2 Executing the Fregean Program

In the pages to follow, we will execute the Fregean program for a fragment of English. Although we will stay very close to Frege's proposals at the beginning, we are not interested in an exegesis of Frege, but in the systematic development of a semantic theory for natural language. Once we get beyond the most basic cases, there will be many small and some not-so-small departures from the semantic analyses that Frege actually defended. But his treatment of semantic composition as functional application (*Frege's Conjecture*), will remain a leading idea throughout.

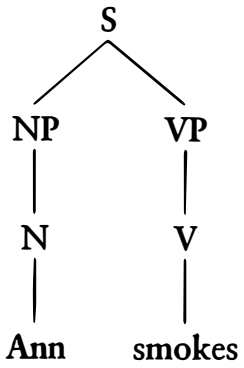
Modern syntactic theory has taught us how to think about sentences and their parts. Sentences are represented as phrase structure trees. The parts of a sentence are subtrees of phrase structure trees. In this chapter, we begin to explore ways of interpreting phrase structure trees of the kind familiar in linguistics. We will proceed slowly. Our first fragment of English will be limited to simple intransitive and transitive sentences (with only proper names as subjects and objects), and extremely naive assumptions will be made about their structures. Our main concern will be with the process of meaning composition. We will see how a precise characterization of this process depends on, and in turn constrains, what we say about the interpretation of individual words.

This chapter, too, has sections which are not devoted to semantics proper, but to the mathematical tools on which this discipline relies. Depending on the reader's prior mathematical experience, these may be supplemented by exercises from other sources or skimmed for a quick review.

2.1 First example of a Fregean interpretation

We begin by limiting our attention to sentences that consist of a proper name plus an intransitive verb. Let us assume that the syntax of English associates these with phrase structures like that in (1).

(1)



We want to formulate a set of semantic rules which will provide denotations for all trees and subtrees in this kind of structure. How shall we go about this? What sorts of entities shall we employ as denotations? Let us be guided by Frege.

Frege took the denotations of sentences to be truth-values, and we will follow him in this respect. But wait. Can this be right? The previous chapter began with the statement "To know the meaning of a sentence is to know its truth-conditions". We emphasized that the meaning of a sentence is not its actual truth-value, and concluded that a theory of meaning for natural language should pair sentences with their truth-*conditions* and explain how this can be done in a compositional way. Why, then, are we proposing truth-*values* as the denotations for sentences? Bear with us. Once we spell out the complete proposal, you'll see that we will end up with truth-conditions after all.

The Fregean denotations that we are in the midst of introducing are also called "*extensions*", a term of art which is often safer to use because it has no potentially interfering non-technical usage. The extension of a sentence, then, is its actual truth-value. What are truth-values? Let us identify them with the numbers 1 (True) and 0 (False). Since the extensions of sentences are not functions, they are saturated in Frege's sense. The extensions of proper names like "Ann" and "Jan" don't seem to be functions either. "Ann" denotes Ann, and "Jan" denotes Jan.

We are now ready to think about suitable extensions for intransitive verbs like "smokes". Look at the above tree. We saw that the extension for the lexical item "Ann" is the individual Ann. The node dominating "Ann" is a non-branching N-node. This means that it should inherit the denotation of its daughter node.[1] The N-node is again dominated by a non-branching node. This NP-node, then, will inherit its denotation from the N-node. So the denotation of the NP-node in the above tree is the individual Ann. The NP-node is dominated by a branching S-node. The denotation of the S-node, then, is calculated from the denotation of the NP-node and the denotation of the VP-node. We know that the denotation of the NP-node is Ann, hence saturated. Recall now that Frege conjectured that all semantic composition amounts to functional application. If that is so, we

must conclude that the denotation of the VP-node must be unsaturated, hence a function. What kind of function? Well, we know what kinds of things its arguments and its values are. Its arguments are individuals like Ann, and its values are truth-values. The extension of an intransitive verb like "smokes", then, should be a function from individuals to truth-values.

Let's put this all together in an explicit formulation. Our semantics for the fragment of English under consideration consists of three components. First, we define our inventory of denotations. Second, we provide a lexicon which specifies the denotation of each item that may occupy a terminal node. Third, we give a semantic rule for each possible type of non-terminal node. When we want to talk about the denotation of a lexical item or tree, we enclose it in double brackets. For any expression α, then, $[\![\alpha]\!]$ is the denotation of α. We can think of $[\![\ ]\!]$ as a function (the *interpretation function*) that assigns appropriate denotations to linguistic expressions. In this and most of the following chapters, the denotations of expressions are extensions. The resulting semantic system is an *extensional semantics*. Towards the end of this book, we will encounter phenomena that cannot be handled within an extensional semantics. We will then revise our system of denotations and introduce *intensions*.

A. Inventory of denotations

Let D be the set of all individuals that exist in the real world. Possible denotations are:

Elements of D, the set of actual individuals.
Elements of {0, 1}, the set of truth-values.
Functions from D to {0, 1}.

B. Lexicon

$[\![\textbf{Ann}]\!]$ = Ann
$[\![\textbf{Jan}]\!]$ = Jan
etc. for other proper names.

$[\![\textbf{works}]\!]$ = f : D $\rightarrow$ {0, 1}
For all x $\in$ D, f(x) = 1 iff x works.
$[\![\textbf{smokes}]\!]$ = f : D $\rightarrow$ {0, 1}
For all x $\in$ D, f(x) = 1 iff x smokes.
etc. for other intransitive verbs.

C. Rules for non-terminal nodes

In what follows, Greek letters are used as variables for trees and subtrees.

(S1) If α has the form $\overset{\text{S}}{\underset{\beta\ \ \gamma}{\wedge}}$, then $[\![\alpha]\!] = [\![\gamma]\!]([\![\beta]\!])$.

(S2) If α has the form $\overset{\text{NP}}{|}{}_{\beta}$, then $[\![\alpha]\!] = [\![\beta]\!]$.

(S3) If α has the form $\overset{\text{VP}}{|}{}_{\beta}$, then $[\![\alpha]\!] = [\![\beta]\!]$.

(S4) If α has the form $\overset{\text{N}}{|}{}_{\beta}$, then $[\![\alpha]\!] = [\![\beta]\!]$.

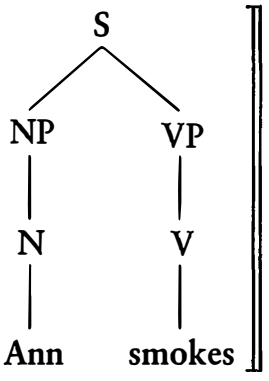
(S5) If α has the form $\overset{\text{V}}{|}{}_{\beta}$, then $[\![\alpha]\!] = [\![\beta]\!]$.

2.1.1 *Applying the semantics to an example*

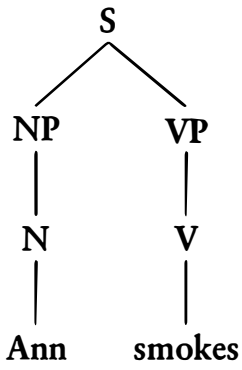
Does this set of semantic rules predict the correct truth-conditions for "Ann smokes"? That is, is "Ann smokes" predicted to be true if and only if Ann smokes? "Of course", you will say, "that's obvious". It's pretty obvious indeed, but we are still going to take the trouble to give an explicit proof of it. As matters get more complex in the chapters to come, it will be less and less obvious whether a given set of proposed rules predict the judgments it is supposed to predict. But you can always find out for sure if you draw your trees and work through them node by node, applying one rule at a time. It is best to get used to this while the calculations are still simple. If you have some experience with computations of this kind, you may skip this subsection.

We begin with a precise statement of the claim we want to prove:

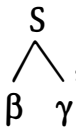
Claim:

$[\![$  $]\!]$ = 1 iff Ann smokes.

