots$	sentence variables
$x^{\langle e,t\rangle}, x^{\langle e,e,t\rangle}$	$X, F, R \ldots$	predicate variables
$x^{\langle t,t,t\rangle}, x^{\langle t,t\rangle}$	$\triangle, \bigcirc$	sentence-operator variables
$c^{\langle\langle e,t\rangle,t\rangle}$	$\boldsymbol{F}, \boldsymbol{G}, \ldots$	higher-order predicate constants
$x^{\langle\langle e,t\rangle,t\rangle}$	$\boldsymbol{X}, \boldsymbol{Y}, \ldots$	higher-order predicate variables
$c^{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}$	$\boldsymbol{q}, \boldsymbol{r}, \ldots$	predicate-functor constants
$x^{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}$	$\boldsymbol{x}, \boldsymbol{y}, \ldots$	predicate-functor variables

5.3 λ-abstraction

5.3.1 Complex predicates

In standard logic, the symbol for conjunction, $\wedge$, is a sentential connective: the only place that it can grammatically occur is between two formulas: $A \wedge B$.

The natural-language word 'and' is more flexible. In addition to functioning as a sentential connective, as in 'Ted is sitting and Ted is eating' (and in addition to constructing plural terms such as 'Daisy and Luke'—recall section 2), it can also occur in *complex predicates*, as in the sentence 'Ted is sitting and eating'. Here 'is sitting and eating' is, grammatically, a predicate (it attaches to 'Ted' to form a sentence); but it is a complex predicate, since it is made up of two other predicates, 'sitting' and 'eating', which are combined with 'and'.[26]

So: natural languages (some, anyway) have complex predicates. Logical languages can be constructed which also have complex predicates. Under a common way of doing so, a complex predicate like 'is sitting and eating' is represented using a new symbol, λ ("lambda"), as follows:

$$\lambda x(Sx \wedge Ex)$$

(Or "$\lambda x.(Sx \wedge Ex)$", "$\lambda x[Sx \wedge Ex]$", or "$(\lambda x.Sx \wedge Ex)$"—different authors use different notation.) It can be read in any of the following ways:

> "is an x such that Sx and Ex"
>
> "is such that: it is sitting and it is eating"
>
> "is sitting and eating"

$\lambda x(Sx \wedge Ex)$ is a one-place complex predicate; but we can also allow complex predicates with more than one place. For example, if B is a two-place predicate meaning "bites", then the following is a two-place predicate meaning "bites or is bitten by":

$$\lambda xy(Bxy \vee Byx)$$

In general, the syntax of λ is:

[26] When I'm teaching introductory logic students how to symbolize natural language sentences like 'Ted is sitting and eating', I tell them that this sentence "means the same as" or "is short for" the sentence 'Ted is sitting and Ted is eating', and hence can be symbolized as $Sc \wedge Ec$. These claims are perhaps correct in some sense, but not if taken as claims about English syntax. In 'Ted is sitting and eating', 'is sitting and eating' is a syntactic unit.

Where $x_1, \ldots, x_n$ are any variables and A is any formula, $\lambda x_1 \ldots x_n A$ is an n-place predicate

$\lambda x_1 \ldots x_n A$ can be read as meaning "are $x_1, \ldots, x_n$ such that A".

Expressions formed with λ are often called "λ abstracts", and the process by which they are formed is called "λ abstraction".

How are λ abstracts used in sentences? Just like other predicates: you attach them to terms to form sentences. For example, if c is a name referring to me, then the following is a sentence symbolizing "Ted is such that he is sitting and eating":

$$\lambda x(Sx \wedge Ex)c$$

(I'll sometimes add parentheses to sentences involving λ-abstracts to improve readability; e.g.: $\lambda x(Sx \wedge Ex)(c)$.)

It's important to be clear about the syntax of λ expressions, so let me re-emphasize it: λxA is a *predicate*. Thus we should not think of '$\lambda x(Sx \wedge Ex)$' as symbolizing 'the property of sitting and eating'. It rather symbolizes 'is sitting and eating'.

