Chapter Two: Invention and Satisfaction

The previous chapter was largely an attempt to explain my claim that phrasal meanings are instructions for how to build conjunctive monadic concepts. This proposal is developed in chapters five and six. Towards the end of chapter zero, I reviewed some reasons—further articulated in chapters three and four—for seeking an alternative to truth-theoretic conceptions of meaning for human languages. This chapter reviews some relevant history and technical notions.

Section one is a reminder that Frege did not provide an algorithm that specified Bedeutungen (or Sinnen) for the boundlessly many expressions of his Begriffsschrift. In this sense, Frege did not provide an explicit semantics for his ideal language; he "merely" offered fecund suggestions for how to interpret his insightful invented syntax. As discussed in sections two and three, Tarski provided a semantics for a first-order fragment of Frege's language, and Church offered a more Fregean semantics for a more Fregean extension of Tarski's language. Davidson suggested that we can provide good models of human languages by drawing on Tarski directly, without the Church-style additions, which Montague then added. My suggestion is that we should go back to Frege and Tarski, try to develop a second-order *extension of a fragment* of a first-order predicate calculus that generates boundlessly many monadic concepts. To understand this dialectic, it helps to first be clear about what Frege didn't do.

1. You Can't Always Get What You Want

In chapter one, I described Frege's taxonomy of concepts in formal terms, setting aside questions about what concepts signify. But Frege wanted expressions of his Begriffsschrift to have a cluster of interpretational properties: each expression of type <e> indicates a particular thing; each expression of type <**t**> indicates one of two truth values; every other expression indicates a function; and each complex expression, which is the result of combining some expression of a type <α, β> with an expression of type <α>, indicates *the result of applying* the corresponding function of type <α, β> to the entity or function indicated by the "saturater" of type <α>. Each expression is also supposed to indicate something in a certain way; where this way of indicating (Sinn) determines that which is indicated (Bedeutung), so that a Sinn—a way of thinking about a Bedeutung—can be a constituent of a thought-content (Gedanke) that can be a premise or conclusion of an argument. Frege's hope was that an ideal thinker would be positioned to know which entity or function is indicated by any ideal expression/concept, given knowledge of which Sinnen the atomic expressions/concepts expressed. But this picture suffered from three serious flaws (see Dummett 1981 and Evans 1982): reference to the unsaturated (§1.1); the persistent specter of Sinn without Bedeutung (§1.2); and lack of an explicit procedure for interpreting *Begriffsschrift* analogs of 'Rxy & Syzw', which can be bound by multiple quantifiers (§1.3).

1.1 Functions and Unsaturateds

As noted in the introductory chapter, Frege accorded a kind of priority to the notion of a function as a procedure that maps inputs onto outputs. By itself, that would be fine. But Frege also suggested that functions, in his sense, are *unsaturated* in a way reflected by ideal predicative symbols. And that lead to headaches.

The basic distinction between procedures and sets is clear enough, at least since Church's (1941) reconstruction of it. Recall that each "function in intension" (procedure) determines exactly one "function in extension" (set of ordered pairs), which can be determined by many procedures. So we can speak of I-functions vs. E-functions, and say that Frege (1892b) characterized E-functions in terms of I-functions. The E-functions indicated with technical predicates like 'Number(_)' and 'Prime(_)' are illustrative cases. In thinking about any such set,

which has infinitely many elements, one thinks about it as the extension of some I-function. So in light of Frege's (1892a) related Sinn/Bedeutung distinction, one might say that an ideal predicative expression indicates an E-function via an I-function that can itself be thought of in many ways. But Frege did not stop here.

Frege also stressed that thoughts, which can be endorsed and so used to make *judgments*, are not mere lists of representations. One needs to be careful with the terminology, since words like 'thought' and 'concept' exhibit symbol/content polysemy; 'thought' and 'judgment' are also like 'assertion', which can be used to describe certain events that be characterized in terms of contents. In speaking of a thought that Sadie is a horse, one might be talking about a mental episode, a mental sentence, or a content shared by various sentences of various languages. But as in chapter one, let's use small capitals so that HORSE(SADIE) is a mental sentence in which the concept SADIE saturates the unsaturated concept HORSE(_), which is effectively a sentence frame. By contrast, HORSEHOOD is a denoting concept with which one thinks about an abstract entity.

This doesn't yet require that sentences indicate truth values, or that monadic concepts indicate functions. But we can stipulate that the "ideal" concept **HORSE**( ) indicates or "Bedeuts" a function from entities to truth values, while **SADIE** indicates a unique entity in the domain of that function, and the ideal mental sentence **HORSE(SADIE)** indicates a truth value.[1] The sense (Sinn) of this sentence, which presents the truth value in a certain way, is then a thought-content (Gedanke) that can be endorsed or rejected by endorsing or rejecting **HORSE(SADIE)**. But the predicative concept **HORSE**( ) is not singular. The indicated Gedanke does not combine a singular way of thinking about Sadie combined with a singular way of thinking about a function. According to Frege, the indicated Gedanke is the result of saturating an *unsaturated* way of thinking about (things mapped to truth iff they are) horses; where saturating an unsaturated can result in a unified judgeable thought, as opposed to a mere list of constituents.

That does not yet explain the unity in question. But Frege's hunch was that unsaturated constituents play a special role in explaining the unity of sentences/Gedanken, while saturating constituents play a special role in explaining how expressions/Sinnen can be about things. Part of the idea was that **SADIE** does not merely indicate its Bedeutung. The saturating concept bears a special relation to a particular horse—a relation that **HORSE**(_) does not bear to the indicated function. We can also say that **SADIE** denotes Sadie, while **HORSE**(_) is of the wrong type to be a denoter; see section four. Frege spoke of **HORSE**(_) having an unsaturated content. And given his interest in ideal contents, as opposed to actual or even idealized symbols, this led Frege to adopt a notion of 'Concept' such that while **HORSE**(_) sort of indicates a Concept, we should not say that the Concept indicated with **HORSE**( ) is a Concept: to use 'the Concept' and 'a Concept' in this way would be to imply that Concepts can be denoted and quantified over.

Frege wanted to refrain from endorsing this implication, and he wanted his ideal language to preclude its formulation; cp. Wittgenstein (1921), Ricketts (199x). But we needn't dwell on ineffability. Suffice it to say that Frege's discussion of Concepts seemed at odds with his claim that we can't talk about them. Even recognizing the motivation for Frege's claim, and giving him the "pinch of salt" he asked for, one wonders if talk of the unsaturated shouldn't be tamed and limited to talk of certain *symbols*.[2] In any case, until Church helped sort things out, one could

[1] We can also define 'indicates' so that ordinary mental sentences indicate truth values, even if there is no independent natural sense in which concepts like HORSE(_) indicate functions.

[2] That leaves the question of what distinguishes a thought from a short list of symbols that exhibit the types <α> and <α, **t**> for some type <α>; cp. note X chapter two. But one can say that in the thought, one concept saturates the other. That labels the distinction without explaining it, since we don't yet know concepts are, or what saturation is.

easily get confused by the various Fregean notions of procedure and determination: Sinnen as procedures that determine Bedeutungen; I-functions that determine E-functions; singular concepts that determine/denote entities; and undenotable Concepts that determine functions. As we'll see, Tarski appealed to none of this, and so avoided the potential confusions.

1.2 Thinking of Nothing

Setting aside reference to unsaturateds, concepts like VULCAN and UNICORN( ) suggest a second difficulty for Frege's intended interpretation of his ideal symbols. Frege was glib about concepts that indicate nothing, as if he assumed that such concepts only figure in episodes of thinking that ideal thinkers would recognize as episodes of story-telling, in a sense of 'story' that excludes sufficiently disciplined attempts to think true thoughts about mind-independent aspects of reality. But a serious thinker can mistakenly think that Vulcan is as real as Neptune, that unicorns are odd horses, that phlogiston is a component of air, etc.

To engage with such thoughts, if only to deny them, it seems that one must interpret symbols that lack Bedeutungen. One can say that such symbols still have Sinnen. But as Evans (1982) stresses, if Sinnen are characterized as ways of presenting Bedeutungen, we need some account of "ways of presenting" that applies to examples like VULCAN. This makes it tempting to identify Sinnen with descriptions; see Russell (1905), Quine (1963). But even setting aside Kripke's (1980) apparently fatal objections, and Frege's inclination to view descriptions like 'the smallest prime' as complex names, one wants to understand conditionals like the following: if there is a greatest prime, then *it* is greater than *its* factorial plus one. Perhaps ideal thinkers know that in proofs by reductio, a singular concept is used in a story-telling way. But in the course of rejecting the thought that there is a greatest prime, it seems that even an ideal thinker briefly entertains a thought with a singular component that purports to indicate the greatest prime.

At a minimum, Frege did not say enough about concepts that indicate nothing. Even a great logician can suspect that every predicate (and hence 'nonselfelemental') has an extension, and hence that a singular concept can be introduced to denote the set of nonselfelemental sets. This point here is not merely that Frege was susceptible to human error in allowing for such a set. Given paradoxes like Russell's, one might worry that the interpretive apparatus of functions is unstable—and that this instability is a potential source of massive error that precludes Fregean thoughts from ever being true (without also being not true). Even idealized symbols can fail to make the intended contact with reality, at least if one is not *very* careful about the kinds of stipulations used to specify interpretations for the symbols. And while Frege's Begriffsschrift was designed to permit quantification into unsaturated positions (see §1.5), his claims about predicates were inconsistent. By contrast, as discussed below, Tarski characterized his notion of satisfaction for languages that do not permit quantification into predicate positions.

Moreover, in a Tarskian language, only sentences have satisfaction conditions. No interpretations are assigned to subsentential expressions. So there is no issue about which interpretation to assign to UNICORN( ) or VULCAN. This makes it easier to stipulate coherent interpretations; though of course, human languages may not conform to these simplifications.

Let me briefly digress, though, to note a different response to concepts without Bedeutungen. One might adopt a hyper-externalistic understanding of 'ideal', so that no ideal thinker has the concept VULCAN, and there is no ideal concept **VULCAN** that lacks a Bedeutung; cp. Evans (1982), McDowell (1995). But then an ideal thinker must not only have her mental house in good order, avoid paradox-inducing claims, and be largely *en rapport* with an environs free of Cartesian demons. In this hyper-externalistic sense, ideal thinkers are immune from certain kinds of errors, perhaps to the point of being unable to think about the same thing in two

ways without knowing it; cp. Kripke (1979a). One can characterize corresponding notions of concept*, language*, and their cognates. These notions, normative in more than one way, may be of interest for certain purposes. Perhaps inquirers aspire to have languages* whose expressions* have meanings* that are instructions to build concepts* that can be constituents of thoughts*. But my aim, more mundane, is to describe the thoughts/concepts/meanings that ordinary humans enjoy. And I see no reason to believe that the best—or even a good—way to study concepts or meanings is by viewing them as imperfect analogs of their hyper-externalistic counterparts, even if we can and should try to acquire concepts*.

Notions like truth and denotation may well be required to interpret philosophical talk of thoughts*/concepts*/meanings*. One can also hypothesize that ordinary words combine to form sentences that express thoughts composed of concepts that denote or "are true of" things in the environment. But this hypothesis does not become inevitable as soon as we talk about thoughts/concepts/meanings. On the contrary, natural concepts seem to be mental symbols that can be used to think about things, even when the concepts apply to nothing. And if words like 'give' fetch *naturally* introduced concepts like GIVE(_), we must be prepared to discover that words fetch concepts that fall short of certain philosophical ideals.

1.3 Thinking of Something and Everything

Worries about the unsaturated, along with Bedeutung-less Sinnen, are serious in the context of Frege's overall conception of thought. But one might hope to quarantine these concerns and deal with them later. Moreover, one might think that these problems stem from Frege having said too much, and that a less ambitious theory would be more defensible. But with regard to Begriffsschrift analogs of 'Rxy & Syzw', which can be bound by multiple quantifiers, the concern is that Frege did not provide an explicit procedure for interpreting such expressions. And this gap, which Tarski and Church filled, was significant.

Frege did not think of ideal expressions as well-formed formulae that a machine could generate and theorists could interpret in various ways. That conception of expressions came later. Frege viewed ideal expressions as contentful representations with which one can think—or perceptible instructions for how to assemble such representations—not as mere forms that are neutral with regard to their interpretation.[3] In this sense, he did not need to specify an intended model for his formalism. But Frege did want it to be possible for a thinker to *figure out* what a complex expression indicates by applying knowledge of what the constituents indicate.

For endlessly many cases, it is clear how this was supposed to work. The concept **&**(ABOVE(VENUS, BESSIE), ∃[COW(_)]) is of type <**t**>, and hence a complete thought. This thought, or at least its ideal analog, indicates ⊤ or ⊥. More specifically, the thought indicates ⊤ if and only if Venus is above Bessie, *and* there is a cow. An ideal thinker can presumably employ the concept ~[COW(_)]—even though ~[…] is of type <**t**, **t**>—by forming a thought like ~[COW(BESSIE)] and abstracting away from the singular concept. And ∀[Φ(_)] can be introduced as a concept of type <e**t**, **t**> equivalent to ~∃~[Φ(_)]. Matters are less clear, though, for endlessly many other examples like ∃X∀Y∃Z[ABOVE(X, Y) ! GAVE(Z, W, Y)]; where as discussed in §3.2 of chapter one, '!' indicates adicity-neutral conjunction, and the variable letters indicate linking of unsaturated conceptual slots.

[3] See, e.g., Heck and May (2006, 2010). Model theory for ideal mental sentences is a bad party game: pretend that *thoughts* can change contents, like people can change shirts, and ask which contents a thought-form can support given constraints pulled from a hat. Chomsky's (1965, 1970, 1986, 1995, 2000a) conception of syntax is like Frege's in this respect, and claims about the "autonomy" of syntax should be understood accordingly (see Berwick et. al. 2011); cp. note 16 of chapter zero.

Since ∃X∀Y∃Z[ABOVE(X, Y) ! GAVE(Z, W, Y)] has exactly one unbound variable/slot, it must indicate a function of type <e, **t**>. In this case, we know which function is intended: the smallest one that maps an entity (w) to ⊤ iff something (x) is above each entity (y) that something (z) gave to the first entity (w). But how is a thinker supposed to *work this out,* as opposed to just "seeing" which ideal thought is intended? *Given* a complete thought like ∃X∀Y∃Z[ABOVE(X, Y) ! GAVE (Z, BESSIE, Y)], an ideal thinker could form the thought-frame ∃X∀Y∃Z[ABOVE(X, Y) ! GAVE (Z, W, Y)]. But the question is how the complete thought is formed in the first place. One can say that $\exists_X[\Phi(X)]$—a.k.a. ∃[Φ(_)]—is a concept of type <e**t**, **t**>. But this concept can neither saturate nor be saturated by ∀Y∃Z[ABOVE(X, Y) ! GAVE (Z, W, Y)]. This latter concept is of the dyadic type <e, e**t**>.