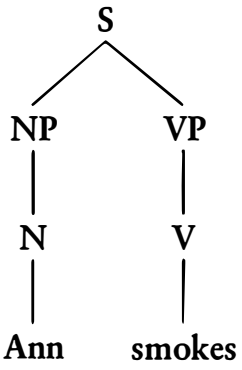
We want to deduce this claim from our lexical entries and semantic rules (S1)–(S5). Each of these rules refers to trees of a certain form. The tree

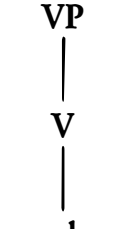


is of the form specified by rule (S1), repeated here, so let's see what (S1) says about it.

(S1) If α has the form  , then $[\![\alpha]\!] = [\![\gamma]\!]([\![\beta]\!])$.

When we apply a general rule to a concrete tree, we must first match up the variables in the rule with the particular constituents that correspond to them in the application. In this instance, α is



so β must be  and γ must be 

The rule says that $[\![\alpha]\!] = [\![\gamma]\!]([\![\beta]\!])$, so this means in the present application that

(2) $[\![$ S (NP: N: Ann; VP: V: smokes) $]\!] = [\![$ VP: V: smokes $]\!]([\![$ NP: N: Ann $]\!])$

Now we apply rule (S3) to the tree

VP
V
smokes

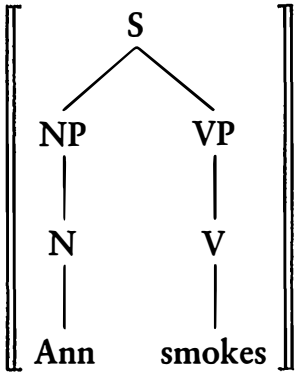
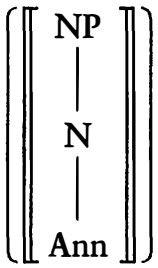
(This time, we skip the detailed justification of why and how this rule fits this tree.) What we obtain from this is

(3) $[\![$ VP: V: smokes $]\!] = [\![$ V: smokes $]\!]$

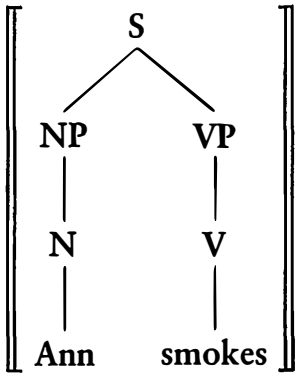
From (2) and (3), by substituting equals for equals, we infer (4).

(4) $[\![$ S (NP: N: Ann; VP: V: smokes) $]\!] = [\![$ V: smokes $]\!]([\![$ NP: N: Ann $]\!])$

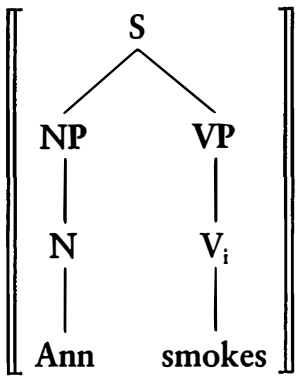
Now we apply rule (S5) to the appropriate subtree and use the resulting equation for another substitution in (4):

(5) $[\![$  $]\!]$ = $[\![\text{smokes}]\!]\left([\![\right.$  $\left.]\!]\right)$

Now we use rule (S2) and then (S4), and after substituting the results thereof in (5), we have (6).

(6) $[\![$  $]\!]$ = $[\![\text{smokes}]\!]([\![\text{Ann}]\!])$

At this point, we look up the lexical entries for **Ann** and **smokes**. If we just use these to substitute equals for equals in (6), we get (7).

(7) $[\![$  $]\!]$ = $\left[\begin{array}{l} f : D \rightarrow \{0, 1\} \\ \quad \text{For all } x \in D,\ f(x) = 1 \text{ iff } x \text{ smokes} \end{array}\right](\text{Ann})$

Let's take a close look at the right-hand side of this equation. It has the gross form "function (argument)', so it denotes the value that a certain function yields for a certain argument. The argument is Ann, and the function is the one which maps those who smoke to 1 and all others to 0. If we apply this function to Ann, we will get 1 if Ann smokes and 0 if she doesn't. To summarize what we have just determined:

(8) $\left[\begin{array}{l} f : D \rightarrow \{0, 1\} \\ \quad \text{For all } x \in D,\ f(x) = 1 \text{ iff } x \text{ smokes} \end{array}\right]$(Ann) = 1 iff Ann smokes.

And now we have reached the goal of our proof: (7) and (8) together imply exactly the claim which we stated at the beginning. QED.

This was not the only way in which we could have constructed the proof of this claim. What matters is (a) that we use each applicable rule or lexical entry to obtain an equation regarding the denotation of a certain subtree; (b) that we keep using some of these equations to substitute equals for equals in others, thereby getting closer and closer to the target equation in our claim; and (c) that we employ the definitions of functions that we find in the lexicon to calculate their values for specified arguments. There is no unique specified order in which we must perform these steps. We can apply rules to the smallest subtrees first, or start at the top of the tree, or anywhere in the middle. We can collect a long list of separate equations before we begin to draw conclusions from any two of them, or else we can keep alternating applications of semantic rules with substitutions in equations derived previously. The soundness of the proof is not affected by these choices (although, of course, some strategies may be easier than others to follow through without getting confused).

We have used the word "proof" a number of times in this section. What exactly do we mean by this term? The notion of "proof" has been made precise in various ways in the history of logic. The most rigorous notion equates a proof with a syntactic derivation in an axiomatic or natural deduction system. Above, we relied on a less regimented notion of "proof" that is common in mathematics. Mathematical proofs are rarely algorithmic derivations. They are usually written in plain English (or some other natural language), supplemented by technical vocabulary that has been introduced through definitions. Conclusions are licensed by inference patterns that are known to be valid but are not spelled out formally. The proofs in this book are all "semi-formal" in this way. The standards of rigor followed in mathematics should be good enough for what we want to accomplish here.

2.1.2 *Deriving truth-conditions in an extensional semantics*

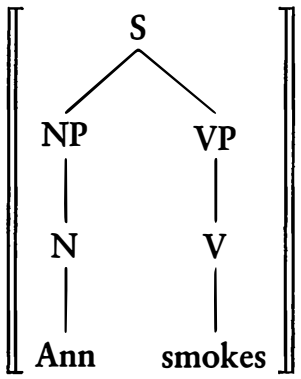
The proof we just gave illustrates how a semantic system based on extensions allows us to compute the truth-conditions, and hence the meaning, of a sentence. If you check the proof again, you will see that we end up with the truth-*conditions* of "Ann smokes" because the lexicon defines the extensions of predicates by specifying a *condition*. Had we defined the function denoted by "smokes" by displaying it in a table, for example, we would have obtained a

mere truth-*value*. We didn't really have a choice, though, because displaying the function in a table would have required more world knowledge than we happen to have. We do not know of every existing individual whether or not (s)he smokes. And that's certainly not what we have to know in order to know the meaning of "smoke". We could look at a fictitious example, though.

Suppose Ann, Jan, and Maria are the only individuals in the actual world, and Ann and Jan are the only smokers. The extension of the verb "smokes" can now be displayed in a table:

$$[\![\textbf{smokes}]\!] = \begin{bmatrix} \text{Ann} & \rightarrow 1 \\ \text{Jan} & \rightarrow 1 \\ \text{Maria} & \rightarrow 0 \end{bmatrix}$$

Using this way of defining the extension of "smokes", our computation would have ended as follows:



$$= \begin{bmatrix} \text{Ann} & \rightarrow 1 \\ \text{Ann} & \rightarrow 1 \\ \text{Maria} & \rightarrow 0 \end{bmatrix} (\text{Ann}) = 1$$

Here, the sentence "Ann smokes" would not be paired with its truth-conditions, but with the value 1.