What is the difference? 'The property of sitting and eating' is a *term*, whose function is to name something, whereas 'is sitting and eating' is a *predicate*, whose function is to describe something. In order to construct a sentence using the former, you need to combine it with a predicate. For instance, you can use the two-place predicate 'instantiates' to say:

Ted instantiates the property of sitting and eating

You *can't* combine it with a term to form a sentence:

\# Ted the property of sitting and eating

That's ungrammatical—terms can't combine with terms to form sentences. But you *can* combine the *predicate* 'is sitting and eating' with a term to form a sentence:

Ted is sitting and eating

What might a formal semantics for λ abstracts look like? Well, they will have meanings just like predicates. Let's continue with the Fregean approach from section 5.2.4, in which (first-order) predicates denote functions from the domain to truth values. Then $\lambda v A$, also being a predicate, also denotes a function from the domain to truth values: the function that maps any member of the domain, o, to T if and only if the formula A is true of o.

For example, consider an interpretation in which S represents "sitting" and E represents "eating" (i.e., S denotes the function mapping any o in the domain to T if and only if it is sitting, and E denotes the function mapping any o in the domain to T if and only if it is eating). Then $\lambda x(Sx \wedge Ex)$ denotes the function that maps any o in the domain to T if and only if '$Sx \wedge Ex$' is true of o—that is, if and only if o is sitting and o is eating. So if c names me in that interpretation, then the sentence $\lambda x(Sx \wedge Ex)c$ is true if and only if Ted is sitting and Ted is eating—i.e., if and only if the sentence $Sc \wedge Ec$ is true.

Similarly, let B be a two-place predicate representing "bites" in some interpretation. Then $\lambda x \exists y Byx$ is a predicate symbolizing "being an x such that something bites x", and denotes the function that maps o to T if and only if something bites o; and so, the sentence $\lambda x \exists y Byx(c)$ is true if and only if something bites Ted—i.e., if and only if the sentence $\exists y Byc$ is true.

Note, then, that $\lambda x(Sx \wedge Ex)c$ is, in a sense, just a long-winded way of saying $Sc \wedge Ec$, and $\lambda x \exists y Byx(c)$ is just a long-winded way of saying $\exists y Byc$. (Similarly, "Ted is such that he is: sitting-and-eating" is just a long-winded way of saying "Ted is sitting and Ted is eating", and "Ted is such that: something bites him" is just a long-winded way of saying "Something bites Ted".) This fact helps a lot in getting the feel for λ abstraction: it's helpful to think of $\lambda x A(c)$ as just meaning $A_{x \mapsto c}$ (i.e., the result of substituting c in for all the free xs in A), and similarly for multi-place complex predicates. Thus we can think of $\lambda x(Sx \wedge Ex)c$ as just meaning $Sc \wedge Ec$, $\lambda x \exists y Byx(c)$ as just meaning $\exists y Byc$, $\lambda xy Bxy(cd)$ as just meaning Bcd, and so on.

If λ just lets us say things more long-windedly, what's the point? We'll think about this more later, but the point is to introduce single syntactic units with complex meanings. Without λ, you can't formulate a *single predicate* which means "is sitting and eating".

5.3.2 Generalizing λ: syntax

Let's think about what the previous section accomplished. In standard predicate logic, all predicates are simple: predicates like $F, G, R, \ldots$. But once we have λ, we can construct complex predicates. Thus for the syntactic category of *predicate*, λ allows us to construct complex expressions of that category.

Once we move from standard predicate logic to higher-order logic, with its very general notion of a syntactic category (type), it is natural to introduce a corresponding generalization of λ abstraction, which lets us construct complex expressions of *any* category.

Let's think in more detail about the syntax of the λ abstracts we introduced in the previous section. An example was:

$$\lambda x(Sx \wedge Ex)$$

There are three important syntactic elements here. (1) The variable attached to λ, namely x. Call this the "abstraction variable". This is an *individual variable*—type e. (2) The expression coming after λx, namely $Sx \wedge Ex$. Call this the "abstraction matrix". This is a *formula*—type t. (3) The entire λ abstract, namely $\lambda x(Sx \wedge Ex)$. This is a *predicate*—type $\langle e, t \rangle$. So: our old λ abstracts were formed by attaching λ to an abstraction variable of type e, and then appending an abstraction matrix of type t, resulting in a λ abstract of type $\langle e, t \rangle$.

To generalize this for higher-order logic: (1) we let the abstraction variables be of any types, $a_1, \ldots, a_n$; (2) we let the abstraction matrix also be of any type, b; and then (3) the resulting λ abstract has type $\langle a_1, \ldots, a_n, b \rangle$. To summarize:

Syntax for λ generalized

For any variables, $x_1^{a_1}, \ldots, x_n^{a_n}$, of types $a_1, \ldots, a_n$, and any expression, E, of type b, the expression

$$\lambda x_1^{a_1} \ldots x_n^{a_n} E$$

is of type $\langle a_1, \ldots, a_n, b \rangle$

(Notice how the complex predicates of section 5.3.1 are special cases of this general rule.)