One can introduce quantificational concepts of type <<e, e**t**>, <e**t**, **t**>>. But the problem recurs: $\forall_Y[\Phi(Y)]$ is saturatable by COW(_), but not by ∃Z[ABOVE(X, Y) ! GAVE (Z, W, Y)], which is of type <e, <e, e**t**>>; and $\exists_Z[\Phi(Z)]$, of the same type as $\exists_X[\Phi(X)]$, cannot be saturated by a concept of type <e, <e, <e, e**t**>>>. In this sense, Frege helped himself to the powerful idea that ∃[Φ(_)] can combine with unsaturated concepts of endlessly many types—in effect, relativizing to assignments of values to any variables not targeted by the quantifier—making his initial suggestion that ∃[Φ(_)] is of type <e**t**, **t**> prescient though somewhat naughty.

Again, one can say that ideal thinkers have the requisite abstractive powers, given the capacities required to form thoughts like ABOVE(BESSIE, VENUS) ! GAVE(SADIE, FIDO, VENUS). But one needs an account of these powers/capacities—which permit of abstraction of a tetradic concept from ABOVE(_, _) and GAVE (_, _, _)—to explain how anyone could form the concepts that Begriffsschift expressions reflect. Communication raises a related issue: how can a thinker perceive a complex Begriffsschrift expression, whose variables can be bound in various ways by multiple quantifiers, as an instruction to build a correspondingly complex concept? This is not to doubt the utility of Frege's logical syntax. On the contrary, I take his proposals about the architecture of thoughts to be major contributions to both logic and psychology. But Frege insightfully invented a logical syntax whose intended interpretation raised important questions that he did not answer.

Tarski showed how to interpret formulae of arbitrary adicity, containing variables that can be bound by arbitrarily many existential/universal quantifiers, for certain simple languages. This was another major achievement. But if all human language phrases are instructions to build monadic concepts—or perhaps predicates that need not be viewed as indicating functions of type <e, **t**>—theories of meaning for human languages may not require all of Tarski's apparatus. And if human languages let us build concepts with quantifiers beyond those of the first-order predicate calculus, we may need to think about *extensions of fragments* of Tarskian languages. That, however, is advertisement. And before turning to Tarski, I want to stress that Frege needed a procedure for interpreting complex quantificational expressions, given how he used his Begriffsschrift. For this is also an opportunity to say how languages can be used to introduce formally new concepts, and how expressions can be Begriffsplans (concept assembly instructions), in the context of examples that motivated Frege.

1.4 Thinking of Numbers

The absence of an explicit semantics for his Begriffsschrift was not a mere technical lacuna in the context of Frege's larger project. Frege maintained that thinkers can employ an ideal language to *introduce* higher-order concepts that are useful in the study of logic.[4] And especially

[4] I have been influenced, here and elsewhere, by Horty's (2007) excellent discussion of Frege's notion of definition.

when technical concepts are being invented, one cannot assume a capacity to intuit what inventers intend when they use symbols in new ways. For these purposes, an explicit semantics really is called for. (This point is relevant to any version of the idea that human languages can be used to introduce concepts like CHASE(_) or BROWN[_, Φ(_)].) In this context, it will be useful to contrast two kinds of concept introduction: explicit definition of new concepts in terms of old ones; and invention of concepts to be used in theorizing.

Given a concept of predicate conjunction (see §2 of chapter one) and a capacity to introduce concepts by stipulation, one can use HORSE(_), FEMALE(_), and ADULT(_) to introduce a fourth concept: MARE(_) $\equiv_{DF}$ ∩[ADULT(_), ∩[FEMALE(_), HORSE(_)]]. From a purely cognitive perspective, it is hard to see the point of such stipulation. But a certain kind of language-user might connect lexical items with atomic concepts. If such a thinker/speaker encounters 'mare', she might introduce MARE(_) with a "meaning postulate" that permits replacement of the atomic but not primitive concept with a conjunction. Perhaps BACHLEOR(_) likewise stands in for ∩[~MARRIED(_), ∩[MARRIAGEABLE(_), MALE(_)]]. Other cases may be more interesting.

Imagine thinkers who use the dyadic concept PARALLEL(_, _) and a concept of existential closure to introduce the triadic concept SHARED-DIRECTION(_, _, _) as follows.[5]

∀L∀L′{∃D[SHARED-DIRECTION(D, L, L′)] $\equiv_{DF}$ PARALLEL(L, L′)}

This does not determine correctness conditions for all uses of the triadic concept. But for such thinkers, the thought that each line is parallel to itself would imply that any given line shares a direction with itself: ∀L∃D[SHARED-DIRECTION(D, L, L)]. Such introduction raises questions about the proper treatment of variables in discourse that aims at truth. But quantifying over directions given lines seems less ontologically inflationary than quantifying over pencils given lines; cp. Wright (200x). Perhaps properly understood, ∃L∃D[SHARED-DIRECTION(D, L, L)] does not imply the existence of a shared direction any more, or less, than ∃L[PARALLEL(L, L)] does.

In any case, theorists often introduce concepts like QUARK(_) in a more ambitious way that is not definitional. Invented concepts can be related, in various ways, to natural concepts. One might regard non-Euclidean ways of thinking about space, or relativistic ways of thinking about mass, as abstract analogs of more familiar concepts—treating generalizations previously held to be universal as special or limiting cases. Though if the world is fundamentally bizarre, we may need to invent bizarre concepts to think about the world as it is fundamentally. This is not the kind of concept introduction Frege discussed. But for some technical concepts, one can imagine minds that *start* with them, and then try to characterize *our* "more primitive" concepts as special case of theirs. Frege undertook such a project with regard to a cluster of arithmetic concepts including NUMBER(_), ZERO, and PRECEDES(_, _).

We can use these concepts to form thoughts that might be regarded as axiomatic: zero is a number; every number precedes another; no number precedes zero; etc. But famously, Frege wanted to encode such thoughts in a way that captures what follows from them, while also making it possible to ask what they follow from. He hoped to reduce (his versions of) the Dedekind-Peano axioms for arithmetic to principles of logic and definitions for the requisite nonlogical vocabulary. He did not fully succeed. But given the definitions he introduced, and using only a consistent fragment of his logic, Frege was able to derive the (re-presented) axioms from "Hume's Principle" concerning the relation of cardinality to correspondence:

(HP) ∀s∀s′{≡[=[$^{<e, e>}$NUMBER-OF(S), $^{<e, e>}$NUMBER-OF(S′)], ONE-TO-ONE(S, S′)]}

[5] Cp. Frege (1884). The triadic concept might then be used to introduce concepts of *having a direction* and *directions*: DIRECTION(D) $\equiv_{DF}$ ∃L[DIRECTION-OF (D, L)]; DIRECTION-OF(D, L) $\equiv_{DF}$ ∃L′[SHARED-DIRECTION (D, L, L′)].

two sets have the same cardinality iff their elements correspond one to one.[6]

As the superscript suggests, $^{<e, e>}$NUMBER-OF(_) is not a concept of type <e, **t**>; though NUMBER-OF(_, _) is of type <e, e**t**>. Saturating $^{<e, e>}$NUMBER-OF(_) with an <e>-type concept that denotes a set yields a complex <e>-type concept of the number that is the set's cardinality. The concepts =[_, _] and ONE-TO-ONE(_, _) are of type <e, e**t**>, taking numbers and sets to be entities, while ≡[_, _] is of type <**t**, **tt**>. So (HP), as encoded above, reflects a relation exhibited by these three equivalence relations. But if the numbers are to be characterized in terms of logic and (HP), as opposed to logic and the unreduced Dedekind-Peano axioms, reconceptualization is required.

In particular, one has to imagine a line of thought in which the concept NUMBER(_)—which applies to things like zero and thirteen—is introduced, as opposed to being a primitive concept. But that's not so surprising if you think about arithmetic induction the way Frege did. It seems obvious that for any property P: if (i) zero has P, and (ii) any given number has P if its predecessor does, then (iii) every number has P. Yet replacing 'number' with 'cow', in (ii) and (iii), does not preserve obviousness. The contents of NUMBER(_), ZERO, and PREDECESSOR(_, _) are evidently related in a way that the contents of COW(_), ZERO, and PREDECESSOR(_, _) are not. This invites the project of re-presenting the contents of arithmetic concepts in more interesting ways, and perhaps *deriving* a reformulated principle of arithmetic induction.

To this end, Frege defined zero as the number of (the set of) nonselfidentical things, and defined the other numbers inductively. One, whose predecessor is zero, is the number of things identical with zero; two, whose predecessor is one, is the number of things identical with zero or one; etc. Each number *n* is the predecessor of another—viz., the number of numbers in the "predecessor series" up to and including *n*; and given (HP), *n* is the number of set s iff the elements of s correspond one to one with the numbers in the predecessor series up to but not including *n*. By itself, this is merely a clever characterization of the numbers. But Frege also introduced *ANCESTRAL*-R(_, _)—a higher-order concept of type <<e, et>, <e, et>>—which can be saturated by a relational concept like PREDECESSOR(_, _) to yield a concept whose extension is the transitive closure of the saturating concept. Indeed, this is where Frege really put his relational logic to work. Given any type <τ>, one can introduce expressions of the higher type <τ, τ>. In particular, *ANCESTRAL*-R(_, _) can be defined shown below.

$$\forall X \forall Y \{\textit{ANCESTRAL}\text{-}R(X, Y) \equiv_{DF}$$
$$\exists Z[R(X, Z)] \,\&\, \forall F[FX \,\&\, \forall W \forall Z\{FW \,\&\, R(W, Z) \supset FZ\} \supset FY]\}.$$

That is, x is an R-ancestor of y iff x bears relation R to something, and every function F of type <e, **t**> meets the following condition: *if* F maps x to **t**, and F maps z to **t** whenever anything that F maps to **t** is R-related to z, *then* F maps y to **t**.[7]

In general, for any relation and any entities: *ANCESTRAL*-R(X, Z) if R(X, Y) and R(Y, Z). The relation of precedence, which zero bears to two, is the ancestral of the predecessor relation that zero bears to one and no other number: ∀X∀Y[PRECEDES(X, Y) ≡ *ANCESTRAL*-PREDECESSOR(X, Y)]. This makes the logical relation between PRECEDES(_, _) and PREDECESSOR(_, _) manifest;

[6] See Wright (198x), Heck (2011), Demopolous (1993), Zalta (200x). Frege's version of (HP) quantified over "courses of values" of functions, as opposed to sets *per se*. And in some settings, one might want to relativize to idealized concepts. For as Frege notes, thousands of soldiers can be a few regiments and a single army.

[7] Put another way, x is an R-ancestor of y iff x bears R to something, and whatever the Fs may be: y is one of them *if* x is one of them, and an entity z is one of them whenever one of them is R-related to z. The idea is that x is an ancestor of y iff y has every "hereditary property" that x "passed on;" where an individual, Eve, may pass on properties like *being human* or *being descended from or identical with Eve*.

compare ANCESTOR(_, _) and PARENT(_, _). One can also say that *n* is in the "predecessor series" up to and including *m* iff *n* precedes—i.e., is ancestral predecessor of—or is identical to *m*.

By representing the contents of PRECEDES(_, _) and ZERO in terms of logical notions and concepts used in (HP), Frege licenced replacement of PRECEDES(_, _) and ZERO with complex concepts that manifest logical relations to $^{<e, e>}$NUMBER-OF(_). He could then define the numbers as those things preceded by, or identical to, the number of nonselfidentical things.

NUMBER(_) ≡ OR[=(ZERO, _), PRECEDES(ZERO, _)]

This showed how a mind furnished with the concepts needed to formulate (HP) could use certain logical capacities—including a capacity to introduce new concepts via logical notions like *ANCESTRAL*-R(_, _)—to reconstruct our concept NUMBER(_), along with related arithmetic concepts, in a way that would permit agreement on the Dedekind-Peano generalizations.

Initially, *we* might view $^{<e, e>}$NUMBER-OF(_) as mental shorthand: *n* is the number of s iff *n* is a number such that the numbers preceding *n* correspond one-to-one with the elements of s. (Think of counting the elements of a set, starting with zero.) This takes our concept NUMBER(_), or perhaps the corresponding Aristotelian predicate, as given. But for Frege, the point is not to reduce $^{<e, e>}$NUMBER-OF(_) to more familiar notions, or to argue that we always had the structured concept OR[=(ZERO, _), PRECEDES(ZERO, _)].[8] The idea is rather that *given* certain "fruitiful" concepts, which we may initially find unfamiliar, an ideal thinker could do arithmetic by introducing NUMBER(_) as an atomic but not primitive monadic concept.

One can say that for us, as opposed to ideal thinkers who might think about our concepts, Frege's definitions are not analytic. Still, his definitions reflect a serious attempt to say what numbers *are*—and how we *can* think of them—in a way that reflects our capacity to appreciate the compellingness of certain inductive inferences.[9] The idealized lines of thought are available to humans, although introducing the requisite concepts takes work and ingenuity. One must, of course, entertain a thought—e.g. that every number precedes another—before offering a Fregean analysis of its content. But we can use concepts like NUMBER(_) and PRECEDES(_, _) to think, and then introduce technical analogs of these concepts, with the aim of re-presenting our initial thought contents in ways that let us describe certain inferences as valid. Then we can replace NUMBER(_) with OR[=(ZERO, _), PRECEDES(ZERO, _)]—or a more structured concept—when this is useful in a proof, and go back to NUMBER(_) when the further structure is irrelevant.

Given a fine-grained notion of content, or thought-*equivalence*, analysis may be impossible or paradoxical. But Frege employed at least two notions of content: one based on his notion of sense (Sinn), and another according to which thoughts are equivalent if each follows from the other. Given latter notion of logical equivalence, or Lewis' (1975, 198x) characterization of contents as sets of logically possible worlds, one can say that our extant ways of representing are not yet perspicuous. We can use our current concepts to ask questions that lead us to *reformulate* the questions in ways that allow for interesting answers. Correlatively, analysis can be a creative activity that need not reflect the structure of the representations we initially use to entertain analysanda; see Horty (2007) for further discussion.

As I have been stressing, Frege introduced higher-order polyadic analogs of monadic concepts. In this respect, my project is the converse of his. Frege invented logically interesting

---

[8] Much less OR[=($^{<e, e>}$NUMBER-OF({X:~=(X, X)}, _), *ANCESTRAL*-PREDECESSOR($^{<e, e>}$NUMBER-OF({X:~=(X, X)}), _)].

[9] One can stipulate that definitions, if such there be, meet demanding conditions not met by dictionary entries or Frege's descriptions of numbers. But I suspect that Frege (like Aristotle) was more interested in defining things—as opposed to words—and refining pre-theoretic concepts, in order to better track theoretically important aspects of the things we think about.

concepts, and treated monadicity as a special case of relationality to truth (see §4.2), as part of a project in logic that prescinds from many details of human psychology. I think humans naturally use concepts of various adicities to introduce logically boring monadic analogs. But I adopt Frege's idea that extant concepts can be used to introduce formally new ones, and that introducing type-distinct analogs of extant concepts can be useful for certain derivational purposes. Frege "unpacked" monadic concepts like NUMBER(_), in ways that let him exploit the power of his sophisticated polyadic logic to derive arithmetic axioms from (HP). I am suggesting that human languages let us ignore adicity distinctions like those exhibited by CHASE(_, _) and GIVE(_, _, _)—in favor of monadic concepts like CHASE(_) and GIVE(_), which can be combined in simple ways that allow for simple inferences like conjunction reduction. But the big idea, which I am applying to the study of human languages in a certain way, is from Frege: languages are not mere tools for expressing available concepts; they can be used to introduce formally new concepts that are useful given certain computational capacities and limitations. This is the main reason I have dwelt so long on Frege's project. For while the idea of concept introduction was important for Frege, it is not the part of Frege's work that semanticists typically draw on.