The issue of how an extensional system can yield a theory of meaning concerns the relationship between what Frege called "*Sinn*" and "*Bedeutung*". Frege's "Bedeutung" corresponds to our term "extension", and is sometimes translated as "reference".[2] Frege's "Sinn" is usually translated as "sense", and corresponds to what we have called "meaning" How does Frege get us from *Bedeutung* to *Sinn*? In his book on Frege's philosophy of language, Michael Dummett answers this question as follows:

> It has become a standard complaint that Frege talks a great deal about the senses of expressions, but nowhere gives an account of what constitutes such a sense. This complaint is partly unfair: for Frege the sense of an expression is the manner in which we determine its reference, and he tells us a great deal about the kind of reference possessed by expressions of different types, thereby specifying the form that the senses of such expressions must

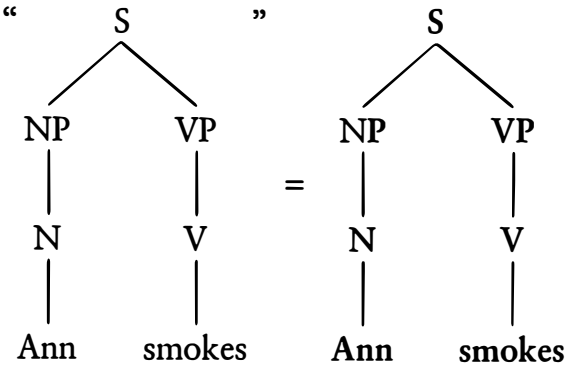
> take. . . . The sense of an expression is the mode of presentation of the referent: in saying what the referent is, we have to choose a particular way of saying this, a particular means of determining something as a referent.[3]

What Dummett says in this passage is that when specifying the extension (reference, *Bedeutung*) of an expression, we have to choose a particular way of presenting it, and it is this manner of presentation that might be considered the meaning (sense, *Sinn*) of the expression. The function that is the extension of a predicate can be presented by providing a condition or by displaying it in a table, for example. Only if we provide a condition do we choose a mode of presentation that "shows"[4] the meaning of the predicates and the sentences they occur in. Different ways of defining the same extensions, then, can make a theoretical difference. Not all choices yield a theory that pairs sentences with their truth-conditions. Hence not all choices lead to a theory of meaning.

2.1.3 Object language and metalanguage

Before we conclude this section, let us briefly reflect on a typographical convention that we have already been using. When we referred to words and phrases of English (represented as strings or trees), we replaced the customary quotes by bold-face. So we had, for example:

"Ann" = **Ann.**



The expressions that are bold-faced or enclosed in quotes are expressions of our *object language*, the language we are investigating. In this book, the object language is English, since we are developing a semantic theory for English. The language we use for theoretical statements is the *metalanguage*. Given that this book is written in English, our metalanguage is English as well. Since we are looking at the English object language in a fairly technical way, our English metalanguage includes a fair amount of technical vocabulary and notational

conventions. The abstraction notation for sets that we introduced earlier is an example. Quotes or typographical distinctions help us mark the distinction between object language and metalanguage. Above, we always used the bold-faced forms when we placed object language expressions between denotation brackets. For example, instead of writing the lexical entry for the name "Ann":

⟦"Ann"⟧ = Ann

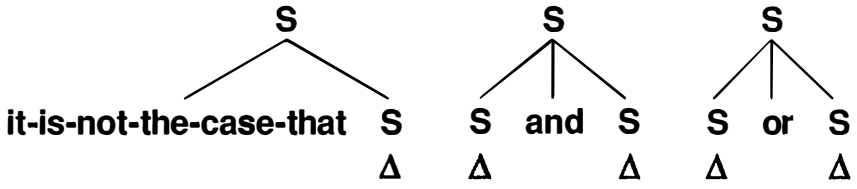
we wrote:

⟦**Ann**⟧ = Ann.

This lexical entry determines that the denotation of the English name "Ann" is the person Ann. The distinction between expressions and their denotations is important, so we will usually use *some* notational device to indicate the difference. We will never write things like "⟦Ann⟧". This would have to be read as "the denotation of (the person) Ann", and thus is nonsense. And we will also avoid using bold-face for purposes other than to replace quotes (such as emphasis, for which we use italics).

Exercise on sentence connectives

Suppose we extend our fragment to include phrase structures of the forms below (where the embedded S-constituents may either belong to the initial fragment or have one of these three forms themselves):



How do we have to revise and extend the semantic component in order to provide all the phrase structures in this expanded fragment with interpretations? Your task in this exercise is to define an appropriate semantic value for each new lexical item (treat "it-is-not-the-case-that" as a single lexical item here) and to write appropriate semantic rules for the new types of non-terminal nodes. To do this, you will also have to expand the inventory of possible semantic values. Make sure that you stick to our working hypothesis that all semantic composition is functional application (Frege's Conjecture).

2.2 Sets and their characteristic functions[5]

We have construed the denotations of intransitive verbs as functions from individuals to truth-values. Alternatively, they are often regarded as *sets* of individuals. This is the standard choice for the extensions of 1-place predicates in logic. The intuition here is that each verb denotes the set of those things that it is true of. For example: $[\![\text{sleeps}]\!] = \{x \in D : x \text{ sleeps}\}$.[6] This type of denotation would require a different semantic rule for composing subject and predicate, one that isn't simply functional application.

Exercise

Write the rule it would require.

Here we have chosen to take Frege's Conjecture quite literally, and have avoided sets of individuals as denotations for intransitive verbs. But for some purposes, sets are easier to manipulate intuitively, and it is therefore useful to be able to pretend in informal talk that intransitive verbs denote sets. Fortunately, this make-believe is harmless, because there exists a one-to-one correspondence between sets and certain functions.

(1) Let A be a set. Then char_A, the *characteristic function* of A, is that function f such that, for any $x \in A$, $f(x) = 1$, and for any $x \notin A$, $f(x) = 0$.

(2) Let f be a function with range {0, 1}. Then char_f, the set characterized by f, is $\{x \in D : f(x) = 1\}$.

Exploiting the correspondence between sets and their characteristic functions, we will often switch back and forth between function talk and set talk in the discussion below, sometimes saying things that are literally false, but become true when the references to sets are replaced by references to their characteristic functions (or vice versa).

Here is an illustration: Suppose our universe consists of three individuals, D = {Ann, Jan, Maria}. Suppose further that Ann and Jan are the ones who sleep, and Ann is the only one who snores. If we treat intransitive verbs as denoting sets, we may then assign the following denotations to sleep and snore:

(3) $[\![\text{sleep}]\!] = \{\text{Ann, Jan}\}$.

(4) $[\![\text{snore}]\!] = \{\text{Ann}\}$.

We can now write things like the following:

(5) Ann $\in$ [[sleep]].

(6) [[snore]] $\subseteq$ [[sleep]].

(7) $|[[\text{snore}]] \cap [[\text{sleep}]]| = 1$.
$|A|$ (the *cardinality* of A) is the number of elements in the set A.

(5) means that Ann is among the sleepers, (6) means that the snorers are a subset of the sleepers, and (7) means that the intersection of the snorers and the sleepers has exactly one element. All these are true, given (3) and (4). Now suppose we want to switch to a treatment under which intransitive verbs denote characteristic functions instead of the corresponding sets.

$$(3') \quad [[\text{sleep}]] = \begin{bmatrix} \text{Ann} & \rightarrow 1 \\ \text{Jan} & \rightarrow 1 \\ \text{Maria} & \rightarrow 0 \end{bmatrix}$$

$$(4') \quad [[\text{snore}]] = \begin{bmatrix} \text{Ann} & \rightarrow 1 \\ \text{Jan} & \rightarrow 0 \\ \text{Maria} & \rightarrow 0 \end{bmatrix}$$

If we want to make statements with the same import as (5)–(7) above, we can no longer use the same formulations. For instance, the statement

Ann $\in$ [[sleep]]

if we read it literally, is now false. According to (3′), [[sleep]] is a function. Functions are sets of ordered pairs, in particular,

[[sleep]] = {<Ann, 1>, <Jan, 1>, <Maria, 0>}.