5.3.3 Generalizing λ: semantics

What do lambda abstracts of arbitrary type mean?

As we saw in section 5.3.1, a λ abstract λxA of type $\langle e, t\rangle$ means "is an x such that A". We can give similar glosses in some other cases. For instance, we can think of λXA as meaning "is a property, X, such that A", and λPA as meaning "is a proposition, P, such that A". (In the previous sentence, the first λ abstract was type $\langle\langle e, t\rangle, t\rangle$, and the second was type $\langle t, t\rangle$.)

But this only gets us so far. The natural language gloss "is a … such that…" is apt only when the abstraction matrix is a formula (type t), since what comes after "such that" needs to be the kind of thing that can be true or false. We still need an explanation of the meanings of λ abstracts whose matrices are not of type t (i.e., λ abstracts of type $\langle a_1, \ldots, a_n, b\rangle$ where $b \neq t$).

Sometimes the best strategy for understanding expressions of complex types is *not* trying to find natural-language glosses for them (since such glosses might not exist), but rather trying to understand what they *do*. By this I mean: understanding how such expressions help determine the meanings of larger expressions of which they are parts.

To understand this for λ abstracts, let's return to a lesson from the end of section 5.3.1. If you attach $\lambda x(Sx \wedge Ex)$ to the name c, the result:

$$\lambda x(Sx \wedge Ex)c$$

is (I said) just a long-winded way of saying:

$$Sc \wedge Ec$$

In general, $\lambda xA(c)$ means the same as $A_{x\mapsto c}$ (i.e., what you get if you start with A and change all the free xs to cs), and similarly for multi-place complex predicates. So what $\lambda x(Sx \wedge Ex)$ *does* is this: when attached to a name, c, it results in a sentence meaning $Sc \wedge Ec$.

The process of changing, e.g., $\lambda x(Sx \wedge Ex)c$ to $Sc \wedge Ec$ is called "β conversion" or "β reduction". In general it works as follows.[27]

[27] Important qualification: no free variables in $A_1, \ldots A_n$ may be "captured" by quantifiers in $E_{x_1^{a_1}\mapsto A_1, \ldots, x_n^{a_n}\mapsto A_n}$. To illustrate the importance of this restriction, consider $\lambda x\exists yBxy$, a complex predicate meaning "bites someone". Without the restriction, $\lambda x\exists yBxy(y)$ would reduce by β

β conversion

The result of applying the λ abstract $\lambda x_1^{a_1} \ldots x_n^{a_n} E$ to expressions $A_1, \ldots, A_n$ (of types $a_1, \ldots, a_n$), namely:

$$\lambda x_1^{a_1} \ldots x_n^{a_n} E(A_1, \ldots, A_n)$$

reduces by β conversion to:

$$E_{x_1^{a_1} \mapsto A_1, \ldots, x_n^{a_n} \mapsto A_n}$$

Actually it's contentious that the result of β conversion means the same as the original. But clearly there is some very close semantic relationship between the two; and in any case thinking of them as meaning the same is a useful heuristic for grasping the meanings of λ abstracts.

β conversion can give us an intuitive handle on the meanings of other λ abstracts. Take another example, $\lambda X(Xc \vee Xd)$ (where c and d are names and X is a type $\langle e, t \rangle$ variable). In this case we do have a natural-language gloss: "being a property that is had either by c or by d". But we can also think about its meaning via what it does. If you attach it to a one-place predicate F, you get this sentence:

$$\lambda X(Xc \vee Xd)F$$

which means the same thing as (via β conversion):

$$Fc \vee Fd$$

Thus what $\lambda X(Xc \vee Xd)$ does is this: it converts any predicate, F, into a sentence meaning that c is F or d is F.

For more practice, let's consider some further examples.

---

conversion to $\exists y Byy$. But the former means "y is such that it bites someone" (with y a free variable) whereas the latter means "something bites itself".

There are other kinds of conversions that also result in expressions that are equivalent (in some sense). "α conversion" is what is often called "relettering of bound variables" in logic: λxFx α-converts to λyFy. "η conversion" is less familiar: λxFx η-converts to F.