1.5 Up the Hierarchy

It is also worth noting that Frege's project requires that we mortals be able to understand *notation* like '*ANCESTRAL*-PREDECESSOR(_, _)'. This does require an explicit semantics, of the sort provided by Tarski and Church, that specifies the intended interpretation of the initially alien formalism. Concepts like *ANCESTRAL*-PREDECESSOR($^{<e, e>}$NUMBER-OF($\{X: X \neq X\}$), _) are supposed to be representations with which an ideal thinker—one who does *not* define these concepts in terms of our logically inferior concepts—can think. So *we* need a clear characterization of what these representations represent. Put another way, the Bedeutungen for atomic ideal concepts must be specified objectively, in terms of concept-independent entities; and the Bedeutungen for complex ideal concepts must be specified recursively, in a way that human thinkers can follow.

Frege came close to providing enough. He had, as we might now put it, the idea of a truth conditional semantics for an ideal language: a procedure for interpreting, or at least specifying the Bedeutungen of, expressions of a possible language of idealized human thought; cp. Wittgenstein (1921). But it was not unproblematic to presuppose unsaturated concepts that indicate functions which map entities (indicated by saturating concepts) onto truth values, especially given conjunctions and negations of polyadic concepts in the scope of multiple quantifiers. So a stipulated semantics for his invented syntax can be valuable without being a model for human languages. It is one thing to note that logicians need to say what their formalism means, in terms of entities that are independent of our natural concepts, in order to describe the sparest bundle of concepts that an ideal mind could start with and still construct "logical analogs" of ours. It is quite another thing to hypothesize that children acquire languages that have a semantics in the sense that Frege envisioned; and this was no part of Frege's view.

A human language may not have an objective semantics in this sense. History suggests that it takes lots of insight and intellectual work to *invent* formalism that lets us represent *some* truth conditions systematically. I think it would be stunning to discover that children naturally acquire languages that are already suitable for representing the truth conditions Frege wanted to get at, as if the only real value of his Begriffsschrift is that it was better than German for purposes of conducting proofs. But in any case, even if each Begriffsschrift sentence specifies a truth condition that is shared with some expression of a human I-language, it is worth being explicit about the hierarchy of *atomic* expression-types that Frege's invented language permits. This highlights both (i) the importance of specifying interpretations for invented expressions that

are not instances of the basic types <e> or <**t**>, and (ii) a respect in which the space of potential atomic Fregean expressions seems to be vastly larger than the space of potential lexical items of a human I-language.

As already noted, expressions of type <τ> can be used to introduce expressions of type <τ, τ>. But this is just a special case of the point that given expressions of types <α> and <β>, one can introduce expressions of type <α, β>. So there are *many* Fregean types. For simplicity, let's assume exactly two truth values, ⊤ and ⊥; where these may be entities of type <e>, perhaps one and zero, chosen at random. The two basic types, <e> and <**t**>, can be viewed as constituting the basic "level"—Level Zero—of a hierarchy whose "next" level includes four types: <e, e>; <e, **t**>; <**t**, e>; and <**t**, **t**>; where each of these types corresponds to a class of functions from entities of some Level Zero sort (numbers or truth values) to entities of some Level Zero sort.

Put another way, Level Zero is exhausted by the two types of kind <0>, and Level One is exhausted by the four types of kind <0, 0>. At the next level of this hierarchy, Level Two, there are the new types that can be formed from those at the lower levels (Zero and One):

eight <0, 1> types, including <e, <e, **t**>> and <**t**, <**t**, e>>;
eight <1, 0> types, including <<e, e>, e>> and <<**t**, **t**>, **t**>; and
sixteen <1, 1> types, including <<e, e>, <e, e>> and <<**t**, **t**>, <**t**, **t**>>.

So at Level Two, there are thirty-two types, each corresponding to a class of functions.[10] At the next level, Level Three, there are the 1408 new types that can be formed from those at the lower levels (Zero, One, and Two):

sixty-four <0, 2> types, including <e, <e, <e, **t**>>>;
sixty-four <2, 0> types, including <<e, <e, **t**>>, **t**>;
one-hundred-and-twenty-eight <1, 2> types, including <<e, **t**>, <<e, t>, **t**>>;
one-hundred-and-twenty-eight <2, 1> types, including <<e, <e, **t**>>, <e, **t**>>; and
one-thousand-and-twenty-four <2, 2> types, including <<e, <e, **t**>>, <e, <e, **t**>>>.

At Level Four, there are more than two million types: <e, <e, <e, <e, **t**>>> and 5631 more examples of type <0, 3> or <3, 0>; 11,264 of type <1, 3> or <3, 1>; 90,112 of type <2, 3> or <3, 2>; and 1,982,464 of type <3, 3>. Let's not worry about Level Five, at which there are more than 5 x 10^{12} types. It is, however, worth thinking about Levels Three and Four.

As noted above, Frege's concept of ancestral is of type <<e, e**t**>, <e, e**t**>>. This type is at Level Three of the hierarchy, along with more than 1400 others. For each of these types, one can introduce boundlessly many concepts of that type, just as one can introduce *ANCESTRAL*-R(_, _) and boundlessly many other concepts of type <<e, e**t**>, <e, e**t**>>. One can imagine minds that think in Fregean terms, find his encoding of arithmetic perfectly natural, are able to introduce concepts of any Level Three type, but suffer from performance limitations (e.g., concerning memory) that keep them from using concepts at Level Four or higher. Such thinkers might appreciate that given concepts of type <e, <e, et>>, there are concepts of the Level Four type <<e, <e, e**t**>>,<e, <e, e**t**>>>. But this appreciation might be purely theoretical.

One can similarly hypothesize that while human I-languages generate expressions of every type <α, β> such that <α> and <β> are types, performance limitations keep us from introducing lexical items of types that go beyond those exhibited in human languages. Even if we can appreciate that induction requires going up the hierarchy in the way that Frege describes, grasping this theoretical point may not allow us to introduce concepts of the Level Three type <<e, e**t**>, <e, e**t**>>—even if we can use certain quantifiers and verbs to express concepts of the

[10] Compare iterative conceptions of the Zermelo-Frankl sets; see e.g. Boolos (1998).

Level Three types <<e, **t**>, <<e, **t**>, **t**>> and <e, <e, e**t**>>. But this raises the question of how to describe the alleged performance limitations, along with the types we do use, in a way that yields a plausible account of why certain types are beyond us linguistically though not conceptually; cp. Chomsky's (1965) discussion of "explanatory adequacy." My suggestion will be that we get a better overall account by imposing a severe limitation on the space of available types, instead of starting with the entire Fregean hierarchy and trying to filter out boundlessly many types.

2. Tarski's Satisfaction

There are many ways of presenting the essentials of Tarski's (1933) semantics for a first-order predicate calculus. Since my aims concern human languages and thought, as opposed to logic *per se*, it will be useful to describe a possible language of thought, PL, that has a Tarskian semantics; cp. Field (197x). And let's think of PL as a procedure (an I-language), which generates endlessly many mental sentences from finitely many elements, though without assuming that any actual thinker (apart from Tarski) ever implemented this idealized procedure without special training. As we'll see, endlessly many sentences of PL are neither true nor false, and the semantics for PL does not specify denotations/Bedeutungen. Still, PL has an objective semantics. Sentences of PL have a kind of significance that is recursively characterized in terms of satisfaction by sequences of entities. And while this significance is not compositional in the way that concepts are, we can define truth for PL in terms of satisfaction. Given this model language, we can go on to ask which additions/substractions yield better models of human languages.

2.1 A Simple Mentalese

The elements of PL include a variable 'X', a variable-extender '′', and finitely many unsaturated concepts of the sort discussed in chapter one: COW(_) and PLANET(_); ABOVE(_, _) and CHASE(_,_); GIVE(_, _, _) and BETWEEN(_, _, _); etc. If *n* is a variable, so is the result of affixing '′' to *n*. More briefly, and without quote marks: if *n* is a variable, so is *n*′. Hence, there are endlessly many variables of PL: X, X ′, X′′, etc. For each unsaturated concept, the result of replacing each of its slots with a variable is a sentence of PL. So endlessly many generable sentences—including COW(X), COW(X′), etc.—have no sentential constituents. As discussed in §2.2 of chapter one, a mind might be able to form ABOVE(BESSIE, VENUS) *without* being able to form ABOVE(BESSIE, _). But we are now imagining a mind that can use the concept ABOVE(_, _) to form mental sentences like ABOVE(X′, X) and ABOVE(X, X′), along with ABOVE(X, X) and endlessly many others that employ different variables.

Each unsaturated element (or thought frame) of PL is thus the progenitor of an unbounded family of sentences that have no sentential constituents. But the elemental concepts are not themselves sentences of PL. One can think of basic sentence-formation as a process in which variables are inserted into all the slots of a unsaturated concept. So while GIVE(_, _, _) and GIVE(X, _, X′) are not sentences of PL, GIVE(X′′, X′, X) and GIVE(X′′, X′′′′, X′) are. The elements of PL also include concepts like BESSIE and VENUS. And for each sentence S of PL, the result of replacing an occurrence of a variable in S with a denoter is another sentence. So the sentences of PL include: COW(BESSIE); ABOVE(X, BESSIE) and ABOVE(VENUS, X); ABOVE(VENUS, BESSIE); etc. We can thus retain the idea that unsaturated concepts can be converted into simple mental sentences by filling slots with singular concepts. But GIVE(X, _, BESSIE) is not a sentence of PL. And unlike saturation, merely inserting a variable does not alter adicity: ABOVE(X′, X) is as dyadic as ABOVE(_, _).

The sentences ABOVE(VENUS, X) and ABOVE(X, BESSIE) are analogs of the partially saturated thought-frames ABOVE(VENUS, _) and ABOVE(_ , BESSIE). But the sentences *share a variable* that reappears in GIVE(X′, X, X′′). By contrast, ABOVE(VENUS, _) and ABOVE(_ , BESSIE)

and GIVE(VENUS, _, BESSIE) are merely similar in having an unsaturated slot. Even if a mind can form these complex monadic concepts, it might be unable to link them. We can—and just did—imagine a process of modifying mere unsaturated slots, so that they admit variables as well as saturating concepts, and then adding a device that generates variables. But since mere slots are not yet variables, we should distinguish ABOVE(VENUS, _) from ABOVE(VENUS, X).[11]

The complex sentences of PL include negations and conjunctions: if S is a sentence, so is ¬S; if S and S* are sentences, so is [S & S*]. Here, I use the usual ampersand, not '!' as in chapter one. As a generable *sentence*, [COW(X) & ¬ABOVE(X′, X″)] is unobjectionable. It is easy to see how combining two formal objects can create a third that has more variables than either input. Likewise, combining a sentence that has one variable with a sentence that has two variables can yield a sentence that has two variables, one of which has two occurrences as in [COW(X) & ABOVE(X, X′)]. The trick is to show how instances of ¬S and [S & S*] can have the intended negative/conjunctive *interpretations*, no matter how many variables are included, in a way that will also allow for quantificational sentences like ∃x′[COW(X) & ¬ABOVE(X′, X″)].

In chapter one, a recurrent question was *how* complex concepts like ABOVE(VENUS, _) can be assembled from atomic concepts. If ABOVE(VENUS, _) is the result of extracting a denoter from a thought, one wants to know why the result is a *concept* as opposed to formal junk. Variable insertion is part of a potential answer: ABOVE(_, _) can be used to build sentences like ABOVE(X′, X). But one cannot form a dyadic concept by putting a dime into the first slot of ABOVE(_, _), a penny into its second slot, and then combining the result with the result of inserting another dime into the slot of COW(_). We can stipulate that variables differ from coins and many other things in this respect. Though even if we assume that [ABOVE(X′, X) & COW(X)] is a dyadic sentence, one wants to know what it signifies, and why it can apply to pairs of things.

2.2 Interpreting Complexity via Sequences

Tarski offered an inspired strategy for assigning interpretations to the sentences of a language like PL. As with the initial discussion of Frege's typology for concepts, I want to go slowly, since it will be important to distinguish two uses of Tarski's notion of satisfaction: providing an objective semantics for an invented language, and thereby characterizing a notion of truth that is important for logic as a normative enterprise; and offering empirical hypotheses about human concepts, words, and modes of composition. (For readers who want to skip or skim this review, let me note that the last two paragraphs of this subsection are important for what follows.)

The leading idea is that the variables of PL range over entities that can be ordered as *sequences* that can satisfy sentences. This does not require that elements of PL denote entities or functions. But it does require entities that can be ordered in a way that corresponds to the generation of variables. As an obvious illustration, if we want to use PL as an invented language for reasoning about sets, we might stipulate a domain of pure sets: ∅, {∅}, {{∅}}, etc. If we are studying animals who seem to use PL as a natural mentalese, the relevant domain is less clear. Is

[11] If any slot can be filled by any variable, and any two sentences can be conjoined, this yields a strong form of *systematic* composability; cp. Evans (1982), Fodor and Pylylsyhn (1988). Given "universal" variables that can be freely replaced with singular concepts, each potential saturater of Φ(_) is a potential saturater of $(_)—even if Φ(_) and $(_) originally required saturaters of different sorts, say because these concepts are rooted in different cognitive modules. But a thinker might have various kinds of variables for various kinds of concepts, and some concepts may not interface with just any variable-inserter, leaving pockets of concepts that are not integrated with others. So one might imagine variants of PL that allow for diverse sentences like SOUND(∇), SURFACE(ζ), CAUSE(α′, α), and BETWEEN(X″, X′, X). I'll return to these points. As suggested in chapter one, I think human concepts are as integrated as they are in part *because* humans enjoy special linguistic capacities; cp. Spelke (2002, 2003), Carruthers (2002).

it the entire world, as a totality of things? Does the domain include all and only the entities countenanced by correct scientific theories? Does it include cabbages, kings, kingdoms, and things that are "commonsense" entities for humans? Is the relevant domain limited to things, or perhaps concept-independent things, about which the animals under study can think? For now, let's set these questions aside, and take the notion of a domain entity (henceforth, d-entity) to be clear enough for purposes of characterizing the notion of a sequence.

A sequence is an ordering of the d-entities. The variables of PL are ordered like natural numbers: X, X′, X″, *etc*. So we can think of sequences as mappings from variables to d-entities. For each sequence σ and any variable *n*, σ(*n*) is a d-entity: σ(X) is the first d-entity of σ, σ(X′) is the second, etc. Since there are boundlessly many variables of PL, even if there are only finitely many d-entities, each d-entity can appear boundlessly many times in each sequence. Crucially, sequences are also plentiful in a technical sense. For each sequence σ, variable *n*, and d-entity Δ, there is a sequence σ* such that: σ*(*n*) = Δ, and for every other variable *m*, σ*(*m*) = σ(*m*).