Ann, who is not an ordered pair, is clearly not among the elements of this set. Likewise,

[[snore]] $\subseteq$ [[sleep]]

is now false, because there is one element of [[snore]], namely the pair <Jan, 0>, which is not an element of [[sleep]]. And

$|[\![\text{snore}]\!] \cap [\![\text{sleep}]\!]| = 1$

is false as well, because the intersection of the two functions described in (3′) and (4′) contains not just one element, but two, namely <Ann, 1> and <Maria, 0>.

The upshot of all this is that, once we adopt (3′) and (4′) instead of (3) and (4), we have to express ourselves differently if we want to make statements that preserve the intuitive meaning of our original (5)–(7). Here is what we have to write instead:

(5′) $[\![\text{sleep}]\!](\text{Ann}) = 1$

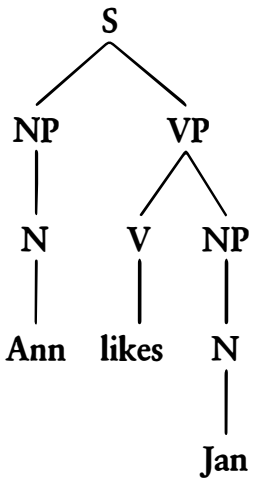
(6′) For all $x \in D$: if $[\![\text{snore}]\!](x) = 1$, then $[\![\text{sleep}]\!](x) = 1$
Or, equivalently:
$\{x : [\![\text{snore}]\!](x) = 1\} \subseteq \{x : [\![\text{sleep}]\!](x) = 1\}$

(7′) $|\{x : [\![\text{snore}]\!](x) = 1\} \cap \{x : [\![\text{sleep}]\!](x) = 1\}| = 1$

As you can see from this, using characteristic functions instead of sets makes certain things a little more cumbersome.

2.3 Adding transitive verbs: semantic types and denotation domains

Our next goal is to extend our semantics to simple transitive clauses like "Ann likes Jan". We take it that the structures that the syntax assigns to these look like this:



The transitive verb combines with its direct object to form a VP, and the VP combines with the subject to form a sentence. Given structures of this kind, what is a suitable denotation for transitive verbs? Look at the above tree. The lexical item "likes" is dominated by a non-branching V-node. The denotation of "likes", then, is passed up to this node. The next node up is a branching VP-node. Assuming that semantic interpretation is local, the denotation of this VP-node must be composed from the denotations of its two daughter nodes. If Frege's Conjecture is right, this composition process amounts to functional application. We have seen before that the denotations of NP-nodes dominating proper names are individuals. And that the denotation of VP-nodes are functions from individuals to truth-values. This means that the denotation of a transitive verb like "likes" is a *function from individuals to functions from individuals to truth-values.*[7]

When we define such a function-valued function in full explicitness, we have to nest one definition of a function inside another. Here is the proposed meaning of "likes".

$$[\![\textbf{like}]\!] = f : D \rightarrow \{g : g \text{ is a function from D to } \{0, 1\}\}$$
$$\text{For all } x \in D,\ f(x) = g_x : D \rightarrow \{0, 1\}$$
$$\text{For all } y \in D,\ g_x(y) = 1 \text{ iff } y \text{ likes } x.$$

This reads: [[like]] is that function f from D into the set of functions from D to {0, 1} such that, for all $x \in D$, f(x) is that function g_x from D into {0, 1} such that, for all $y \in D$, $g_x(y) = 1$ iff y likes x. Fortunately, this definition can be compressed a bit. There is no information lost in the following reformulation:[8]

$$[\![\textbf{like}]\!] = f : D \rightarrow \{g : g \text{ is a function from D to } \{0, 1\}\}$$
$$\text{For all } x, y \in D,\ f(x)(y) = 1 \text{ iff } y \text{ likes } x.$$

[[like]] is a 1-place function, that is, a function with just one argument. The arguments of [[like]] are interpreted as the individuals which are liked; that is, they correspond to the grammatical *object* of the verb "like". This is so because we have assumed that transitive verbs form a VP with their direct object. The direct object, then, is the argument that is closest to a transitive verb, and is therefore semantically processed before the subject.

What is left to spell out are the interpretations for the new kinds of non-terminal nodes:

(S6) If α has the form $\begin{array}{c}\text{VP}\\ \beta\ \ \gamma\end{array}$, then $[\![\alpha]\!] = [\![\beta]\!]([\![\gamma]\!])$.

What we just proposed implies an addition to our inventory of possible denotations: aside from individuals, truth-values, and functions from individuals to truth-values, we now also employ functions from individuals to functions from individuals to truth-values. It is convenient at this point to introduce a way of systematizing and labeling the types of denotations in this growing inventory. Following a common practice in the tradition of Montague, we employ the labels "e" and "t" for the two basic types.[9]

(1) e is the type of individuals.
$D_e := D$.

(2) t is the type of truth-values.
$D_t := \{0, 1\}$.

Generally, D_τ is the set of possible denotations of type τ. Besides the basic types e and t, which correspond to Frege's *saturated* denotations, there are derived types for various sorts of functions, Frege's *unsaturated* denotations. These are labeled by ordered pairs of simpler types: $\langle\sigma,\tau\rangle$ is defined as the type of functions whose arguments are of type σ and whose values are of type τ. The particular derived types of denotations that we have employed so far are <e,t> and <e,<e,t>>:

(3) $D_{\langle e,t\rangle} := \{f : f \text{ is a function from } D_e \text{ to } D_t\}$

(4) $D_{\langle e,\langle e,t\rangle\rangle} := \{f : f \text{ is a function from } D_e \text{ to } D_{\langle e,t\rangle}\}$

Further additions to our type inventory will soon become necessary. Here is a general definition:

(5) **Semantic types**
(a) e and t are semantic types.
(b) If σ and τ are semantic types, then $\langle\sigma,\tau\rangle$ is a semantic type.
(c) Nothing else is a semantic type.

Semantic denotation domains
(a) $D_e := D$ (the set of individuals).
(b) $D_t := \{0, 1\}$ (the set of truth-values).
(c) For any semantic types σ and τ, $D_{\langle\sigma,\tau\rangle}$ is the set of all functions from D_σ to D_τ.

(5) presents a recursive definition of an infinite set of semantic types and a parallel definition of a typed system of denotation domains. Which semantic

types are actually used by natural languages is still a matter of debate. The issue of "type economy" will pop up at various places throughout this book, most notably in connection with adjectives in chapter 4 and quantifier phrases in chapter 7. So far, we have encountered denotations of types e, <e,t>, and <e,<e,t>> as possible denotations for lexical items: D_e contains the denotations of proper names, $D_{<e,t>}$ the denotations of intransitive verbs, and $D_{<e,<e,t>>}$ the denotations of transitive verbs. Among the denotations of non-terminal constituents, we have seen examples of four types: D_t contains the denotations of all Ss, D_e (= D) the denotations of all Ns and NPs, $D_{<e,t>}$ the denotations of all VPs and certain Vs, and $D_{<e,<e,t.>}$ the denotations of the remaining Vs.

2.4 Schönfinkelization

Once more we interrupt the construction of our semantic component in order to clarify some of the underlying mathematics. Our current framework implies that the denotations of transitive verbs are 1-place functions. This follows from three assumptions about the syntax–semantics interface that we have been making:

Binary Branching
In the syntax, transitive verbs combine with the direct object to form a VP, and VPs combine with the subject to form a sentence.

