Blackwell Textbooks in Linguistics

The books included in this series provide comprehensive accounts of some of the most central and most rapidly developing areas of research in linguistics. Intended primarily for introductory and post-introductory students, they include exercises, discussion points and suggestions for further reading.

1. Liliane Haegeman
   Introduction to Government and Binding Theory (Second Edition)

2. Andrew Spencer
   Morphological Theory

3. Helen Goodluck
   Language Acquisition

4. Ronald Wardhaugh
   Introduction to Sociolinguistics (Fourth Edition)

5. Martin Atkinson
   Children’s Syntax

6. Diane Blakeimore
   Understanding Utterances

7. Michael Kenstowicz
   Phonology in Generative Grammar

8. Deborah Schiffrin
   Approaches to Discourse

9. John Clark and Colin Yallop
   An Introduction to Phonetics and Phonology (Second Edition)

10. Natsuko Tsujimura
    An Introduction to Japanese Linguistics

11. Robert D. Borsley
    Modern Phrase Structure Grammar

12. Nigel Fabb
    Linguistics and Literature

13. Irene Heim and Angelika Kratzer
    Semantics in Generative Grammar

14. Liliane Haegeman and Jacqueline Guérón
    English Grammar: A Generative Perspective

15. Stephen Crain and Diane Lillo-Martin
    An Introduction to Linguistic Theory and Language Acquisition

16. Joan Bresnan
    Lexical-Functional Syntax

17. Barbara A. Fennell
    A History of English: A Sociolinguistic Approach

18. Henry Rogers
    Writing Systems: A Linguistic Approach

19. Benjamin W. Fortson IV
    Indo-European Language and Culture: An Introduction

Semantics in Generative Grammar

Irene Heim and Angelika Kratzer
Massachusetts Institute of Technology and University of Massachusetts at Amherst
Contents

Preface ix

1 Truth-conditional Semantics and the Fregean Program 1
1.1 Truth-conditional semantics 1
1.2 Frege on compositionality 2
1.3 Tutorial on sets and functions 3
1.3.1 Sets 4
1.3.2 Questions and answers about the abstraction notation for sets 5
1.3.3 Functions 10

2 Executing the Fregean Program 13
2.1 First example of a Fregean interpretation 13
2.1.1 Applying the semantics to an example 16
2.1.2 Deriving truth-conditions in an extensional semantics 20
2.1.3 Object language and metalanguage 22
2.2 Sets and their characteristic functions 24
2.3 Adding transitive verbs: semantic types and denotation domains 26
2.4 Schönfinkelization 29
2.5 Defining functions in the \( \lambda \)-notation 34

3 Semantics and Syntax 43
3.1 Type-driven interpretation 43
3.2 The structure of the input to semantic interpretation 45
3.3 Well-formedness and interpretability 47
3.4 The \( \Theta \)-Criterion 49
3.5 Argument structure and linking 53

4 More of English: Nonverbal Predicates, Modifiers, Definite Descriptions 61
4.1 Semantically vacuous words 61
4.2 Nonverbal predicates 62
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Predicates as restrictive modifiers</td>
<td>63</td>
</tr>
<tr>
<td>4.3.1 A new composition rule</td>
<td>65</td>
</tr>
<tr>
<td>4.3.2 Modification as functional application</td>
<td>66</td>
</tr>
<tr>
<td>4.3.3 Evidence from nonintersective adjectives?</td>
<td>68</td>
</tr>
<tr>
<td>4.4 The definite article</td>
<td>73</td>
</tr>
<tr>
<td>4.4.1 A lexical entry inspired by Frege</td>
<td>73</td>
</tr>
<tr>
<td>4.4.2 Partial denotations and the distinction between</td>
<td>75</td>
</tr>
<tr>
<td>presupposition and assertion</td>
<td></td>
</tr>
<tr>
<td>4.4.3 Uniqueness and utterance context</td>
<td>80</td>
</tr>
<tr>
<td>4.4.4 Presupposition failure versus uninterpretablity</td>
<td>81</td>
</tr>
<tr>
<td>4.5 Modifiers in definite descriptions</td>
<td>82</td>
</tr>
<tr>
<td>5 Relative Clauses, Variables, Variable Binding</td>
<td>86</td>
</tr>
<tr>
<td>5.1 Relative clauses as predicates</td>
<td>86</td>
</tr>
<tr>
<td>5.2 Semantic composition inside the relative clause</td>
<td>89</td>
</tr>
<tr>
<td>5.2.1 Does the trace pick up a referent?</td>
<td>90</td>
</tr>
<tr>
<td>5.2.2 Variables</td>
<td>92</td>
</tr>
<tr>
<td>5.2.3 Predicate abstraction</td>
<td>96</td>
</tr>
<tr>
<td>5.2.4 A note on proof strategy: bottom up or top down?</td>
<td>99</td>
</tr>
<tr>
<td>5.3 Multiple variables</td>
<td>106</td>
</tr>
<tr>
<td>5.3.1 Adding “such that” relatives</td>
<td>106</td>
</tr>
<tr>
<td>5.3.2 A problem with nested relatives</td>
<td>108</td>
</tr>
<tr>
<td>5.3.3 Amending the syntax: co-indexing</td>
<td>109</td>
</tr>
<tr>
<td>5.3.4 Amending the semantics</td>
<td>110</td>
</tr>
<tr>
<td>5.4 What is variable binding?</td>
<td>115</td>
</tr>
<tr>
<td>5.4.1 Some semantic definitions</td>
<td>116</td>
</tr>
<tr>
<td>5.4.2 Some theorems</td>
<td>120</td>
</tr>
<tr>
<td>5.4.3 Methodological remarks</td>
<td>121</td>
</tr>
<tr>
<td>5.5 Interpretability and syntactic constraints on indexing</td>
<td>123</td>
</tr>
<tr>
<td>6 Quantifiers: Their Semantic Type</td>
<td>131</td>
</tr>
<tr>
<td>6.1 Problems with individuals as DP-denotations</td>
<td>131</td>
</tr>
<tr>
<td>6.1.1 Predictions about truth-conditions and</td>
<td>132</td>
</tr>
<tr>
<td>entailment patterns</td>
<td></td>
</tr>
<tr>
<td>6.1.2 Predictions about ambiguity and the effects of</td>
<td>135</td>
</tr>
<tr>
<td>syntactic reorganization</td>
<td></td>
</tr>
<tr>
<td>6.2 Problems with having DPs denote sets of individuals</td>
<td>138</td>
</tr>
<tr>
<td>6.3 The solution: generalized quantifiers</td>
<td>140</td>
</tr>
<tr>
<td>6.3.1 “Something”, “nothing”, “everything”</td>
<td>140</td>
</tr>
<tr>
<td>6.3.2 Problems avoided</td>
<td>142</td>
</tr>
<tr>
<td>6.4 Quantifying determiners</td>
<td>145</td>
</tr>
<tr>
<td>6.5 Quantifier meanings and relations between sets</td>
<td>147</td>
</tr>
<tr>
<td>6.5.1 A little history</td>
<td>147</td>
</tr>
<tr>
<td>6.5.2 Relational and Schönfinkel denotations for determiners</td>
<td>149</td>
</tr>
<tr>
<td>6.6 Formal properties of relational determiner meanings</td>
<td>151</td>
</tr>
<tr>
<td>6.7 Presuppositional quantifier phrases</td>
<td>153</td>
</tr>
<tr>
<td>6.7.1 “Both” and “neither”</td>
<td>154</td>
</tr>
<tr>
<td>6.7.2 Presuppositionality and the relational theory</td>
<td>154</td>
</tr>
<tr>
<td>6.7.3 Other examples of presupposing DPs</td>
<td>157</td>
</tr>
<tr>
<td>6.8 Presuppositional quantifier phrases: controversial cases</td>
<td>159</td>
</tr>
<tr>
<td>6.8.1 Strawson’s reconstruction of Aristotelian logic</td>
<td>159</td>
</tr>
<tr>
<td>6.8.2 Are all determiners presuppositional?</td>
<td>162</td>
</tr>
<tr>
<td>6.8.3 Nonextensional interpretation</td>
<td>165</td>
</tr>
<tr>
<td>6.8.4 Nonpresuppositional behavior in weak determiners</td>
<td>170</td>
</tr>
<tr>
<td>7 Quantification and Grammar</td>
<td>178</td>
</tr>
<tr>
<td>7.1 The problem of quantifiers in object position</td>
<td>178</td>
</tr>
<tr>
<td>7.2 Repairing the type mismatch in situ</td>
<td>179</td>
</tr>
<tr>
<td>7.2.1 An example of a “flexible types” approach</td>
<td>180</td>
</tr>
<tr>
<td>7.2.2 Excursion: flexible types for connectives</td>
<td>182</td>
</tr>
<tr>
<td>7.3 Repairing the type mismatch by movement</td>
<td>184</td>
</tr>
<tr>
<td>7.4 Excursion: quantifiers in natural language and predicate logic</td>
<td>190</td>
</tr>
<tr>
<td>7.4.1 Separating quantifiers from variable binding</td>
<td>189</td>
</tr>
<tr>
<td>7.4.2 1-place and 2-place quantifiers</td>
<td>190</td>
</tr>
<tr>
<td>7.5 Choosing between quantifier movement and in situ</td>
<td>193</td>
</tr>
<tr>
<td>interpretation: three standard arguments</td>
<td></td>
</tr>
<tr>
<td>7.5.1 Scope ambiguity and “inverse” scope</td>
<td>194</td>
</tr>
<tr>
<td>7.5.2 Antecedent-contained deletion</td>
<td>198</td>
</tr>
<tr>
<td>7.5.3 Quantifiers that bind pronouns</td>
<td>200</td>
</tr>
<tr>
<td>8 Syntactic and Semantic Constraints on Quantifier Movement</td>
<td>209</td>
</tr>
<tr>
<td>8.1 Which DPs may move, and which ones must?</td>
<td>210</td>
</tr>
<tr>
<td>8.2 How much moves along? And how far can you move?</td>
<td>212</td>
</tr>
<tr>
<td>8.3 What are potential landing sites for moving quantifiers?</td>
<td>214</td>
</tr>
<tr>
<td>8.4 Quantifying into VP</td>
<td>215</td>
</tr>
<tr>
<td>8.4.1 Quantifiers taking narrow scope with respect to</td>
<td>215</td>
</tr>
<tr>
<td>auxiliary negation</td>
<td></td>
</tr>
<tr>
<td>8.4.2 Quantifying into VP, VP-internal subjects, and flexible types</td>
<td>217</td>
</tr>
<tr>
<td>8.5 Quantifying into PP, AP, and NP</td>
<td>221</td>
</tr>
<tr>
<td>8.5.1 A problem of undergeneration</td>
<td>221</td>
</tr>
<tr>
<td>8.5.2 PP-internal subjects</td>
<td>225</td>
</tr>
<tr>
<td>8.5.3 Subjects in all lexically headed XPs?</td>
<td>228</td>
</tr>
</tbody>
</table>
Contents