Consider a particular sequence, $^{\text{BVS}}\sigma$, whose first three positions are occupied by (respectively) Bessie, Venus, and Sadie. This sequence, along with endlessly many others, is of the form <Bessie, Venus, Sadie, …>. Assume that Bessie is a cow, Venus is not a cow, and Venus is above Bessie. Then $^{\text{BVS}}\sigma(\text{X})$ is a cow, and $^{\text{BVS}}\sigma(\text{X}')$ is above $^{\text{BVS}}\sigma(\text{X})$. However, $^{\text{BVS}}\sigma(\text{X}')$ is not a cow, and $^{\text{BVS}}\sigma(\text{X})$ is not above $^{\text{BVS}}\sigma(\text{X}')$. Now consider the sequence $^{\text{VBS}}\sigma$ that is just like $^{\text{BVS}}\sigma$ except that first two entities are reversed: $^{\text{VBS}}\sigma(\text{X}) = {}^{\text{BVS}}\sigma(\text{X}')$ = Venus; and $^{\text{VBS}}\sigma(\text{X}') = {}^{\text{BVS}}\sigma(\text{X})$ = Bessie.[12] Then $^{\text{VBS}}\sigma$ is of the form <Venus, Bessie, Sadie, …>. So $^{\text{VBS}}\sigma(\text{X})$ is not a cow, and $^{\text{VBS}}\sigma(\text{X}')$ is not above $^{\text{VBS}}\sigma(\text{X})$; but $^{\text{VBS}}\sigma(\text{X}')$ is a cow, and $^{\text{VBS}}\sigma(\text{X})$ is above $^{\text{VBS}}\sigma(\text{X}')$. If Venus is also above Sadie, then $^{\text{VBS}}\sigma(\text{X})$ is above $^{\text{VBS}}\sigma(\text{X}'')$, and $^{\text{BVS}}\sigma(\text{X}')$ is above $^{\text{BVS}}\sigma(\text{X}'')$.

Given the notion of a sequence, we can say that many sentences of PL are *satisfied by* many sequences. In particular, we can say that for any variable *n*, a sequence σ satisfies COW(*n*) iff COW(_) applies to σ(*n*). Though for simplicity, let's just say that σ satisfies COW(*n*) iff σ(*n*) is a cow. Then $^{\text{BVS}}\sigma$ satisfies COW(X) iff $^{\text{BVS}}\sigma(\text{X})$—a.k.a. Bessie—is a cow. So $^{\text{BVS}}\sigma$ satisfies COW(X). But this sentence of PL is not satisfied by $^{\text{VBS}}\sigma$, since $^{\text{VBS}}\sigma(\text{X})$—a.k.a. Venus—is not a cow. Though $^{\text{VBS}}\sigma$ does satisfy COW(X′), since $^{\text{VBS}}\sigma(\text{X}')$ is Bessie.

Similarly, let's say that for any variables *n* and *m*, σ satisfies ABOVE(*n*, *m*) iff σ(*n*) is above σ(*m*). Then $^{\text{BVS}}\sigma$ satisfies ABOVE(X′, X) iff $^{\text{BVS}}\sigma(\text{X}')$ is above $^{\text{BVE}}\sigma(\text{X})$. Since Venus is above Bessie, $^{\text{BVS}}\sigma$ satisfies ABOVE(X′, X); but this sentence of PL is not satisfied by $^{\text{VBS}}\sigma$. By contrast, $^{\text{VBS}}\sigma$ satisfies the distinct sentence ABOVE(X, X′); though $^{\text{BVS}}\sigma$ does not satisfy ABOVE(X, X′). Just as COW(X) differs from COW(X′), ABOVE(X′, X) differs from ABOVE(X, X′).

Likewise, [COW(X) & ABOVE(X′, X)] differs from both [COW(X′) & ABOVE(X′, X)] and [COW(X) & ABOVE(X, X′)]. This makes it easy to specify a sense in which instances of [S & S*] are conjunctions: a sequence σ satisfies a sentence of this form iff σ satisfies both S and S*. So in particular, $^{\text{BVS}}\sigma$ satisfies [COW(X) & ABOVE(X′, X)], since $^{\text{BVS}}\sigma(\text{X})$ is a cow *and* $^{\text{BVS}}\sigma(\text{X}')$ is above $^{\text{BVS}}\sigma(\text{X})$. But neither $^{\text{BVS}}\sigma$ nor $^{\text{VBS}}\sigma$ satisfies [COW(X′) & ABOVE(X′, X)], since $^{\text{BVS}}\sigma(\text{X}')$ is not a cow, and $^{\text{VBS}}\sigma(\text{X}')$ is not above $^{\text{VBS}}\sigma(\text{X})$. By contrast, $^{\text{VBS}}\sigma$ satisfies [COW(X′) & ABOVE(X, X′)].

Dealing with negation and sentences of arbitrary adicity is also easy. A sentence of the form ¬S is satisfied by σ iff it is not the case that σ satisfies S. For any variables *n*, *m*, and *k*, σ satisfies GAVE(*n*, *m*, *k*) iff σ(*n*) gave σ(*m*) to σ(*k*). So $^{\text{VBS}}\sigma$ satisfies ¬[GAVE(X, X′, X″) & COW(X′)]

[12] Since sequences are plentiful, $^{\text{BVS}}\sigma$ ensures a sequence just like it, except that Venus is the first element. And that sequence, $^{\text{VVS}}\sigma$, ensures a sequence just like it, except that Bessie is the second element.

iff it is not the case that $^{VBS}\sigma$ satisfies both GAVE(X, X′, X″) and COW(X′). That is, $^{VBS}\sigma$ satisfies the complex sentence iff it is not the case that: Venus gave Bessie to Sadie, and Bessie is a cow. Likewise, $^{VBS}\sigma$ satisfies [COW(X″) & ABOVE(X′, X)] iff Sadie is a cow and Bessie is above Venus. So $^{VBS}\sigma$ does not satisfy this complex sentence, but it does satisfy ¬[COW(X″) & ABOVE(X′, X)].

For any variable *n* and sequence σ, we can say that σ(*n*) is the value of *n* relative to σ. And if there are no "empty" singular concepts like VULCAN, we can say that each singular element of PL is a degenerate variable—a constant—whose value is same relative to each sequence. For each sequence σ: σ(BESSIE) = Bessie; σ(VENUS) = Venus; etc.[13] It follows that *some* sequence satisfies ABOVE(VENUS, BESSIE) iff *every* sequence satisfies this sentence of PL.

If any sequence σ is such that σ(VENUS) is above σ(BESSIE), then for each sequence σ*, σ*(VENUS) is above σ*(BESSIE). So if a sentence of PL has no inconstant variables, it is satisfied by each sequence or none. Thus, we can say that any such sentence is *true* iff it is satisfied by all sequences, and *false* iff it is satisfied by no sequence. This characterizes truth and falsity, for some sentences, in terms of the more basic and recursively characterized notion of satisfaction: σ satisfies ¬[COW(BESSIE) & ABOVE(VENUS, SADIE)] iff it is not the case that σ *satisfies* both COW(BESSIE) and ABOVE(VENUS, SADIE). This condition is satisfied by all sequences or none. So the negated sentence is false or true, depending on whether or not Bessie is a cow and Venus is above Sadie. But this truth condition is characterized in terms of the satisfaction condition for the negated sentence, not in terms of the falsity/truth of [COW(BESSIE) & ABOVE(VENUS, SADIE)], much less the truth/falsity of COW(BESSIE) and ABOVE(VENUS, SADIE).[14]

In PL, the symbols '¬' and '&' are not of types <**t**, **t**> and <**t**, <**t**, **t**>>. That is why the sentences of PL can include [COW(X″) & ¬ABOVE(X′, X)] and ¬[COW(X″) & ABOVE(VENUS, X)]. Sentential significance is specified in terms of sequences of d-entities, not in terms of truth. The variable-free sentences of PL do not denote (truth) values of functions indicated by predicative constituents given arguments indicated by constants. Sentences of PL specify conditions on ways of ordering d-entities. Indeed, each sentence is significant in and only in the following sense: it divides *the ways of ordering a domain* into those that meet a certain condition, and the rest.

In general, potential values of variables will not themselves determine whether they meet the condition imposed by [GIVE(X, X′, X″) & ¬COW(X′)]. Given three d-entities, they might meet the condition or not, depending on which one is assigned to which variable. But a *sequence* of variable values is just the sort of thing that can meet this condition or not, depending on whether or not: the first d-entity of the sequence gave the second to the third, and the second is not a cow. One can say that each sentence of PL is of type <*PLTF*>, for pre-linked thought-frame; see §4.2 of chapter one. One can add that '¬' and '&' are of the types <*PLTF*, *PLTF*> and <*PLTF*, <*PLTF*, *PLTF*>>. But whatever our notation, sequences are the *only* things that can satisfy a sentence of PL, regardless of how many or how few variables/constants the sentence includes.[15]

---

[13] For explicitness, order the constants of PL, and let each sequence σ have the corresponding (finite) *n*-tuple of d-entities as a "zeroth" element; cp. Larson and Segal's (1995) encoding Kaplan's (1989) treatment of indexicals.

[14] A true sentence can be conjoined with COW(X) to form a sentence that is satisfied by some but not all sequences, and so neither true nor false. Even for purposes of specifying truth, *satisfaction* conditions are what matter.

[15] One can say that ABOVE(X′, X″) is "tuplefied" by ordered pairs ⟨α, β⟩ such that for some sequence σ: σ(x′) = α, σ(x″) = β, and σ satisfies ABOVE(X′, X″); GIVE(X, X′, X″) is "tuplefied" by ordered triples; etc. For any sentence S of PL, some *n* is such that only the first *n* entities of a sequence σ will be relevant to whether or not σ satisfies S. But for any finite *n*, endlessly many sentences of PL have more than *n* variables. So sequences need to be unbounded, even given an upper bound on the adicities of atomic sentences.

2.3 Quantification via Sequences

Elementary quantification is easily introduced as follows: for any variable *n* and sentence S of **PL**, $\exists n$[S] is also a sentence of PL. Thus, $\exists$X[ABOVE(X′, X)] and $\exists$X[ABOVE(X, X′)] are sentences, as are $\exists$X″[GIVE(X, X″, X′)] and $\exists$X′[[COW(X) & ABOVE(X′, X)]]. Let's also permit elimination of doubled brackets, so that the last sentence can be respelled as $\exists$X′[COW(X) & ABOVE(X′, X)].

Like all sentences of PL, these new examples specify conditions on sequences. But for each sequence σ, a sentence of the form $\exists n$[S] imposes a weak condition on σ: *some* sequence that differs minimally from σ, in the respect specified by *n*, meets the condition imposed by S. Tarski's notion of a sequence variant, implicit in the plentitude of sequences, captures the relevant notion of minimal difference. For any sequence σ and variable *n*, a sequence σ* is an *n*-variant of σ—more briefly, $\sigma^* \approx_n \sigma$—iff σ* differs from σ *at most* with respect to the value of *n*; where this means that either σ* does not differ at all from σ (i.e., σ* = σ), or σ* differs from σ only in the *n*th position. The relation indicated with '$\sigma^* \approx_n \sigma$' is reflexive, symmetric, and transitive. But note that ${}^{BVS}\sigma$ differs from ${}^{VBS}\sigma$ in *two* respects: ${}^{BVS}\sigma(X) \neq {}^{VBS}\sigma(X)$, and ${}^{BVS}\sigma(X') \neq {}^{VBS}\sigma(X')$. So ${}^{BVS}\sigma$ is neither an X-variant nor an X′-variant of ${}^{VBS}\sigma$. If ${}^{SVS}\sigma$ is just like ${}^{BVS}\sigma$ except that ${}^{SVS}\sigma(X)$ is Sadie the horse, then ${}^{SVS}\sigma \approx_X \sigma^{BVS}$; but these X-variants are not X′-variants. Sequences alike at every position except the *n*th are *n*-variants. But if *n* and *m* are distinct variables, sequences that differ at the *n*th position cannot be *m*-variants.

Sequence σ satisfies a sentence of the form $\exists n$[S] iff some *n*-variant of σ satisfies S. Put another way, σ satisfies $\exists n$[S] iff some sequence σ* such that $\sigma^* \approx_n \sigma$ is such that σ* satisfies S. This treats '$\exists n$' as a kind of quantification over sequences. For example, σ satisfies $\exists$X[COW(X)] iff some X-variant of σ satisfies COW(X). Hence, ${}^{SVS}\sigma$ satisfies $\exists$X[COW(X)], even though ${}^{SVS}\sigma(X)$ is Sadie the horse. For at least one sequence—${}^{BVS}\sigma$ is an example—is such that it is an X-variant of ${}^{SVS}\sigma$, and its first element is cow. Obviously, ${}^{BVS}\sigma$ also satisfies $\exists$X[COW(X)]; since ${}^{BVS}\sigma$ itself satisfies COW(X), and each sequence is an X-variant of itself. And so long as there is a cow in the domain, there will be many sequences whose first element is a cow; in which case, every sequence will satisfy $\exists$X[COW(X)], even if endlessly many sequences do not satisfy COW(X). But if there is no cow in the domain, no sequence will have a cow as its first element. In which case, no sequence will satisfy COW(X), and so no sequence will satisfy $\exists$X[COW(X)]. Thus, $\exists$X[COW(X)] is satisfied by all sequences or none, depending on whether or not there is a cow in the domain.

Likewise, σ satisfies $\exists$X[ABOVE(X′, X) & COW(X)] iff some sequence σ* such that $\sigma^* \approx_X \sigma$ is such that: σ* satisfies [ABOVE(X′, X) & COW(X)]; i.e., σ* satisfies ABOVE(X′, X), and σ* satisfies COW(X); i.e., σ*(X′) is above σ*(X), and σ*(X) is a cow. One can paraphrase this condition on sequences given two facts: if $\sigma^* \approx_X \sigma$, then σ*(X′) = σ(X′); and if σ*(X) is a cow, then being above σ*(X) is being above a cow. So σ satisfies $\exists$X[ABOVE(X′, X) & COW(X)] iff σ(X′) is above a cow. Thus, despite the asterisks and squiggles in many specifications of this condition on sequences, the condition is simple: the second sequence element is above a cow.

Let's abbreviate instances of '_ satisfies S iff ...' as instances of '*Sat*{_, S} ⇔ ...', and abbreviate instances of 'sequence _ such that ...' with instances of '_: ... ', as shown below.

Sat{σ, $\exists$X′[$\exists$X[ABOVE(X′, X) & COW(X)]]} ⇔ some σ^*:$\sigma^* \approx_{X'} \sigma$ is such that *Sat*{σ*, $\exists$X[ABOVE(X′, X) & COW(X)]}

Repeating the reasoning above, without the paraphrase, yields the following biconditional.

Sat{σ*, $\exists$X[ABOVE(X′, X) & COW(X)]} ⇔ some σ**:$\sigma^{**} \approx_X \sigma^*$ is such that σ**(X′) is above σ**(X), and σ**(X) is a cow

So the following sequence specification condition can be derived, given that '⇔' is transitive.[16]

$Sat\{\sigma, \exists X'[\exists X[\text{ABOVE}(X', X) \,\&\, \text{COW}(X)]]\} \Leftrightarrow$ some $\sigma^*{:}\sigma^* \approx_{X'} \sigma$ is such that some $\sigma^{**}{:}\sigma^{**} \approx_{X} \sigma$ is such that $\sigma^{**}(X')$ is above $\sigma^{**}(X)$, and $\sigma^{**}(X)$ is a cow.

That is, σ satisfies $\exists X'[\exists X[\text{ABOVE}(X', X) \,\&\, \text{COW}(X)]]$ iff some d-entity (that is the first element of some X′-variant of σ) is a above a cow. And this condition is satisfied by all sequences or none.