Locality
Semantic interpretation rules are local: the denotation of any non-terminal node is computed from the denotations of its daughter nodes.

Frege's Conjecture
Semantic composition is functional application.

If transitive verbs denote 1-place function-valued functions, then our semantics contrasts with the standard semantics for 2-place predicates in logic, and it is instructive to reflect somewhat systematically on how the two approaches relate to each other.

In logic texts, the extension of a 2-place predicate is usually a set of ordered pairs: that is, a relation in the mathematical sense. Suppose our domain D contains just the three goats Sebastian, Dimitri, and Leopold, and among these, Sebastian is the biggest and Leopold the smallest. The relation "is-bigger-than" is then the following set of ordered pairs:

R_{bigger} = {<Sebastian, Dimitri>, <Sebastian, Leopold>, <Dimitri, Leopold>}.

We have seen above that there is a one-to-one correspondence between sets and their characteristic functions. The "functional version" of R_{bigger} is the following function from D × D to {0, 1}.[10]

$$f_{bigger} = \begin{bmatrix} \langle L, S\rangle \rightarrow 0 \\ \langle L, D\rangle \rightarrow 0 \\ \langle L, L\rangle \rightarrow 0 \\ \langle S, L\rangle \rightarrow 1 \\ \langle S, D\rangle \rightarrow 1 \\ \langle S, S\rangle \rightarrow 0 \\ \langle D, L\rangle \rightarrow 1 \\ \langle D, S\rangle \rightarrow 0 \\ \langle D, D\rangle \rightarrow 0 \end{bmatrix}$$

f_{bigger} is a 2-place function.[11] In his paper "Über die Bausteine der mathematischen Logik,"[12] Moses Schönfinkel showed how n-place functions can quite generally be reduced to 1-place functions. Let us apply his method to the 2-place function above. That is, let us *Schönfinkel*[13] the function f_{bigger}. There are two possibilities. f'_{bigger} is the left-to-right Schönfinkelization of f_{bigger}. f''_{bigger} is the right-to-left Schönfinkelization of f_{bigger}.

$$f'_{bigger} = \begin{bmatrix} L \rightarrow \begin{bmatrix} L \rightarrow 0 \\ S \rightarrow 0 \\ D \rightarrow 0 \end{bmatrix} \\ S \rightarrow \begin{bmatrix} L \rightarrow 1 \\ S \rightarrow 0 \\ D \rightarrow 1 \end{bmatrix} \\ D \rightarrow \begin{bmatrix} L \rightarrow 1 \\ S \rightarrow 0 \\ D \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

f'_{bigger} is a function that applies to the first argument of the "bigger" relation to yield a function that applies to the second argument. When applied to Leopold, it yields a function that maps any goat into 1 if it is smaller than Leopold. There is no such goat. Hence we get a constant function that assigns 0 to all the goats. When applied to Sebastian, f'_{bigger} yields a function that maps any goat into 1 if it is smaller than Sebastian. There are two such goats, Leopold and Dimitri. And when applied to Dimitri, f'_{bigger} yields a function that maps any goat into 1 if it is smaller than Dimitri. There is only one such goat, Leopold.

$$f''_{\text{bigger}} = \begin{bmatrix} L \rightarrow \begin{bmatrix} L \rightarrow 0 \\ S \rightarrow 1 \\ D \rightarrow 1 \end{bmatrix} \\ S \rightarrow \begin{bmatrix} L \rightarrow 0 \\ S \rightarrow 0 \\ D \rightarrow 0 \end{bmatrix} \\ D \rightarrow \begin{bmatrix} L \rightarrow 0 \\ S \rightarrow 1 \\ D \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

f''_{bigger} is a function that applies to the second argument of the "bigger" relation to yield a function that applies to the first argument. When applied to Leopold, it yields a function that maps any goat into 1 if it is bigger than Leopold. These are all the goats except Leopold. When applied to Sebastian, f''_{bigger} yields a function that maps any goat into 1 if it is bigger than Sebastian. There is no such goat. And when applied to Dimitri, f''_{bigger} maps any goat into 1 if it is bigger than Dimitri. There is only one such goat, Sebastian.

On both methods, we end up with nothing but 1-place functions, and this is as desired. This procedure can be generalized to any n-place function. You will get a taste for this by doing the exercises below.

Now we can say how the denotations of 2-place predicates construed as relations are related to the Fregean denotations introduced above. The Fregean denotation of a 2-place predicate is the right-to-left Schönfinkeled version of the characteristic function of the corresponding relation. Why the *right-to-left* Schönfinkelization? Because the corresponding relations are customarily specified in such a way that the grammatical object argument of a predicate corresponds to the *right* component of each pair in the relation, and the subject to the left one. (For instance, by the "love"-relation one ordinarily means the set of lover–loved pairs, in this order, and not the set of loved–lover pairs.) That's an arbitrary convention, in a way, though suggested by the linear order in which English realizes subjects and objects. As for the Fregean denotations of 2-place predicates, remember that it is not arbitrary that their (unique) argument corresponds to the grammatical object of the predicate. Since the object is closest to the predicate in hierarchical terms, it must provide the argument for the function denoted by the predicate.

Exercise 1

Suppose that our universe D contains just two elements, Jacob and Maria. Consider now the following binary and ternary relations:

R_{adores} = {<Jacob, Maria>, <Maria, Maria>}
$R_{assigns\ to}$ = {<Jacob, Jacob, Maria>, <Maria, Jacob, Maria>}

In standard predicate logic, these would be suitable extensions for the 2-place and 3-place predicate letters "F^2" and "G^3" as used in the following scheme of abbreviation:

"F^2": "a adores b"
"G^3": "a assigns b to c"

Find the characteristic functions for both of these relations, and then Schönfinkel them from right to left. Could the two Schönfinkeled functions be suitable denotations for the English verbs "adore" and "assign (to)" respectively? If yes, why? If not, why not?

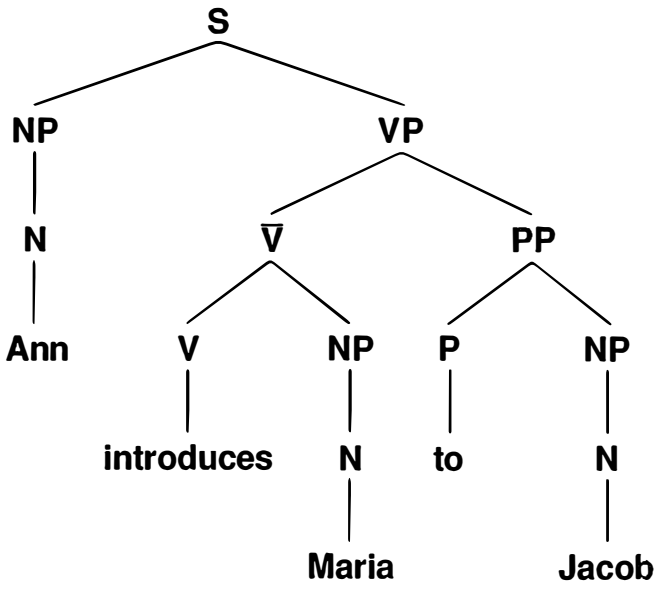
Exercise 2

In the exercise on sentence connectives in section 2.1, we stipulated ternary branching structures for sentences with "and" and "or". Now assume that all English phrase structures are at most binary branching, and assign accordingly revised syntactic analyses to these sentences. (Whether you choose right-branching or left-branching structures does not matter here, but stick to one option.) Then revise the semantics accordingly. As always, be sure to provide every subtree with a semantic value, as well as to adhere to our current assumptions about the semantic interpretation component (Locality and Frege's Conjecture).