8.6 Quantifying into DP
   8.6.1 Readings that can only be represented by DP adjunction? 230
   8.6.2 Indirect evidence for DP adjunction: a problem with free IP adjunction? 233
   8.6.3 Summary 234

9 Bound and Referential Pronouns and Ellipsis
   9.1 Referential pronouns as free variables 239
      9.1.1 Deictic versus anaphoric, referential versus bound-variable pronouns 239
      9.1.2 Utterance contexts and variable assignments 242
   9.2 Co-reference or binding? 245
   9.3 Pronouns in the theory of ellipsis 248
      9.3.1 Background: the LF Identity Condition on ellipsis 248
      9.3.2 Referential pronouns and ellipsis 252
      9.3.3 The “sloppy identity” puzzle and its solution 254

10 Syntactic and Semantic Binding
    10.1 Indexing and Surface Structure binding 260
    10.2 Syntactic binding, semantic binding, and the Binding Principle 262
    10.3 Weak Crossover 265
    10.4 The Binding Principle and strict and sloppy identity 266
    10.5 Syntactic constraints on co-reference? 269
    10.6 Summary 274

11 E-Type Anaphora
    11.1 Review of some predictions 277
    11.2 Referential pronouns with quantifier antecedents 280
    11.3 Pronouns that are neither bound variables nor referential 286
    11.4 Paraphrases with definite descriptions 288
    11.5 Cooper’s analysis of E-Type pronouns 290
    11.6 Some applications 293

12 First Steps Towards an Intensional Semantics
    12.1 Where the extensional semantics breaks down 299
    12.2 What to do: intensions 301
    12.3 An intensional semantics 303
    12.4 Limitations and prospects 310

Index 313

Preface

This book is an introduction to the craft of doing formal semantics for linguists. It is not an overview of the field and its current developments. The many recent handbooks provide just that. We want to help students develop the ability for semantic analysis, and, in view of this goal, we think that exploring a few topics in detail is more effective than offering a bird’s-eye view of everything. We also believe that foundational and philosophical matters can be better discussed once students have been initiated into the practice of semantic argumentation. This is why our first chapter is so short. We dive right in.

The students for whom we created the lectures on which this book is based were graduate or advanced undergraduate students of linguistics who had already had a basic introduction to some formal theory of syntax and had a first understanding of the division of labor between semantics and pragmatics. Not all of them had had an introduction to logic or set theory. If necessary, we filled in gaps in formal background with the help of other books.

We learned our craft from our teachers, students, and colleagues. While working on this book, we were nurtured by our families and friends. Paul Hirschbühler, Molly Diesing, Kai von Fintel, Jim Higginbotham, the students in our classes, and reviewers whose names we don’t know gave us generous comments we could use. The staff of Blackwell Publishers helped us to turn lecture notes we passed back and forth between the two of us into a book. We thank them all.

Irene Heim and Angelika Kratzer
Cambridge, Mass., and Amherst, April 1997
1 Truth-conditional Semantics and the Fregean Program

1.1 Truth-conditional semantics

To know the meaning of a sentence is to know its truth-conditions. If I say to you

(1) There is a bag of potatoes in my pantry

you may not know whether what I said is true. What you do know, however, is what the world would have to be like for it to be true. There has to be a bag of potatoes in my pantry. The truth of (1) can come about in ever so many ways. The bag may be paper or plastic, big or small. It may be sitting on the floor or hiding behind a basket of onions on the shelf. The potatoes may come from Idaho or northern Maine. There may even be more than a single bag. Change the situation as you please. As long as there is a bag of potatoes in my pantry, sentence (1) is true.

A theory of meaning, then, pairs sentences with their truth-conditions. The results are statements of the following form:

Truth-conditions
The sentence “There is a bag of potatoes in my pantry” is true if and only if there is a bag of potatoes in my pantry.