In chapter four, I return to the importance of *deriving*—as opposed to merely formulating and paraphrasing—specifications of conditions on sequences. But to foreshadow, and to take a break from mere technicalia: while Tarski offered his semantics as an algorithm that specifies a certain relation that invented expressions bear to an idealized domain of entities, one can imagine a mind that already enjoys a *richer* language of thought than PL. In particular, imagine a mind that can conduct derivations like those conducted in a Tarskian *metalanguage*. Such a mind would not merely generate sentences of PL. It would be able to represent such sentences as having satisfaction conditions by using mental analogs of '$Sat\{_, S\} \Leftrightarrow \ldots$', sequence variables, '$\sigma^*{:}\sigma^* \approx_n \sigma$', and other apparatus required to derive specifications of conditions on sequences.

One can hypothesize that competent speakers of a human language **H** likewise represent satisfaction conditions for sentences of **H**. Then while each true sentence would bear the same semantic relation to the domain—being satisfied by all sequences—true sentences could still be distinguished by virtue of being derivationally paired with distinct *specifications of* conditions on sequences. In this sense, true sentences could differ in meaning, even if the domain does not include sentential meanings that true sentences have. Initially, it might seem crazy to suggest that ordinary speakers tacitly conduct Tarski-style derivations. But it may be worse to deny this hypothesis, yet still follow Davidson (1967a) in supposing that English has a Tarski-style semantics. One wants know else sentences of a human language **H** could *be* satisfied by sequences if competent speakers/signers of **H** *don't* represent the relevant satisfaction conditions.

A lucky thinker might have concepts that are related to d-entities (and to each other) in the right way, so that the thinker enjoys a language of thought (e.g., PL) that has a Tarskian semantics that the thinker does not represent. But pronounceable expressions are presumably related to d-entities indirectly, via uses that are mediated by concepts; cp. Chomsky (195x). So even if some concepts have satisfaction conditions that go unrepresented, it is hard to see how words could have satisfaction conditions except via mentalistic specifications.[17] In the end, I reject the hypothesis that ordinary speakers "internalize Tarski," but not because I think children could not have the requisite mental language and use it as a meta-language. Animal minds are impressive; see, e.g., Gallistel (200x) on insect cognition. I have doubts about appealing to mental analogs of '$Sat\{_, S\}$'. But given the quantificational thoughts that ordinary humans enjoy, it is reasonable to posit mental analogs of quantification over sequence variants. So I take "psychologized" versions of Tarskian semantics (e.g., Larson and Segal [1995]) seriously; see chapter four.

At any rate, one can view *derivations* of specifications of conditions on sequences—derivations of "Sat-sentences"—as models of how a mind equipped with classical semantic

[16] See Larson and Segal (1995). Any system of derivation will require a version of 'P iff Q, and Q iff R, so P iff R', the sentential analog of 'a = b and b = c, so a = c'. But while the Fregean biconditional '≡' must be flanked by truth value denoters, the double arrow used here can be flanked by Tarskian sentences.

[17] Davidson famously appealed to a notion of idealized interpretation as an alleged sufficient condition: a speaker's word 'cow' is true of cows if her behavior would lead a certain kind of interpreter to adopt a theory that includes this hypothesis. But there is little reason for thinking that this condition is sufficient or ever met by actual humans.

concepts might use an extant mental language to generate (i) hypotheses regarding the satisfaction conditions of foreign sentences, or (ii) stipulations regarding the satisfaction conditions of sentences generated by a new procedure implemented by that very mind. At least in case (ii), the new sentences could be articulable devices of communication, as opposed to instructions for how to build sentences of the older mental language. So again, I take this view seriously. And note that thinkers might agree on which sequences satisfy COW(X), even if they think about cows differently. The new sentences—e.g., ∃X′[∃X[ABOVE(X′, X) & COW(X)]]—would impose conditions on sequences indirectly, by virtue of being paired with mental analogs of 'some σ^*:$\sigma^* \approx_{X'} \sigma$ is such that some σ^{**}:$\sigma^{**} \approx_X \sigma$ is such that σ^{**}(X′) is above σ^{**}(X), and σ^{**}(X) is a cow'. But depending on one's goals, this might be perfectly acceptable; cp. Katz and Fodor (1963), Lewis (1972), Dummett (1976).

I'll return to these issues. But let me get back to a few last technical details regarding PL. Given negation and the existential quantifer introduced above, we can define a universal quantifier. Or we can say that for any sequence σ, variable *n*, and sentence S of PL: σ satisfies ∀*n*S iff *every n*-variant of σ satisfies S. That is, σ satisfies ∀*n*[S] iff every σ^*:$\sigma^* \approx_n \sigma$ is such that σ^* satisfies S. For example, *Sat*{σ, ∀X[COW(X)]} ⇔ every σ^*:$\sigma^* \approx_X \sigma$ is such that *Sat*{σ^*, COW(X)}. So $^{BVS}\sigma$ does not satisfy ∀X[COW(X)], even though $^{BVS}\sigma$(X) is a cow. If even one d-entity is not a cow, then each sequence has an X-variant that does not satisfy COW(X), and no sequence satisfies ∀X[COW(X)]. But if everything in the domain is a cow, each (X-variant of) each sequence satisfies COW(X). So ∀X[COW(X)] is satisfied by all sequences or none, depending on whether or not each d-entity is a cow. Likewise, ∀X¬[COW(X) & ¬BROWN(X)]—and the equivalent ∀X[COW(X) ⊃ BROWN(X)]—is satisfied by all sequences or none, depending on whether or not each d-entity fails to be both a cow and not brown.

Sentences with both quantifiers, as in ∀X′[∃X[ABOVE(X′, X) & COW(X)]], are already accommodated by the interpretive rules for instances of ∀*n*S and ∃*n*S.

Sat{σ, ∀X′[∃X[ABOVE(X′, X) & COW(X)]]} ⇔ every σ^*:$\sigma^* \approx_{X'} \sigma$ is such that *Sat*{σ^*, ∃X[ABOVE(X′, X) & COW(X)]}

Since we just specified what it is for σ^* to satisfy ∃X[ABOVE(X′, X) & COW(X)], the following sequence specification condition can be derived.

Sat{σ, ∀X′[∃X[ABOVE(X′, X) & COW(X)]]} ⇔ every σ^*:$\sigma^* \approx_{X'} \sigma$ is such that some σ^{**}:$\sigma^{**} \approx_X \sigma$ is such that σ^{**}(X′) is above σ^{**}(X), and σ^{**}(X) is a cow.

That is, σ satisfies ∀X′[∃X[ABOVE(X′, X) & COW(X)]] iff every X′-variant of σ is such that it has an X-variant whose second element is above its first, and its second element is a cow. Or more simply, a sequence satisfies this sentence of PL iff each d-entity is above a cow.

Instances of ∀*n*S impose a strong condition on sequences. If σ satisfies ∀*n*S, then σ satisfies S, and every other *n*-variant of σ satisfies S. Instances of ∃*n*S impose a weak condition: σ can satisfy ∃*n*S without satisfying S, so long as some *n*-variant of σ satisfies S. Yet in either case, if S has no inconstant variables apart from *n*, each of these conditions will be met by every sequence or none. So we can say that a sentence of PL is *true* iff it is satisfied by all sequences.[18] And since universal/existential quantification is construed in terms of strong/weak constraints on sequences, ∀X[Φ(X)] implies Φ(BESSIE), which implies ∃X[Φ(X)]; though not conversely.

[18] At least if the sentence has no free (i.e., inconstant and unbound) variables. It is a matter of decision whether we count open sentences like COW(X) as true given a domain that includes only cows.

2.4 Semantic Recursion without Semantic Composition

The significance of PL sentences is characterized recursively in terms of satisfaction. Given how truth is related to satisfaction, each closed sentence of PL is true or false, depending on whether or not some (and hence every) sequence meets a certain condition. In this sense, each closed sentence S of PL has a truth condition that can be specified with an instance of the following schema: True(S) ⇔ F(σ). But in general, the sequences that satisfy a sentence are not composed of sequences that satisfy parts of the sentence; cp. Salmon (200x). One can say that sets of sequences are "T-meanings," which do not compose. This is not a problem given Tarski's aims. If one just needs a method that *determines* interpretations for generable formulae, as opposed to a theory of some natural generative procedure, one may not even to need to specify the interpretations recursively in terms of constituents of the formulae; see Szabo (200x). But since expressions of PL are often viewed as models of human linguistic expressions, it is worth highlighting the distinction between being specifiable and being composite. For it may be that human languages are unlike PL in being less systemtically connected to d-entities, but significant in way that is more demandingly compositional.

A sequence can satisfy COW(X) without satisfying ∀X[COW(X)]. But the sequences that satisfy ∀X[COW(X)]—either all of them or none—are not composed of the sequences, if any, that satisfy COW(X); and since ∀X is not a sentence, it cannot be satisfied by any sequences. Perhaps even more obviously, the satisfiers of ∃X[COW(X)] are not built up from whatever sequences satisfy COW(X). A sequence can satisfy ∃X[COW(X)] without satisfying COW(X). We can say that the satisfiers of ∃X[COW(X)] are *determined* by the satisfiers of COW(X), given the rule for interpreting instances of ∃*n*S, and likewise ∀X[COW(X)]. This assumes that quantificational sentences of PL exhibit a kind of *syntactic* composition that corresponds to their recursive specification. In particular, one can view ∃X[COW(X)] and ∀X[COW(X)] as results of embedding the sentence COW(X) in the frames ∀X[…] and ∀X[…], thereby forming larger sentences that have COW(X) as a shared part. And *specifications* of conditions on sequences, mentalistic or orthographic, can be composite. Expressions of a metalanguage can exhibit part-whole relations that mirror syntactic composition in an object language. But sequences don't compose.[19]

Relatedly, embedding a Tarskian sentence in a sentential frame is *not* a notational variant of saturating a Fregean concept. The concept COW(_), which is not a sentence of PL, can be saturated by a concept of type <e>; and COW(_) can saturate the concept ∀[Φ(_)], which is of type <e**t**, **t**>. But ∀X is not a Fregean concept of any type. It is a kind of prefix that extends one sentence into another. That is why ∀X[ABOVE(X″, X′)] is a sentence with satisfiers—as opposed to a mere concatenation of two mismatched concepts, one of type <e**t**, **t**> and one of type <e, e**t**>. Likewise, ¬ is not of type <**t**, **t**> in PL. The satisfiers of ¬COW(X) are determined by the satisfiers of COW(X), given the interpretive rule for instances of ¬S. But the sequences that satisfy ¬COW(X) are not composed of those that satisfy COW(X); and ¬ cannot be satisfied by sequences. Attaching a prefix to COW(X) differs from saturating COW(_) or the prefix. Prefixing turns one predicate of sequences into another. Saturating reduces the adicity of a concept.

Similarly, Tarski's ampersand cannot be satisfied or saturated. From a Fregean perspective, this ampersand is not a concept. It is, at best, a syncategorematic device that can be used to form complex concepts. In this respect, & is like the variable extender ′. The satisfiers of

[19] Fregean Bedeutungen don't compose either. The truth value of an ideal thought, of the form Φ(α), is not built up from the function and entity indicated with Φ(_) and α. But Sinnen are supposed to exhibit part-whole relations, perhaps on analogy with interlocking pieces of a jigsaw puzzle, or valenced atoms that combine to form molecules.

[ABOVE(X″, X′) & COW(X)] are *determined* by the satisfiers of ABOVE(X″, X′) and COW(X), given the rule for interpreting instances of [S & S*]; and a conjunctive *sentence* is built up from its parts. But sentences of PL do not have meanings that are determined by, much less composed of, the meanings of subsentential expressions like ABOVE(_, _) and the ampersand. One can say that nonsentential symbols of PL have a kind of derived significance, as potential sentence parts. But in PL, not even BESSIE and COW(_) are formatives that have significance of their own. The significance of COW(BESSIE), as a sentence of PL, is related to the contents of the Fregean concepts BESSIE and COW(_); and likewise for [ABOVE(X″, BESSIE) & COW(X)]. But to repeat, there is no Fregean counterpart of the Tarskian ampersand. And in PL, satisfaction by sequences is the only kind of significance that expressions (i.e., sentences) can have.

I have been describing PL as a possible generative system whose atomic concepts, which exhibit a saturated/unsaturated distinction, can be used to form mental sentences that contain bindable variables. To show that the resulting formulae still count as concepts—representations with which thinkers can think, as opposed to mere results of jamming variables into slots for genuine saturaters—one must show how the sentences of PL can have a systematic kind of significance. Tarski provided the resources for doing this. So we can see how, at least in principle, concepts like ABOVE(_, _) and COW(_) could be used to form truth-evaluable representations like ∀X′[∃X[ABOVE(X′, X) & COW(X)]].

That was a major achievement. But crucially, ∀X′[∃X[ABOVE(X′, X) & COW(X)]] exhibits a kind of composition that differs from the kind exhibited by COW(BESSIE). And one might suspect that with respect to composition, meanings are more like concepts. But even setting aside questions about the sense(s) in which meaning is compositional, one might want to consider variants of PL that allow for certain kinds of subsentential significance; cp. Lewis on grammars, as discussed in §2.3 of chapter zero. Human languages generate expressions that seem not to have analogs in PL—e.g., adverbial modifiers, indexicals, demonstratives, that-clauses, plural noun phrases, and quantificational sentences whose interpretations cannot be captured in terms of Tarskian quantification over sequences. This last topic was especially important in the development of formalisms for describing meanings. So let me move ahead with the quasi-historical review.

3. Supplementing PL: Taking Tarski to Church

In this section, I review Church's (1941, 195x) extension of Tarski-style semantic theories. Church used his independently developed lambda calculus (see §… of chapter zero) to provide a semantics more like the one Frege envisioned for his Begriffsschrift, with many meaningful expressions having denotations or Bedeutungen. Church's invented language, unlike Tarski's, was also designed to permit representation of any computable function. So as Montague (1974) illustrated, Church's toolkit provides more resources for modeling human languages in truth-theoretic terms. But unsurprisingly, this highlights questions concerning interpretations that human linguistics expressions do not exhibit.

3.1 Truth Values, Entities, and Functions as Denotations

In thinking about how human languages apparently differ from PL, one might focus on expressions used in reference and quantification. A name like 'Bessie' seems somehow like a Fregean concept of type <e>, with which one can think about a particular thing, at least if one ignores the fact that individuals share names. Names can be modeled with variables whose values cannot vary. But this suggests a special kind of significance—viz., denoting. One might also think that real variables are for binding, and that phrases like 'every cow' are binders of the type <e**t**, **t**>. That raises questions, to which I return in chapter five, about 'every Bessie I know';

cp. Burge (197x). But given a categorematic conception of names and quantifiers, predicates presumably exhibit a kind of significance that sentences of PL do not. If 'is brown' can combine with expressions of type <e> or <e**t**, **t**> form a sentence, then 'is brown' is like BROWN(_) and COW(_) in being of type <e**t**>. Still, even if one posits unsaturated concepts, one might want to avoid Frege's appeal to undenotable unsaturateds. Church (1941, 195x) showed how to extend a Tarskian language, in a way that reflects these considerations, by introducing combinable denoters of various types.