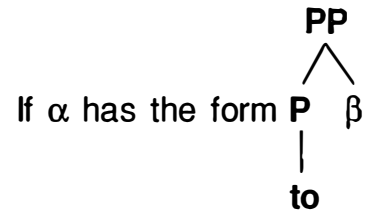
Using the labeling system introduced at the end of section 2.3, specify the type of denotation for each node in your binary branching structure for the sentence "Jan works, and it is not the case that Jan smokes".

Exercise 3

(a) Extend the fragment in such a way that phrase structure trees of the following kind are included.



Add the necessary semantic rules and lexical entries, sticking to Locality and Frege's Conjecture. Assume that the preposition "to" is a semantically vacuous element; that is, assume the *ad hoc* rule below:



(b) Suppose now that the actual world contains just three individuals, Ann, Maria, and Jacob. And suppose further that Ann introduces Maria to Jacob, and Maria introduces Jacob to Ann, and no further introductions take place. Which particular function is ⟦**introduce**⟧ in this case? Display it in a *table*.

(c) Using the *table* specification of ⟦**introduce**⟧ from (b) and the lexical entries for the names, calculate the denotations of each non-terminal node in the tree under (a).

(d) Under standard assumptions, a predicate logic formalization of the English sentence "Ann introduces Maria to Jacob" might look like this:

I^3 (A M J)
Scheme of abbreviation:
"A": "Ann"
"M": "Maria"
"J": "Jacob"
"I^3": "a introduces b to c"

The extension of "I[3]" under this scheme of abbreviation is the following set X of ordered triples:

$X := \{\langle x, y, z\rangle \in D \times D \times D : x \text{ introduces } y \text{ to } z\}$.

How is this extension related to the extension – let's call it f – for **introduce** that you defined in (a)? Give your answer by completing the following statement:

For any $x, y, z \in D$, $f(x)(y)(z) = 1$ iff $\ldots \in X$.

2.5 Defining functions in the λ-notation

The final section of this chapter is devoted to yet another technical matter. You have already had a taste of the ubiquity of functions among the denotations that our Fregean semantics assigns to the words and phrases of natural languages. We will now introduce another notation for describing functions, which will save us some ink in future chapters.

The format in which we have defined most of our functions so far was introduced in section 1.3 with the following example:

(1) $F_{+1} = f : \mathbb{N} \rightarrow \mathbb{N}$
For every $x \in \mathbb{N}$, $f(x) = x + 1$.

The same definition may henceforth be expressed as follows:

(2) $F_{+1} := [\lambda x : x \in \mathbb{N} . x + 1]$

The λ-term, "$\lambda x : x \in \mathbb{N} . x + 1$", is to be read as "the (smallest) function which maps every x such that x is in IN to x + 1".

Generally, λ-terms are constructed according to the following schema:

(3) $[\lambda\alpha : \phi . \gamma]$

We say that α is the *argument variable*, φ the *domain condition*, and γ the *value description*. The domain condition is introduced by a colon, and the value description by a period.[14] α will always be a letter standing for an arbitrary argument of the function we are trying to define. In (2), this is the letter "x", which we generally use to stand for arbitrary individuals. The domain condition φ defines the domain of our function, and it does this by placing a condition on

possible values for α. In our example, ϕ corresponds to "$x \in \mathbb{N}$", which encodes the information that the domain of F_{+1} contains all and only the natural numbers. The value description γ, finally, specifies the value that our function assigns to the arbitrary argument represented by α. In (2), this reads "$x + 1$", which tells us that the value that F_{+1} assigns to each argument is that argument's successor. The general convention for reading λ-terms in (semi-mathematical) English is such that (3) reads as (3′):

(3′) the smallest function which maps every α such that ϕ to γ

We will typically omit "smallest", but it is always understood, and it is strictly speaking necessary to pick out the intended function uniquely. Notice, for instance, that besides F_{+1}, there are lots of other functions which also map every natural number to its successor: namely, all those functions which are supersets of F_{+1}, but have larger domains than $\mathbb{N}$. By adding "smallest", we make explicit that the domain condition ϕ delimits the domain exactly; that is, that in (2), for instance, "$x \in \mathbb{N}$" is not only a sufficient, but also a necessary, condition for "$x \in \mathrm{dom}(F_{+1})$".[15]

Like other function terms, λ-terms can be followed by argument terms. So we have:

(4) $[\lambda x : x \in \mathbb{N} \,.\, x + 1](5) = 5 + 1 = 6.$

The λ-notation as we have just introduced it is not as versatile as the format in (1). It cannot be used to abbreviate descriptions of functions which involve a distinction between two or more cases. To illustrate this limitation, let's look at the function G defined in (5).

(5) $G = f : \mathbb{N} \to \mathbb{N}$
For every $x \in \mathbb{N}$, $f(x) = 2$, if x is even, and $f(x) = 1$ otherwise.

The problem we encounter if we attempt to press this into the shape "$[\lambda\alpha : \phi \,.\, \gamma]$" is that there is no suitable value description γ. Obviously, neither "1" nor "2" is the right choice. $[\lambda x : x \in \mathbb{N} \,.\, 1]$ would be that function which maps every natural number to 1, clearly a different function from the one described in (5). And similarly, of course, "$[\lambda x : x \in \mathbb{N} \,.\, 2]$" would be inappropriate.[16] This implies that, as it stands, the new notation is unsuitable for precisely the kinds of functions that figure most prominently in our semantics. Take the extension of an intransitive verb.

(6) $[\![\textbf{smoke}]\!] = f : D \to \{0, 1\}$
For every $x \in D$, $f(x) = 1$ iff x smokes.

(6) stipulates that f(x) is to be 1 if x smokes and 0 otherwise.[17] So we face the same difficulty as with (5) above when it comes to deciding on the value description in a suitable λ-term.

We will get rid of this difficulty by defining an extended use of the λ-notation. So far, you have been instructed to read "[λα : ϕ . γ]" as "the function which maps every α such that ϕ to γ". This paraphrase makes sense only when the value description γ is a noun phrase. If γ had the form of a sentence, for instance, we would get little more than word salad. Try reading out a λ-term like "[λx : x ∈ ℕ . x is even]". What comes out is: "the function which maps every x in ℕ to x is even". This is neither colloquial English nor any kind of technical jargon; it just doesn't make any sense.

We could, of course, change the instructions and stipulate a different way to read the notation. Suppose we decided that "[λα : ϕ . γ]" was to be read as follows: "the function which maps every α such that ϕ to 1 if γ and to 0 otherwise". Then, of course, "[λx : x ∈ ℕ . x is even]", would make perfect sense: "the function which maps every x in ℕ to 1, if x is even, and to 0 otherwise". With this new convention in force, we could also use the λ-notation to abbreviate our lexical entry for "smoke".

(7) $[\![\textbf{smoke}]\!] := [\lambda x : x \in D \,.\, x \text{ smokes}]$

If this reads: "let $[\![\textbf{smoke}]\!]$ be the function which maps every x in D to 1, if x smokes, and to 0 otherwise", it is easily seen as an equivalent reformulation of (6). If we add an argument term, we have:

(8) $[\![\textbf{smoke}]\!](\text{Ann}) = [\lambda x : x \in D \,.\, x \text{ smokes}](\text{Ann}) = 1$ if Ann smokes
$= 0$ otherwise.

Instead of (8), we'll write:

(8′) $[\![\textbf{smoke}]\!](\text{Ann}) = [\lambda x : x \in D \,.\, x \text{ smokes}](\text{Ann}) = 1$ iff Ann smokes.

The down side of substituting this new convention for reading λ-terms for the previous one would be that it makes garbage of those cases which we considered at first. For instance, we could no longer write things like "[λx : x ∈ ℕ . x + 1]" if this had to be read: "the function which maps every x in ℕ to 1, *if x + 1* [sic], and to 0 otherwise".