Example 1:

$$\lambda P\, P$$

(P is a variable of type t.) Since both the abstraction variable and the abstraction matrix are of type t, the entire λ abstract has type $\langle t, t\rangle$. That is, it combines with a formula to make a formula. So it's a one-place sentence operator (like $\sim$).

What does it mean? Since the matrix is of type t, we can gloss it using "such that": it means "is a proposition, P, such that P". Or, one might say, "is a true proposition" (though really the concept of truth is not involved).

But let's also think about what it does. If you attach it to a formula, A, you get this formula:

$$\lambda P\, P(A)$$

which reduces by β conversion to:

$$A$$

Thus what $\lambda P\, P$ does is attaches to a sentence A to form a sentence that means A. So it is a redundant sentence operator.

Example 2:

$$\lambda x \left(\boldsymbol{q}(R)(x) \wedge \boldsymbol{g}(R)(x)\right)$$

where x is a variable of type e, $\boldsymbol{q}$ and $\boldsymbol{g}$ are predicate functor constants (type $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$, and R is a one-place predicate constant (type $\langle e, t\rangle$). Since x is type e, and $\boldsymbol{q}(R)(x) \wedge \boldsymbol{g}(R)(x)$ is type t, the entire λ abstract is type $\langle e, t\rangle$—it's a one-place predicate of individuals. But what does it mean?

Since the matrix expression is of type t, we can gloss the λ abstract using "such that". It means: "is an entity, x, such that $\boldsymbol{q}(R)(x)$ and $\boldsymbol{g}(R)(x)$". Suppose we think of $\boldsymbol{q}$ as meaning "quickly", $\boldsymbol{g}$ as meaning "gracefully", and R as meaning "runs". Then $\boldsymbol{q}(R)$ is a one-place predicate meaning "runs quickly", and $\boldsymbol{g}(R)$ is a one place predicate meaning "runs gracefully"; and so, the entire λ abstract means "is an x such that x runs quickly and x runs gracefully". In other words, it is a complex one-place predicate meaning "runs quickly and gracefully".

We can reach the same conclusion by thinking about what the λ abstract does. If you attach it to a name, c, symbolizing "Ted", say, you get:

$$\lambda x \left(\boldsymbol{q}(R)(x) \wedge \boldsymbol{g}(R)(x)\right)(c)$$

which reduces by β conversion to:

$$\boldsymbol{q}(R)(c) \wedge \boldsymbol{g}(R)(c)$$

which means that Ted runs quickly and runs gracefully. That is exactly what the complex predicate 'runs quickly and gracefully' does.

Example 3:

$$\lambda Y\, \lambda x \left(\boldsymbol{q}(Y)(x) \wedge \boldsymbol{g}(Y)(x)\right)$$

This one is a little more complicated. Here the matrix variable Y is of type $\langle e, t\rangle$, and the matrix expression is itself a λ abstract:

$$\lambda x \left(\boldsymbol{q}(Y)(x) \wedge \boldsymbol{g}(Y)(x)\right)$$

This "inner" λ abstract is of type $\langle e, t\rangle$, since its abstraction variable x is type t and its matrix $\boldsymbol{q}(Y)(x) \wedge \boldsymbol{g}(Y)(x)$ is type t. Thus the type of the "outer" λ abstract (i.e., the entire original λ abstract $\lambda Y\, \lambda x \left(\boldsymbol{q}(Y)(x) \wedge \boldsymbol{g}(Y)(x)\right)$) is $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$. It's a predicate functor.

We can't gloss the outer λ abstract using "such that" since its matrix isn't of type t. We'll need to think instead about what it does. If you attach it to a one-place predicate, R, you get:

$$\lambda Y\, \lambda x \left(\boldsymbol{q}(Y)(x) \wedge \boldsymbol{g}(Y)(x)\right)(R)$$

which β-reduces to:

$$\lambda x \left(\boldsymbol{q}(R)(x) \wedge \boldsymbol{g}(R)(x)\right)$$

This is just Example 2. If we think of R as meaning "runs", then this is a predicate meaning "runs quickly and gracefully". So, here is what the outer λ abstract did: it converted a predicate meaning "runs" into a predicate meaning "runs quickly and gracefully". Similarly, if attached to a predicate meaning "walks" it will result in a predicate meaning "walks quickly and gracefully". So what it does in general is convert any predicate G into a predicate meaning "Gs quickly and gracefully".

We now have a pretty good handle on what the λ abstract does. In many cases, that will be the best we can do. However, in this case, we can supply a gloss: the λ abstract symbolizes this *complex adverb*: "quickly and gracefully"!