I will not, however, follow Church's presentation in detail. My aim is to show how idealized minds might *introduce* a language of thought that has a Church-style semantics, given PL and a few additional concepts. For one might think children have or acquire a mental language that lets them specify denotations (a.k.a. "semantic values") for all meaningful expressions, as opposed to merely specifying satisfaction conditions for sentences (open or closed). Indeed, this is tempting construal of much recent work in semantics. I also want to describe this extended language—call it CPL—in a way that highlights contrasts with PL, while noting that CPL is a Fregean language that avoids the difficulties discussed in section two. This will serve as a model for how human minds might come to have a less powerful language of thought (SMPL), described in chapters five and six. Specialists can keep skimming. But it is easy to forget how Church-style denotation of Fregean functions (in extension) is related to Tarski's notion of satisfaction. And later, it will be important to have been clear about this.

Like PL, CPL is an idealized generative procedure. Though for purposes of interpreting the generable expressions, let the domain of PL be a subset of the domain of CPL. The domain of PL is the *domain of entities*, of type <e>, for CPL. Let CPL also have a *domain of truth values*, of type <**t**>, exhausted by two entities ⊤ and ⊥. It does not matter if truth values are distinctive abstracta, or two entities of type <e> chosen at random. What matters is that there are exactly two such values, so that any function of type <e, **t**> maps each entity of type <e> to ⊤ or ⊥. In which case, any such function maps an entity to ⊥ if and only if that function does not map that entity to ⊤. Likewise, each function of type <**t**, **t**> maps each truth value to ⊤ or ⊥; each function of type <e**t**, **t**> maps each function of type <e, **t**> to ⊤ or ⊥; etc. And let CPL have a *domain of functions*, comprising (sub)domains of endlessly many types, recursively characterized in terms of the two basic domains of type <e> and <**t**>: given a domain of type <α> and a domain of type <β>, CPL also has a domain of functions of type <α, β>.

The domain for CPL thus includes subdomains that mirror the Fregean hierarchy of conceptual types; see §1.5 above. Things of type <α> differ from concepts of type <α>, apart from concepts of type <e>, which are also entities. But I'll adopt the standard if potentially misleading practice of using the same typological notation for subdomains and concepts. However, the domain of CPL does not include any "unsaturateds" in Frege's intended sense (see §1.1 above). The functions that are stipulated to be in the domain for CPL—given the entities and truth values—are *denotable sets* of ordered pairs.[20] So each thing in the domain of CPL is denotable, and one can extend PL to include corresponding denoters.

Let the sentences of CPL include all the sentences of PL. And let's say that for any sentence S of PL, **T-VALUE**{S} is a denoting expression of CPL. As this suggests, the relevant notion of denotation will be sequence relative, since denoting expressions may contain unbound variables. Given a closed sentence like ∃x[COW(x)], one might simply say that the corresponding

[20] These sets (E-functions) may be specified in terms of procedures (I-functions) that we can think about. Though one also can interpret CPL in terms of a hierarchical domain of procedures; see §2.1 of chapter zero.

truth value is ⊤ or ⊥: **T-VALUE**{∃X[COW(X)]} denotes ⊤ if ∃X[COW(X)] is satisfied by each sequence; otherwise **T-VALUE**{∃X[COW(X)]} denotes ⊥. But we don't want to say that COW(X) has a truth value, once and for all, much less the same truth value as ∃X[COW(X)] or ∀X[COW(X)]. Still, we want CPL to generate the expression **T-VALUE**{COW(X)}, so that COW(X) can be part of a representation of a corresponding function of type <e, **t**>. So let's say that denotation is sequence relative, allowing for expressions whose denotation is constant across sequences.

We can think of sequences as ways of assigning values to unbound variables, and say that relative to any sequence σ: the denotation of **T-VALUE**{S} is ⊤ if σ satisfies S; otherwise, the denotation of **T-VALUE**{S} is ⊥. Since a closed sentence is satisfied by all sequences or none, the denotation of **T-VALUE**{∃X[COW(X)]} is the same relative to each sequence: ⊤ or ⊥, depending on whether or not the domain of entities includes a cow. But relative to each sequence σ, the denotation of **T-VALUE**{COW(X)} is ⊤ if σ satisfies COW(X), and otherwise ⊥. Put another way, for any assignment of a value to the variable in **T-VALUE**{COW(X)}, this expression denotes ⊤ or ⊥, depending on whether or not the assigned value is a cow. So if some but not all entities of type <e> are cows, **T-VALUE**{COW(X)} may denote ⊤ relative to σ and denote ⊥ relative to some other sequence σ*. Likewise, the denotation of **T-VALUE**{ABOVE(X′, X)} relative to σ will depend on whether or not σ satisfies ABOVE(X′, X): ⊤ if σ(X′) is above σ(X), and otherwise ⊥.

Each constant of PL is a denoter of CPL. But let's continue to say that sequences map constants/denoters to entities. Relative to any sequence σ, BESSIE denotes σ(BESSIE)—a.k.a. Bessie the cow. And for any sequences σ and σ*, σ(BESSIE) = σ*(BESSIE).

In section three, I used notation like the following to specify conditions on sequences.

Sat{σ, ∃X[COW(X)]} ⇔ some σ*:σ*≈$_X$σ is such that *Sat*{σ*, COW(X)}

Sat{σ*, COW(X)} ⇔ σ*(X) is a cow

Sat{σ, ∃X[COW(X)]} ⇔ some σ*:σ*≈$_X$σ is such that σ*(X) is a cow

Recall that '*Sat*{σ, S} ⇔ ...' abbreviates 'σ satisfies S iff ...', and 'some σ*:σ*≈$_X$σ' is short for 'some sequence σ* such that σ*≈$_X$σ'. Now let '*Den*{D, σ} = …' abbreviate 'the denotation of D relative to σ is …'; and let '⊤/⊥[…]' be short for '⊤ if … and ⊥ otherwise', as shown below.

Den{BESSIE, σ} = σ(BESSIE) = Bessie

Den{**T-VALUE**{COW(X)}, σ} = ⊤/⊥[*Sat*{σ, COW(X)}] = ⊤/⊥[σ(X) is a cow]

Read '⊤/⊥[σ(X) is a cow]' as follows: ⊤ or ⊥, depending on whether not σ(X) is a cow.

The sentences of PL correspond to sequence-relative denoters of CPL, and **T-VALUE**{…} can be viewed as an expression frame. But there are no instances of **T-VALUE**{**T-VALUE**{…}}; and truth values do not have truth values. We can, however, introduce new denoting expressions as follows: for any variable *v* of type <α> and denoter D of type <β>, the expression λ*v*.D is a denoting expression of type <α, β>. Since **T-VALUE**{COW(X)} and **T-VALUE**{ABOVE(X′, X)} are denoting expressions of CPL, so are λX.**T-VALUE**{COW(X)} and λX.**T-VALUE**{ABOVE(X′, X)} and λX′.λX.**T-VALUE**{ABOVE(X′, X)}. The idea is that these lambda expressions denote functions that can be specified recursively as indicated below.

Den{λX.**T-VALUE**{COW(X)}, σ} = the smallest function F of type <e, **t**> such that for each σ*:σ*≈$_X$σ, F maps σ*(X) to *Den*{**T-VALUE**{COW(X)}, σ*}

Relative to any sequence σ, λX.**T-VALUE**{COW(X)} denotes the smallest function of type <e, **t**> that maps each entity in the relevant domain (i.e., each d-entity) to the truth value assigned to COW(X) given that entity as the value of the variable. For each sequence σ and d-entity δ, σ

has exactly one x-variant that assigns δ to x. So given that *Den*{**T-VALUE**{COW(x)}, σ*} = ⊤/⊥[σ*(x) is a cow], the denotation of λx.**T-VALUE**{COW(x)} can be described as follows.

Den{λx.**T-VALUE**{COW(x)}, σ} = the smallest function F of type <e, **t**> such that for each d-entity δ, F maps δ to ⊤/⊥[δ is a cow]

This function—the **COW**-function, for short—maps Bessie to ⊤ and Venus to ⊥. So for any sequence σ, the **COW**-function maps *Den*{BESSIE, σ} to ⊤ and *Den*{VENUS, σ} to ⊥.

With more than one variable, the pattern is repeated.

Den{λx′.λx.**T-VALUE**{ABOVE(x′, x)}, σ} =
the smallest function F′ of type <e, e**t**> such that for each $\sigma^*{:}\sigma^* \approx_{x'} \sigma$,
F′ maps σ*(x′) to *Den*{λx.**T-VALUE**{ABOVE(x′, x)}, σ*}

The (constant) denotation of λx′.λx.**T-VALUE**{ABOVE(x′, x)} is thus specified in terms of the (inconstant) denotation of λx.**T-VALUE**{ABOVE(x′, x)}, which is specified in terms of the (inconstant) denotation of **T-VALUE**{ABOVE(x′, x)}.

Den{λx.**T-VALUE**{ABOVE(x′, x)}, σ*} =
the smallest function F of type <e, **t**> such that for each $\sigma^{**}{:}\sigma^{**} \approx_{x} \sigma^*$,
F maps σ**(x) to *Den*{**T-VALUE**{ABOVE(x′, x)}, σ**}

So given a specification of the (inconstant) denotation of **T-VALUE**{ABOVE(x′, x)},

Den{**T-VALUE**{ABOVE(x′, x)}, σ**} = ⊤/⊥[σ**(x′) is above σ**(x)]

the denotation of {λx.**T-VALUE**{ABOVE(x′, x)} can be characterized as shown below.

the smallest function F of type <e, **t**>
such that for each $\sigma^{**}{:}\sigma^{**} \approx_{x} \sigma^*$, F maps σ**(x) to ⊤/⊥[σ**(x′) is above σ**(x)]

Or more briefly, and exploiting the fact that σ**(x′) = σ*(x′) if σ** and σ* are x-variants,

Den{λx.**T-VALUE**{ABOVE(x′, x)}, σ*} = the smallest function F of type <e, **t**>
such that for each d-entity δ, F maps δ to ⊤/⊥[σ*(x′) is above δ].

In which case, the denotation of λx′.λx.**T-VALUE**{ABOVE(x′, x)} can be characterized as the smallest function F′ of type <e, e**t**> such that

for each $\sigma^*{:}\sigma^* \approx_{x'} \sigma$, F′ maps σ*(x′) to the smallest function F of type <e, **t**>
such that for each d-entity δ, F maps δ to ⊤/⊥[σ*(x′) is above δ],

or more simply,

for each d-entity δ′, F′ maps δ′ to the smallest function F of type <e, **t**>
such that for each d-entity δ, F maps δ to ⊤/⊥[δ′ is above δ].

This function F′ maps Bessie to a function F that maps each d-entity δ to ⊤ or ⊥ depending on whether or not Bessie is above δ. So for any σ, *Den*{λx′.λx.**T-VALUE**{ABOVE(x′, x)}, σ} maps *Den*{BESSIE, σ} to a function that maps *Den*{VENUS, σ} to ⊤ if Bessie is above Venus, and otherwise maps *Den*{VENUS, σ} to ⊥. By contrast,

Den{λx.λx′.**T-VALUE**{ABOVE(x′, x)}, σ} = the smallest function F of type <e, e**t**>
such that for each d-entity δ, F maps δ to the function F′ of type <e, **t**> such that
for each d-entity δ′, F′ maps δ′ to T/⊥[δ′ is above δ].

This function maps Bessie to a function that maps each entity δ′ to ⊤ or ⊥ depending on whether or not δ′ is above Bessie. So for any σ, *Den*{λx.λx′.**T-VALUE**{ABOVE(x′, x)}, σ} maps *Den*{BESSIE, σ} to a function that maps *Den*{VENUS, σ} to ⊤ if Venus is above Bessie, and otherwise maps *Den*{VENUS, σ} to ⊥.

3.2 Lambdas and Quantification over Sequences

The details just reviewed, somewhat tediously, illustrate the parallels between Church's lambda-binders and Tarski's quantifiers. And I want to highlight this point, which will be quite important in chapter six, before turning to combinations of CPL denoters.

If one viewed λ as a primitive symbol, unrelated to the core Tarskian idea of quantifying over sequence variants, it would be hard to see how—except by stipulation or magic—prefixing **T-VALUE**{COW(X)} with 'λx.' could create a function denoter. It would be even harder to see how prefixing **T-VALUE**{ABOVE(X′, X)} with two lambda binders could, depending on the order, create either of two function denoters. One wants to know how embedding COW(X) in λX.**T-VALUE**{…} could have such an effect in any mind, ideal or natural—and likewise for embedding ABOVE(X′, X) in λX′.λX.**T-VALUE**{…}, GIVE(X″, X′, X) in λX″.λX′.λX.**T-VALUE**{…}, etc. Any sense of mystery evaporates, though, once one sees the relation between lambda abstraction and quantification over sequence variants (e.g., 'some $\sigma^*{:}\sigma^*\approx_x\sigma$').

We don't know how natural minds implement such quantification, or how they represent functions (or entities). But *given* PL as a language of thought that can be extended in modest ways, forming the complex denoting expressions of CPL would not be a huge leap forward, despite initial appearances. This raises the question of whether PL already creates possibilities that are not open to natural human thought, even if such thought goes beyond PL in other ways. But lambda abstraction can be viewed as a way of unleashing the power of quantifying over sequence variants within a language that extends PL only in a few modest ways. And for Church (194x), the neo-Fregean application of his lambda calculus—recursive specification of functions that mirror Frege's hierarchy of ideal concepts, rooted in the basic types <e> and <**t**>—merited only a passing mention (p. xx) as a special but not especially interesting case of how a certain class of functions can be specified with lambda expressions, given some simple primitives. Church invented his calculus to offer proofs concerning the entire space of Turing-computable functions; see §2.1 of chapter zero. Correlatively, lambdas unleash significant power in modest extensions of PL.

The flip side of this point is that lambda abstraction is not an alternative to quantification over sequence variants. It is a way of putting such quantification to work. This is no complaint. In chapter six, I posit a mental language that employs a restricted version of 'some $\sigma^*{:}\sigma^*\approx_x\sigma$' to generate certain monadic concepts from monadic and dyadic atoms. If appeal to quantification over sequence variants is unavoidable, in some form, we may as well make the most of whatever form we have to posit—even if we can't yet say how natural minds implement some version of the trick whose explicit formulation required Tarski's mind. So I have no objection to positing a language of thought with restricted analogs of lambdas. But what *further* mental resources do we need posit to account for word meanings, how they compose, and the cognitive productivity of expressible concepts? In particular, do meanings exhibit more than a smidgeon of dyadicity? If not, do we need to posit words that fetch concepts of *functions* that map entities to truth values? If not, what do we need to assume instead, and what are the implications for how meaning is related to truth?

Such questions are not easily answered. But suppose that given a certain mentalese, the sparest descriptively adequate form of quantification over sequence variants would yield representational capacities that humans do not naturally have. Then prima facie, human minds do not employ that mentalese. In thinking about the implications of this point, for theories of human languages, it is worth remembering two others. First, the power that 'some $\sigma^*{:}\sigma^*\approx_x\sigma$' unleashes depends partly on the number of variables that can be left open in a sentence. Second, the history

of semantics makes it easy to assume (even if one shouldn't) that word meanings can be polyadic in ways that create multiple openings for abstraction. Indeed, Frege's invention of modern logic—and his appeal to truth values as Bedeutungen of ideal thoughts (Gedanken)—was interwoven with his insight, which Church preserved, that apparent monadicity can be reconstructed as a special case of polyadicity. And it is worth pausing to be clear about this.