We will have our cake and eat it too, by stipulating that λ-terms may be read in either one of the two ways which we have considered, whichever happens to make sense in the case at hand.

(9) Read "[λα : ϕ . γ]" as either (i) or (ii), whichever makes sense.
  (i) "the function which maps every α such that ϕ to γ"
  (ii) "the function which maps every α such that ϕ to 1, if γ, and to 0 otherwise"

Luckily, this convention doesn't create any ambiguity, because only one clause will apply in each given case. If γ is a sentence, that's clause (ii), otherwise (i).[18]

You may wonder why the same notation has come to be used in two distinct senses. Wouldn't it have been wiser to have two different notations? This is a legitimate criticism. There does exist a precise *and uniform* interpretation of the λ-notation, where λ-terms can be shown to have the same meaning in the two cases, after all, despite the superficial disparity of the English paraphrases. This would require a formalization of our current informal metalanguage, however. We would map phrase structure trees into expressions of a λ-calculus, which would in turn be submitted to semantic interpretation.[19] Here, we use λ-operators and variables informally in the metalanguage, and rely on a purely intuitive grasp of the technical locutions and notations involving them (as is the practice, by the way, in most mathematical and scientific texts). Our use of the λ-notation in the metalanguage, then, has the same status as our informal use of other technical notation, the abstraction notation for sets, for example.

The relative conciseness of the λ-notation makes it especially handy for the description of function-valued functions. Here is how we can express the lexical entry of a transitive verb.

(10) $[\![\textbf{love}]\!] := [\lambda x : x \in D . [\lambda y : y \in D . y \text{ loves } x]]$

In (10), we have a big λ-term, whose value description is a smaller λ-term. Which clause of the reading convention in (9) applies to each of these? In the smaller one ("[λy : y ∈ D . y loves x]"), the value description is evidently sentential, so we must use clause (ii) and read this as "the function which maps every y in D to 1, if y loves x, and to 0 otherwise". And since this phrase is a noun phrase, clause (i) must apply for the bigger λ-term, and thus that one reads: "the function which maps every x in D to the function which maps every y in D to 1, if y loves x, and to 0 otherwise".

Functions can have functions as arguments. Here, too, the λ-notation is handy. Take:

(11) $[\lambda f : f \in D_{<e,t>} . \text{ there is some } x \in D_e \text{ such that } f(x) = 1]$

The function in (11) maps functions from $D_{<e,t>}$ into truth-values. Its arguments, then, are functions from individuals to truth-values. The function in (12) is a possible argument:

(12) $[\lambda y : y \in D_e . y \text{ stinks}]$

If we apply the function in (11) to the function in (12), we get the value 1 if there is some $x \in D_e$ such that $[\lambda y : y \in D_e . y \text{ stinks}](x) = 1$. Otherwise, the value is 0. That is, we have:

(13) $[\lambda f : f \in D_{<e,t>} .$ there is some $x \in D_e$ such that $f(x) = 1]([\lambda y : y \in D_e .$ y stinks$]) = 1$
iff there is some $x \in D_e$ such that $[\lambda y : y \in D_e . y \text{ stinks}](x) = 1$
iff there is some $x \in D_e$ such that x stinks.

Let us introduce a couple of abbreviatory conventions which will allow us to describe the most common types of functions we will be using in this book even more concisely. First, we will sometimes omit the outermost brackets around a λ-term which is not embedded in a larger formal expression. Second, we will contract the domain condition when it happens to be of the form "$\alpha \in \beta$". Instead of "$\lambda\alpha : \alpha \in \beta . \gamma$", we will then write "$\lambda\alpha \in \beta . \gamma$". (This corresponds to shortening the paraphrase "every α such that α is in β" to "every α in β".) And sometimes we will leave out the domain condition altogether, notably when it happens to be "$x \in D$". So the lexical entry for "love" may appear, for example, in either of the following shorter versions:

(14) $[\![\textbf{love}]\!] := \lambda x \in D . [\lambda y \in D . y \text{ loves } x]$
$[\![\textbf{love}]\!] := \lambda x . [\lambda y . y \text{ loves } x]$

You have to be careful when λ-terms are followed by argument terms. Without further conventions, 15(a) is not legitimate, since it is ambiguous between 15(b) and 15(c), which are not equivalent:

(15) (a) $\lambda x \in D . [\lambda y \in D . y \text{ loves } x](\text{Sue})$
(b) $[\lambda x \in D . [\lambda y \in D . y \text{ loves } x](\text{Sue})] = \lambda x \in D . \text{Sue loves } x$
(c) $[\lambda x \in D . [\lambda y \in D . y \text{ loves } x]](\text{Sue}) = \lambda y \in D . y \text{ loves Sue}$

In cases of this kind, use either the formulation in (b) or the one in (c), whichever corresponds to the intended meaning.

There is a close connection between the abstraction notation for sets and the λ-notation for functions. The characteristic function of the set $\{x \in \mathbb{N} : x \neq 0\}$ is $[\lambda x \in \mathbb{N} . x \neq 0]$, for example. Much of what you have learned about the abstraction notation for sets in chapter 1 can now be carried over to the λ-notation for functions. Set talk can be easily translated into function talk. Here are the correspondences for some of the cases we looked at earlier:

Set talk	**Function talk**
$29 \in \{x \in \mathbb{N} : x \neq 0\}$ iff $29 \neq 0$	$[\lambda x \in \mathbb{N} . x \neq 0](29) = 1$ iff $29 \neq 0$
Massachusetts $\in \{x \in D :$ California is a western state$\}$ iff California is a western state.	$[\lambda x \in D .$ California is a western state$]$(Massachusetts) = 1 iff California is a western state.
$\{x \in D :$ California is a western state$\} = D$ if California is a western state. $\{x \in D :$ California is a western state$\} = \emptyset$ if California is not a western state.	$[\lambda x \in D .$ California is a western state$](x) = 1$ for all $x \in D$ if California is a western state. $[\lambda x \in D .$ California is a western state$](x) = 0$ for all $x \in D$ if California is not a western state.
$\{x \in \mathbb{N} : x \neq 0\} = \{y \in \mathbb{N} : y \neq 0\}$	$[\lambda x \in \mathbb{N} : x \neq 0] = [\lambda y \in \mathbb{N} : y \neq 0]$
$\{x \in \mathbb{N} : x \in \{x \in \mathbb{N} : x \neq 0\}\} = \{x \in \mathbb{N} : x \neq 0\}$	$[\lambda x \in \mathbb{N} . [\lambda x \in \mathbb{N} . x \neq 0](x)] = [\lambda x \in \mathbb{N} . x \neq 0]$
$\{x \in \mathbb{N} : x \in \{y \in \mathbb{N} : y \neq 0\}\} = \{x \in \mathbb{N} : x \neq 0\}$	$[\lambda x \in \mathbb{N} . [\lambda y \in \mathbb{N} . y \neq 0](x)] = [\lambda x \in \mathbb{N} . x \neq 0]$

If you are still unclear about some of the statements in the left column, go back to chapter 1 and consult the questions and answers about the abstraction notation for sets. Once you understand the set talk in the left column, the transition to the function talk in the right column should be straightforward.

Exercise 1

Describe the following functions in words:

(a) $\lambda x \in \mathbb{N} . x > 3$ and $x < 7$
(b) $\lambda x :$ x is a person . x's father
(c) $\lambda X \in \text{Pow}(D) . \{y \in D : y \notin X\}$[20]
(d) $\lambda X : X \subseteq D . [\lambda y \in D . y \notin X]$

Exercise 2

In this exercise, simple functions are described in a rather complicated way. Simplify the descriptions as much as possible.