Let's end by examining how the formal semantics of section 5.2.4 can be applied to λ abstracts. In any interpretation, the denotation of a λ abstract of a certain

type will just be the same sort of animal that is generally denoted by expressions of that type: a function of the appropriate sort. Which function? The rule is this:

Semantics for λ abstracts

In any interpretation, the λ abstract

$$\lambda x_1^{a_1} \ldots x_n^{a_n} E$$

denotes the n-place function that maps any n denotations, of types $a_1, \ldots, a_n$, respectively, to the object that E denotes when the variables $x_1^{a_1}, \ldots, x_n^{a_n}$ are assigned those denotations

For example, $\lambda x(Cx \wedge Ex)$ denotes the one-place function that maps any denotation of type e—i.e., any member of the domain—to the object that the formula $Cx \wedge Ex$ denotes when x is assigned that individual. But formulas denote truth values, so this amounts to saying that $\lambda x(Cx \wedge Ex)$ denotes the function that maps a member of the domain to T if and only if the formula $Cx \wedge Ex$ is true of that entity—which is exactly what we said in section 5.3.1.

5.4 Alternate systems

I have been presenting one sort of lambda abstraction and higher-order logic, but there are others out there which you may encounter.

5.4.1 Binary-only types and schönfinkelization

In the approach to types that I have introduced, complex types look like this: $\langle a_1, \ldots, a_n, b \rangle$. A complex type can have any finite number of "constituent types". But in some other systems, complex types are only allowed to have two constituent types; they must always look like this: $\langle a, b \rangle$. That is, complex types must always be binary.

In fact this is not a substantive restriction, since there is a way of simulating the more complex types using only binary types. Let's illustrate this with a two-place predicate, L, for "loves". In standard predicate logic, you can write things like Lcd, meaning that c loves d. The two-place predicate L attaches

to two names to form a sentence. Thus L is of type $\langle e, e, t\rangle$. How would we represent L if we were only allowed to use expressions of binary types?

Here is the trick. (It's called "schönfinkelization" or "schoenfinkelization", after Moses Schönfinkel, or "currying", after Haskell Curry.) We represent L as being of type $\langle e, \langle e, t\rangle\rangle$. Thus it combines with a name or individual variable (type e) to form a *predicate* (type $\langle e, t\rangle$). $L(c)$ therefore is a predicate, meaning *is loved by* c; and we can then attach this predicate to d to form a sentence, $L(c)(d)$, meaning that d is such that it is loved by c.[28]

In a theory allowing only binary types, we would play this same trick with other "multi-place" expressions, such as two-place sentential connectives like $\wedge$, $\vee$, and $\rightarrow$. These are usually treated as having the type $\langle t, t, t\rangle$: they attach to a pair of formulas to make a formula. But in a binary type theory they are treated as having type $\langle t, \langle t, t\rangle\rangle$: they attach to an expression of type t to make an expression of type $\langle t, t\rangle$. That is, they attach to formulas to make one-place sentence operators, which can in turn attach to formulas to make formulas. So, for example, instead of writing $A \wedge B$ (where A and B are formulas), we would instead write $\wedge(A)(B)$.

5.4.2 Relational types

The types we have been discussing are sometimes called "functional" types. There is an alternative system of types, which are often called "relational" types, in which the only types in addition to the types of names and formulas are predicate types, albeit of arbitrarily high level.

To define relational types, you begin with just one primitive type, e. Then, for any types $a_1, \ldots a_n$, there is another type: $(a_1, \ldots, a_n)$.[29] This further type is to be understood as the type of expressions that combine with n expressions, of types $a_1, \ldots, a_n$, respectively, to make a *formula*. The fact that the resulting expression is a formula needn't be explicitly represented in relational type $(a_1, \ldots, a_n)$ (in contrast to the functional type $\langle a_1, \ldots, a_n, b\rangle$, which explicitly represents the type b of the resulting expression), because the resulting expression for

[28] We made an arbitrary choice here, in deciding that $L(c)$ is to mean *is loved by* c. We could have instead decided that it would mean *loves* c. In that case, $L(c)(d)$ would have meant that d loves c. See Heim and Kratzer (1998, section 2.4) on "left-" and "right-" schönfinkelizing.