The apparent banality of such statements has puzzled generations of students since they first appeared in Alfred Tarski’s 1935 paper “The Concept of Truth in Formalized Languages.” Pairing English sentences with their truth-conditions seems to be an easy task that can be accomplished with the help of a single schema:

Schema for truth-conditions
The sentence “_____” is true if and only if _____.
A theory that produces such schemata would indeed be trivial if there wasn't another property of natural language that it has to capture: namely, that we understand sentences we have never heard before. We are able to compute the meaning of sentences from the meanings of their parts. Every meaningful part of a sentence contributes to its truth-conditions in a systematic way. As Donald Davidson put it:

The theory reveals nothing new about the conditions under which an individual sentence is true; it does not make those conditions any clearer than the sentence itself does. The work of the theory is in relating the known truth conditions of each sentence to those aspects ("words") of the sentence that recur in other sentences, and can be assigned identical roles in other sentences. Empirical power in such a theory depends on success in recovering the structure of a very complicated ability—the ability to speak and understand a language.¹

In the chapters that follow, we will develop a theory of meaning composition. We will look at sentences and break them down into their parts. And we will think about the contribution of each part to the truth-conditions of the whole.

1.2 Frege on compositionality

The semantic insights we rely on in this book are essentially those of Gottlob Frege, whose work in the late nineteenth century marked the beginning of both symbolic logic and the formal semantics of natural language. The first worked-out versions of a Fregean semantics for fragments of English were by Lewis, Montague, and Cresswell.²

It is astonishing what language accomplishes. With a few syllables it expresses a countless number of thoughts, and even for a thought grasped for the first time by a human it provides a clothing in which it can be recognized by another to whom it is entirely new. This would not be possible if we could not distinguish parts in the thought that correspond to parts of the sentence, so that the construction of the sentence can be taken to mirror the construction of the thought. . . If we thus view thoughts as composed of simple parts and take these, in turn, to correspond to simple sentence-parts, we can understand how a few sentence-parts can go to make up a great multitude of sentences to which, in turn, there correspond a great multitude of thoughts. The question now arises how the construction of the thought proceeds, and by what means the parts are put together so that the whole is something more than the isolated parts. In my essay "Negation," I considered the case of a thought that appears to be composed of one part which is in need of completion or, as one might say, unsaturated, and whose linguistic correlate is the negative particle, and another part which is a thought. We cannot negate without negating something, and this something is a thought. Because this thought saturates the unsaturated part or, as one might say, completes what is in need of completion, the whole hangs together. And it is a natural conjecture that logical combination of parts into a whole is always a matter of saturating something unsaturated.⁴

Frege, like Aristotle and his successors before him, was interested in the semantic composition of sentences. In the above passage, he conjectured that semantic composition may always consist in the saturation of an unsaturated meaning component. But what are saturated and unsaturated meanings, and what is saturation? Here is what Frege had to say in another one of his papers.

Statements in general, just like equations or inequalities or expressions in Analysis, can be imagined to be split up into two parts; one complete in itself, and the other in need of supplementation, or "unsaturated." Thus, e.g., we split up the sentence "Caesar conquered Gaul" into "Caesar" and "conquered Gaul." The second part is "unsaturated"—it contains an empty place; only when this place is filled up with a proper name, or with an expression that replaces a proper name, does a complete sense appear. Here too I give the name "function" to what this "unsaturated" part stands for. In this case the argument is Caesar.⁵

Frege construed unsaturated meanings as functions. Unsaturated meanings, then, take arguments, and saturation consists in the application of a function to its arguments. Technically, functions are sets of a certain kind. We will therefore conclude this chapter with a very informal introduction to set theory. The same material can be found in the textbook by Partee et al.⁶ and countless other sources. If you are already familiar with it, you can skip this section and go straight to the next chapter.

1.3 Tutorial on sets and functions

If Frege is right, functions play a crucial role in a theory of semantic composition. "Function" is a mathematical term, and formal semanticists nowadays use it in
exactly the way in which it is understood in modern mathematics. Since functions are sets, we will begin with the most important definitions and notational conventions of set theory.

1.3.1 Sets

A set is a collection of objects which are called the "members" or "elements" of that set. The symbol for the element relation is "∈". "x ∈ A" reads "x is an element of A". Sets may have any number of elements, finite or infinite. A special case is the empty set (symbol "∅"), which is the (unique) set with zero elements.

Two sets are equal if they have exactly the same members. Sets that are not equal may have some overlap in their membership, or they may be disjoint (have no members in common). If all the members of one set are also members of another, the former is a subset of the latter. The subset relation is symbolized by "⊂ ".

A ⊆ B reads "A is a subset of B".

There are a few standard operations by which new sets may be constructed from given ones. Let A and B be two arbitrary sets. Then the intersection of A and B (in symbols: A ∩ B) is that set which has as elements exactly the members that A and B share with each other. The union of A and B (in symbols: A ∪ B) is the set which contains all the members of A and all the members of B and nothing else. The complement of A in B (in symbols: B - A) is the set which contains precisely those members of B which are not in A.

Specific sets may be defined in various ways. A simple possibility is to define a set by listing its members, as in (1).

(1) Let A be the set whose elements are a, b, and c, and nothing else.

A more concise rendition of (1) is (1').

(1') A := {a, b, c}.

Another option is to define a set by abstraction. This means that one specifies a condition which is to be satisfied by all and only the elements of the set to be defined.

(2) Let A be the set of all cats.

(2') Let A be that set which contains exactly those x such that x is a cat.

(2'), of course, defines the same set as (2); it just uses a more convoluted formulation. There is also a symbolic rendition:

(2'') A := {x : x is a cat}.

Read "{x : x is a cat}" as "the set of all x such that x is a cat". The letter "x" here isn't meant to stand for some particular object. Rather, it functions as a kind of place-holder or variable. To determine the membership of the set A defined in (2''), one has to plug in the names of different objects for the "x" in the condition "x is a cat". For instance, if you want to know whether Kaline ∈ A, you must consider the statement "Kaline is a cat". If this statement is true, then Kaline ∈ A; if it is false, then Kaline ∈ A ("x ∈ A" means that x is not an element of A).

1.3.2 Questions and answers about the abstraction notation for sets

Q1: If the "x" in "{x : x is a positive integer less than 7}" is just a place-holder, why do we need it at all? Why don't we just put a blank as in "{ _ : _ is a positive integer less than 7}"?

A1: That may work in simple cases like this one, but it would lead to a lot of confusion and ambiguity in more complicated cases. For example, which set would be meant by "{ _ : _ likes _ = ∅ }"? Would it be, for instance, the set of objects which don't like anything, or the set of objects which nothing likes? We certainly need to distinguish these two possibilities (and also to distinguish them from a number of additional ones). If we mean the first set, we write "{x : [y : x likes y} = ∅]". If we mean the second set, we write "{x : [y : y likes x} = ∅]".

Q2: Why did you just write "{x : [y : y likes x} = ∅]" rather than "{y : [x : x likes y} = ∅]"?

A2: No reason. The second formulation would be just as good as the first, and they specify exactly the same set. It doesn't matter which letters you choose; it only matters in which places you use the same letter, and in which places you use different ones.

Q3: Why do I have to write something to the left of the colon? Isn't the condition on the right side all we need to specify the set? For example, instead of "{x : x is a positive integer less than 7}" wouldn't it be good enough to write simply "{x is a positive integer less than 7}"?
A3: You might be able to get away with it in the simplest cases, but not in more complicated ones. For example, what we said in A1 and A2 implies that the following two are different sets:

\[ \{x : \{y : x \text{ likes } y\} = \emptyset\} \]
\[ \{y : \{x : x \text{ likes } y\} = \emptyset\} \]

Therefore, if we just wrote "\(\{x : y \text{ likes } y\} = \emptyset\)", it would be ambiguous. A mere statement enclosed in set braces doesn't mean anything at all, and we will never use the notation in this way.

Q4: What does it mean if I write "\{California : California is a western state\}"?