Instead of adopting a subject-predicate conception of thought—in which monadic concepts play a central role, but PRECEDES-EVERY-NUMBER-THAT-HAS-A-PREDECESSOR presents serious difficulties—Frege urged a function-argument conception of thought that initially seems to leave no place for traditional monadic concepts. Note that the predecessor function is usually specified with *one* variable: P(x) = x–1; λx.x–1; etc. The dyadic character of the underlying concept, PREDECESSOR(_, _) is reflected by the fact that functions map inputs onto outputs; cp. FATHER-OF(_, _). The addition function is usually specified with two variables, while the underlying concept is intuitively triadic: SUM(_, _, _). This raises the question of how to represent (the contents of) monadic concepts as functions. In the case of NUMBER(_), Frege suggested a higher-order dyadic reanalysis; see §1.4 above. Perhaps COW(_), CALF(_), and FARMER(_) are also better represented in terms of dyadic relations that individuals bear to times or events. But some concepts, including NUMBER(_) itself, classify entities. So what do the corresponding functions map entities onto?

Frege's answer is that ideal concepts indicate relations to truth values. The ideal concept **PREDECESSOR**(_, _) applies to ordered pairs of the form ⟨⟨x, x′⟩, **t**⟩; where the second element is a truth value, the first is an ordered pair of numbers, and the second element is ⊤ iff x is the predecessor of x′. Likewise, **SUM**(_, _) applies to ordered pairs ⟨⟨x, x′, x″⟩, **t**⟩ such that **t** = ⊤ iff x″ is the sum of x and x′. This makes room for an analog of COW(_), since **COW**(_, _) can apply to pairs ⟨x, **t**⟩ such that **t** = ⊤ iff x is a cow. This idea is reflected in the semantics for CPL, without problematic appeal to unsaturateds: COW(X) is used to form λX.**T-VALUE**{COW(X)}, whose denotation is a function whose extension is a set of entity/truth-value pairs.

One might speculate that humans naturally employ, at least for purposes of linguistic expression and comprehension, a system of thought that is fundamentally polyadic in this sense. As we'll see in chapter five, many facts tell against this hypothesis. Still, one can at least imagine thinkers who naturally introduce concepts that denote functions that map entities to truth values. Given unsaturated concepts but no variables, a thinker might be able to form the thought EVERY[ABOVE(_, BESSIE), COW(_)], yet be unable to form concepts like ABOVE(VENUS, _)—much less COW-THAT-VENUS-IS-ABOVE(_), COW-THAT-CHASED-EVERY-DOG(_), SAW-A-COW-THAT-CHASED-EVERY-DOG(_), etc. But given CPL, a mind could use concepts like ABOVE(_, _) and COW(_) to introduce complex denoters like λX.λX′.**T-VALUE**{ABOVE(X′, X)}, λX′.λX.**T-VALUE**{ABOVE(X′, X)}, λX′.λX.**T-VALUE**{ABOVE(X′, X) & COW(X)}, etc. This could be useful, since denoters like BESSIE and λX.**T-VALUE**{COW(X)} can be combined in CPL.

3.3 Frege in Church: Polyadicity and Applying Functions

A mere pair of denoting concepts is not a concept, much less a thought. But in CPL, denoters of types <e, **t**> and <**t**> can combine to form denoters of type <**t**>. More generally, if D is a denoting expression of type <α, β> and D* is a denoting expression of type <α>, then D can combine with D* to form a complex denoting expression of the type <β>. And relative to any sequence σ, the denotation of the complex is the result of applying the function that is the denotation of D to the denotation of D*. Thus, "Apply Function"

$$\text{(AF)}\quad \mathit{Den}\{[\mathrm{D}_{<\alpha,\beta>}\ \mathrm{D}^*_{<\alpha>}], \sigma\} = \mathit{Den}\{\mathrm{D}_{<\alpha,\beta>}, \sigma\}(\mathit{Den}\{\mathrm{D}^*_{<\alpha>}, \sigma\})$$

is an interpretation principle for CPL. But note that (AF) differs from saturation in Frege's sense.

In this metalanguage, $\Phi(\alpha)$ is *the value of* the function Φ given the argument α. So if we take functions to be sets/procedures—abstract things, as opposed to Fregean unsaturateds—then $\Phi(\alpha)$ is importantly different than the Fregean thought COW(BESSIE), in which BESSIE saturates COW(_). Denoters cannot combine via saturation. Since λX.**T-VALUE**{COW(X)}(BESSIE) and **T-VALUE**{COW(BESSIE)} have the same denotation, the former might be replaced with the latter in a derivation, suggesting an *analogy* to saturation. But λX.**T-VALUE**{COW(X)} is still a denoter, not a general concept like COW(_). So BESSIE cannot saturate λX.**T-VALUE**{COW(X)}. This is no objection to CPL. Languages like PL and CPL are designed to overcome the limitations, described in section one, of saturation without variables. So it is no surprise if combination in CPL is not interpreted as saturation.

Indeed, CPL permits construction of λX′.λX.**T-VALUE**{ABOVE(X′, X)}(VENUS). This expression denotes the <e, **t**> function that maps an entity to T iff Venus is above that entity. Thus, CPL permits construction of an analog of ABOVE(VENUS, _). And we can rewrite λX′.λX.**T-VALUE**{ABOVE(X′, X)}(VENUS) as the co-denoting λX.**T-VALUE**{ABOVE(VENUS, X)}. But the latter expression is *not* formed by saturating λX′.λX.**T-VALUE**{ABOVE(X′, X)} with VENUS. On the contrary, λX′.λX.**T-VALUE**{ABOVE(X′, X)} cannot be saturated, and that is why it merited introduction: the lambda expression *denotes* a function that maps Venus to the intended *Bedeutung* of ABOVE(VENUS, _). Instead of simply assuming that VENUS can skip over the first/rightmost slot of a polyadic concept and saturate another—or stipulating that BESSIE can be extracted from ABOVE(VENUS, BESSIE), leaving a saturatable concept—CPL provides a possible mechanism for generating a simulacrum of ABOVE(VENUS, _): λX.**T-VALUE**{ABOVE(VENUS, X)} is formed by building λX′.λX.**T-VALUE**{ABOVE(X′, X)}(VENUS) and then rewriting this denoter in accord with an obvious rule for how to re-represent the value of a function given an argument.

If saturation is the only "direct" mode of combining concepts, then combining denoters and interpreting the results as Church-combinations is as "indirect" as combining monadic concepts and interpreting the results as M-junctions; see note 20 of chapter one). Not that there's anything wrong with that. Recall that one might analyze the M-junction BROWN^COW(_) as the result of twice saturating a concept of conjunction—∩[Φ(_), (_)]—of type <e**t**, <e**t**, e**t**>>. Correlatively, the instruction **M-join**[*fetch*@'cow', *fetch*@'brown'] might be spelled out in terms of two instructions to saturate, at least if such instructions are generable.

Saturate[**Saturate**[*fetch*@'∩', *fetch*@'cow'], *fetch*@'brown']]

Executing the embedded step this mini-program would be a way of forming ∩[_, COW(_)]. And this concept of type <e**t**, e**t**> could be saturated with BROWN(_).

Likewise, one might analyze **T-VALUE**{COW(BESSIE)} as the result of twice saturating a dyadic concept of application: APPLY(λX.**T-VALUE**{COW(X)}, BESSIE); where APPLY(_, _) exhibits the abstract type $<D_{<\alpha>}, <D_{<\alpha, \beta>}, D_{<\beta>}>>$, and so saturating APPLY(_, _) with two suitable *denoters* creates a third. The instruction **Saturate**[*fetch*@'cow', *fetch*@'Bessie'] will not be executable if λX.**T-VALUE**{COW(X)} is the only concept at the 'cow' address. But a mind equipped with CPL might generate instructions that call for application rather than saturation: **Apply**[*fetch*@'cow', *fetch*@'Bessie']. And the instruction might be spelled out in terms of two instructions to saturate, given the concept APPLY(_, _).

Saturate[**Saturate**[*fetch*@'APPLY', *fetch*@'Bessie'], *fetch*@'cow']

Executing the embedded complex instruction would be a way of forming APPLY(_, BESSIE), a concept of the abstract type $<D_{<e, \beta>}, D_{<\beta>}>>$. If the lexical address for 'cow' is shared by COW(_) and λX.**T-VALUE**{COW(X)}, there are two ways of executing the lexical instruction *fetch@*'cow'; see §4.2 of chapter one. But since APPLY(_, _) can only be saturated by denoters, executing the program above would require that λX.**T-VALUE**{COW(X)} be the concept fetched via 'cow'.

A rewriting instruction might then replace APPLY(λX.**T-VALUE**{COW(X)}, BESSIE) with **T-VALUE**{COW(BESSIE)}. For simple examples, one need not appeal to APPLY(_, _). If COW(_) and λX.**T-VALUE**{COW(X)} reside at the same lexical address, then executing the instruction **Saturate**[*fetch@*'cow', *fetch@*'Bessie'] is still a way of forming COW(BESSIE). But the denoter λX.**T-VALUE**{ABOVE(BESSIE, X)} can be formed by saturating APPLY(_, BESSIE) with the denoter λX′.λX.**T-VALUE**{ABOVE(X′, X)} and rewriting the result. Still, BESSIE cannot saturate λX′.λX.**T-VALUE**{ABOVE(X′, X)} even if it can saturate the first position of APPLY(_, _).

I don't think that humans understand linguistic expressions by employing the concept APPLY(_, _) along with Church-style denoters of functions/sets and capacities for rewriting such denoters. But just as one might hypothesize that children use a mental analog of PL to specify satisfaction conditions for human language sentences, one can hypothesize a mental analog of CPL. Moreover, my own proposal also involves constructing simulacra of Fregean concepts, by means of operations other than saturation, and letting lexical addresses be shared by concepts of different types. So it is important to be clear that in offering a Church-style reconstruction of Frege's semantics, one appeals to operations *other than* saturation. The question is *which* further operations are needed, not *whether* further operations are needed.

I think that human languages invoke conjunction operations, that they do not invoke function-application, and that such languages are massively monadic as opposed to massively denotational. But this is an empirical hypothesis. One might have thought that since combining denoters cannot yield truth-evaluable thoughts, words like 'cow' *cannot* fetch denoting concepts that combine with others. This is not my view. I don't think that 'cow' (or any word) fetches a denoter. But if thoughts can denote truth values, then denoters *of the right sort* can be truth-evaluable. Perhaps each thought needs at least one nondenoting constituent, as Frege suggested. But CPL can systematically generate denoters that can saturate APPLY(_, _)—and thereby form analogs of Fregean thoughts whose constituents include concepts like COW(_), as opposed to APPLY(_, _) and a denoter like λX.**T-VALUE**{COW(X)}.

Indeed, my hypothesis that linguistic meanings direct construction of predicates can be viewed as a variant of the idea that linguistic meanings direct construction of CPL denoters. In principle, a semanticist can reject the latter hypothesis and still say that the "semantic value" of the word 'cow' is the function λX.**T-VALUE**{COW(X)}—or abbreviating, λX.**COW**(X). But one wants to know how the word, which children can acquire, could come to have this semantic value if *not* because speakers associate 'cow' with the concept COW(_), and use this concept to construct the corresponding denoter of type <e**t**>. So I take the psychological hypothesis seriously.[21] On this view, meanings can (qua assembly instructions) exhibit part-whole relations even when the corresponding denotations do not. And CPL, unlike PL, generates expressions that are both significant and subsentential: λX.**T-VALUE**{COW(X)} and BESSIE belong to semantic

[21] As discussed in chapter six, this proposal seems to be implicit in Heim and Kratzer's (1998) textbook presentation of a Church/Montague-style semantics for English.

categories, as does **T-VALUE**{COW(BESSIE)}. One can similarly treat '¬', which is syncategorematic in PL, as a denoter of type <**t**, **t**> in CPL; and one can introduce a range of denoters for the range of conjunction functions (cp. section three of chapter one). But I haven't yet said how to assign denotations to quantificational expressions, which are also syncategorematic in PL.

3.4 Functions and Quantification

PL only allows for variables in positions occupied by Fregean singular concepts of type <e>. CPL, as defined thus far, only allows for lambda abstraction into such positions. But one can allow for open sentences like Φ(BESSIE) and ∃x[Φ(x)], extend the notion of sequence accordingly, and permit abstraction into predicate positions as in λΦ.**T-VALUE**{∃x[Φ(x)]}. This lambda expression denotes the function ℑ of type <e**t**, **t**> such that for each function F of type <e, **t**>: ℑ maps F to ⊤ iff F maps some entity of type <e> to ⊤. This higher-order function maps the **COW**-function to ⊤ iff there is a cow.

The new twist adds variables of type <e, **t**>: Φ, Φ′, Φ″, etc. The old variables—x, x′, x″, etc.—are of type <e>. The domain already includes functions of type <e, **t**>. So we can say that *assignments* map the new variables to functions and the old variables of type <e> to entities. Think of Tarski-sequences, n-tuples of entities, as special cases of n-tuples of entity-function n-tuples. For example, think about replacing ⟨Bessie, Venus, Sadie, …⟩ with an n-tuple, each element of which is a sequence of entity-function pairs with constant entity as indicated below;

⟨⟨Bessie, the **COW**-function⟩, ⟨Bessie, the **HORSE**-function⟩, … ⟩
⟨⟨Venus, the **COW**-function⟩, ⟨Venus, the **HORSE**-function⟩, … ⟩
⟨⟨Sadie, the **COW**-function⟩, ⟨Sadie, the **HORSE**-function⟩, … ⟩

where each such sequence orders the functions in the same way, allowing for two dimensions of variation in how values are assigned to variables. Let '𝒜' range over such assignments, each of which can be viewed as a mapping from variable-pairs like <x, Φ> to entity-function pairs.

For example, let ALONZO be a particular assignment of the sort indicated above. Then ALONZO(<x, Φ>) is the ordered pair ⟨Bessie, the **COW**-function⟩, while ALONZO(<x′, Φ″>) is ⟨Venus, the **HORSE**-function⟩. Given how assignments are ordered, we can still speak of *the* entity that ALONZO associates with the variable x—viz., Bessie. Each entity e corresponds to a "horizontal" sequence of entity-function pairs of the form <e, F>. Likewise, the **COW**-function is *the* function that ALONZO associates with the variable Φ. Each function F of type <e, **t**> corresponds to a "vertical" sequence of entity-function pairs of the form <e, F>.

The notion of an assignment variant remains the same. If 𝒜* differs from 𝒜 at most with regard to what 𝒜* assigns to the variable x, then 𝒜* and 𝒜 assign the same value to every other variable, including all the variables of type <e, **t**>. For example, ALONZO has an x-variant of the sort indicated below, with Fido replacing Bessie in the x-row.

⟨⟨Fido, the **COW**-function⟩, ⟨Fido, the **HORSE**-function⟩, … ⟩
⟨⟨Venus, the **COW**-function⟩, ⟨Venus, the **HORSE**-function⟩, … ⟩
⟨⟨Sadie, the **COW**-function⟩, ⟨Sadie, the **HORSE**-function⟩, … ⟩

Likewise, if 𝒜* differs from 𝒜 at most with regard to what 𝒜* assigns to the variable Φ, then 𝒜* and 𝒜 assign the same value to every other variable, including all the variables of type <e>. For example, ALONZO has a Φ-variant of the sort indicated below, with the **DOG**-function replacing the **COW**-function in the Φ-column.