[29] Relational types are often written with angled rather than rounded brackets: $\langle a_1, \ldots, a_n\rangle$. I'm using a different notation to distinguish them from functional types (which are, by the way, often written as $(a \rightarrow b)$ when they're binary).

relationally typed expressions is *always* a formula; this is hard-wired into the idea of relational types.

For relational types $(a_1,\ldots,a_n)$, the case where $n = 0$ is allowed. That is, there is a type (). This is the type of expressions that combine with zero expressions to make formulas. How can an expression combine with *no* formulas to make a formula? Easy: by already *being* a formula. So what I am saying is that () is the type of expressions that are formulas.

Thus the definition of relational types is as follows:

Definition of relational types

Undefined type: e.

(The type of singular terms)

For any types $a_1,\ldots a_n$ (where n may be 0), $(a_1,\ldots,a_n)$ is also a type.

(The type of expressions that combine with n expressions, of types $a_1,\ldots,a_n$, respectively, to make a formula)

There is an easy to miss, but very important, difference between relational and functional types. Expressions of functional type $\langle a, b\rangle$ convert an a into a b, so to speak, whereas expressions of relational type (a, b) convert an a and a b into a formula. The former are one-place expressions whereas the latter are two-place; and the latter are always predicates whereas the former are not unless b happens to be t. For instance, the functional type $\langle e, e\rangle$ is the type of *one-place function symbols*. Although we haven't discussed this type explicitly, we have met one expression of this type: the successor sign $'$ from the language of arithmetic, which combines with a term (such as 0) to form a term (0′). But the similar-looking relational type (e, e) is the type of two place (first-order) predicates, such as B for "bites".

Just as we introduced a logical language based on functional types at the end of section 5.2.2, we can also introduce a logical language based on relational types. As before, for each relational type there will be variables and perhaps constants; propositional connectives will be assigned appropriate types (for instance, $\sim$ will have type (()) and $\wedge$ will have type ((),())); and for any variable v of any

type and any expression E of type () (i.e., any formula), $\forall vE$ and $\exists vE$ will be expressions of type (). But the rule for forming complex expressions will be a little different. The old rule for functionally typed expressions looked like this:

> If $E_1,\ldots,E_n$ are expressions of types $a_1,\ldots,a_n$, and E is an expression of type $\langle a_1,\ldots,a_n,b\rangle$, then $E(E_1,\ldots,E_n)$ is an expression of type b

whereas the new rule for relationally typed expressions looks like this:

> If $E_1,\ldots,E_n$ are expressions of types $a_1,\ldots,a_n$, and E is an expression of type $(a_1,\ldots,a_n)$, then $E(E_1,\ldots,E_n)$ is an expression of type ().

This new language is in many ways similar to the language introduced at the end of section 5.2.2, though not perfectly so. Each language has names, formulas, sentential connectives, quantifiers, n-place first-order predicates for each n, n-place second-order predicates for each n, and so on. (Notice that in most cases, these expressions are assigned different-looking types in the two systems. For instance, formulas are of functional type t, but relational type (); one-place predicates are of functional type $\langle e,t\rangle$, but relational type (e); binary sentential connectives (like $\wedge$) are of functional type $\langle t,t,t\rangle$, but relational type $((),())$.) But the functional language has some additional expressions without direct counterparts in the relational language. For example, only the functional language has one-place predicate-functors (i.e., expressions that combine with one-place predicates to form one-place predicates; i.e., expressions of type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$; i.e., adverbs). The relational language lacks such expressions because every complex relational type is that of an expression which, when combined with appropriate other expressions, produces a *formula*, whereas predicate-functors combine with predicates to produce *predicates*.

So there is a sense in which functional types are "syntactically more expressive" than relational types: on the natural way of associating types with syntactic categories, relational syntactic categories are a proper subset of functional syntactic categories. But in a way this is superficial, since there is a sense in which relational types are just as *semantically* expressive as functional types. See Dorr (2016, Appendix 2) for details, but the basic idea is that we can use a trick akin to schönfinkelization to simulate the meanings of, e.g., predicate functors with relationally typed expressions. Given the semantics of section

5.2.4, a predicate functor (type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$) expresses a function from predicate meanings to predicate meanings—i.e., a one-place function from (i) functions-from-entities-to-truth-values, to (ii) functions-from-entities-to-truth-values. But the information encoded in any such function could also be encoded in a function that moves the argument of the function in (ii) into (i)—i.e., by a two place function from (ia) functions-from-entities-to-truth-values and (ib) entities, to (ii) truth values. For example, the functionally typed adverb 'quickly' expresses the function, q, that maps any function g (from entities to truth values) to the "g-ing quickly" function—the function that maps any entity x to T iff x "gs quickly". We can simulate this function with the closely related function q' that maps any g and any entity x to T if and only if x gs quickly. And since this is a function to truth values, it is the kind of meaning that can be expressed by a predicate (if we modify the semantics of section 5.2.4 in the obvious way for relational types), namely a predicate with two arguments, the first of which is a predicate and the second of which is a name. And such predicates do exist in the relationally typed language: they have relational type $((e),e)$.