A4: Nothing, it doesn't make any sense. If you want to give a list specification of the set whose only element is California, write "\{California\}". If you want to give a specification by abstraction of the set that contains all the western states and nothing else but those, the way to write it is "\{x : x \text{ is a western state}\}". The problem with what you wrote is that you were using the name of a particular individual in a place where only place-holders make sense. The position to the left of the colon in a set-specification must always be occupied by a place-holder, never by a name.

Q5: How do I know whether something is a name or a place-holder? I am familiar with "California" as a name, and you have told me that "x" and "y" are place-holders. But how can I tell the difference in other cases? For example, if I see the letter "a" or "d" or "s", how do I know if it's a name or a place-holder?

A5: There is no general answer to this question. You have to determine from case to case how a letter or other expression is used. Sometimes you will be told in so many words that the letters "b", "c", "t", and "u" are made-up names for certain individuals. Other times, you have to guess from the context. One very reliable clue is whether the letter shows up to the left of the colon in a set-specification. If it does, it had better be meant as a place-holder rather than a name; otherwise it doesn't make any sense. Even though there is no general way of telling names apart from place-holders, we will try to minimize sources of confusion and stick to certain notational conventions (at least most of the time). We will normally use letters from the end of the alphabet as place-holders, and letters from the beginning of the alphabet as names. Also we will never employ words that are actually used as names in English (like "California" or "John") as place-holders. (Of course, we could so use them if we wanted to, and then we could also write things like "\{California : California is a western state\}", and it would be just another way of describing the set \{x : x \text{ is a western state}\}. We could, but we won't.)

Q6: In all the examples we have had so far, the place-holder to the left of the colon had at least one occurrence in the condition on the right. Is this necessary for the notation to be used properly? Can I describe a set by means of a condition in which the letter to the left of the colon doesn't show up at all? What about "\{x : California is a western state\}"?

A6: This is a strange way to describe a set, but it does pick out one thing. Which one? Well, let's see whether, for instance, Massachusetts qualifies for membership in it. To determine this, we take the condition "California is a western state" and plug in "Massachusetts" for all the "x"s in it. But there are no "x"s, so the result of this "plug-in" operation is simply "California is a western state" again. Now this happens to be true, so Massachusetts has passed the test of membership. That was trivial, of course, and it is evident now that any other object will qualify as a member just as easily. So \{x : California is a western state\} is the set containing everything there is. (Of course, if that's the set we mean to refer to, there is no imaginable good reason why we'd choose this of all descriptions.) If you think about it, there are only two sets that can be described at all by means of conditions that don't contain the letter to the left of the colon. One, as we just saw, is the set of everything; the other is the empty set. The reason for this is that when a condition doesn't contain any "x" in it, then it will either be true regardless of what value we assign to "x", or it will be false regardless of what value we assign to "x".

Q7: When a set is given with a complicated specification, I am not always sure how to figure out which individuals are in it and which ones aren't. I know how to do it in simpler cases. For example, when the set is specified as "\{x : x + 2 = x^2\}" and I want to know whether, say, the number 29 is in it, I know what I have to do: I have to replace all occurrences of "x" in the condition that follows the colon by occurrences of "29", and then decide whether the resulting statement about 29 is true or false. In this case, I get the statement "29 + 2 = 29^2"; and since this is false, 29 is not in the set. But there are cases where it's not so easy. For example, suppose a set is specified as "\{x : x \in \{x : x \neq 0\}\}" and I want to figure out whether 29 is in this one. So I try replacing "x" with "29" on the right side of the colon. What I get is "29 \in \{29 : 29 \neq 0\}". But I don't understand this. We just learned that names can't occur to the left of the colon; only place-holders make sense there. This looks just like the example "\{California : California is a western state\}" that I brought up in Q5. So I am stuck. Where did I go wrong?
A7: You went wrong when you replaced all the “x” by “29” and thereby went from “\(\{x : x \in \{x : x \neq 0\}\}\)” to “\(29 \in \{29 : 29 \neq 0\}\)”. The former makes sense, the latter doesn’t (as you just noted yourself); so this cannot have been an equivalent reformulation.

Q8: Wait a minute, how was I actually supposed to know that “\(\{x : x \in \{x : x \neq 0\}\}\)” made sense? For all I knew, this could have been an incoherent definition in the first place, and my reformulation just made it more transparent what was wrong with it.

A8: Here is one way to see that the original description was coherent, and this will also show you how to answer your original question: namely, whether 29 \(\in\) \(\{x : x \in \{x : x \neq 0\}\}\). First, look only at the most embedded set description, namely “\(\{x : x \neq 0\}\)”. This transparently describes the set of all objects distinct from 0. We can refer to this set in various other ways: for instance, in the way I just did (as “the set of all objects distinct from 0”), or by a new name that we especially define for it, say as “\(S := \{x : x \neq 0\}\)”, or by “\(\{y : y \neq 0\}\)”. Given that the set “\(\{x : x \neq 0\}\)” can be referred to in all these different ways, we can also express the condition “\(x \in \{x : x \neq 0\}\)” in many different, but equivalent, forms — for example, these three:

“\(x \in \) the set of all objects distinct from 0"
“\(x \in S\) (where \(S\) is as defined above)"
“\(x \in \{y : y \neq 0\}\)"

Each of these is fulfilled by exactly the same values for “\(x\)” as the original condition “\(x \in \{x : x \neq 0\}\)”. This, in turn, means that each can be substituted for “\(x \in \{x : x \neq 0\}\)” in “\(\{x : x \in \{x : x \neq 0\}\}\)”, without changing the set that is thereby defined. So we have:

\[
\begin{align*}
\{x : x \in \{x : x \neq 0\}\} & = \{x : x \in \text{the set of all objects distinct from 0}\} \\
& = \{x : x \in S\} \text{ (where } S \text{ is as defined above)} \\
& = \{x : x \in \{y : y \neq 0\}\}.
\end{align*}
\]

Now if we want to determine whether 29 is a member of \(\{x : x \in \{x : x \neq 0\}\}\), we can do this by using any of the alternative descriptions of this set. Suppose we take the third one above. So we ask whether 29 \(\in\) \(\{x : x \in S\}\). We know that it is if and only if 29 \(\in\) \(\{x : x \neq 0\}\). And this in turn is the case if and only if 29 \(\neq 0\). Now we have arrived at an obviously true statement, and we can work our way back and conclude, first, that 29 \(\in\) \(S\), second, that 29 \(\in\) \(\{x : x \in S\}\), and third, that 29 \(\in\) \(\{x : x \in \{x : x \neq 0\}\}\).

Q9: I see for this particular case now that it was a mistake to replace all occurrences of “x” in the condition “\(x \in \{x : x \neq 0\}\)” by “29”. But I am still not confident that I wouldn’t make a similar mistake in another case. Is there a general rule or fool-proof strategy that I can follow so that I’ll be sure to avoid such illegal substitutions?

A9: A very good policy is to write (or rewrite) your conditions in such a way that there is no temptation for illegal substitutions in the first place. This means that you should never reuse the same letter unless this is strictly necessary in order to express what you want to say. Otherwise, use new letters wherever possible. If you follow this strategy, you won’t ever write something like “\(\{x : x \in \{x : x \neq 0\}\}\)” to begin with, and if you happen to read it, you will quickly rewrite it before doing anything else with it. What you would write instead would be something like “\(\{x : x \in \{y : y \neq 0\}\}\)”. This (as we already noted) describes exactly the same set, but uses distinct letters “x” and “y” instead of only “x”s. It still uses each letter twice, but this, of course, is crucial to what it is meant to express. If we insisted on replacing the second “x” by a “z”, for instance, we would wind up with one of those strange descriptions in which the “x” doesn’t occur to the right of the colon at all, that is, “\(\{z : z \in \{y : y \neq 0\}\}\)”. As we saw earlier, sets described in this way contain either everything or nothing. Besides, what is “z” supposed to stand for? It doesn’t seem to be a place-holder, because it’s not introduced anywhere to the left of a colon. So it ought to be a name. But whatever it is a name of, that thing was not referred to anywhere in the condition that we had before changing “x” to “z”, so we have clearly altered its meaning.