(a) $[\lambda x \in D . [\lambda y \in D . [\lambda z \in D . z \text{ introduced } x \text{ to } y]]](Ann)(Sue)$
(b) $[\lambda x \in D . [\lambda y \in D . [\lambda z \in D . z \text{ introduced } x \text{ to } y](Ann)](Sue)]$
(c) $[\lambda x \in D . [\lambda y \in D . [\lambda z \in D . z \text{ introduced } x \text{ to } y](Ann)]](Sue)$
(d) $[\lambda x \in D . [\lambda y \in D . [\lambda z \in D . z \text{ introduced } x \text{ to } y]](Ann)](Sue)$
(e) $[\lambda f \in D_{<e,t>} . [\lambda x \in D_e . f(x) = 1 \text{ and } x \text{ is gray}]]([\lambda y \in D_e . y \text{ is a cat}])$
(f) $[\lambda f \in D_{<e,<e,t>>} . [\lambda x \in D_e . f(x)(Ann) = 1]]([\lambda y \in D_e . [\lambda z \in D_e . z \text{ saw } y]])$
(g) $[\lambda x \in |N . [\lambda y \in |N . y > 3 \text{ and } y < 7](x)]$
(h) $[\lambda z \in |N . [\lambda y \in |N . [\lambda x \in |N . x > 3 \text{ and } x < 7](y)](z)]$

Exercise 3

Suppose "and" and "or" have Schönfinkeled denotations, that is **[[and]]** and **[[or]]** are both members of $D_{<t,<t,t>>}$. They are functions that map truth-values into functions from truth-values to truth-values. Specify the two functions using the λ-notation.

Exercise 4

Replace the "?" in each of the following statements (you may want to review definition (5) of section 2.3 before tackling this exercise):

(a) $[\lambda f \in D_{<e,t>} . [\lambda x \in D_e . f(x) = 1 \text{ and } x \text{ is gray}]] \in D_?$
(b) $[\lambda f \in D_{<e,<e,t>>} . [\lambda x \in D_e . f(x)(Ann) = 1]] \in D_?$
(c) $[\lambda y \in D_e . [\lambda f \in D_{<e,t>} . [\lambda x \in D_e . f(x) = 1 \text{ and } x \text{ is in } y]]] \in D_?$
(d) $[\lambda f \in D_{<e,t>} . \text{there is some } x \in D_e \text{ such that } f(x) = 1] \in D_?$
(e) $[\lambda f \in D_{<e,t>} . Mary] \in D_?$
(f) $[\lambda f \in D_{<e,t>} . [\lambda g \in D_{<e,t>} . \text{there is no } x \in D_e \text{ such that } f(x) = 1 \text{ and } g(x) = 1]] \in D_?$

Notes

1 Here and below, when we speak of the denotation of a *node* in a tree, we really mean the denotation of the subtree dominated by that node.
2 M. Black and P. Geach, *Translations from the Philosophical Writings of Gottlob Frege* (Oxford, Basil Blackwell, 1960), use "reference" to translate Frege's "Bedeutung".

Some translations, for instance, in A. P. Martinich (ed.), *The Philosophy of Language*, 2nd edn (New York and Oxford, Oxford University Press, 1990), and J. L. Garfield and M. Kiteley (eds), *The Essential Readings in Modern Semantics* (New York, Paragon House, 1991), use "nominatum". This is the translation Rudolf Carnap introduced in *Meaning and Necessity. A Study in Semantics and Modal Logic* (Chicago, University of Chicago Press, 1947).

3 M. Dummett, *Frege. Philosophy of Language*, 2nd edn (Cambridge, Mass., Harvard University Press, 1981), p. 227.

4 The use of Wittgenstein's term "show" in this connection is due to Dummett, ibid.

5 This is another mathematical-background section, which the mathematically sophisticated need only skim.

6 The notation "{x ∈ D : x sleeps}" is a standard abbreviation for "{x : x ∈ D and x sleeps}". (Recall that when we first introduced the set-abstraction notation, we allowed only a variable to the left of the colon.)

7 This conclusion, even though we are motivating it by using his general proposal about semantic composition, Frege himself would not have endorsed. As discussed by Dummett (*Frege*, pp. 40ff.), he did not allow for function-valued functions.

8 Notice that the implicit bracketing in "f(x)(y)" is "[f(x)](y)", not "f[(x)(y)]". The latter wouldn't make any sense. (What could we possibly mean by "(x)(y)"?) So it is not necessary to make the correct parse explicit in the notation.

9 R. Montague, *Formal Philosophy* (New Haven, Yale University Press, 1974); "e" is for "*e*ntity", "t" for "*t*ruth-value".

10 D × D is the *Cartesian product* of D with D, which is defined as the set of ordered pairs of elements of D.

11 A 2-place function on a domain A is a function with domain A × A, which is defined as {<x, y> : x ∈ A and y ∈ A}.

12 *Mathematische Annalen*, 92 (1924), pp. 305–16.

13 This procedure is also called "Currying" after the logician H. B. Curry, who built on Schönfinkel's work. Molly Diesing informs us that by Stephen Jay Gould's "Brontosaurus principle", one could argue that we should use "Currying": generality of use takes priority over temporal precedence. We are not sure how general the use of "Currying" is at this time, however, hence we don't know whether the Brontosaurus Principle applies to this case. We'll stick to temporal priority, then. W. Kneale and M. Kneale, *The Development of Logic* (Oxford, Clarendon Press, 1962), pp. 522f., credit Schönfinkel for Schönfinkelization.

14 Most versions of the λ-notation in the literature look a little different. What we are calling the "domain condition" is typically absent, and the intended domain is indicated instead by using argument variables that are assigned to fixed semantic types. The value description is frequently enclosed in brackets rather than introduced by a period. The terms "argument variable", "domain condition", and "value description" are also our own invention.

15 The attentive reader may have noticed that another piece of information seems to get lost in the reformulation from (1) to (2): viz. information about which set F_{+1} is *into*. Nothing in (2) corresponds to the part "→ ℕ" in (1). Is this a problem? No. If you go back to our initial definition of "function" in section 1.3, you can see that the information supplied in (2) is entirely sufficient to define a unique function. It already follows from (2) that all values of F_{+1} are in ℕ. In other words, the format employed in (1) is actually redundant in this respect.

16 The best we can do, if we insist on using the λ-notation to define G, is to describe G as the union of two separate functions with smaller domains:

$G := \lambda x : x \in \mathbb{N}$ & x is even . $2 \cup \lambda x : x \in \mathbb{N}$ & x is odd . 1

But this is not much of an *abbreviation* of (5). Remember, we were planning to save ink.

17 Incidentally, the indication that f is into {0, 1} is not redundant in (6). Without this information, we could not conclude that f(x) = 0 when x doesn't smoke. From the condition "f(x) = 1 iff x smokes" by itself, we can infer only that $f(x) \neq 1$ in this case. If we want to render "$\rightarrow$ {0, 1}" as redundant here as "$\rightarrow \mathbb{N}$" was in (1) and (5), we have to replace "f(x) = 1 iff x smokes" by something like the formulation in the text below: "f(x) = 1 if x smokes and 0 otherwise'.

18 Even with this disjunctive convention, by the way, the λ-notation remains unsuitable for abbreviating definitions like the one for G in (5). But we can live with this, because functions like G are not common in semantic applications.

19 A. Church, "A Formulation of a Simple Theory of Types," *Journal of Symbolic Logic*, 5 (1940), pp. 56–68. See also L. T. F. Gamut, *Logic, Language and Meaning*, vol. 2: *Intensional Logic and Logical Grammar* (Chicago, University of Chicago Press, 1991).

20 "Pow(D)" reads "the power set of D". The power set of a set is the set of all its subsets. (More formally: for any set X, $Pow(X) := \{Y : Y \subseteq X\}$.)