5.4.3 Quantifiers as higher-order predicates

In first-order logic, quantifiers do two things: bind variables and express quantity. λ abstracts also bind variables, and in fact can take over variable-binding from quantifiers. Here is how.[30]

Why not think of a quantified natural language sentence like:

Something is sitting and eating

as meaning:

Sitting-and-eating has at least one instance

? Here 'sitting-and-eating' is a (complex) predicate, and 'has at least one instance' is a higher-order predicate—a predicate of a predicate. This in turn suggests a symbolization like this:

$$\text{has-at-least-one-instance}\big(\lambda x(Sx \wedge Ex)\big)$$

where 'has-at-least-one-instance' is a predicate applied to the predicate $\lambda x(Sx \wedge Ex)$ representing 'sitting-and-eating'. Only, to save writing, let's write $\exists$ instead

[30] See Stalnaker (1977). This is inspired by Frege, e.g. (1892, p. 187).

of 'has-at-least-one-instance', and drop the outer parentheses:

$$\exists \lambda x(Sx \wedge Ex)$$

Compare, now, the standard symbolization:

$$\exists x(Sx \wedge Ex)$$

In the standard symbolization, the quantifier both binds the variable in the matrix formula $Sx \wedge Ex$, and also makes a statement of quantity: that at least one thing satisfies the matrix. In the new symbolization, λ binds the variable, and then all the quantifier $\exists$ has to do is make the statement of quantity: that the predicate formed using the λ has at least one instance.

So our new syntax for the quantifiers is this:

> If Π is a one-place predicate then $\forall\Pi$ and $\exists\Pi$ are formulas

As we've seen, instead of writing $\exists x(Sx \wedge Ex)$ in the new syntax we now write $\exists x \lambda x(Sx \wedge Ex)$. Similarly, instead of writing $\forall x(Fx \rightarrow Gx)$ we write $\forall \lambda x(Fx \rightarrow Gx)$. Instead of writing $\exists x Fx$, we write $\exists x \lambda x Fx$, or better, just $\exists F$. And instead of writing $\exists x \forall y Rxy$ for "there is someone who respects everyone", we instead write:

$$\exists \lambda x \forall \lambda y Rxy$$

Let's think about this last example carefully, working backwards: $\lambda y Rxy$ is a predicate meaning "is a y that x respects", or more concisely, "is respected by x" (x is free here), so $\forall \lambda y Rxy$ is a sentence meaning "everyone is respected by x", that is, "x respects everyone" (x is still free), so $\lambda x \forall \lambda y Rxy$ is a predicate (in which x is no longer free) meaning "is an x that respects everyone", or more concisely, "respects everyone"; and so, finally, $\exists \lambda x \forall \lambda y Rxy$ is a sentence saying that there is someone who respects everyone.

We can play this trick in higher-order logic too. Instead of symbolizing "Ted has at least one property" as:

$$\exists X Xc$$

we can instead symbolize it thus:

$$\exists \lambda X Xc$$

meaning: "the property of *being a property had by Ted* has at least one instance—i.e., is had by at least one property". And instead of symbolizing "every proposition is either true or false" as:

$$\forall P(P \vee \sim P)$$

we can instead write:

$$\forall \lambda P(P \vee \sim P)$$

meaning: "the property of *being a proposition that is either true or false* is such that every proposition has it".