Exercise

The same set can be described in many different ways, often quite different superficially. Here you are supposed to figure out which of the following equalities hold and which ones don’t. Sometimes the right answer is not just plain “yes” or “no”, but something like “yes, but only if . . .”. For example, the two sets in (i) are equal only in the special case where \(a = b\). In case of doubt, the best way to check whether two sets are equal is to consider an arbitrary individual, say John, and to ask if John could be in one of the sets without being in the other as well.

\[
\begin{align*}
(i) & \quad \{a \} = \{b \} \\
(ii) & \quad \{x : x = a\} = \{a\}
\end{align*}
\]
1.3.3 Functions

If we have two objects x and y (not necessarily distinct), we can construct from them the ordered pair \( \langle x, y \rangle \). \( \langle x, y \rangle \) must not be confused with \( \{x, y\} \). Since sets with the same members are identical, we always have \( \{x, y\} = \{y, x\} \). But in an ordered pair, the order matters: except in the special case of \( x = y \), \( \langle x, y \rangle \neq \langle y, x \rangle \).

A \((2\text{-place})\) relation is a set of ordered pairs. Functions are a special kind of relation. Roughly speaking, in a function (as opposed to a non-functional relation), the second member of each pair is uniquely determined by the first. Here is the definition:

3. A relation \( f \) is a function iff it satisfies the following condition:
   For any \( x \): if there are \( y \) and \( z \) such that \( \langle x, y \rangle \in f \) and \( \langle x, z \rangle \in f \), then \( y = z \).

Each function has a domain and a range, which are the sets defined as follows:

4. Let \( f \) be a function.
   Then the domain of \( f \) is \( \{ x \text{ where there is a } y \text{ such that } \langle x, y \rangle \in f \} \), and the range of \( f \) is \( \{ x \text{ where there is a } y \text{ such that } \langle y, x \rangle \in f \} \).

When \( A \) is the domain and \( B \) the range of \( f \), we also say that \( f \) is from \( A \) and onto \( B \). If \( C \) is a superset\(^1\) of \( f \)'s range, we say that \( f \) is into (or to) \( C \). For “\( f \) is from \( A \) (into) \( B \)”, we write “\( f: A \rightarrow B \)”. The uniqueness condition built into the definition of functionhood ensures that whenever \( f \) is a function and \( x \) an element of its domain, the following definition makes sense:

5. \( f(x) = \) the unique \( y \) such that \( \langle x, y \rangle \in f \).

For “\( f(x) \)” read “\( f \) applied to \( x \)” or “\( f \) of \( x \)”. \( f(x) \) is also called the “value of \( f \) for the argument \( x \)”, and we say that \( f \) maps \( x \) to \( y \). “\( f(x) = y \)” (provided that it is well-defined at all) means the same thing as “\( \langle x, y \rangle \in f \)” and is normally the preferred notation.

Functions, like sets, can be defined in various ways, and the most straightforward one is again to simply list the function’s elements. Since functions are sets of ordered pairs, this can be done with the notational devices we have already introduced, as in (6), or else in the form of a table like the one in (7), or in words such as (8).

(6) \( F := \{<a, b>, <c, b>, <d, e>\} \)

(7) \[
\begin{array}{c}
\text{in} \\
\text{range}
\end{array}
\begin{array}{c}
a \\
b \\
d \\
e
\end{array}
\]

(8) Let \( F \) be that function \( f \) with domain \([a, c, d]\) such that \( f(a) = f(c) = b \) and \( f(d) = e \).

Each of these definitions determines the same function \( F \). The convention for reading tables like the one in (7) is transparent: the left column lists the domain and the right column the range, and an arrow points from each argument to the value it is mapped to.

Functions with large or infinite domains are often defined by specifying a condition that is to be met by each argument-value pair. Here is an example.

(9) Let \( F_{+1} \) be that function \( f \) such that
   \( f: \mathbb{N} \rightarrow \mathbb{N} \), and for every \( x \in \mathbb{N} \), \( f(x) = x + 1 \).
   (\( \mathbb{N} \) is the set of all natural numbers.)

The following is a slightly more concise format for this sort of definition:

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

Read (10) as: “\( F_{+1} \) is to be that function \( f \) from \( \mathbb{N} \) into \( \mathbb{N} \) such that, for every \( x \in \mathbb{N} \), \( f(x) = x + 1 \).” An even more concise notation (using the \( \lambda \)-operator) will be introduced at the end of the next chapter.

Notes

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).
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 “smokes” 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 N-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 \( \alpha \), then, \([\alpha]\) is the denotation of \( \alpha \). We can think of \([\cdot]\) 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

\[
\begin{align*}
[\text{Ann}] &= \text{Ann} \\
[\text{Jan}] &= \text{Jan} \\
\text{etc. for other proper names.}
\end{align*}
\]

\[
\begin{align*}
[\text{works}] &= f : D \to [0, 1] \\
&\quad \text{For all } x \in D, f(x) = 1 \text{ iff } x \text{ works.}
\end{align*}
\]

\[
\begin{align*}
[\text{smokes}] &= f : D \to [0, 1] \\
&\quad \text{For all } x \in D, f(x) = 1 \text{ iff } x \text{ smokes.}
\end{align*}
\]

etc. for other intransitive verbs.

C. Rules for non-terminal nodes
In what follows, Greek letters are used as variables for trees and subtrees.
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

\[
S \\
NP \quad VP \\
N \quad V \\
Ann \quad \text{smokes}
\]

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

(S1) If \( \alpha \) has the form \( \langle \land \rangle \), then \( \llbracket \alpha \rrbracket = \llbracket \beta \rrbracket \llbracket \gamma \rrbracket \).

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, \( \alpha \) is

\[
S \\
NP \quad VP \\
N \quad V \\
Ann \quad \text{smokes}
\]

so \( \beta \) must be \( NP \) and \( \gamma \) must be \( VP \).
The rule says that $[\alpha] = [\gamma][\beta]$, so this means in the present application that

(2) \[
\begin{array}{c}
S \\
NP & VP \\
N & V \\
Ann & smokes
\end{array}
\] = \[
\begin{array}{c}
VP \\
V \\
smokes \\
Ann
\end{array}
\] \[
\begin{array}{c}
NP \\
N \\
(smokes) \\
Ann
\end{array}
\]

Now we apply rule (S3) to the tree \[
\begin{array}{c}
VP \\
V \\
smokes
\end{array}
\]

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

(3) \[
\begin{array}{c}
VP \\
V \\
smokes
\end{array}
\] = \[
\begin{array}{c}
V \\
smokes
\end{array}
\]

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

(4) \[
\begin{array}{c}
S \\
NP & VP \\
N & V \\
Ann & smokes
\end{array}
\] = \[
\begin{array}{c}
V \\
smokes \\
Ann
\end{array}
\] \[
\begin{array}{c}
NP \\
N \\
(smokes) \\
Ann
\end{array}
\]

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

(5) \[
\begin{array}{c}
S \\
NP & VP \\
N & V \\
Ann & smokes
\end{array}
\] = \[
\begin{array}{c}
NP \\
N \\
(smokes) \\
Ann
\end{array}
\]

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

(6) \[
\begin{array}{c}
S \\
NP & VP \\
N & V \\
Ann & smokes
\end{array}
\] = \[
\begin{array}{c}
(smokes)[([Ann])]
\end{array}
\]