⟨⟨Bessie, the **DOG**-function⟩, ⟨Bessie, the **HORSE**-function⟩, … ⟩
⟨⟨Venus, the **DOG**-function⟩, ⟨Venus, the **HORSE**-function⟩, … ⟩
⟨⟨Sadie, the **DOG**-function⟩, ⟨Sadie, the **HORSE**-function⟩, … ⟩

Each entity-function pair corresponds to a truth value in an obvious way. And the notion of satisfaction can be extended accordingly. The open sentence $\mathcal{A}(x)$ is satisfied by $\mathcal{A}$ if and only if $\top$ is the result of applying the function $\mathcal{A}(\Phi)$ to the entity $\mathcal{A}(x)$.

$$Sat\{\mathcal{A}, \Phi(x)\} \Leftrightarrow \mathcal{A}(\Phi)(\mathcal{A}(x)) = \top$$

This extends the original notion of satisfaction via the notion of function application. But the interpretation for the Tarskian existential quantifier remains the same.

$$Sat\{\mathcal{A}, \exists x[\Phi(x)]\} \Leftrightarrow \text{some } \mathcal{A}^*{:}\mathcal{A}^*\approx_x\mathcal{A} \text{ is such that } Sat\{\mathcal{A}^*, \Phi(x)]\}$$

So the condition imposed by $\exists x[\Phi(x)]$ can be specified as follows.

$$Sat\{\mathcal{A}, \exists x[\Phi(x)]\} \Leftrightarrow \text{some } \mathcal{A}^*{:}\mathcal{A}^*\approx_x\mathcal{A} \text{ is such that } \mathcal{A}^*(\Phi)(\mathcal{A}^*(x)) = \top$$

If $\mathcal{A}^*\approx_x\mathcal{A}$, then $\mathcal{A}^*(\Phi) = \mathcal{A}(\Phi)$. So $\mathcal{A}$ satisfies $\exists x[\Phi(x)]$ iff some $\mathcal{A}^*{:}\mathcal{A}^*\approx_x\mathcal{A}$ is such that $\mathcal{A}(\Phi)$ maps $\mathcal{A}^*(x)$ to $\top$. That is, $\mathcal{A}$ satisfies $\exists x[\Phi(x)]$ iff the function that $\mathcal{A}$ assigns to Φ maps some entity to $\top$. In which case, relative to any assignment $\mathcal{A}$, **T-VALUE**$\{\exists x[\Phi(x)]\}$ denotes $\top$ or $\bot$, depending on whether or not $\mathcal{A}(\Phi)$ maps some entity to $\top$. This denotation condition can be specified more formally as follows.

$$Den\{\textbf{T-VALUE}\{\exists x[\Phi(x)]\}, \mathcal{A}\} = \top/\bot[Sat\{\mathcal{A}, \exists x[\Phi(x)]\}] =$$
$$\top/\bot[\text{some } \mathcal{A}^*{:}\mathcal{A}^*\approx_x\mathcal{A} \text{ is such that } \mathcal{A}^*(\Phi)((\mathcal{A}^*(x)) = \top]$$

So the lambda abstract $\lambda\Phi.$**T-VALUE**$\{\exists x[\Phi(x)]\}$ denotes the intended higher-order function.

$$Den\{\lambda\Phi.\textbf{T-VALUE}\{\exists x[\Phi(x)]\}, \mathcal{A}\} = \text{the smalleset function } \Im \text{ of type } <\mathbf{e}\mathbf{t}, \mathbf{t}>$$
$$\text{such that for each function F of type } <\mathrm{e}, \mathbf{t}>, \text{ some } \mathcal{A}^*{:}\mathcal{A}^*\approx_\Phi\mathcal{A} \text{ is such that}$$
$$\mathcal{A}^*(\Phi) = \mathrm{F}, \text{ and } \Im \text{ maps } \mathcal{A}^*(\Phi) \text{ to } Den\{\textbf{T-VALUE}\{\exists x[\Phi(x)]\}, \mathcal{A}^*\}$$

Or more briefly, relative to any assignment $\mathcal{A}$, $\lambda\Phi.$**T-VALUE**$\{\exists x[\Phi(x)]\}$ denotes the function $\Im$ that maps each function F to $\top$ if and only if F maps some entity to $\top$.

An assignment $\mathcal{A}$ satisfies $[\Phi'(x) \,\&\, \Phi(x)]$ iff $\mathcal{A}$ satisfies both $\Phi'(x)$ and $\Phi(x)$—i.e., iff both $\mathcal{A}(\Phi')$ and $\mathcal{A}(\Phi)$ map $\mathcal{A}(x)$ to $\top$. A function $\Im^2$ of type $<\mathbf{e}\mathbf{t}, <\mathbf{e}\mathbf{t}, \mathbf{t}>>$ maps each function F′ of type $<\mathrm{e}\mathbf{t}>$ to a function $\Im$ of type $<\mathbf{e}\mathbf{t}, \mathbf{t}>$. So relative to any assignment, the "doubled" lambda abstract $\lambda\Phi'.\lambda\Phi.$**T-VALUE**$\{\exists x[\Phi'(x) \,\&\, \Phi(x)]\}$ denotes the (smallest) function $\Im^2$ that maps each function F′ to the function $\Im$ that maps each function F to $\top$ iff some entity e is such that both F and F′ map e to $\top$. This dyadic higher-order function maps the **COW**-function to a function that maps the **BROWN**-function to $\top$ iff some entity is both a cow and brown. Likewise, relative to any assignment, $\lambda\Phi'.\lambda\Phi.$**T-VALUE**$\{\forall x[\Phi'(x) \supset \Phi(x)]\}$ denotes the function $\Im^2$ that maps each function F′ to the function $\Im$ that maps each function F to $\top$ iff every entity e is such that F maps e to $\top$ if F′ maps e to $\top$. This dyadic higher-order function maps the **COW**-function to a function that maps the **BROWN**-function to $\top$ iff every entity is such that it is brown if it is a cow.

Abstracts like $\lambda\Phi'.\lambda\Phi.\textbf{T-VALUE}\{\forall x[\Phi'(x) \supset \Phi(x)]\}$ have Tarskian quantifiers, each binding one variable, as constituents. But since the variable of type <e> is bound, one can abbreviate in a way that highlights the double predicate abstraction: $\lambda\Phi'.\lambda\Phi.\textbf{EVERY}\{\Phi', \Phi\}$. Combining $\lambda\Phi'.\lambda\Phi.\textbf{EVERY}\{\Phi', \Phi\}$ with $\lambda x.\textbf{COW}(x)$ yields $\lambda\Phi'.\lambda\Phi.\textbf{EVERY}\{\Phi', \Phi\}(\lambda x.\textbf{COW}(x))$, which can be rewritten as $\lambda\Phi.\textbf{EVERY}\{\lambda x.\textbf{COW}(x), \Phi\}$; and this denoter of type <e**t**, **t**> can combine with $\lambda x.\textbf{BROWN}(x)$ to form $\textbf{EVERY}\{\lambda x.\textbf{COW}(x), \lambda x.\textbf{BROWN}(x)\}$, which denotes $\top$ or $\bot$ depending on whether or not every cow is brown.

Double predicate abstraction makes it possible to introduce endlessly many concepts—including $\lambda\Phi'.\lambda\Phi.\textbf{TWO}\{\Phi', \Phi\}$, $\lambda\Phi'.\lambda\Phi.\textbf{THREE}\{\Phi', \Phi\}$, etc.—in Fregean fashion.[22] The thought ONETOONE[COW(_), HORSE(_)] is true iff the cows correspond one to one with the horses; and the concept ONETOONE[_, _] is saturated by two function denoters. So let ONETOONE[Φ', Φ] be satisfied by $\mathcal{A}$ iff the (set of) things that $\mathcal{A}$ (Φ') maps to $\top$ correspond one to one with the (set of) things that $\mathcal{A}(\Phi)$ maps to $\top$. Then $\lambda\Phi'.\lambda\Phi.\ \textbf{ONETOONE}[\Phi', \Phi]\}$ denotes a function $\Im^2$ that maps each function F' to the function $\Im$ that maps each function F to $\top$ iff the elements of $\{x: F'(x) = \top\}$ correspond one to one with the elements of $\{x: F(x) = \top\}$. One can likewise introduce ONETOONEPLUS[_, _] so that $\lambda\Phi'.\lambda\Phi.\textbf{ONETOONEPLUS}[\Phi', \Phi]$ denotes a function $\Im^2$ that maps each function F' to the function $\Im$ that maps each function F to $\top$ iff: some but not all of the elements of $\{x: F'(x) = \top\}$ correspond one to one with (all) the elements of $\{x: F(x) = \top\}$. Note that most of the cows are brown iff the brown cows outnumber the cows that are not brown. So given the resources of CPL, $\lambda\Phi'.\lambda\Phi.\textbf{MOST}\{\Phi', \Phi\}$ might be defined in terms of one-to-one correspondence, conjunction, and negation; see note 28 of chapter one.

Thus, CPL supports introduction of many concepts that cannot be defined in PL, many of which humans enjoy; see Rescher (1967), Wiggins (1980). Indeed, CPL supports introduction of function-denoters of every Fregean type, at every level of hierarchy discussed in §1.5 above. The notion of an assignment can be extended to cover variables of any type. And even with regard to monadic predicate abstraction, CPL supports introduction of concepts that go beyond those humans can naturally fetch with words; see chapter five. Staying with quantificational examples here, no word like 'every' fetches $\lambda\Phi'.\lambda\Phi.\textbf{ONETOONE}\{\Phi', \Phi\}$ and combines with two predicates to form a sentence; see Barwise and Cooper (1981), Higginbotham and May (1981). To express this simple second-order concept, we need to form complex locutions like 'The dogs *are equinumerous with* the cows'. I return to the details and questions raised in chapter six. For now, it is enough to note that CPL allows for concept introduction by exploiting quantification over assignments (abstraction) in ways that may not be natural for humans.

Since the expressions of CPL include endlessly many denoters, and lambda expressions can be used to specify *procedures* (I-functions as opposed to E-functions), CPL is also suited to the Fregean idea that expressions have Sinnen as well as Bedeutungen. One can say that each denoter of CPL denotes its denotation in a certain way, and that two denoters—HESPERUS and PHOSPHORUS, $\lambda x.|x - 1|$ and $\lambda x.{}^{+}\sqrt{(x^2 - 2x + 1)}$, $\lambda x.\textbf{WOODCHUCK}(x)$ and $\lambda x.\textbf{GROUNDHOG}(x)$, etc.—can denote the same entity/function in different ways. So while $\lambda x.\textbf{WOODCHUCK}$(HESPERUS) and $\lambda x.\textbf{GROUNDHOG}$(PHOSPHORUS) both denote the truth value $\bot$, one might say that these sentences differ semantically by virtue of denoting in different ways.

[22] That is, without resorting to first-order specifications like $\exists x[\Phi'(x) \,\&\, \Phi(x) \,\&\, \exists x'[\Phi'(x') \,\&\, \Phi(x') \,\&\, \neg(x = x') \,\&\, \forall x''[\Phi'(x'') \,\&\, \Phi(x'') \supset [(x'' = x) \vee (x'' = x')]]]]$. Introducing $\lambda\Phi'.\lambda\Phi.\textbf{SEVEN}\{\Phi', \Phi\}$ this way is less than ideal.

We needn't dwell on whether this ends up being a formal variant of an idea noted in section three: WOODCHUCK(HESPERUS) and GROUNDHOG(PHOSPHORUS) differ semantically, even though neither is satisfied by any sequence, because these sentences of PL correspond to different *derivable specifications of* conditions on sequences. The point here is that a Tarskian semantics for PL can be extended in a way that provides at least much of what Frege wanted. This provides a potential model for human languages. Though again, CPL puts quantification over sequence variants to work in a way that permits representation of any computable function. And this may be more work of a kind that human minds cannot do in any natural way.

5. Summary and Transition

Thanks to Frege, we have modern quantificational logic and a vivid case study of how a language can be used to introduce formally new concepts that are useful for certain cognitive purposes. Thanks to Tarski, we have a paradigm of what it is for a language to have a truth conditional semantics. Thanks to Church, we have a clear example of how such a language can be extended in interesting ways. This invites an exciting project to which many theorists have contributed a great deal of insightful work: try to successively modify PL and/or CPL in ways that yield increasingly better models of human languages. My own proposals are anchored within this tradition. In my view, humans enjoy a language of thought that can be viewed as an extension of a fragment of PL: PL does not allow various constructions that human languages permit; though in other respects, PL (and hence, CPL) is far more permissive. But there are at least two ways—sketched in this final section—to think about modifying a language *designed* to have a truth conditional semantics, in the service of providing better models of human languages.

One possibility, urged by Davidson (1967b, 1984) and Montague (1974), is that each human language *has* a truth conditional semantics. From this perspective, modifications of (C)PL can be viewed as attempts to characterize the satisfaction/denotation conditions that human linguistic expressions impose on the entities/functions that speakers can talk about and quantify over. As discussed in the introductory chapter, (D1) is a compact formulation of this hypothesis.

(D1) for each human language **H**, there is a correct theory of truth for **H**.

And if (D1) is plausible, or at least not obviously wrong, then (D2) can be tempting.

(D2) a suitably formulated theory of truth for a human language **H**
can serve as the core of a correct theory of meaning for **H**

The alternative urged here is that human languages let their users introduce, combine, and express concepts that are useful but not ideal in Frege's sense.

It is an old idea that serious inquiry includes and may require attempts to *construct* languages that mirror reality, and to understand how actual modes of thought and talk *diverge* from this scientific ideal.[23] From this perspective, it would take a miracle—a universe ordered so that human biology naturally reflects it—for a human I-language to *have* a truth conditional semantics. One can view sentences of (C)PL as models of the thoughts that humans can form by using *all* the resources, natural and theoretical, at our disposal. But if we want models of the thoughts that we regularly form in a natural way, then we may need to invent model languages—perhaps extensions of fragments of PL—with *that* purpose in mind. We may also need to think of spoken/signed languages as tools used in the assembly of natural thoughts, rather than mere systems for communicating thoughts constructed in some other way.

[23] Descartes tried to think with clear and distinct ideas governed by the *Principles of Philosophy*. Leibniz wanted a *Characteristica Universalis*. Following Frege and Russell, Wittgenstein envisioned the possibility of a language that has a truth conditional semantics as outlined in the *Tractatus*.

Perhaps (1) is like the ideal sentence PRECEDES(ZERO, ONE) in having a truth condition.

(1) Snow is white.

Theorists can hypothesize that for some condition C on assignments of values to variables in (1), snow is white iff for each assignment $\mathcal{A}$, C($\mathcal{A}$). But is this a correct hypothesis about a human linguistic expressions, or an idealization whose limitations need to be highlighted in an account of what meanings are? Should we grant that (1) is true iff snow is white, and try to figure out how this could be so, or should we treat biconditionals like (2)

(2) 'Snow is white.' is true iff snow is white.

as initially useful but ultimately distorting simplifications? In the next two chapters, I offer reasons for not following the first path; in my view, (D1) and (D2) are both implausible. In chapters five and six, I return to my proposed alternative.