In general, for any type a, instead of writing:

$$\exists x^a A \qquad\qquad \forall x^a A$$

(where A is a formula) we now write instead:

$$\exists^a \lambda x^a A \qquad\qquad \forall^a \lambda x^a A$$

(In the previous two examples I left out the superscripts on $\exists$ or $\forall$, and we might continue to do so in informal contexts, but the idea here is that they are officially required.[31]) $\exists^a$ and $\forall^a$ are of type $\langle\langle a, t\rangle, t\rangle$: they attach to predicates of type

[31]To require the superscripts is to treat quantifiers of different types as being distinct logical constants—e.g., $\forall^t$ is not the same symbol as $\forall^e$. This isn't mandatory; one could instead continue with just two quantifiers, $\exists$ and $\forall$, and write the semantics so that the truth conditions of sentences with quantifiers depends on the type of the variable to which the quantifiers are attached. But one nice thing about requiring the superscripts is that quantifiers can then be treated "categorematically". A categorematic expression is one that has a meaning in isolation; a "syncategorematic" expression has no meaning in isolation; rather, larger expressions containing it have a meaning. ("Having a meaning" needs to be understood narrowly in order for this to make sense, as signifying that the expression is assigned an entity as a meaning by a certain semantic theory.) In the standard semantics for first-order logic, quantifiers and sentential connectives are syncategorematic, since instead of assigning them denotations (in the way that we assign predicates and names denotations) we state rules that govern sentences containing them, such as "$A \wedge B$ is true if and only if A is true and B is true" or "$\forall v A$ is true if and only if A is true of every member of the domain". But in higher-order logic, sentential connectives can be treated categorematically. E.g., recall from section 5.2.4 that $\sim$ (which is of type $\langle t, t\rangle$, denotes (in any interpretation) the function that maps T to F and F to T. Now, if in higher-order logic we had just the two untyped quantifiers, they would still need to be syncategorematic. But with typed quantifiers, each one can receive a denotation in isolation, as in the text. Note, however, that even in higher-order logic we still need a syncategoramic expression: λ.

$\langle a, t\rangle$ (which is the type of the λ abstracts they're attached to) to make formulas. We can gloss $\exists^a$ and $\forall^a$ as meaning "applies to at least one a-entity" and "applies to every a-entity", respectively. Given the formal semantics for higher-order logic we have been developing, in any interpretation the denotations of these logical constants would be the following:

> $\exists^a$ denotes the function that maps any $\langle a, t\rangle$ denotation, d, to T if and only if for some a denotation, d', $d(d') = T$
>
> $\forall^a$ denotes the function that maps any $\langle a, t\rangle$ denotation, d, to T if and only if for every a denotation, d', $d(d') = T$

References

Bacon, Andrew (2022). *A Philosophical Introduction to Higher-order Logics*. Routledge. Forthcoming.

Boolos, George (1971). "The Iterative Conception of Set." *Journal of Philosophy* 68: 215–31. Reprinted in Boolos 1998: 13–29.

— (1998). *Logic, Logic, and Logic*. Cambridge, MA: Harvard University Press.

Church, Alonzo (1940). "A Formulation of the Simple Theory of Types." *Journal of Symbolic Logic* 5: 56–68.

Dorr, Cian (2016). "To Be F is to Be G." *Philosophical Perspectives* 30(1): 39–134.

Dorr, Cian, John Hawthorne and Juhani Yli-Vakkuri (2021). *The Bounds of Possibility: Puzzles of Modal Variation*. Oxford: Oxford University Press.

Enderton, Herbert (1977). *Elements of Set Theory*. New York: Academic Press.

Fine, Kit (2007). "Relatively Unrestricted Quantification." In Agustín Rayo and Gabriel Uzquiano (eds.), *Absolute Generality*, 20–44. Oxford: Oxford University Press.

Frege, Gottlob (1892). "On Concept and Object." In Frege (1997*a*), 181–93.

— (1952/1892). "On Sense and Reference." In Peter Geach and Max Black (eds.), *Translations of the Philosophical Writings of Gottlob Frege*. Oxford: Blackwell.

— (1997*a*). *The Frege Reader*. Ed. Michael Beany. Oxford: Blackwell.

— (1997*b*). "Function and Concept." In Frege (1997*a*), 130–48.

Heim, Irene and Angelika Kratzer (1998). *Semantics in Generative Grammar*. Malden, MA: Blackwell.

Kaplan, David (1994). "A Problem in Possible Worlds Semantics." In Walter Sinnott-Armstrong (ed.), *Modality, Morality, and Belief*, 41–52. New York: Cambridge University Press.

MacFarlane, John (2005). "Logical Constants." Stanford Encyclopedia of Philosophy. Available at `http://plato.stanford.edu/entries/logical-constants/`.

Shapiro, Stewart (1991). *Foundations without Foundationalism: A Case for Second-Order Logic*. Oxford: Clarendon Press.

Stalnaker, Robert (1977). "Complex Predicates." *The Monist* 60: 327–39.