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) \[
\begin{array}{c}
S \\
NP & VP \\
N & V_i \\
Ann & smokes
\end{array}
\] = \[
\begin{array}{c}
f : D \rightarrow \{0, 1\} \\
For all x \in D, f(x) = 1 \iff x \text{ smokes}
\end{array}
\]

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:
Executing the Fregean Program

(8) \[ f : D \rightarrow \{0, 1\} \]
\[ \text{For all } x \in D, f(x) = 1 \text{ iff } x \text{ smokes.} \]

(Ann) = 1 \text{ iff } Ann \text{ smokes.}

And now we have reached the goal of our proof: (7) and (8) together imply the exact 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:

\[
\begin{array}{c}
\text{[smokes]} = \\
\text{Ann} \rightarrow 1 \\
\text{Jan} \rightarrow 1 \\
\text{Maria} \rightarrow 0
\end{array}
\]

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

\[
\begin{array}{c}
\begin{array}{c}
\text{S} \\
\text{NP} & \text{VP} \\
\text{N} & \text{V} \\
\text{Ann} & \text{smokes}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{Ann} \rightarrow 1 \\
\text{Ann} \rightarrow 1 \\
\text{Maria} \rightarrow 0
\end{array}
\end{array}
\]

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". 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” 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:


```
    S
  NP   VP
     |   |
    N  V   N  V
```

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:


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):

```
S
```

```
S S
```

```
S or S
```

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

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: \([\text{sleep}] = \{x \in D : x \text{ sleeps}\}\). 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 \(\text{char}_A\), 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 \(\text{char}_f\), 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, \(D = \{\text{Ann}, \text{Jan}, \text{Maria}\}\). Suppose further that \(\text{Ann}\) and \(\text{Jan}\) are the ones who sleep, and \(\text{Ann}\) is the only one who snores. If we treat intransitive verbs as denoting sets, we may then assign the following denotations to \textit{sleep} and \textit{snores}:

\[
\begin{align*}
(3) \quad [\text{sleep}] &= \{\text{Ann}, \text{Jan}\}.
\end{align*}
\]

\[
\begin{align*}
(4) \quad [\text{snores}] &= \{\text{Ann}\}.
\end{align*}
\]

We can now write things like the following:

\[
\begin{align*}
(5) \quad \text{Ann} &\in [\text{sleep}].
\end{align*}
\]

\[
\begin{align*}
(6) \quad [\text{snores}] &\subseteq [\text{sleep}].
\end{align*}
\]

\[
\begin{align*}
(7) \quad [\text{snores}] \cap [\text{sleep}] &= 1.
\quad |A| (\text{the cardinality of } A) \text{ is the number of elements in the set } A.
\end{align*}
\]

(5) means that \(\text{Ann}\) 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.

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

\[
\begin{align*}
(4') \quad [\text{snores}] &= \begin{bmatrix}
\text{Ann} & 1 \\
\text{Jan} & 0 \\
\text{Maria} & 0
\end{bmatrix}
\end{align*}
\]

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

\[
\begin{align*}
\text{Ann} &\in [\text{sleep}]
\end{align*}
\]

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

\[
\begin{align*}
[\text{sleep}] &= \{<\text{Ann}, 1>, <\text{Jan}, 1>, <\text{Maria}, 0>\}.
\end{align*}
\]

\(\text{Ann}\), who is not an ordered pair, is clearly not among the elements of this set. Likewise,

\[
\begin{align*}
[\text{snores}] &\subseteq [\text{sleep}].
\end{align*}
\]

is now false, because there is one element of \([\text{snores}]\), namely the pair \(<\text{Jan}, 0>\), which is not an element of \([\text{sleep}]\). And
Executing the Fregean Program

\[ [\text{snore}] \land [\text{sleep}] = 1 \]

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

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:

\[
\begin{array}{c}
S \\
NP \quad VP \\
N \quad V \quad NP \\
\text{Ann} \quad \text{likes} \quad \text{N} \\
\end{array}
\]

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.

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".

\[ [\text{like}] = f : D \to [g : g \text{ is a function from } D \text{ to } \{0, 1\}] \]

For all \(x \in D\), \(f(x) = g_x : D \to \{0, 1\}\)

\[ \text{For all } y \in D, g_x(y) = 1 \text{ iff } y \text{ likes } x. \]

This reads: \([\text{like}]\) 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:

\[ [\text{like}] = f : D \to [g : g \text{ is a function from } D \text{ to } \{0, 1\}] \]

For all \(x, y \in D\), \(f(x)(y) = 1\) iff \(y\) likes \(x\).

\([\text{like}]\) is a 1-place function, that is, a function with just one argument. The arguments of \([\text{like}]\) 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 \(\alpha\) has the form \(\beta \gamma\), 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, \( D_i \) is the set of possible denotations of type \( i \). 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 \( \sigma \) and whose values are of type \( \tau \). The particular derived types of denotations that we have employed so far are \( \langle e, t \rangle \) and \( \langle e, e, t, t \rangle \):

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

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

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 \( \sigma \) and \( \tau \) 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 \( \sigma \) and \( \tau \), \( D_{e, \sigma, \tau} \) is the set of all functions from \( D_e \) to \( D_\tau \).

(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 \( \langle e, t \rangle \) and \( \langle e, e, t, t \rangle \) as possible denotations for lexical items: \( D_e \) contains the denotations of proper names, \( D_{e, t} \), the denotations of intransitive verbs, and \( D_{e, e, t, t} \), the denotations of transitive verbs. Among the denotations of non-terminal constituents, we have seen examples of four types: \( D_e \) contains the denotations of all Ss, \( D_{t, t} (= D) \) the denotations of all Ns and NPs, \( D_{e, e, t, t} \) the denotations of all VPs and certain Vs, and \( D_{e, e, t, t, t, t} \) 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:
Executing the Fregean Program

\( R_{\text{bigger}} = \{<\text{Sebastian, Dimitri}>, <\text{Sebastian, Leopold}>, <\text{Dimitri, Leopold}>, \} \).

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

\[
\begin{align*}
\text{f}_{\text{bigger}} &= \begin{cases} 
<L, S> & \rightarrow 0 \\
<L, D> & \rightarrow 0 \\
<L, L> & \rightarrow 0 \\
<S, L> & \rightarrow 1 \\
<S, D> & \rightarrow 1 \\
<S, S> & \rightarrow 0 \\
<D, L> & \rightarrow 0 \\
<D, S> & \rightarrow 0 \\
<D, D> & \rightarrow 0
\end{cases}
\end{align*}
\]

\( f_{\text{bigger}} \) is a 2-place function.\(^1\) In his paper "Über die Bausteine der mathematischen Logik,"\(^2\) 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 \( \text{Schönfinkel} \)\(^3\) the function \( f_{\text{bigger}} \). There are two possibilities. \( f'_{\text{bigger}} \) is the left-to-right Schönfinkelization of \( f_{\text{bigger}} \). \( f''_{\text{bigger}} \) is the right-to-left Schönfinkelization of \( f_{\text{bigger}} \).

\[
\begin{align*}
\text{f}'_{\text{bigger}} &= \begin{cases} 
L & \rightarrow 0 \\
S & \rightarrow 0 \\
D & \rightarrow 0 \\
L & \rightarrow 1 \\
S & \rightarrow 0 \\
D & \rightarrow 1 \\
L & \rightarrow 1 \\
S & \rightarrow 0 \\
D & \rightarrow 0
\end{cases}
\end{align*}
\]

\( f'_{\text{bigger}} \) 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, \( f'_{\text{bigger}} \) 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, \( f'_{\text{bigger}} \) yields a function that maps any goat into 1 if it is smaller than Dimitri. There is only one such goat, Leopold.

\[
\begin{align*}
\text{f}''_{\text{bigger}} &= \begin{cases} 
L & \rightarrow 0 \\
S & \rightarrow 1 \\
D & \rightarrow 1 \\
L & \rightarrow 0 \\
S & \rightarrow 0 \\
D & \rightarrow 0
\end{cases}
\end{align*}
\]

\( f''_{\text{bigger}} \) 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, \( f''_{\text{bigger}} \) 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, \( f''_{\text{bigger}} \) 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 Schönfinkelized 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:
$R_{addresses} = \{ <Jacob, Maria>, <Maria, Maria> \}$
$R_{assigns to} = \{ <Jacob, Jacob, Maria>, <Maria, Jacob, Maria> \}$

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

"$F^a$": "a adores b"
"$G^a$": "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 Schönfinkel 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 Freges’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.

![Phrase Structure Tree]

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

\[ \text{If } \alpha \text{ has the form } P \beta, \text{ then } [\alpha] = [\beta]. \]

(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:

$I^a (A M J)$

Scheme of abbreviation:

"A": "Ann"
"M": "Maria"
"J": "Jacob"
"$I^a$": "a introduces b to c"
The extension of \( "\mathcal{P}" \) under this scheme of abbreviation is the following set \( X \) of ordered triples:

\[
X := \{ <x, y, z> \in D \times D \times D : x \text{ introduces } y \text{ to } z \}.
\]

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 \( \ldots \in X \).

\[3.5\] **Defining functions in the \( \lambda \)-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:

\[1\] \( F_{s1} = f : \mathcal{N} \rightarrow \mathcal{N} \)

For every \( x \in \mathcal{N} \), \( f(x) = x + 1 \).

The same definition may henceforth be expressed as follows:

\[2\] \( F_{s1} := [\lambda x : x \in \mathcal{N} \cdot x + 1] \)

The \( \lambda \)-term, \( \"[\lambda x : x \in \mathcal{N} \cdot x + 1]\" \), is to be read as “the (smallest) function which maps every \( x \) such that \( x \in \mathcal{N} \) to \( x + 1 \).”

Generally, \( \lambda \)-terms are constructed according to the following schema:

\[3\] \( [\lambda \alpha : \phi \cdot \gamma] \)

We say that \( \alpha \) is the **argument variable**, \( \phi \) the **domain condition**, and \( \gamma \) the **value description**. The domain condition is introduced by a colon, and the value description by a period.\(^{14}\) \( \alpha \) 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 \( \phi \) defines the domain of our function, and it does this by placing a condition on possible values for \( \alpha \). In our example, \( \phi \) corresponds to \( "x \in \mathcal{N}" \), which encodes the information that the domain of \( F_{s1} \) contains all and only the natural numbers.

The value description \( \gamma \), finally, specifies the value that our function assigns to the arbitrary argument represented by \( \alpha \). In (2), this reads \( "x + 1" \), which tells us that the value that \( F_{s1} \) assigns to each argument is that argument’s successor.

The general convention for reading \( \lambda \)-terms in (semi-mathematical) English is such that (3) reads as (3’):

\(3'\) the smallest function which maps every \( \alpha \) such that \( \phi \) to \( \gamma \).

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 \( F_{s1} \), there are lots of other functions which also map every natural number to its successor: namely, all those functions which are supersets of \( F_{s1} \), but have larger domains than \( \mathcal{N} \). By adding “smallest”, we make explicit that the domain condition \( \phi \) delimits the domain exactly; that is, that in (2), for instance, \( "x \in \mathcal{N}" \) is not only a sufficient, but also a necessary, condition for \( "x \in \text{dom}(F_{s1})" \).\(^{15}\)

Like other function terms, \( \lambda \)-terms can be followed by argument terms. So we have:

\[4\] \( [\lambda x : x \in \mathcal{N} \cdot x + 1](5) = 5 + 1 = 6. \)

The \( \lambda \)-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).

\[5\] \( G = f : \mathcal{N} \rightarrow \mathcal{N} \)

For every \( x \in \mathcal{N} \), \( f(x) = 2 \), if \( x \) is even, and \( f(x) = 1 \) otherwise.

The problem we encounter if we attempt to press this into the shape \( "[\lambda x : \phi \cdot \gamma]" \) is that there is no suitable value description \( \gamma \). Obviously, neither “\( 1 \)” nor “\( 2 \)” is the right choice. \( [\lambda x : x \in \mathcal{N} \cdot 1] \) 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, \( "[\lambda x : x \in \mathcal{N} \cdot 2]" \) 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.

\[6\] \( [\text{smoke}] = f : \mathcal{D} \rightarrow [0, 1] \)

For every \( x \in \mathcal{D} \), \( f(x) = 1 \) 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 \( \lambda \)-term.

We will get rid of this difficulty by defining an extended use of the \( \lambda \)-notation. So far, you have been instructed to read \( \langle \lambda x : \phi . \gamma \rangle \) as “the function which maps every \( x \) such that \( \phi \) to \( \gamma \)”. This paraphrase makes sense only when the value description \( \gamma \) is a noun phrase. If \( \gamma \) had the form of a sentence, for instance, we would get little more than word salad. Try reading out a \( \lambda \)-term like \( \langle \lambda x : x \in \text{IN} . x \text{ is even} \rangle \). What comes out is: “the function which maps every \( x \) in \( \text{IN} \) to \( x \text{ is even} \)”. 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 \( \langle \lambda x : \phi . \gamma \rangle \) was to be read as follows: “the function which maps every \( x \) such that \( \phi \) to 1 if \( \gamma \) and to 0 otherwise”. Then, of course, \( \langle \lambda x : x \in \text{IN} . x \text{ is even} \rangle \), would make perfect sense: “the function which maps every \( x \) in \( \text{IN} \) to 1, if \( x \) is even, and to 0 otherwise”. With this new convention in force, we could also use the \( \lambda \)-notation to abbreviate our lexical entry for “smoke”.

(7) \([\text{smoke}] := \langle \lambda x : x \in \text{D} . x \text{ smokes} \rangle\)

If this reads: “let \([\text{smoke}] \) be the function which maps every \( x \) in \( \text{D} \) 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) \([\text{smoke}](\text{Ann}) := \langle \lambda x : x \in \text{D} . x \text{ smokes} \rangle(\text{Ann}) = 1 \text{ if Ann smokes} = 0 \text{ otherwise.}\)

Instead of (8), we’ll write:

(8') \([\text{smoke}](\text{Ann}) := [\lambda x : x \in \text{D} . x \text{ smokes}](\text{Ann}) = 1 \text{ iff Ann smokes.}\)

The down side of substituting this new convention for reading \( \lambda \)-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 “\( \langle \lambda x : x \in \text{IN} . x + 1 \rangle \)” if this had to be read: “the function which maps every \( x \) in \( \text{IN} \) to 1, if \( x + 1 \) [sic], and to 0 otherwise”.

We will have our cake and eat it too, by stipulating that \( \lambda \)-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 \( \langle \lambda x : \phi . \gamma \rangle \) as either (i) or (ii), whichever makes sense.

(i) “the function which maps every \( x \) such that \( \phi \) to \( \gamma \)”

(ii) “the function which maps every \( x \) such that \( \phi \) to 1, if \( \gamma \), and to 0 otherwise”.

Luckily, this convention doesn’t create any ambiguity, because only one clause will apply in each given case. If \( \gamma \) 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 wise to have two different notations? This is a legitimate criticism. There does exist a precise and uniform interpretation of the \( \lambda \)-notation, where \( \lambda \)-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 metalinguage, however. We would map phrase structure trees into expressions of a \( \lambda \)-calculus, which would in turn be submitted to semantic interpretation.\(^{19}\) Here, we use \( \lambda \)-operators and variables informally in the metalinguage, and rely on a purely intuitive grasp of the technical locations and notations involving them (as is the practice, by the way, in most mathematical and scientific texts). Our use of the \( \lambda \)-notation in the metalinguage, 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 \( \lambda \)-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) \([\text{love}] := [\lambda x : x \in \text{D} . [\lambda y : y \in \text{D} . y \text{ loves x}]]\)

In (10), we have a big \( \lambda \)-term, whose value description is a smaller \( \lambda \)-term. Which clause of the reading convention in (9) applies to each of these? In the smaller one (“\( [\lambda y : y \in \text{D} . y \text{ loves x}] \)”), the value description is evidently sentential, so we must use clause (ii) and read this as “the function which maps every \( y \) in \( \text{D} \) 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 \( \lambda \)-term, and thus that one reads: “the function which maps every \( x \) in \( \text{D} \) to the function which maps every \( y \) in \( \text{D} \) to 1, if \( y \) loves \( x \), and to 0 otherwise”.

Functions can have functions as arguments. Here, too, the \( \lambda \)-notation is handy. Take:

(11) \( [\lambda f : f \in D_{\alpha \rightarrow} . \text{there is some } x \in D_\alpha \text{ such that } f(x) = 1] \)

The function in (11) maps functions from \( D_{\alpha \rightarrow} \) 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 \cdot f \in D_{\text{num}} . \text{ there is some } x \in D_e \text{ such that } f(x) = 1])([\lambda y : y \in D_e . y \text{ stinks}](x)) = 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 \(\lambda\)-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 "\(\lambda \alpha : \alpha \in \beta . y\)," we will then write "\(\lambda \alpha \in \beta . y\)." (This corresponds to shortening the paraphrase "every \(\alpha\) such that \(\alpha \in \beta\)" to "every \(\alpha\) in \(\beta\).") 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) \([\text{love}] = \lambda \alpha \in D . \lambda y \in D . y \text{ loves } x\]

\([\text{love}] = \lambda x . \lambda y . y \text{ loves } x\]

You have to be careful when \(\lambda\)-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 \(\lambda\)-notation for functions. The characteristic function of the set \([x \in \text{IN} : x \neq 0]\) is \([\lambda x \in \text{IN} . x \neq 0]\), for example. Much of what you have learned about the abstraction notation for sets in chapter 1 can now be carried over to the \(\lambda\)-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

\[29 \in [x \in \text{IN} : x \neq 0] \iff 29 \neq 0\]

\[\text{Massachusetts} \in [x \in D : \text{California} \text{ is a western state}] \iff \text{California is a western state}\]

\[\{x \in D : \text{California is a western state}\} = D \text{ if California is a western state}\]

\[\{x \in D : \text{California is a western state}\} = \emptyset \text{ if California is not a western state}\]

\[x \in \text{IN} : x \neq 0 = [y \in \text{IN} : y \neq 0]\]

\[x \in \text{IN} : x \in [x \in \text{IN} : x \neq 0] = [x \in \text{IN} : x \neq 0]\]

\[x \in \text{IN} : x \in [y \in \text{IN} : y \neq 0] = [x \in \text{IN} : x \neq 0]\]

Function talk

\([\lambda x \in \text{IN} . x \neq 0](29) = 1 \iff 29 \neq 0\]

\([\lambda x \in D . \text{California is a western state}](\text{Massachusetts}) = 1 \iff \text{California is a western state}\]

\([\lambda x \in D . \text{California is a western state}](x) = 1 \text{ for all } x \in D \text{ if} \]

\([\lambda x \in D . \text{California is a western state}](x) = 0 \text{ for all } x \in D \text{ if} \]

\([\lambda x \in \text{IN} . x \neq 0]\]

\([\lambda x \in \text{IN} . x \in [x \in \text{IN} : x \neq 0]](x)\]

\([\lambda x \in \text{IN} . x \in [y \in \text{IN} : y \neq 0]](x)\]

\([\lambda x \in \text{IN} . x \in [y \in \text{IN} : y \neq 0]](x)\]

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) \(\lambda x \in [N . x > 3 \text{ and } x < 7]\)

(b) \(\lambda x : x \text{ is a person} . x's \text{ father}\)

(c) \(\lambda X \in \text{Pow}(D) . [y \in D : y \neq X]\)

(d) \(\lambda x : x \subseteq D . [y \in D . y \in X]\)

---

**Exercise 2**

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

40

(a) \([\lambda x \in D. [\lambda y \in D. [\lambda z \in D. z \text{ introduced } x \text{ to } y]](\text{Ann})\)(Sue)

(b) \([\lambda x \in D. [\lambda y \in D. [\lambda z \in D. z \text{ introduced } x \text{ to } y]](\text{Ann}) (\text{Sue})\]

(c) \([\lambda x \in D. [\lambda y \in D. [\lambda z \in D. z \text{ introduced } x \text{ to } y]](\text{Ann})\)(Sue)

(d) \([\lambda x \in D. [\lambda y \in D. [\lambda z \in D. z \text{ introduced } x \text{ to } y]](\text{Ann})\)(Sue)

(e) \([\lambda x \in D. [\lambda y \in D. f(x) = 1 \text{ and } x \text{ is gray}]][\lambda y \in D. y \text{ is a cat}]\)

(f) \([\lambda x \in D. [\lambda y \in D. f(x)(\text{Ann}) = 1]](\lambda y \in D. [\lambda z \in D. z \text{ saw } y])\)

(g) \([\lambda x \in [N. [\lambda y \in [N. y > 3 \text{ and } y < 7]x]\]

(h) \([\lambda x \in [N. [\lambda y \in [N. [\lambda z \in [N. x > 3 \text{ and } x < 7]y]](z)]\]

Exercise 3

Suppose “and” and “or” have Schönfinkelized denotations, that is [and] and [or]
are both members of D_{\lambda e h b}. They are functions that map truth-values into
functions from truth-values to truth-values. Specify the two functions using the
\lambda-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) \([D \in D. [\lambda x \in D. f(x) = 1 \text{ and } x \text{ is gray}]][D_2]

(b) \([D \in D_{\lambda e h b}. [\lambda x \in D. f(x)(\text{Ann}) = 1]](D_2)

(c) \([\lambda y \in D_2. [\lambda z \in D_{\lambda e h b}. [\lambda x \in D_2. f(x) = 1 \text!\text{ and } x \text{ is in } y]]]D_2

(d) \([\lambda x \in D_{\lambda e h b}. [\lambda y \in D_2. [\lambda x \in D_2. f(x) = 1 \text{ and } x \text{ is gray}]]]D_2

(e) \([\lambda x \in D_{\lambda e h b}. [\lambda y \in D_2. f(x) = 1 \text{ and } x \text{ is gray}]](D_2)

(f) \([\lambda x \in D_{\lambda e h b}. [\lambda y \in D_{\lambda e h b}. [\lambda z \in D_2. f(x) = 1 ]]\]

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. Kitley (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).


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 mathematician
sophisticated need only skim.

6 The notation "[x \in D : x sleeps]" is a standard abbreviation for "[x : x \in 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)))? 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 "entity", "=" for "truth-value".

10 D \times 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 \times A, which is defined
as \{x, y : x \in A \text{ and } y \in 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's" 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,

14 Most versions of the \lambda-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 intro-
duced 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 "\rightarrow \text{IN}" 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_2 are in \text{IN}. 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 \lambda-notation to define G, is to describe
G as the union of two separate functions with smaller domains: